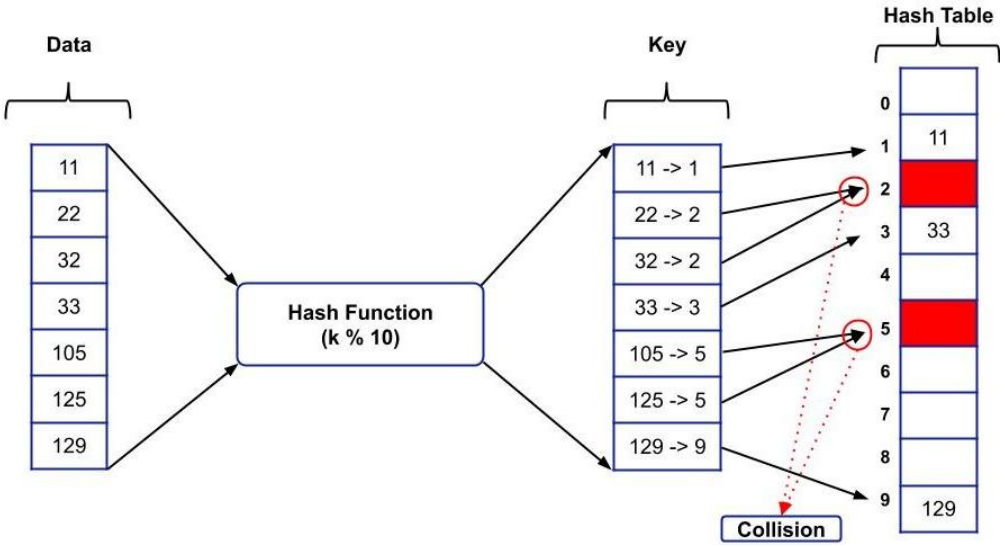
CHƯƠNG 5. HÀM BĂM BẢO MẬT

I. BĂM

Băm (hashing) là quá trình **chuyển đổi dữ liệu đầu vào có kích thước bất kỳ thành một chuỗi đầu ra có độ dài cố định, được gọi là giá trị băm.** Quá trình này đóng vai trò quan trọng trong việc lưu trữ, truy xuất và bảo mật dữ liệu.

Băm bao gồm *ba thành phần chính:*

- ***Khóa (key)*** có thể là chuỗi ký tự, số nguyên hoặc bất kỳ dạng dữ liệu nào, được sử dụng để tạo ra giá trị băm. Khóa giúp xác định vị trí lưu trữ hoặc chỉ số của dữ liệu trong một cấu trúc dữ liệu.
- ***Hàm băm (Hash function)*** là một thuật toán nhận khóa đầu vào và sinh ra một chỉ số băm (hash index), xác định vị trí của dữ liệu trong bảng băm. Hàm này được thiết kế để biểu diễn dữ liệu một cách ngắn gọn và đồng nhất.
- ***Bảng băm (Hash table)*** là một cấu trúc dữ liệu lưu trữ các cặp khóa-giá trị, trong đó hàm băm ánh xạ các khóa đến các vị trí cụ thể trong bảng. Mỗi giá trị trong bảng băm được liên kết với một chỉ số duy nhất, giúp truy xuất dữ liệu nhanh chóng.



Vì sao cần cấu trúc dữ liệu Hash?

- Dữ liệu trên internet tăng nhanh mỗi ngày, khiến việc lưu trữ và xử lý hiệu quả trở thành thách thức. Một cấu trúc dữ liệu rất phổ biến được sử dụng cho mục đích như vậy là cấu trúc dữ liệu Mảng.
- Trong lập trình, dù dữ liệu không lớn, vẫn cần lưu trữ và truy cập nhanh. Mảng là cấu trúc phổ biến, cho phép lưu trữ trong thời gian $T(1)$, nhưng tìm kiếm mất ít nhất $T(\log n)$, không hiệu quả với dữ liệu lớn.
- Vì vậy, cần một cấu trúc lưu trữ và tìm kiếm trong thời gian không đổi $T(1)$. Cấu trúc dữ liệu Hash ra đời để đáp ứng nhu cầu này, cho phép lưu trữ và truy xuất dữ liệu nhanh chóng trong $T(1)$.

Băm hoạt động như thế nào?

- Giả sử ta có một tập hợp các chuỗi {“ab”, “cd”, “efg”} và ta muốn lưu trữ nó trong một bảng.
- Mục tiêu chính là tìm kiếm hoặc cập nhật các giá trị được lưu trữ trong bảng một cách nhanh chóng trong thời gian ***T*(1)** và ta không quan tâm đến thứ tự của các chuỗi trong bảng.
- Vậy tập hợp các chuỗi trên có thể **vừa là khóa và vừa là giá trị.** Nhưng làm cách nào để lưu trữ giá trị tương ứng với khóa?
- **Bước 1:** Các hàm băm (là một số công thức toán học) được sử dụng để tính giá trị băm đóng vai trò là index của cấu trúc dữ liệu, nơi giá trị sẽ được lưu trữ.
- **Bước 2:** Gán số cho từng ký tự: “a” = 1, “b” = 2, ...
- **Bước 3:** Giá trị số bằng tổng của tất cả các ký tự của chuỗi **[sum(string)]**: “ab”=1+2=3, “cd”=3+4=7, “efg”= 5+6+7=18.
- **Bước 4:** Giả sử rằng ta có một bảng kích thước 7 phần để lưu trữ các chuỗi này. Hàm băm được sử dụng ở đây là **tổng của các ký tự trong key mod (Table size).** Từ đó, ta có thể tính toán vị trí **(index)** của từng chuỗi trong mảng bằng cách lấy **sum(string) mod (table size).**
- **Bước 5:** Vì vậy, ta sẽ lưu trữ “ab”: 3 mod 7 = 3 (index = 3), “cd”: 7 mod 7 = 0 (index = 0), “efg”: 18 mod 7 = 4 (index = 4). Minh họa như hình dưới:

0	1	2	3	4	5	6
cd			ab	efg		

Lưu ý

- **Đụng độ (collision)**: Ví dụ trên không có đụng độ (các chỉ số 0, 3, 4 khác nhau). Nếu hai chuỗi có cùng chỉ số, cần phương pháp giải quyết đụng độ (như danh sách liên kết hoặc dò tuyến tính).
- Thực tế, cần hàm băm phức tạp hơn ví dụ trên để phân bố đều dữ liệu, giảm đụng độ.

II. HÀM BĂM (HASH FUNCTION)

Hàm băm chuyển đổi dữ liệu có kích thước bất kỳ thành chuỗi ngắn gọn với độ dài cố định, đảm bảo tính toàn vẹn của dữ liệu mà không yêu cầu khóa. Chúng thường được xây dựng thông qua các thuật toán lặp đặc biệt, tạo ra giá trị băm độc đáo và hiệu quả. Ba đặc điểm chính của hàm băm:

- *Tính kháng tiền ảnh thứ nhất:* Khi một hàm băm h tạo ra giá trị băm a, việc tìm ra bất kỳ giá trị đầu vào x nào có giá trị băm a là rất khó khăn. Điều này đảm bảo tính chất mã hóa một chiều của hàm băm.
- *Tính kháng tiền ảnh thứ hai:* Khi một hàm băm h cho đầu vào x tạo ra giá trị băm $h(x)$, việc tìm thấy bất kỳ giá trị đầu vào y nào khác để $h(y) = h(x)$ là rất khó khăn.
- *Tính kháng va chạm:* Đối với một hàm băm h, rất khó để tìm thấy bất kỳ hai giá trị đầu vào x, y khác nhau sao cho $h(x) = h(y)$.

Tính bảo mật trong hàm băm:

- **Không thể đảo ngược**: Chưa chứng minh được sự tồn tại của hàm băm hoàn toàn không thể đảo ngược (tức là không thể tìm nguyên mẫu từ giá trị băm). Tuy nhiên, hàm băm mạnh phải khiến việc này cực kỳ khó về mặt tính toán.
- **Kháng xung đột (collision resistance)**:

- Tấn công "birthday" cho thấy: Với hàm băm có giá trị dài *n* bit, kẻ tấn công có thể tìm xung đột (hai đầu vào khác nhau cho cùng giá trị băm) trong khoảng $2^{\frac{n}{2}}$ phép tính.
- ***Ví dụ:*** Với SHA-256 ($n = 256$), độ phức tạp tìm xung đột là 2^{128}, vẫn rất lớn.
- Một hàm băm được coi là mạnh nếu độ phức tạp tìm xung đột gần với $2^{\frac{n}{2}}$.

- **Hiệu ứng tuyết lở (avalanche effect)**:
  - Một thay đổi nhỏ trong đầu vào (dù chỉ 1 bit) phải dẫn đến thay đổi lớn trong giá trị băm (khoảng 50% bit thay đổi).
  - Điều này đảm bảo giá trị băm không tiết lộ thông tin về đầu vào, ngay cả các bit riêng lẻ, tăng tính bảo mật.
- **Ứng dụng trong băm mật khẩu**: Thuộc tính trên giúp hàm băm tạo khóa an toàn từ mật khẩu người dùng, vì giá trị băm không để lộ thông tin về mật khẩu gốc.

Ứng dụng của hàm băm:

- **Xác minh chữ ký số**: Dùng trong chữ ký số (như ECDSA, RSA) để xác minh tính xác thực của dữ liệu. Ví dụ: SHA-256 băm thông điệp trước khi ký, đảm bảo tính toàn vẹn và xác thực.
- **Băm mật mã**: Tăng tính riêng tư và giảm tải cho cơ sở dữ liệu trung tâm bằng cách lưu trữ dữ liệu dưới dạng giá trị băm.
- **SSL handshake**: Trong "bắt tay" SSL/TLS, các hàm băm (như SHA) giúp trình duyệt và máy chủ đồng ý về khóa mã hóa và xác thực dữ liệu, chuẩn bị kết nối an toàn.
- **Kiểm tra tính toàn vẹn**: Dùng SHA-256 để kiểm tra tính toàn vẹn dữ liệu, đảm bảo dữ liệu không bị thay đổi trong quá trình truyền tải.

**** Phương pháp để tính hàm băm:***

- **Phương pháp phân chia:** Đây là phương pháp đơn giản nhất và dễ dàng nhất để tạo giá trị băm.
  - *Thực hiện chia giá trị khóa k cho kích thước bảng băm M, lấy phần dư làm giá trị băm.*
  - **Công thức**: $H(k) = k \bmod M$
    - *k:* Giá trị khóa.
    - *M:* Kích thước bảng băm (nên là *số nguyên tố* để phân bố đều).
  - **Ưu điểm**:
    - Đơn giản, nhanh (chỉ cần 1 phép chia).
    - Hoạt động tốt với nhiều giá trị *M*.
  - **Nhược điểm**:
    - Các khóa liên tiếp ánh xạ đến giá trị băm liên tiếp, dễ gây cụm (clustering).
    - Cần chọn *M* cẩn thận để tránh va chạm.

- **Phương pháp giữa bình phương (Mid Square Method)**
  - *Thực hiện bình phương giá trị khóa: k^2, sau đó lấy các chữ số ở giữa của k^2 làm giá trị băm.*
  - **Công thức**: H(*k*) = các chữ số giữa của (*k*·*k*).
  - **Ví dụ:** Bảng có 100 vị trí (cần 2 chữ số), k = 50. Tính $k^2 = 50 \cdot 50 = 2500$. Sau đó, lấy 2 chữ số giữa: 50 $\Rightarrow$ H(*k*) = 50.
  - **Ưu điểm**:

        - Hiệu suất tốt, vì tất cả chữ số của khóa đều góp phần vào kết quả.
        - Không phụ thuộc vào phân phối của chữ số đầu/cuối của khóa.
    - **Nhược điểm**:
        - Khóa lớn làm bình phương tăng gấp đôi số chữ số, khó xử lý.
        - Vẫn có thể xảy ra va chạm.

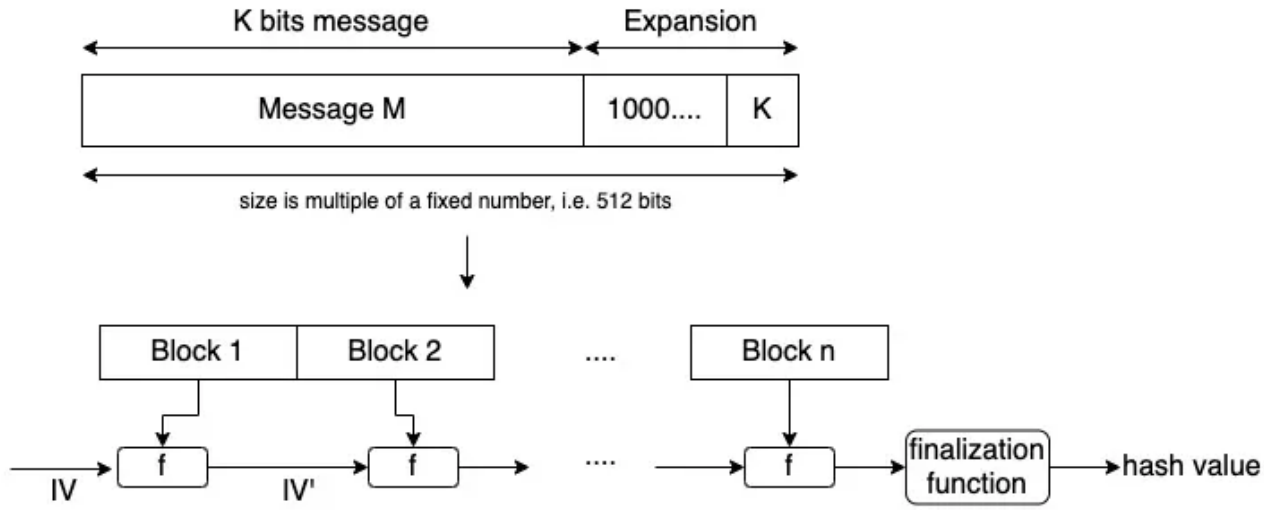
- **Phương pháp gấp (Folding Method):**
    - ***Bước 1:*** *Chia khóa k thành các phần có cùng số lượng chữ số (trừ phần cuối có thể ít số chữ số hơn).*
    - ***Bước 2:*** *Cộng các phần lại. Nếu tổng s có số chữ số vượt quá số chữ số mong muốn (dựa trên kích thước bảng băm), thì* ***bỏ đi chữ số cao nhất*** *(chữ số "mang" – carry), chỉ giữ lại các chữ số thấp hơn làm giá trị băm.*
    - **Công thức:** $\begin{cases} k = k_1, k_2, ..., k_n \\ s = k_1 + k_2 + ... + k_n \\ H(k) = s \end{cases}$
    - Ví dụ: k = 1234567, bảng kích thước 100 (mỗi phần 2 chữ số).
        - Chia: 12, 34, 56, 7
        - Cộng: 12 + 34 + 56 + 7 = 109.
        - Nhưng bảng chỉ cần 2 chữ số (vì kích thước bảng là 100), nên chữ số cao nhất: 1 là "lần mang cuối cùng". Bỏ chữ số 1, giữ lại 2 chữ số thấp hơn: 09.
        - Vậy giá trị băm H(k) = 09.

- **Phương pháp nhân (Multiplication Method)**
    - ***Bước 1:*** *Chọn hằng số a (0 < a < 1).*
    - ***Bước 2:*** *Nhân khóa k với a: $k \cdot a$.*
    - ***Bước 3:*** *Lấy phần thập phân của $k \cdot a$.*
    - ***Bước 4:*** *Nhân kết quả với kích thước bảng M.*
    - ***Bước 5:*** *Lấy phần nguyên (FLOOR) làm giá trị băm.*
    - **Công thức:** $H(k) = FLOOR\left[M.(k.a \bmod 1)\right]$
    - **Lưu ý:** Số lượng chữ số mỗi phần thay đổi tùy thuộc vào kích thước của bảng băm. Ví dụ: kích thước của bảng băm là 100 thì mỗi phần phải có 2 chữ số, trừ phần cuối cùng có thể có số chữ số ít hơn.
    - **Ví dụ:**
        - $k = 123456, a = 0.1516, M = 100.$
        - $k \cdot a = 123456 \cdot 0.1516 = 41654.0416.$
        - Phần thập phân: $0.0416.$
        - $M \cdot 0.0416 = 100 \cdot 0.0416 = 4.16.$
        - $H(k) = \text{FLOOR}(4.16) = 4.$
    - **Ưu điểm**:
        - Hoạt động với nhiều giá trị a từ 0 đến 1, mặc dù có 1 số giá trị có xu hướng cho kết quả tốt hơn các giá trị còn lại.
        - Nhanh nếu M là lũy thừa của 2.

- **Nhược điểm**:
  - Phù hợp nhất khi *M* là lũy thừa của 2, nếu không sẽ chậm hơn.
  - Cần chọn *a* cẩn thận để giảm va chạm.

III. CẤU TRÚC MERKLE-DAMGARD

Hàm băm Merkle-Damgard là một phương pháp xây dựng các hàm băm mật mã chống va chạm từ các hàm nén một chiều chống va chạm. Cấu trúc này được sử dụng trong thiết kế của nhiều thuật toán băm phổ biến như *MD5, SHA-1, SHA-2.*



Đầu tiên, ta tiến hành mở rộng thông điệp (*M*) đến độ dài là bội số của một số bit cụ thể, vì hàm nén chỉ xử lý đầu vào có kích thước xác định. Quá trình mở rộng bao gồm hai phần:

Khối đệm: Bắt đầu bằng bit '1' và theo sau là các bit '0'. Bit '1' đầu tiên đảm bảo tránh va chạm, ngăn trường hợp thông điệp M thêm một khối đệm '0', và *M'* là *M* thêm '0' ở cuối, tạo ra cùng giá trị băm, gây khó xác định số 0 sau *M* là khối đệm được thêm hay đó là phần cuối của *M'*.

Kích thước của khối thông điệp gốc: 64 bit cuối cùng (khối *K*) lưu trữ độ dài của thông điệp gốc, giúp ngăn chặn các cuộc tấn công mở rộng độ dài. Tổng độ dài của thông điệp gốc và khối đệm phải là bội số của kích thước khối trừ đi 64 bit dành cho khối kích thước.

Tiếp theo, ta tiến hành xử lý thông điệp đã mở rộng. Thông điệp đã mở rộng được chia thành các khối nhỏ và xử lý tuần tự. Mỗi khối được kết hợp với một vector khởi tạo (*IV*) thông qua hàm nén một chiều *f*, sử dụng các phép toán logic như *AND, OR, XOR*, v.v. Giá trị *IV* được cập nhật sau mỗi lần nén, và giá trị cuối cùng chính là giá trị băm. Kích thước của *IV* quyết định độ dài của giá trị băm đầu ra.

Cấu trúc Merkle-Damgard đảm bảo tính kháng va chạm nhờ hàm nén một chiều và quy trình mở rộng thông điệp chặt chẽ. Việc sử dụng khối đệm và khối kích thước giúp tăng cường bảo mật, ngăn chặn các cuộc tấn công như tấn công mở rộng độ dài, đồng thời duy trì tính toàn vẹn của dữ liệu.

IV. THUẬT TOÁN BĂM

- **Tổng quan:** Họ thuật toán SHA bao gồm các hàm băm: SHA-1, SHA-224, SHA-256, SHA-384, SHA-512.
  - **SHA-1**: Do NSA phát triển (1995), dài 160-bit, hiện không còn an toàn.
  - **SHA-2**: Gồm SHA-224, SHA-256, SHA-384, SHA-512, cũng do NSA phát triển, công bố bởi NIST (FIPS PUB 180-2, 2002). An toàn hơn SHA-1 nhờ độ dài bit lớn, bắt buộc dùng trong chữ ký số, chứng chỉ từ 2016.

- **SHA-3**: Thế hệ mới (2015), dựa trên cấu trúc bọt biển (sponge), gồm SHA-3-224, SHA-3-256, SHA-3-384, SHA-3-512.
- **Ứng dụng**: SSL, SSH, S/MIME, DNSSEC, X.509, PGP, IPSec, BitTorrent.

- **Các thuật toán băm an toàn phổ biến**
  - **SHA-0**: 160-bit, NIST đề xuất 1993, đã bị thay thế.
  - **SHA-1**: 160-bit, NIST đề xuất 1995, không còn an toàn, không được chấp nhận trong các chứng chỉ mới.
  - **SHA-2**: Nhóm hàm 224, 256, 384, 512-bit, an toàn hơn SHA-1.
  - **SHA-3**: Dựa trên cấu trúc bọt biển, khác với cấu trúc Merkle-Damgård của SHA-1, SHA-2.
  - **RIPEMD [RACE Integrity Primaries Assessment Message Digest (Chức năng Tóm tắt Đánh giá Toàn vẹn Nguyên thủy RACE)]**: Dựa trên MD4, có các phiên bản 128, 160, 256, 320-bit.
  - **Whirlpool**: Dựa trên Rijndael, dùng hàm nén Miaguchi-Preneel, tạo đầu ra cố định từ hai đầu vào.

- **Mô tả thuật toán**

Thuật toán	*Độ dài kết quả (bit)*	*Độ dài trạng thái (bit)*	*Độ dài khối (bit)*	*Độ dài DL tối đa (bit)*	*Độ dài từ (bit)*	*Số vòng lặp*
SHA-1	160	160	512	$< 2^{64}$	32	80
SHA-224	224	256	512	$< 2^{64}$	32	64
SHA-256	256	256	512	$< 2^{64}$	32	64
SHA-384	384	512	1024	$< 2^{128}$	64	80
SHA-512	512	512	1024	$< 2^{128}$	64	80

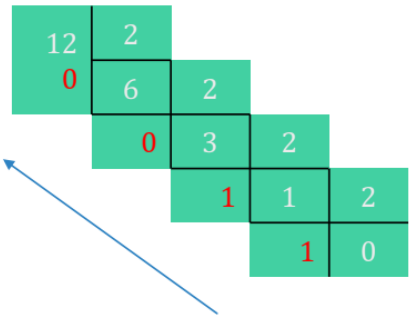
- **Tính bảo mật**
  - **SHA-1**: Từ 2011-2015 là chuẩn chính, nhưng bị phát hiện điểm yếu (Google tạo va chạm năm 2017). Không còn an toàn.
  - **SHA-2**: Từ 2016, trở thành tiêu chuẩn mới, an toàn hơn, bắt buộc trong chứng chỉ SSL/TLS.
  - **Tấn công brute-force**: Với SHA-2 (ví dụ SHA-256), độ phức tạp tìm va chạm là 2^{128}, mất nhiều năm/thập kỷ để phá.
  - **SHA-3**: Dùng cấu trúc bọt biển, cải tiến hơn, nhưng chưa phổ biến bằng SHA-2.

V. SECURE HASH ALGORITHM 256 – SHA256

1. Một số phép toán sử dụng trong SHA-256:

Bit	A binary digit having a value of 0 or 1
Byte	A group of eight bits
Hexadecimal Notation	Numbering system using a base of 16
Word	A group of 32 bits (4 bytes)
Hex Digit	Representation of a 4-bit string

Dec → Bin



Bin → Dec

$1100_2 \rightarrow 12_{10}$

2^3	2^2	2^1	0	$= 1\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 0\cdot2^0 = 12$
1	1	0	0	

Bin → Hex

$11100_2 \rightarrow 1C_{16}$

2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
0	0	0	1	1	1	0	0
$0\cdot2^3 + 0\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 = 1$				$1\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 0\cdot2^0 = 12 = C_{16}$			
1C							

Hex → Bin

$1C_{16} \rightarrow 11100_2$

1 = 1				C = 8 + 4			
2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
8	4	2	1	8	4	2	1
0	0	0	1	1	1	0	0

- **Toán tử bitwise** ∧, ∨, ⊕, ¬, ≪, ≫ (and, or, xor, not, shift left, shift right).

Phép toán dịch chuyển sang phải ($SHR^n(x) = x \gg n$)

- $SHR^1(1010) = 1010 \gg 1 = 0101_{(1)}$
- $SHR^3(1010) = 1010 \gg 3 = 0101_{(1)} = 0010_{(2)} = 0001_{(3)}$

Dịch tuần hoàn sang phải ($ROTR^n(x)$)

- $ROTR^1(1010) = 0101$
- $ROTR^3(1010) = 0101_{(1)} = 1010_{(2)} = 0101_{(3)}$

Cộng Modulo 2^{32} ($x = (a + b)\ mod\ 2^{32}$)

Ví dụ:

a = $100_{(10)} = 1100100_{(2)}$

b = $200_{(10)} = 11001000_{(2)}$

Dec:

$x = (100 + 200)\ mod\ 2^7 = 300\ mod\ 2^7 = 44_{10} = 101100_2$

Bin:

$x = (1100100 + 11001000)\ mod\ 10000000 = 100101100\ mod\ 10000000 = 101100$

$$\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$$

x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$ROTR^7$	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
$ROTR^{18}$	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SHR^3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
$\sigma_0(x)$	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	0	0	0	0	0	0	0

XOR

XOR

Dịch tuần hoàn sang phải 7 đơn vị.

Dịch sang phải 3 đơn vị.

$$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$$

x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$ROTR^7$	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$ROTR^{19}$	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
SHR^{10}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
$\sigma_1(x)$	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	1	1

XOR

XOR

$$\sum_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$$

x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$ROTR^2$	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
$ROTR^{13}$	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
$ROTR^{22}$	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
$\sum_0^{\{256\}}(x)$	0	0	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	0

XOR

XOR

$$\sum_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$$

x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$ROTR^6$	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
$ROTR^{11}$	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
$ROTR^{25}$	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
$\sum_1^{\{256\}}(x)$	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0

XOR

XOR

$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$

x	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
z	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Ch	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$

x	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
z	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Maj	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

2. Chi tiết về SHA-256:

SHA-256 (Secure Hash Algorithm 256-bit) là một thuật toán băm mật mã thuộc gia đình SHA-2, được phát triển bởi Cơ quan An ninh Quốc gia Hoa Kỳ (NSA) và công bố bởi Viện Tiêu chuẩn và Công nghệ Quốc gia (NIST) vào năm 2001. Thuật toán này tạo ra một giá trị băm có độ dài cố định 256 bit (tương đương 32 byte) từ dữ liệu đầu vào bất kỳ.

SHA-256 xử lý dữ liệu đầu vào thông qua một loạt các bước tính toán phức tạp, bao gồm nhiều vòng lặp và các phép toán mật mã. Quá trình này được thiết kế để đảm bảo tính bảo mật, bao gồm tính kháng va chạm, kháng tiền ảnh, và khả năng bảo vệ tính toàn vẹn dữ liệu. Kết quả là một chuỗi băm 256 bit duy nhất, đại diện cho dữ liệu gốc một cách an toàn và hiệu quả.

SHA-256 được sử dụng rộng rãi trong các ứng dụng bảo mật như xác minh chữ ký số, bảo vệ mật khẩu, và đảm bảo an toàn cho các giao thức như SSL/TLS. Với độ dài đầu ra cố định và khả năng bảo mật cao, SHA-256 là một công cụ quan trọng trong việc duy trì tính an toàn và tin cậy của dữ liệu trong các hệ thống hiện đại.

Quy trình thực hiện của thuật toán băm SHA-256:

2.1. Tiền xử lý

- Chuyển đổi tin nhắn ban đầu sang hệ nhị phân bằng cách sử dụng bảng mã ASCII.
  - **Ví dụ:** input: **abc** → byte [97,98,99] → Bin: 01100001 01100010 01100011

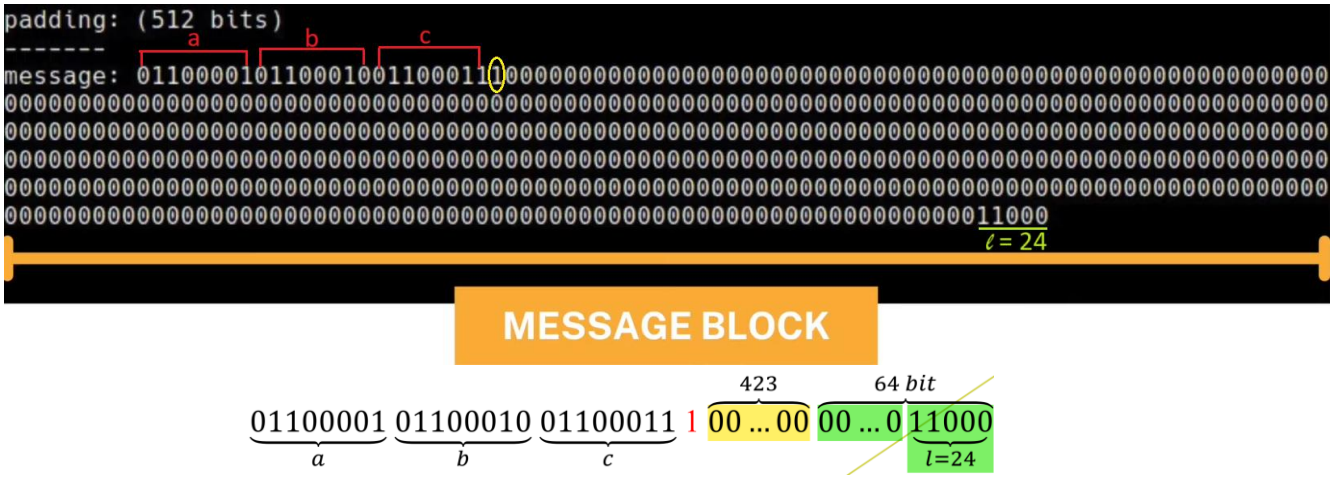
Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

- Thêm bộ đệm tin nhắn (padding the message) như đã mô tả ở mục III, để đảm bảo rằng chiều dài của tin nhắn được đệm là bội số của 512 bits. Ta tiến hành thêm 1 bit "1" vào cuối chuỗi tin nhắn đã chuyển sang nhị phân, sau đó đệm thêm *k* bit '0', với *k* là nghiệm không âm nhỏ nhất của phương trình:

$$l + 1 + k \equiv 448 \bmod 512$$

Với l là chiều dài của chuỗi tin nhắn nhị phân. Cuối cùng, 64 bits cuối được dành riêng để chỉ ra độ dài của tin nhắn thực tế ở dạng nhị phân.



- *Phân tích cú pháp tin nhắn đệm:*
  - Nếu chuỗi tin nhắn đã được đệm dài hơn 512 bits, thì chia chuỗi tin nhắn đó thành *N* khối, mỗi khối có kích thước 512 bits, được ký hiệu lần lượt là $M^{(1)}, M^{(2)}, ..., M^{(N)}$.
  - Mỗi khối thứ *i* có kích thước 512 bits lại được chia nhỏ thành 16 khối nữa, mỗi khối có kích thước 32 bits, và được ký hiệu lần lượt là $M_0^{(i)}, M_1^{(i)}, ...M_{15}^{(i)}$. Đây chính là *lịch trình tin nhắn ban đầu (message schedule).*

```
----------------
message schedule:
----------------
M0 = W0  01100001011000100110001110000000
M1 = W1  00000000000000000000000000000000
M2 = W2  00000000000000000000000000000000
M3 = W3  00000000000000000000000000000000
M4 = W4  00000000000000000000000000000000
M5 = W5  00000000000000000000000000000000
M6 = W6  00000000000000000000000000000000
M7 = W7  00000000000000000000000000000000
M8 = W8  00000000000000000000000000000000
M9 = W9  00000000000000000000000000000000
M10 = W10 00000000000000000000000000000000
M11 = W11 00000000000000000000000000000000
M12 = W12 00000000000000000000000000000000
M13 = W13 00000000000000000000000000000000
M14 = W14 00000000000000000000000000000000
M15 = W15 00000000000000000000000000011000
```

  - Với mỗi khối *i* thuộc $M^{(1)}, M^{(2)}, ..., M^{(N)}$ ($i = 1, ..., N$), ta tiến hành mở rộng lịch trình tin nhắn bằng cách thêm 48 từ 32 bit theo công thức sau:

$$W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}$$

Với $M_t^{(i)}$: từ 32 bit lấy từ 512 bit của khối tin nhắn.

W_t: từ 32 bit được tạo từ công thức trên.

Lưu ý: $\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$

$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$

$ROTR^n(x)$: phép dịch tuần hoàn sang phải *n* vị trí của chuỗi nhị phân *x*.

$SHR^n(x)$: Phép toán dịch chuyển *n* bit sang phải.

```
W0  01100001011000100110001110000000 ->    01100001011000100110001110000000
W1  00000000000000000000000000000000 -> σ0 00000000000000000000000000000000
W2  00000000000000000000000000000000
W3  00000000000000000000000000000000
W4  00000000000000000000000000000000
W5  00000000000000000000000000000000
W6  00000000000000000000000000000000
W7  00000000000000000000000000000000
W8  00000000000000000000000000000000
W9  00000000000000000000000000000000 ->    00000000000000000000000000000000
W10 00000000000000000000000000000000
W11 00000000000000000000000000000000
W12 00000000000000000000000000000000
W13 00000000000000000000000000000000
W14 00000000000000000000000000000000 -> σ1 00000000000000000000000000000000
W15 00000000000000000000000000011000
W16 01100001011000100110001110000000 = σ1(t-2) + (t-7) + σ0(t-15) + (t-16)
```

...

```
W47 00000110010111000100001111011010
W48 11111011001111101000100111001011
W49 11001100011101100001011111011011
W50 10111001111001100110110000110100
W51 10101001100110010011011001100111
W52 10000100101110101101111011011101
W53 11000010000101000110001010111100
W54 00010100100001110100011100101100
W55 10110010000011110111101010011001
W56 11101111010101111011100111001101
W57 11101011111001101011001000111000
W58 10011111111000110000100101011110
W59 01111000101111001000110101001011
W60 10100100001111111100111100010101
W61 01100110100010110010111111111000
W62 11101110101010111010001011001100
W63 00010010101100011110110111101011
```

- ***Thiết lập giá trị băm ban đầu:*** Bao gồm 8 từ (word), mỗi từ 32 bit, bắt nguồn từ 32 bits đầu tiên các phần thập phân của căn bậc hai 8 số nguyên tố đầu tiên, bao gồm:

$$H_0^{(0)} = tp\left(\sqrt{2}\right) = 6a09e667 \qquad H_4^{(0)} = tp\left(\sqrt{11}\right) = 510e527f$$

$$H_1^{(0)} = tp\left(\sqrt{3}\right) = bb67ae85 \qquad H_5^{(0)} = tp\left(\sqrt{13}\right) = 9b05688c$$

$$H_2^{(0)} = tp\left(\sqrt{5}\right) = 3c6ef372 \qquad H_6^{(0)} = tp\left(\sqrt{17}\right) = 1f83d9ab$$

$$H_3^{(0)} = tp\left(\sqrt{7}\right) = a54ff53a \qquad H_7^{(0)} = tp\left(\sqrt{19}\right) = 5be0cd19$$

```
H0 = 01101010000010011110011001100111
H1 = 10111011011001111010111010000101
H2 = 00111100011011101111001101110010
H3 = 10100101010011111111010100111010
H4 = 01010001000011100101001001111111
H5 = 10011011000001010110100010001100
H6 = 00011111100000111101100110101011
H7 = 01011011111000001100110100011001
```

- ***Hằng số (constants):*** Bao gồm 64 từ, mỗi từ 32 bit, từ các **phần thập phân căn bậc ba của 64 số nguyên tố đầu tiên** được ký hiệu lần lượt: $K_0^{\{256\}}, K_1^{\{256\}}, ..., K_{63}^{\{256\}}$. Các từ hằng số theo hệ 16 này là (từ trái sang phải):

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2
```

2.2. Tính toán băm

- *a)* Thực hiện mở rộng lịch trình tin nhắn sao cho có tổng cộng 64 từ (word) theo mục 5.3.1.
- *b)* Khởi tạo 8 biến làm việc a, b, c, d, e, f, g, h, là vị trí nơi các từ sẽ được lưu trữ trong quá trình nén, và chính là 8 giá trị băm ban đầu.

$$a = H_0^{(i-1)} = tp\left(\sqrt{2}\right) \qquad e = H_0^{(i-1)} = tp\left(\sqrt{11}\right)$$
$$b = H_1^{(i-1)} = tp\left(\sqrt{3}\right) \qquad f = H_1^{(i-1)} = tp\left(\sqrt{13}\right)$$
$$c = H_2^{(i-1)} = tp\left(\sqrt{5}\right) \qquad g = H_2^{(i-1)} = tp\left(\sqrt{17}\right)$$
$$d = H_3^{(i-1)} = tp\left(\sqrt{7}\right) \qquad h = H_3^{(i-1)} = tp\left(\sqrt{19}\right)$$

- *c)* *Chạy chu kỳ nén:* Cập nhật các giá trị của các biến làm việc bằng cách:

For $t = 0$ to 63:

{

$$T_1 = h + \sum_1^{\{256\}}(e) + Ch(e, f, g) + K_t^{\{256\}} + W_t$$
$$T_2 = \sum_0^{\{256\}}(a) + Maj(a, b, c)$$
$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

}

Với: T_1: từ tạm 1.

T_2: từ tạm 2.

a, b, c, d, e, f, g, h: các biến làm việc.

$\sum_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x)$.

$\sum_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x)$.

$K_t^{\{256\}}$: các hằng số (constants).

W_t : Các từ trong lịch trình tin nhắn.

$Maj(a,b,c) = (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c)$: Hàm Majority.

$Ch(e,f,g) = (e \wedge f) \oplus (\neg e \wedge g)$: Hàm chọn.

Ví dụ: Xét $t = 0$:

```
compression: (T1)
-----------
W0 = 01100001011000100110001110000000 (message schedule)
K0 = 01000010100010100010111110011000 (constant)

T1 = Σ1(e) + Ch(e, f, g) + h + K0  + W0 = 01010100110110100101000011101000


a = 01101010000010011110011001100111
b = 10111011011001111010111010000101
c = 00111100011011101111001101110010
d = 10100101010011111111101010011010
e = 01010001000011100101001001111111    00110101100001110010011100101011 Σ1(e)
f = 10011011000001010110100010001100
g = 00011111100000111101100110101011    00011111110000101110010011000110 Ch(e, f, g)
h = 01011011111000000110011010011001    01011011111100000110011010011001 h
```

```
compression: (T2)
-----------
W0 = 01100001011000100110001110000000 (message schedule)
K0 = 01000010100010100010111110011000 (constant)

T1 = Σ1(e) + Ch(e, f, g) + h + K0  + W0
T2 = Σ0(a) + Maj(a, b, c) = 00001000100100000100110101110010 1

a = 01101010000010011110011001100111    11001110001000001011010001111110 Σ0(a)
b = 10111011011001111010111010000101
c = 00111100011011101111001101110010
d = 10100101010011111111101010011010    00111010011011111111001100111 Maj(a, b, c)
e = 01010001000011100101001001111111
f = 10011011000001010110100010001100
g = 00011111100000111101100110101011
h = 01011011111000000110011010011001
```

Cập nhật giá trị các biến làm việc:

```
a = 01011101011010101110101111001101
b = 01101010000010011110011001100111
c = 10111011011001111010111010000101
d = 00111100011011101111001101110010
e = 11111010001010100100011000100010
f = 01010001000011100101001001111111
g = 10011011000001010110100010001100
h = 00011111100000111101100110101011
```

Thực hiện 63 vòng còn lại tương tự như trên. Cuối cùng ta được giá trị các biến làm việc như sau:

```
a = 01010000011011100011000001011000
b = 11010011100110100010000101100101
c = 00000100110100100100110101101100
d = 10111000010111100010110011101001
e = 01011110111101010000111100100100
f = 11111011000100100001001000010000
g = 10010100100011010010010110110110
h = 10010110000111111010010010010100
```

- *d) Tính giá trị hàm trung gian:* Lấy giá trị băm ban đầu, rồi cộng thêm biến làm việc được cập nhật tương ứng.

$$H_0^{(i)} = a + H_0^{(i-1)} \qquad H_4^{(i)} = a + H_4^{(i-1)}$$
$$H_1^{(i)} = b + H_1^{(i-1)} \qquad H_5^{(i)} = b + H_5^{(i-1)}$$
$$H_2^{(i)} = a + H_2^{(i-1)} \qquad H_6^{(i)} = a + H_6^{(i-1)}$$
$$H_3^{(i)} = a + H_3^{(i-1)} \qquad H_7^{(i)} = a + H_7^{(i-1)}$$

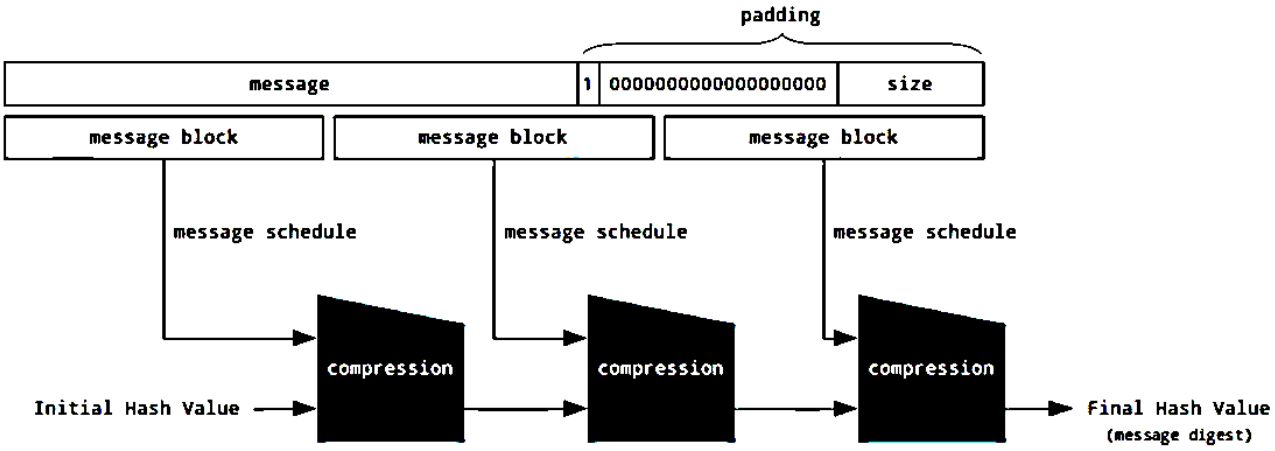
Tính H^1

```
   H0:
H0 = 01101010000010011110011001100111 + 01010000011011100011000001011000
H1 = 10111011011001111010111010000101 + 11010011100110100010000101100101
H2 = 00111100011011011111001101110010 + 00000100110100100100110101101100
H3 = 10100101010011111111010100111010 + 10111000010111100010110011101001
H4 = 01010001000011100101001001111111 + 01011110111101010000111100100100
H5 = 10011011000001010110100010001100 + 11111011000100100001001000010000
H6 = 00011111100000111101100110101011 + 10010100100011010010010110110110
H7 = 01011011111000001100110100011001 + 10010110000111111010010010010100
```

Sau tính toán, ta được các giá trị hàm trung gian là:

```
H0 = 10111010011110000001011010111111
H1 = 10001111000000011100111111101010
H2 = 01000001010000010100000011011110
H3 = 01011101101011100010001000100011
H4 = 10110000000000110110000110100011
H5 = 10010110000101110111101010011100
H6 = 10110100000100001111111101100001
H7 = 11110010000000000001010110101101
```

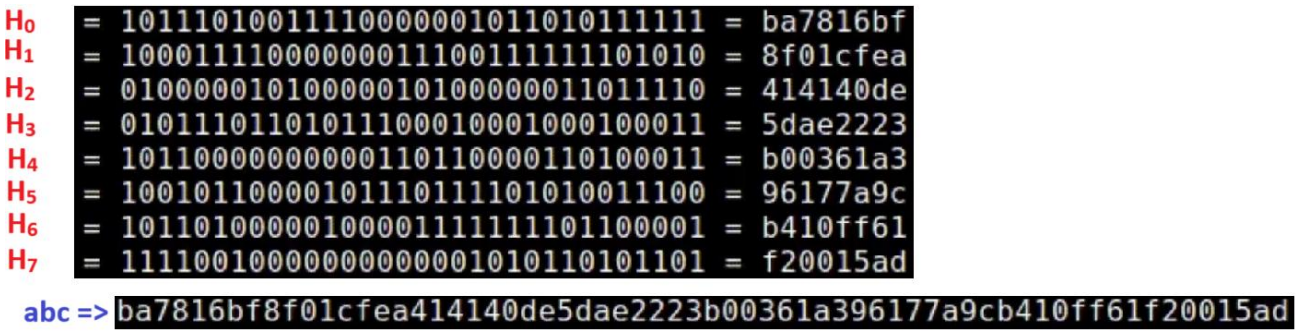
Tiếp tục thực hiện $N-1$ lần các bước *a), b), c), d)* với các khối 512 bits còn lại (nếu có).



Sau khi thực hiện ta có các giá trị băm:

$$H_0^{(N)}, H_1^{(N)}, H_2^{(N)}, H_3^{(N)}, H_4^{(N)}, H_5^{(N)}, H_6^{(N)}, H_7^{(N)}$$

- *e)* Chuyển đổi giá trị băm $H_0^{(N)},...,H_7^{(N)}$ từ hệ nhị phân sang hệ Hex(16), sau đó nối các giá trị này lại với nhau. Cuối cùng ta được một chuỗi tin nhắn đã được chuyển đổi có kích thước 256 bits.



abc => ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad