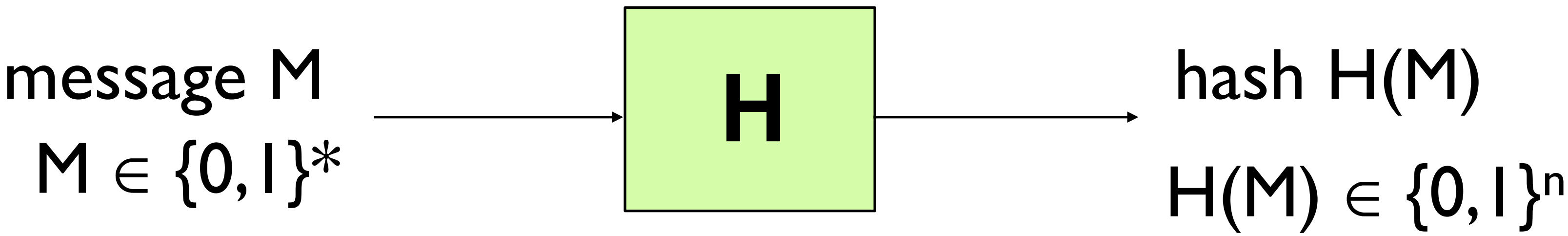
Symmetric Crypto
Hash function

Pierre-Alain Fouque

Hash Function

- Def:



A hash function H compute a hash value, a.k.a. fingerprint of n bits for a given arbitrary long message M

$$H : \{0,1\}^* \rightarrow \{0,1\}^n$$

- Usage: integrity, password storage, signature, ...
- Eg: SHA-1 (160 bits), MD5 (128 bits), SHA-2, ...

Use cases: File integrity

- Idea : we want to detect if a file has been modified

  by recomputing its fingerprint

```c
// Fichier code.c

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char** argv)
{
  if (argc <2)
    {
      …
    }
}
```

SHA-1 →

Hash Length of 160 bits :

SHA-1 (code.c) =
A51F 07BB 62EC 44A3 F118

Use cases: Passwords

- Instead of storing a password on a machine, we store its hash

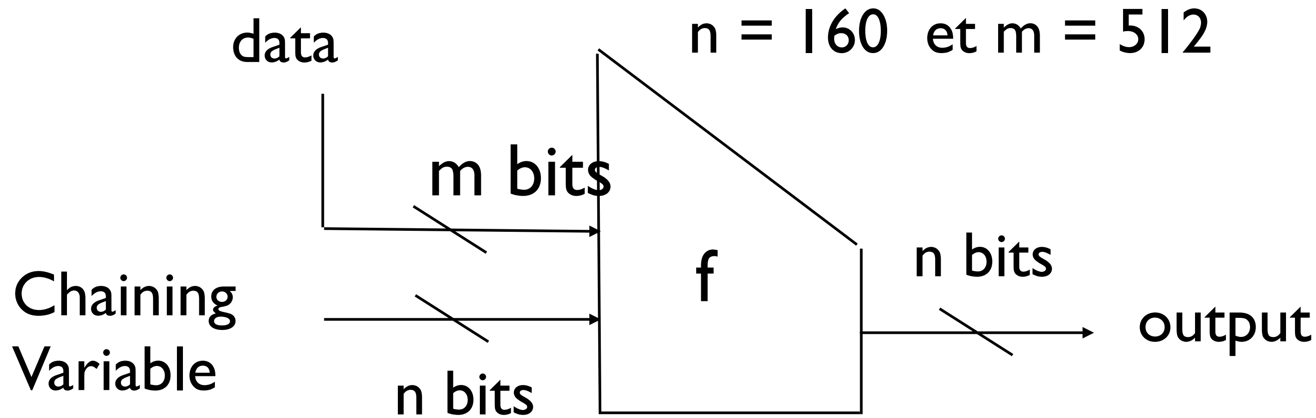
$$h = H(\textit{password})$$

- To authenticate, the user must send h
- On the web, the server sends a random value N and the user must answer with H(N||Password)

Compression Function

- f a compression function $f:\{0,1\}^{m+n} \rightarrow \{0,1\}^n$
- Fixed-Length hashing function

For SHA-1 :
n = 160 et m = 512

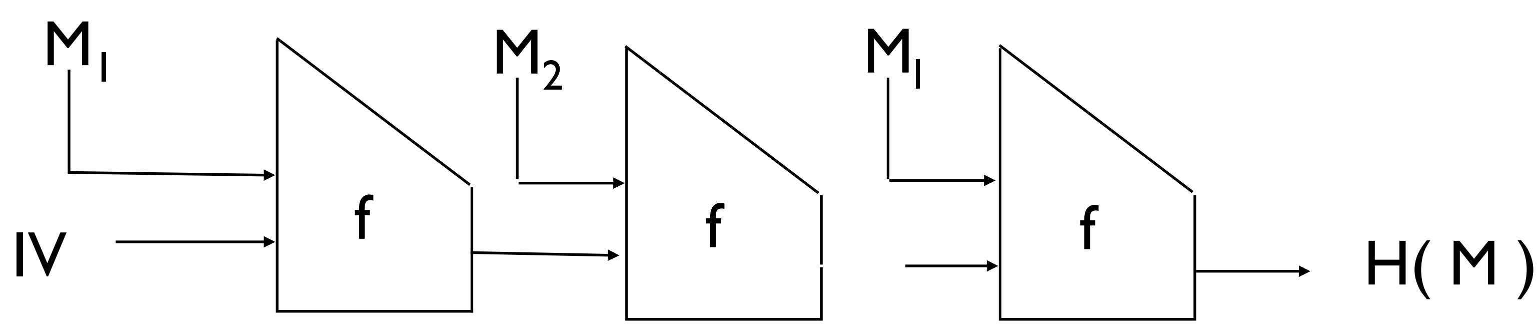


Merkle-Damgard

- f a compression function $f:\{0,1\}^{m+n} \rightarrow \{0,1\}^n$
- Let $M = M_1 \,||\, \ldots \,||\, M_m$ a message to hash (l blocks of m bits)
- Pad includes the length of M
- <u>Construction</u>:

$$H^f(M): h_1=f(IV,M_1),\ h_2=f(h_1,M_2),\ \ldots,\ h_n=f(h_{n-1},Pad)$$

- <u>Th</u>: If we have a collision on H^f, then we have a collision on f



Security notions

- Collision Resistance

  Find M_1 and M_2 such that $H(M_1) = H(M_2)$ ($2^{n/2}$ + Pollard)

- Second-preimage Resistance

  Given M_1, find M_2 such that $H(M_1) = H(M_2)$ (2^n)

- Preimage Resistance

  Given x, find M such that $H(M) = x$ (2^n)

Security expectations

- One-way: given y, find x s.t. H(x)=y (One-Time Password)
- Random Oracle: there should be no shortcut for knowing H(m) better than computing it !
- Usages: Key derivations, MAC, signatures

Length extension

- Attacks:
  - Could you predict the value of H(M) without having to recompute from the beginning ?

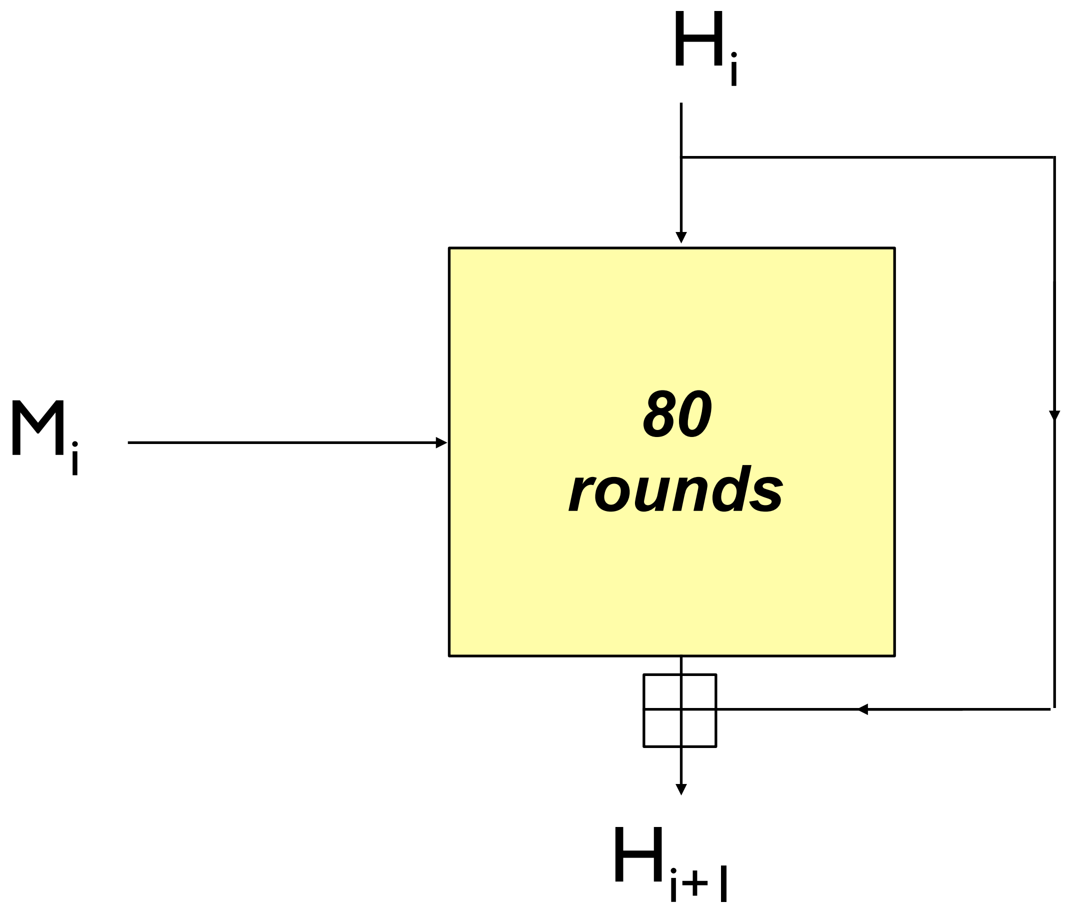
SHA

- SHA published by NIST in 1993
- Tweaked in 1995 : version SHA-1
- New versions in 2002, SHA-2 (SHA-256, 384, 512)

- Iterated Hash Function
- Compression Function: Generalized Feistel

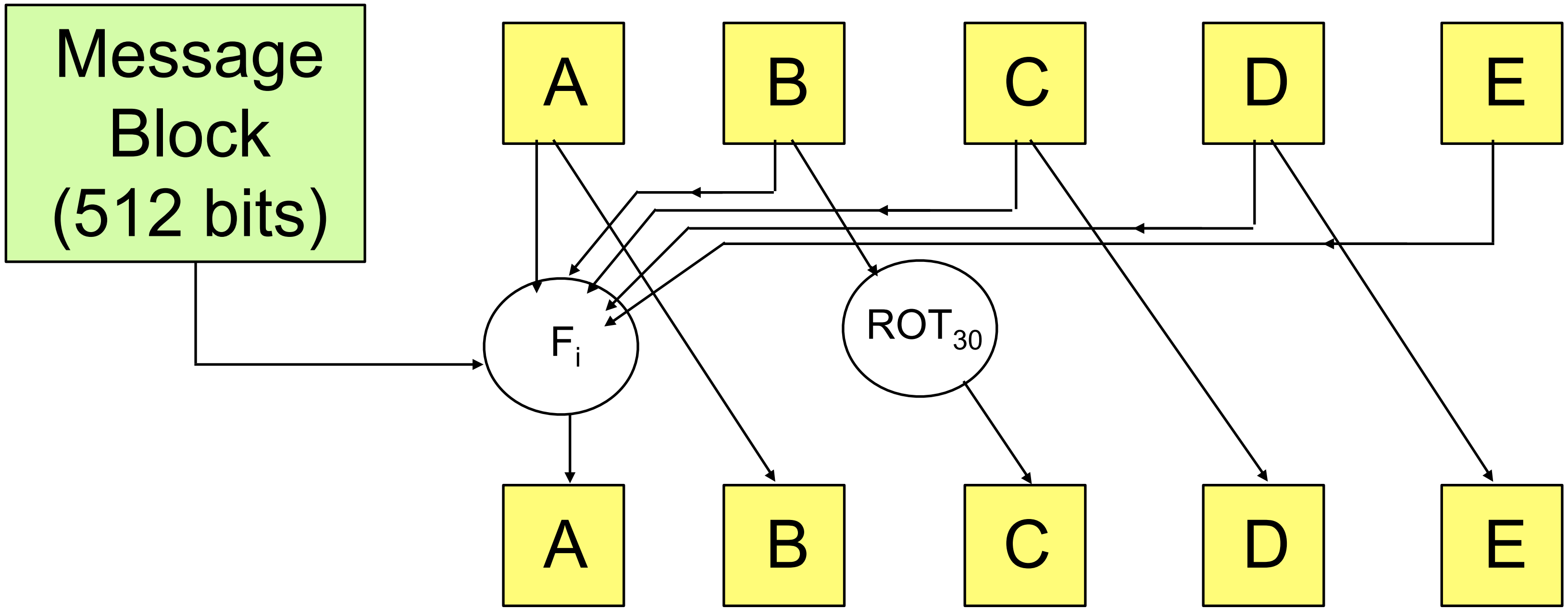
SHA

The round function is invertible (Generalized Feistel) !
We apply Davies - Meyer



SHA

The compression function uses 80 rounds :



All words have 32 bits

SHA

5 words of 32 bits A,B,C,D,E

A = Iv[0]; B = Iv[1]; C = Iv[2]; D = Iv[3]; E = Iv[4];

```
For i = 1, … , 80
    {
        A = (A<<<5) + f_i(B,C,D) + E + Cst[i] + W[i]
        B = A
        C = (B<<<30)
        D = C
        E = D
    }
```

Derived from the message block (points to W[i])

SHA

- The function $f_i(B,C,D)$ is a boolean bitwise function, chosen among IF, XOR, MAJORITY
- The 32-bit words W[i] are derived from the message blocks $(M[i])_{i=0,\ldots,15}$ using

$$W[i] = M[i] \text{ for } i=0 \ldots 15$$

$$W[i] = (W[i-3] \oplus W[i-8] \oplus W[i-14] \oplus W[i-16])_{<<<1}$$

$$\text{for } i \geq 16$$

Difference with SHA-0

SHA-3

- Keccak hash function in 2008
- Designed by G. Bertoni, J. Daemen, M. Peters and G. van Asche (ST and NXP)