MAC

Hank Chen

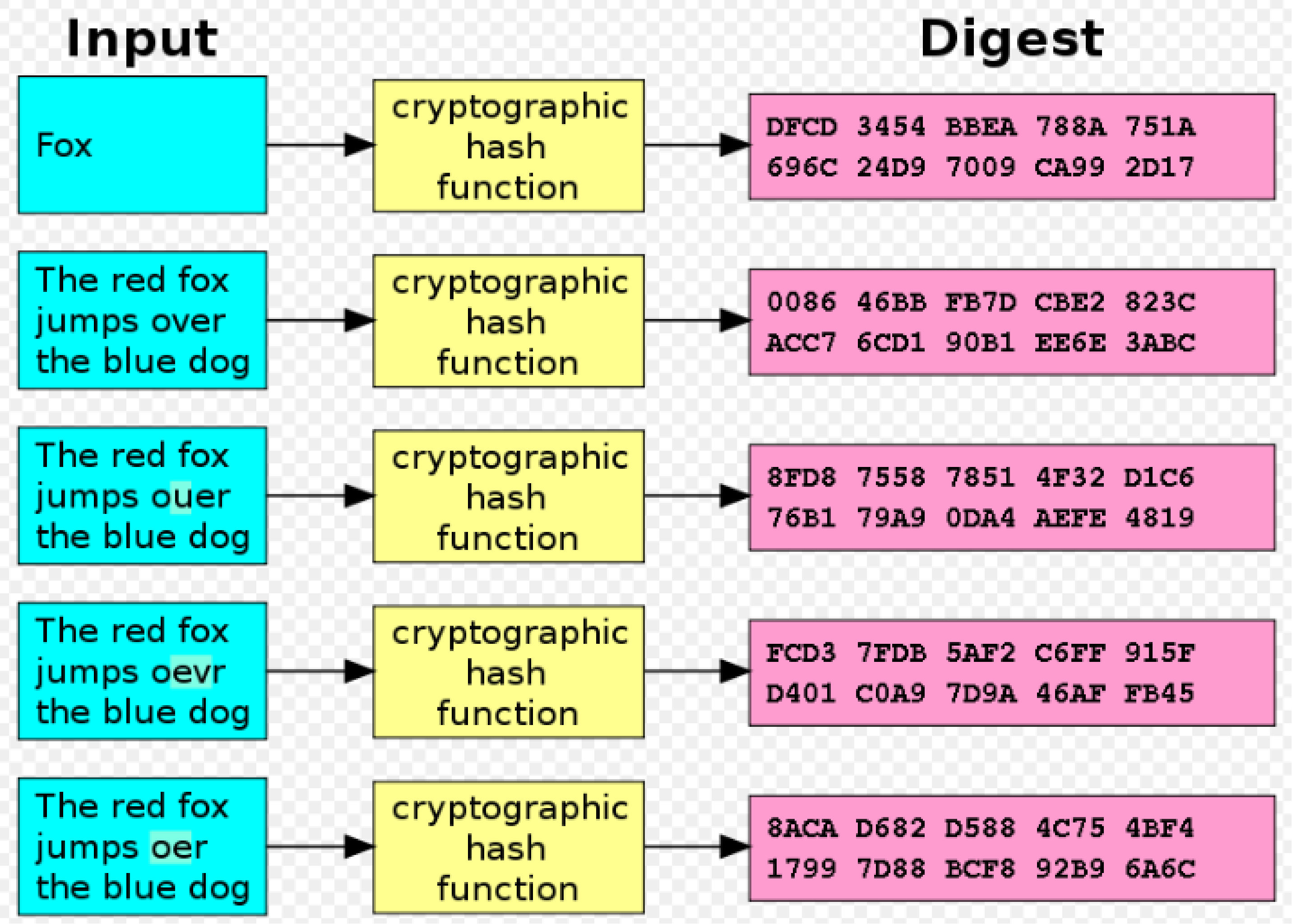
Outline

- Cryptographic Hash Function Criteria
- Collision Theorem
- Merkle–Damgård Construction
- Message Authentication Code (MAC)

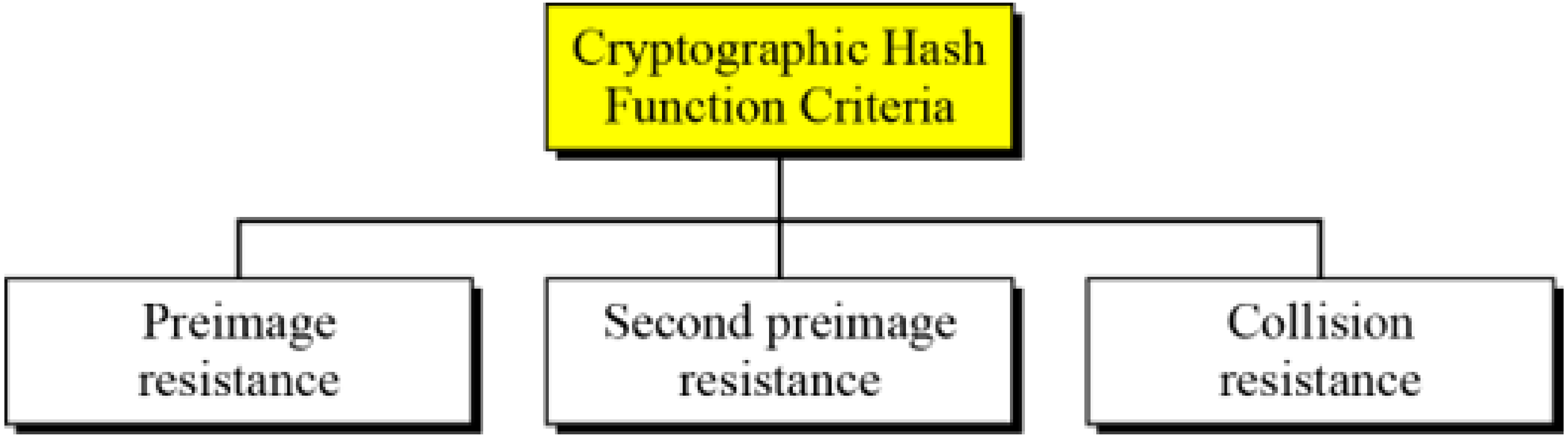
Cryptographic Hash Function Criteria

Cryptographic hash function

- variant input size
- fixed output size (digest)



Cryptographic Hash Function Criteria

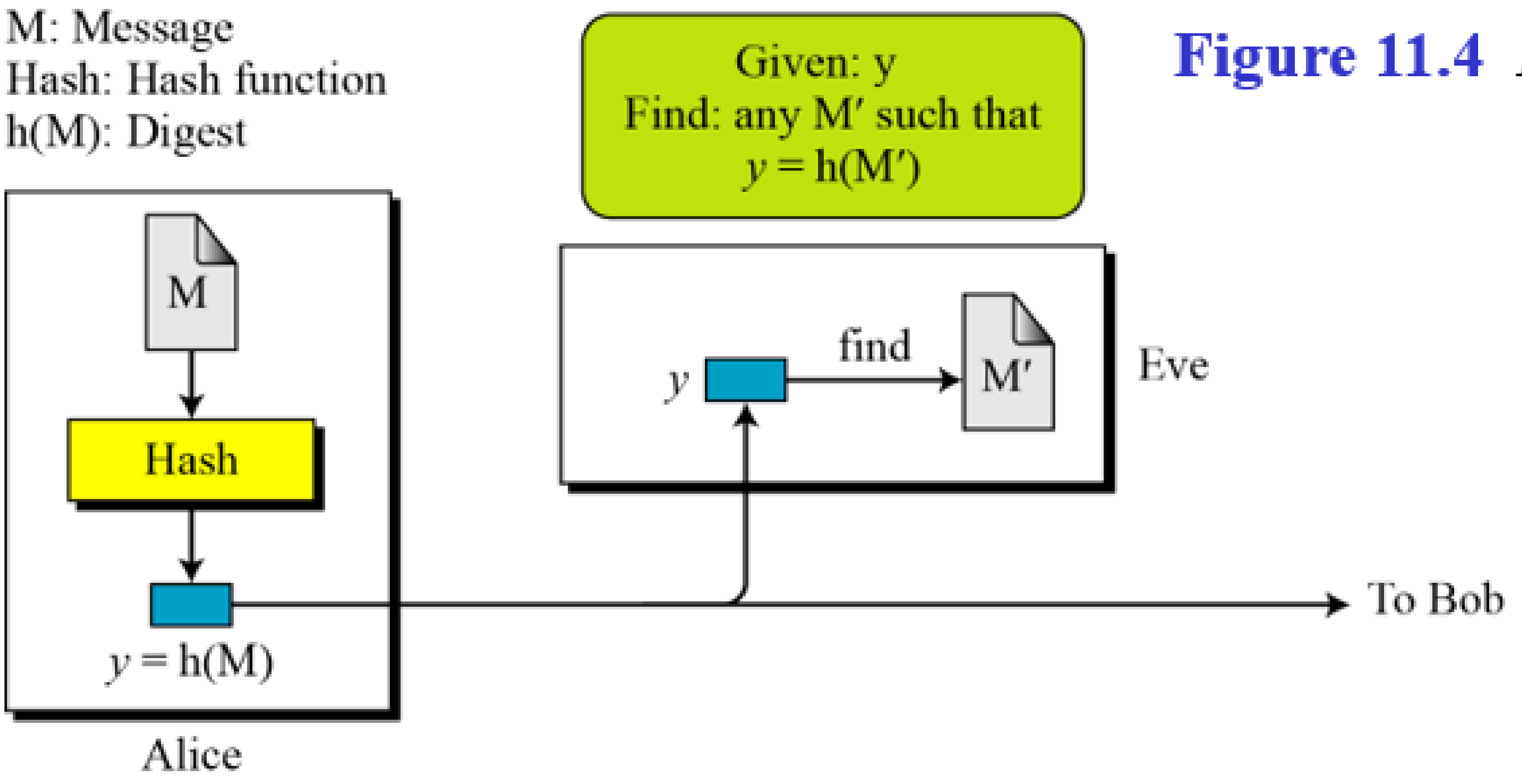


Preimage Resistance

Preimage Attack

Given: $y = h(M)$ **Find: M' such that $y = h(M')$**

Figure 11.4 ***Preimage***

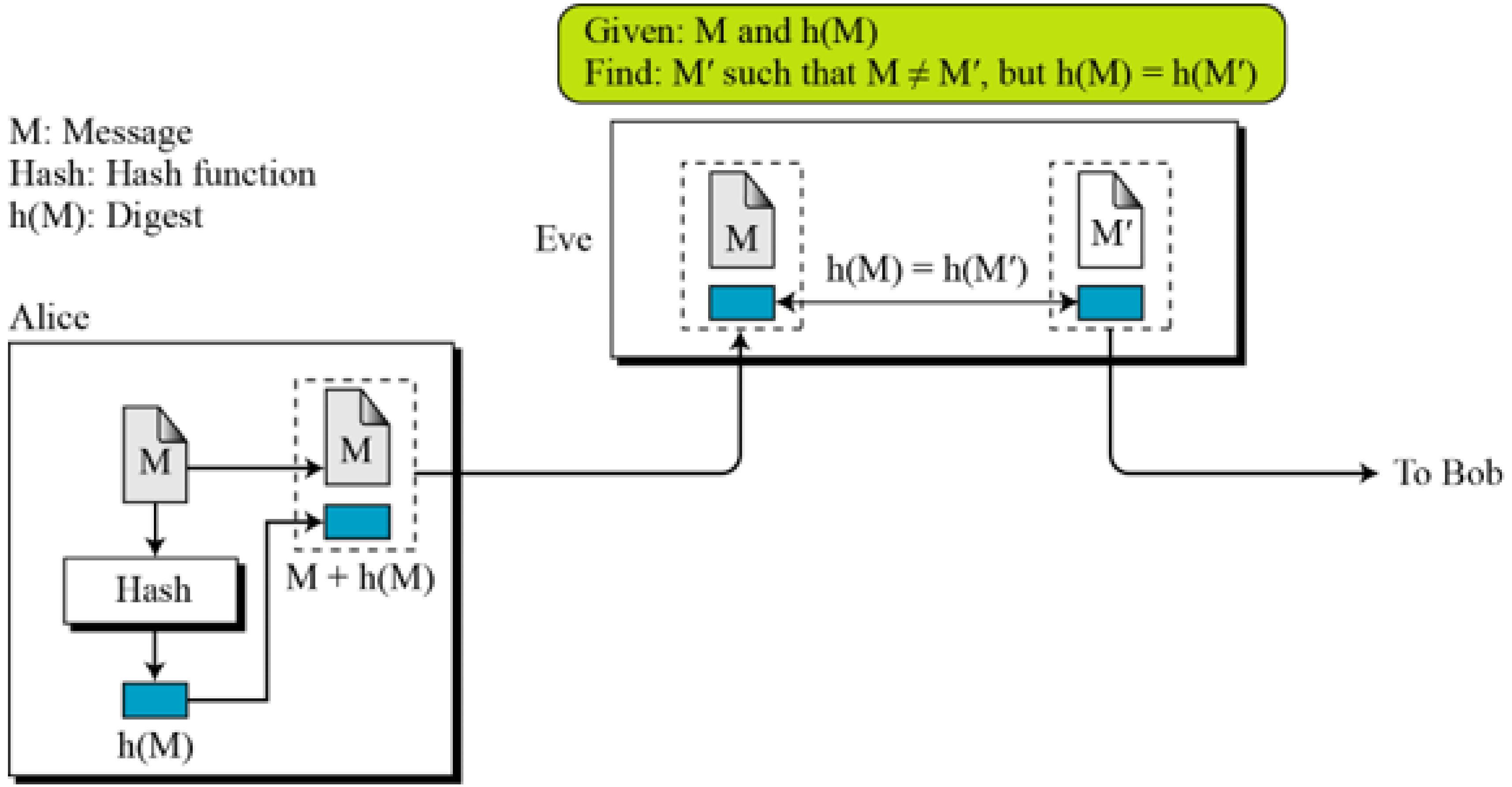


Second Preimage Resistance

Second Preimage Attack

Given: M and h(M) **Find: M′ ≠ M such that h(M) = h(M′)**

Figure 11.5 ***Second preimage***

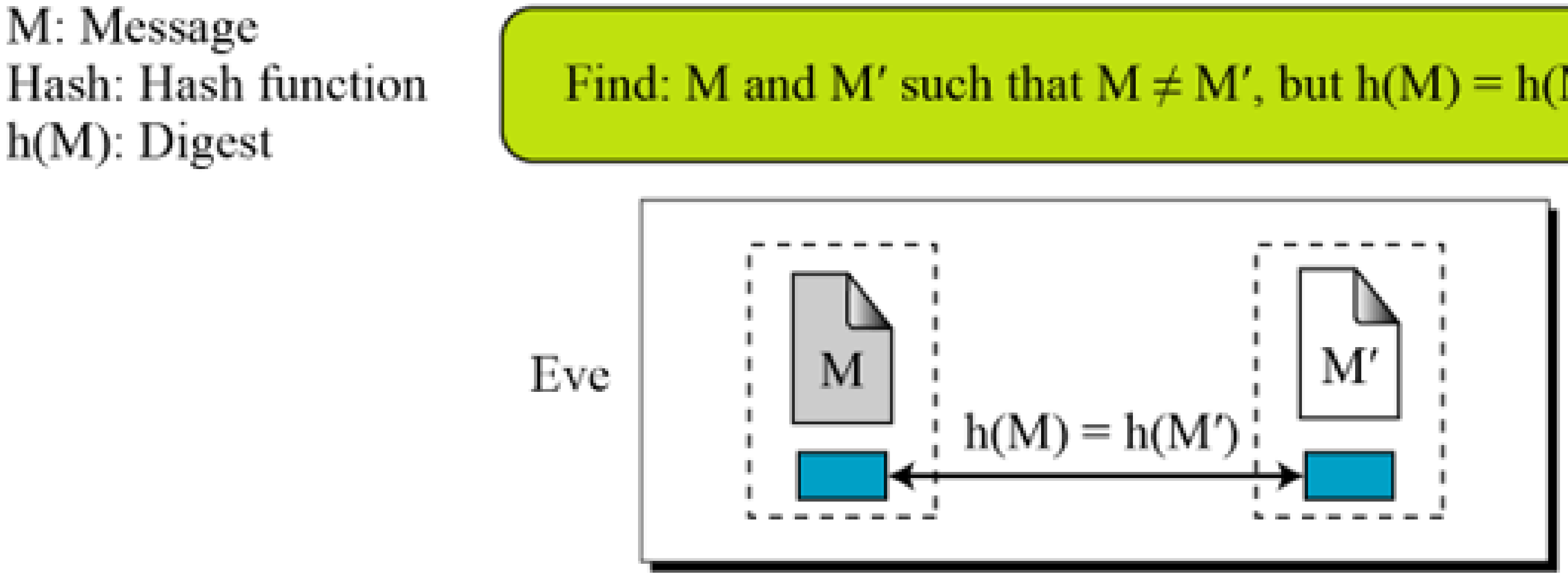


Collision Resistance

Collision Attack

Given: none **Find: $M' \neq M$ such that $h(M) = h(M')$**

Figure 11.6 *Collision*



Collision Theorem

Birthday Paradox

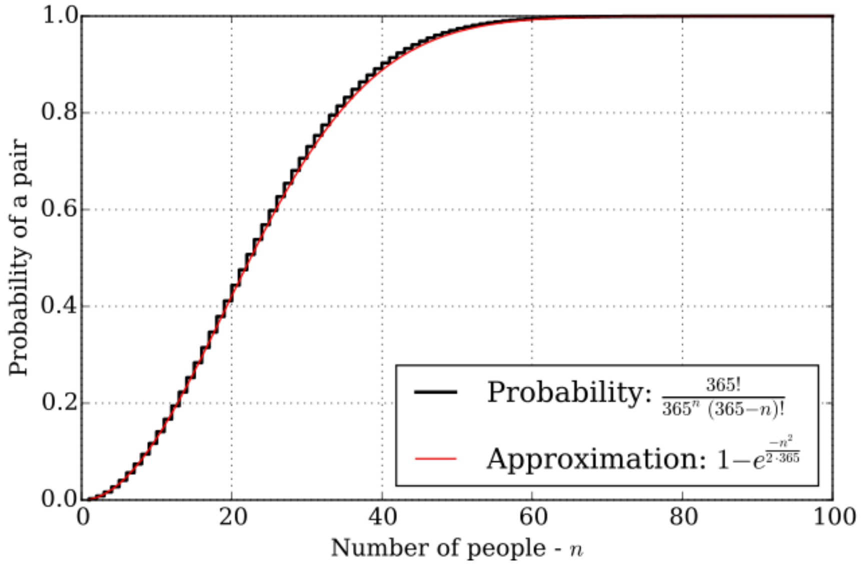
- In a group of 40 people
  1. What is the probability that someone has the same birthday as you? (弱碰撞)

$$
\begin{aligned}
\Pr\begin{pmatrix}\text{someone has}\\ \text{your birthday}\end{pmatrix} &= 1 - \Pr\begin{pmatrix}\text{none of the 40 people}\\ \text{has your birthday}\end{pmatrix}\\
&= 1 - \prod_{i=1}^{40} \Pr\begin{pmatrix}i^{\text{th}}\text{ person does not}\\ \text{have your birthday}\end{pmatrix}\\
&= 1 - \left(\frac{364}{365}\right)^{40}\\
&\approx 10.4\%.
\end{aligned}
$$

Birthday Paradox

- In a group of 40 people
    2. What is the probability that at least two people share the same birthday? (強碰撞)

$$
\begin{aligned}
\Pr\begin{pmatrix}\text{two people have}\\ \text{the same birthday}\end{pmatrix} &= 1 - \Pr\begin{pmatrix}\text{all 40 people have}\\ \text{different birthdays}\end{pmatrix}\\
&= 1 - \prod_{i=1}^{40} \Pr\begin{pmatrix} i^{\text{th}} \text{ person does not have}\\ \text{the same birthday as any}\\ \text{of the previous } i-1 \text{ people}\end{pmatrix}\\
&= 1 - \prod_{i=1}^{40} \frac{365-(i-1)}{365}\\
&= 1 - \frac{365}{365}\cdot\frac{364}{365}\cdot\frac{363}{365}\cdots\frac{326}{365}\\
&\approx 89.1\%.
\end{aligned}
$$

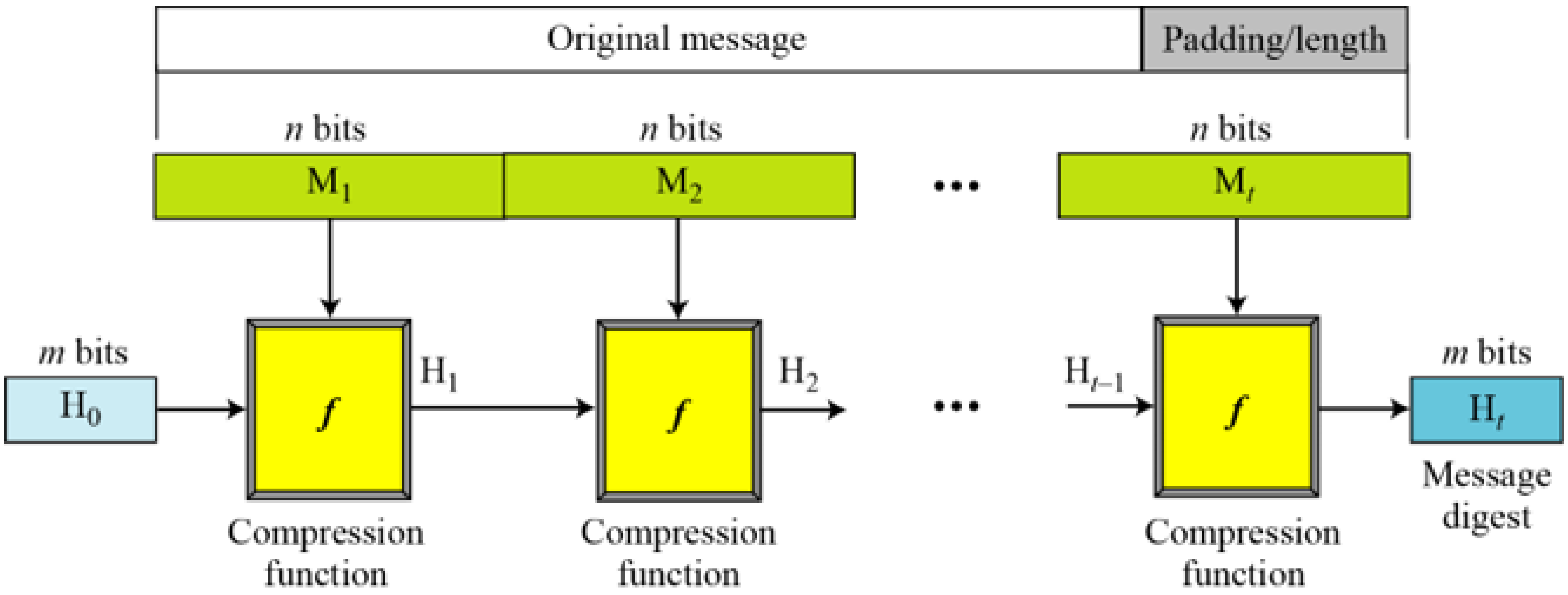


- Birthday Attack
- Given f, find f(x1) == f(x2), for distinct x1 and x2
  - $n \approx \sqrt{2H \times p(n)}$
- For example
  - A hash with 32 bits digest size
  - probability of collision is 2^{-20}
  - $n \approx \sqrt{2 \times 2^{32} \times 2^{-20}} = \sqrt{2^{1+32-20}} = \sqrt{2^{13}} = 2^{6.5} \approx 90.5$

Merkle–Damgård Construction

Merkle–Damgård Construction

- Suffer from Length Extension Attack
- Padding

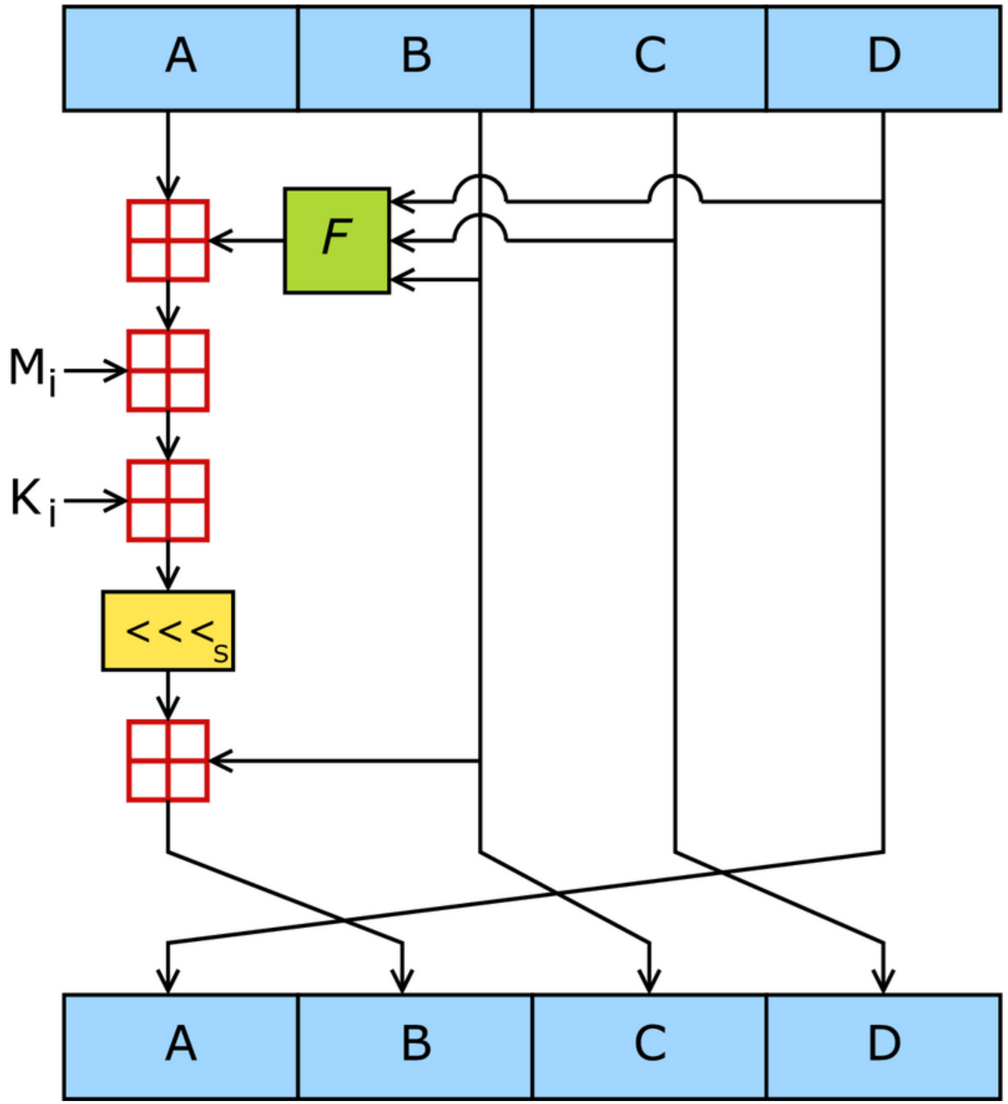


MD5

- digest size: 128 bits (16 bytes)
- Much faster than SHA-family

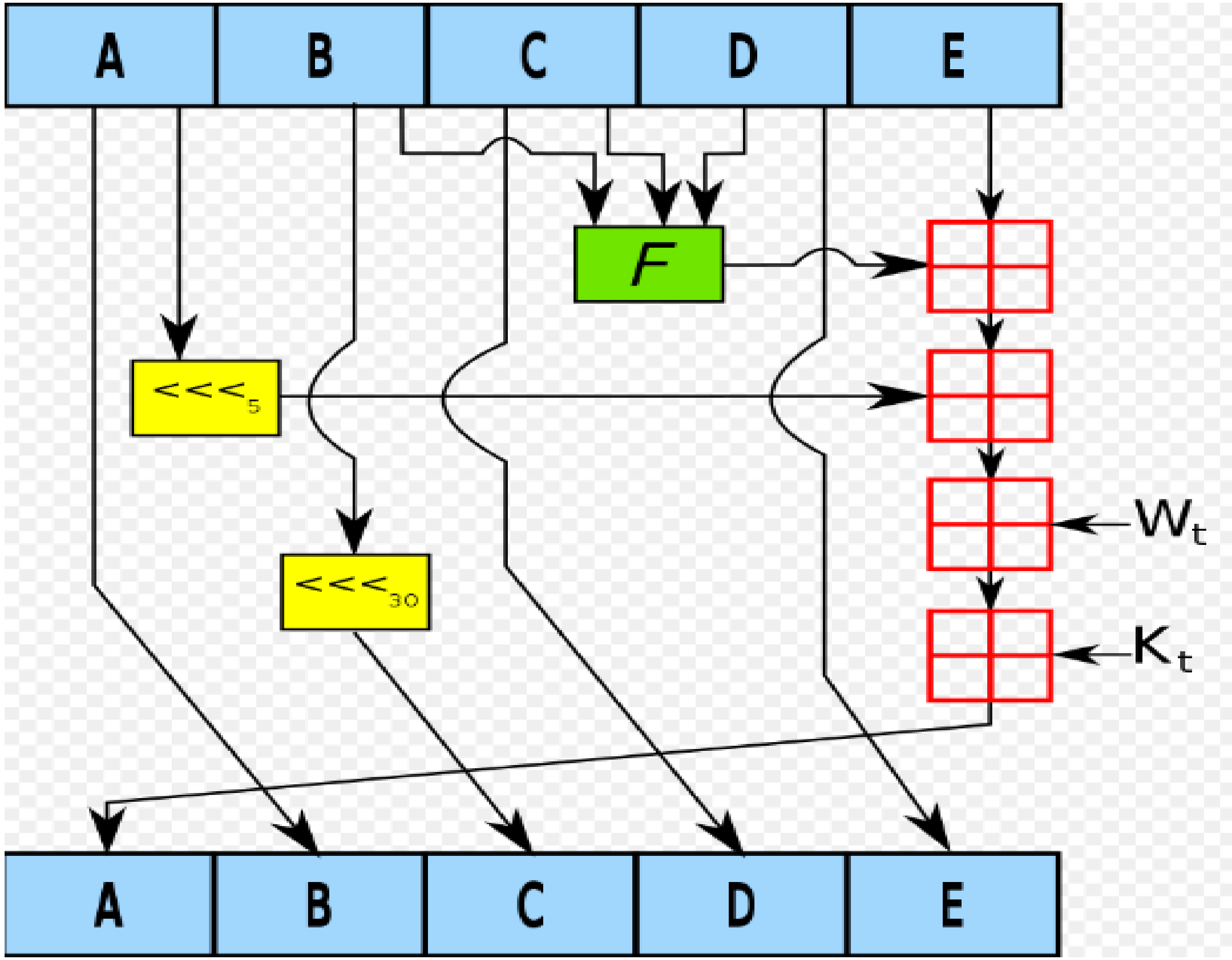
Security properties:

✓ Pre-image resistance
✓ Second pre-image resistance
✗ Collision resistance: 2^18



SHA1

- digest size: 160 bits (20 bytes)



Security properties:

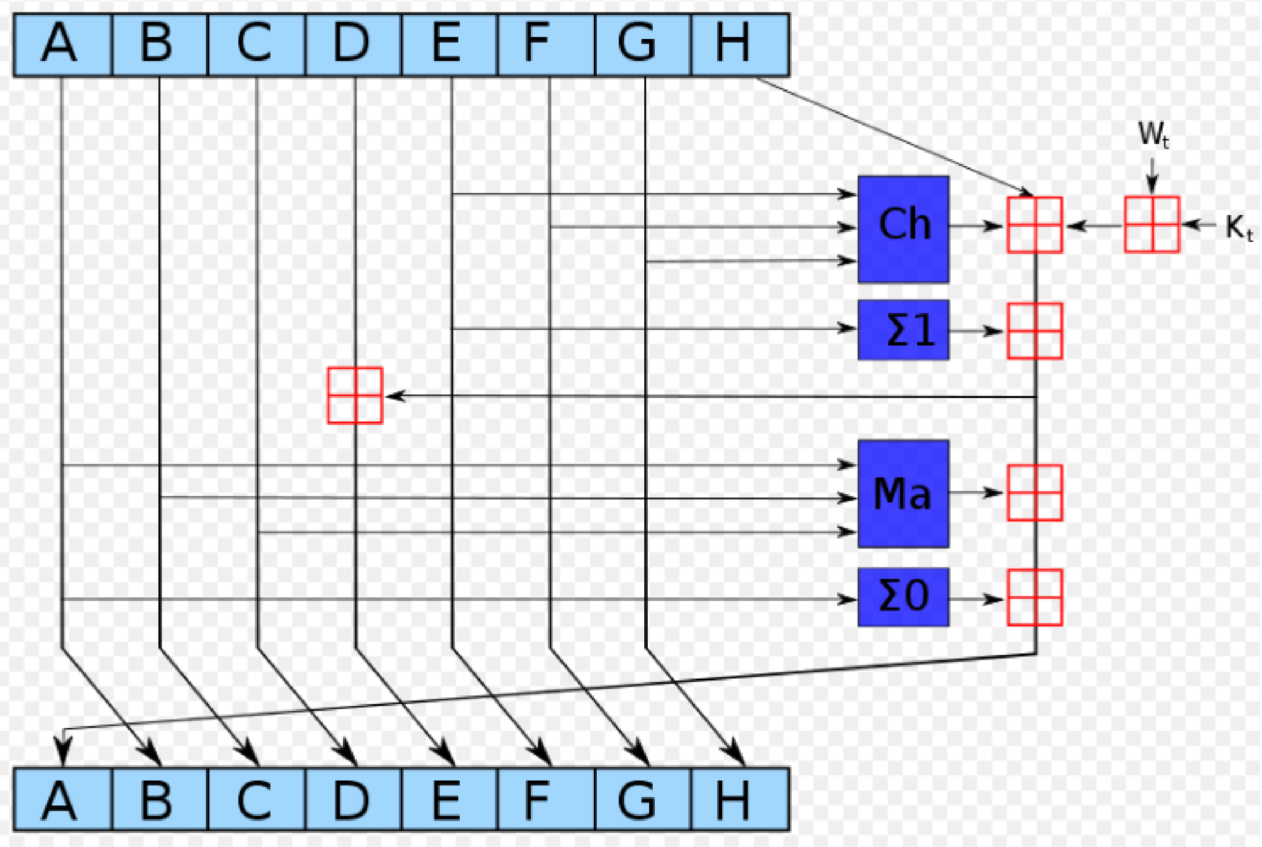
✓ Pre-image resistance
✓ Second pre-image resistance
✗ Collision resistance: 2^60

SHA2

- digest size:
  - 224, 256, 384, 512 bits

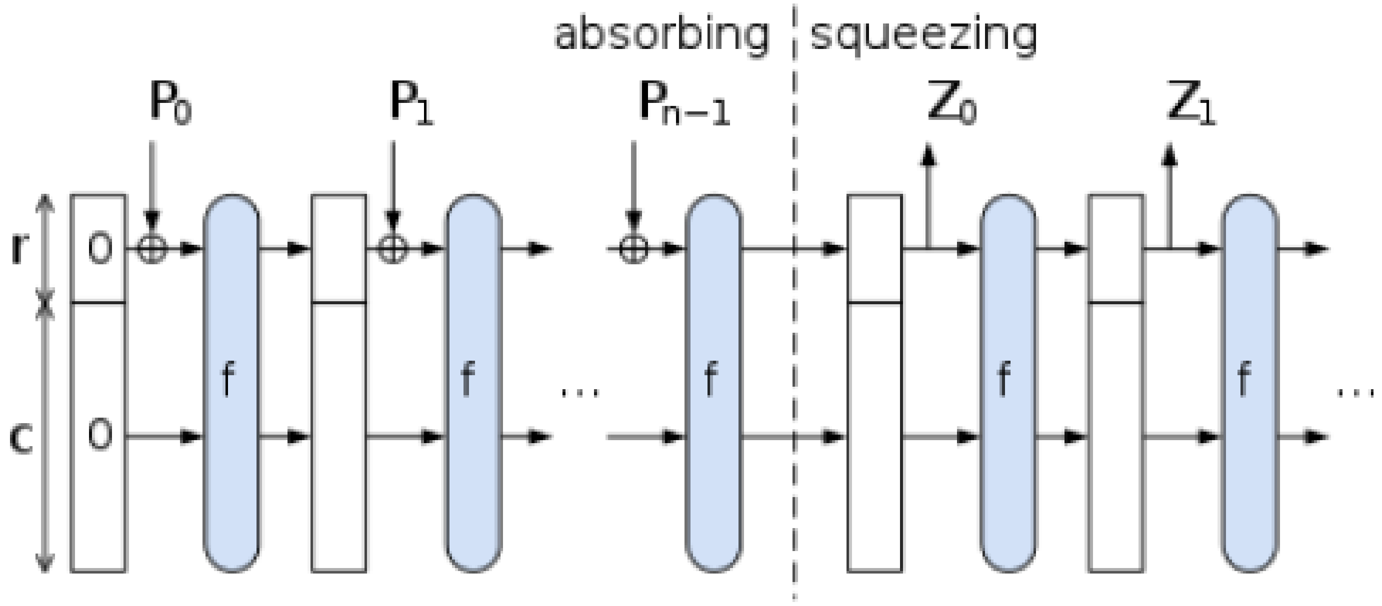
Security properties:

✓ Pre-image resistance
✓ Second pre-image resistance
✓ Collision resistance



SHA3

- Sponge construction
- arbitary digest size



Security properties:

✓ Pre-image resistance

✓ Second pre-image resistance

✓ Collision resistance

Lab1 – SHA256LEA

Length Extension Attack

```python
def main():
    try:
        token = b64encode(b"user=someone")
        inside = salt + b64decode(token)
        auth = hashlib.sha256(inside).hexdigest()

        token2 = input("input your token: ").strip()
        inside2 = salt + b64decode(token2.encode('ascii'))
        auth2 = input("input your authentication code: ").strip()

        secret_auth = hashlib.sha256(inside2).hexdigest()

        if auth2 == secret_auth:
            if b"user=admin" in b64decode(token2):
                print(flag)
            else:
                print("YOU ARE NOT ADMIN, GO AWAY!")
        else:
            print("YOU ARE NOT ALLOW TO CHANGE MY TOKEN!")
    except:
        exit(0)
```

Length Extension Attack

- nc 140.114.77.172 60004
- HashPump

```
IV = | 00 00 00 00 |, X = | 12 34 |,       H(x) = | 27 0a 19 4e |

IV = | 00 00 00 00 |, X = | 12 34 56 78 |, H(x) = | 51 b4 c0 ad |

IV = | 27 0a 19 4e |, X = | 56 78 |,       H(x) = | 51 b4 c0 ad |
```

```
hank292@islabserver:~$ hashpump -s 8dad6cd30ea1682cf6d8864e53ab954127ecc88c52f12fc858f88e208af33506 -d user=someone -k 44 -a user=admin
a4968525c9f8ec6a02b8725bb87597aced97906189e987c1efd5e545d68d8c32
user=someone\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x01\xc0user=admin
```

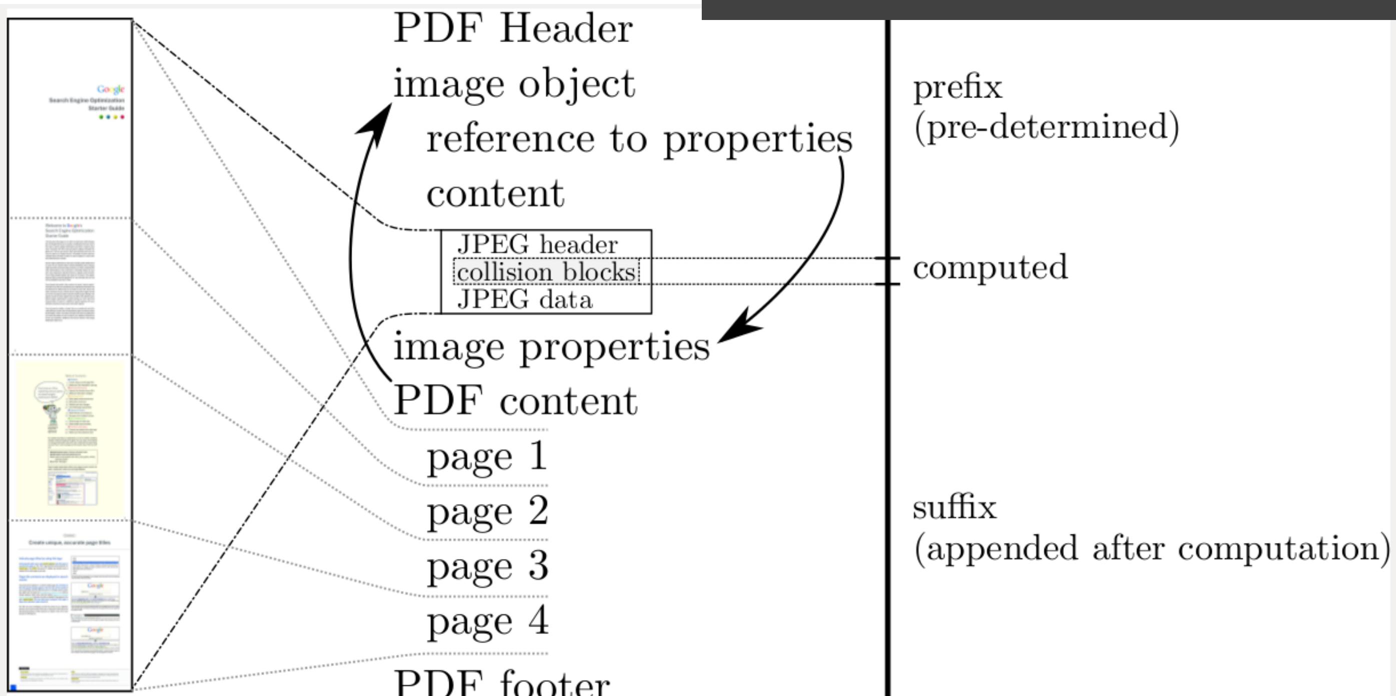
Lab2 – SHA1 is dead

```python
def main():
    print("Give me the collision pair of SHA1")
    file1 = input("[>] :")
    file2 = input("[>] :")

    try:
        file1 = b64decode(file1.encode('ascii'))
        file2 = b64decode(file2.encode('ascii'))
    except:
        print("Not base64!")
        exit(1)
    if (file1 == b'') | (file2 == b'' ):
        print("Empty input!")
        exit(1)
    if (sha1(file1).hexdigest() == sha1(file2).hexdigest()):
        print(flag)
    else:
        print("SHA1 is alive!!!")
    exit(0)
```

SHA1 Collision

- nc 140.114.77.172 60005
- https://shattered.io/



Lab3 – MD5 c011isi0n

```python
def main():
    key = os.urandom(32)

    usr = input('[>] Username (hex): ')
    usr = bytes.fromhex(usr)
    mac = hashlib.md5(b'Crypto is fun' + usr + key).digest()
    print(f'[+] Mac: {mac.hex()}')

    usr2 = input('[>] Username (hex): ')
    mac2 = input('[>] Mac: ')
    usr2 = bytes.fromhex(usr2)
    mac2 = bytes.fromhex(mac2)
    mac2 = hashlib.md5(b'Crypto is fun' + usr2 + key).digest()
    if mac2 == mac:
        if usr2 != usr:
            print(f'[+] {flag}')
        else:
            print(f'[!] Try harder')
    else:
        print(f'[!] Invalid mac')
    exit(0)
```

MD5 tunneling

- nc 140.114.77.172 60006
- md5-tunneling
- hashclash

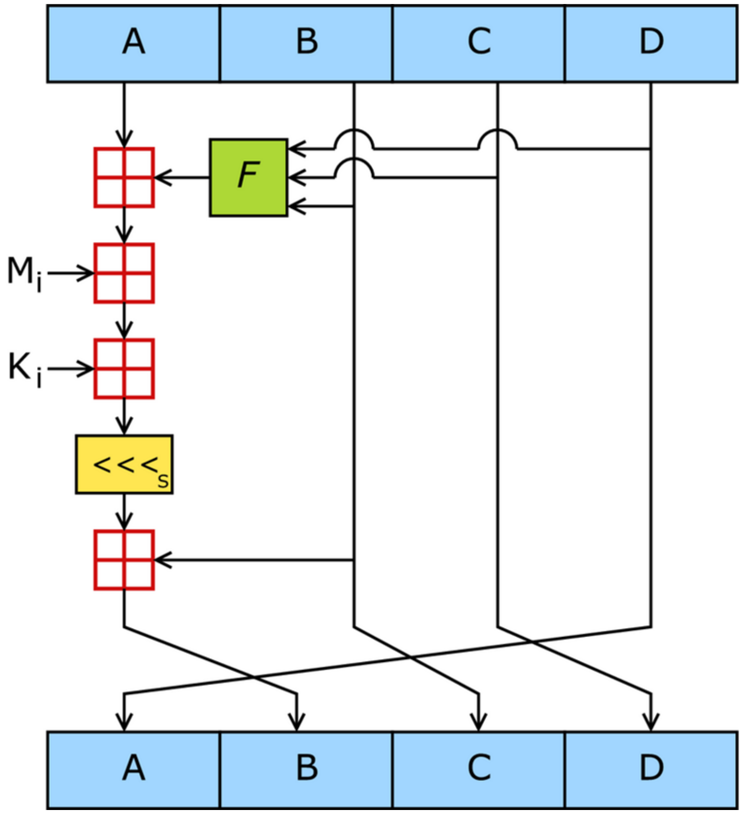
partial_md5.c

```
#include <openssl/md5.h>
```

/usr/include/openssl/md5.h

```
MD5_Init(&ctx);
MD5_Update(&ctx, p, st.st_size);
printf("Partial MD5 is %08X %08X %08X %08X\n", ctx.A, ctx.B, ctx.C, ctx.D);

MD5_Final(final, &ctx);
printf("Final MD5 is %08X %08X %08X %08X\n", ctx.A, ctx.B, ctx.C, ctx.D);
```



```
### ./par_md5 prefix ###
'''
Partial MD5 is C08603BE 3CE1AA68 4B43510F 4B432744
Final MD5 is C7AB12D9 C9692A59 C4FA067E 52DDA4A5
'''
```

Prefix

```
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000: 43 72 79 70 74 6F 20 69 73 20 66 75 6E 20 20 20    Crypto.is.fun...
00000010: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20    ................
00000020: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20    ................
00000030: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20    ................
```

```
### ./md5-tunneling 0xC08603BE 0x3CE1AA68 0x4B43510F 0x4B432744 ###

m0 = [
    0x31,0xA7,0x9A,0x89,0xBB,0xDB,0xFD,0x08,0x8C,0xB1,0x57,0x6D,0x0E,0x2D,0xB5,0xD4,
    0x6D,0x01,0x62,0x4C,0x33,0x1E,0xBC,0x3D,0xD9,0x08,0xBD,0x4A,0x3A,0x3E,0x25,0x0D,
    0x55,0x2D,0x35,0x05,0x02,0x7A,0x54,0xE2,0x67,0xBB,0xAA,0x7C,0x54,0x77,0x21,0x94,
    0xF6,0x88,0x29,0xCE,0xEC,0x28,0xAC,0xBC,0xC5,0x31,0xFD,0x5C,0x9E,0xFC,0xF5,0xD4,
    0xB9,0x29,0xD7,0x3B,0x33,0xA1,0x8C,0x27,0xE7,0xF8,0xDB,0xB8,0x2A,0xD4,0xCF,0xD3,
    0x99,0xF2,0x35,0xB5,0x8B,0xF6,0x7F,0xFC,0x16,0x9D,0xBD,0x26,0x4B,0xBD,0x00,0xBD,
    0x1D,0x10,0xFB,0x20,0x26,0x93,0xF3,0x66,0x66,0x03,0x25,0x6A,0x9D,0xFC,0x90,0xBD,
    0xFD,0x78,0x2D,0xF3,0x7C,0xA6,0xFB,0x20,0xA8,0x7F,0xC5,0xCB,0xAF,0x92,0x8D,0xD4
]

m1 = [
    0x31,0xA7,0x9A,0x89,0xBB,0xDB,0xFD,0x08,0x8C,0xB1,0x57,0x6D,0x0E,0x2D,0xB5,0xD4,
    0x6D,0x01,0x62,0xCC,0x33,0x1E,0xBC,0x3D,0xD9,0x08,0xBD,0x4A,0x3A,0x3E,0x25,0x0D,
    0x55,0x2D,0x35,0x05,0x02,0x7A,0x54,0xE2,0x67,0xBB,0xAA,0x7C,0x54,0xF7,0x21,0x94,
    0xF6,0x88,0x29,0xCE,0xEC,0x28,0xAC,0xBC,0xC5,0x31,0xFD,0xDC,0x9E,0xFC,0xF5,0xD4,
    0xB9,0x29,0xD7,0x3B,0x33,0xA1,0x8C,0x27,0xE7,0xF8,0xDB,0xB8,0x2A,0xD4,0xCF,0xD3,
    0x99,0xF2,0x35,0x35,0x8B,0xF6,0x7F,0xFC,0x16,0x9D,0xBD,0x26,0x4B,0xBD,0x00,0xBD,
    0x1D,0x10,0xFB,0x20,0x26,0x93,0xF3,0x66,0x66,0x03,0x25,0x6A,0x9D,0x7C,0x90,0xBD,
    0xFD,0x78,0x2D,0xF3,0x7C,0xA6,0xFB,0x20,0xA8,0x7F,0xC5,0x4B,0xAF,0x92,0x8D,0xD4
]
```

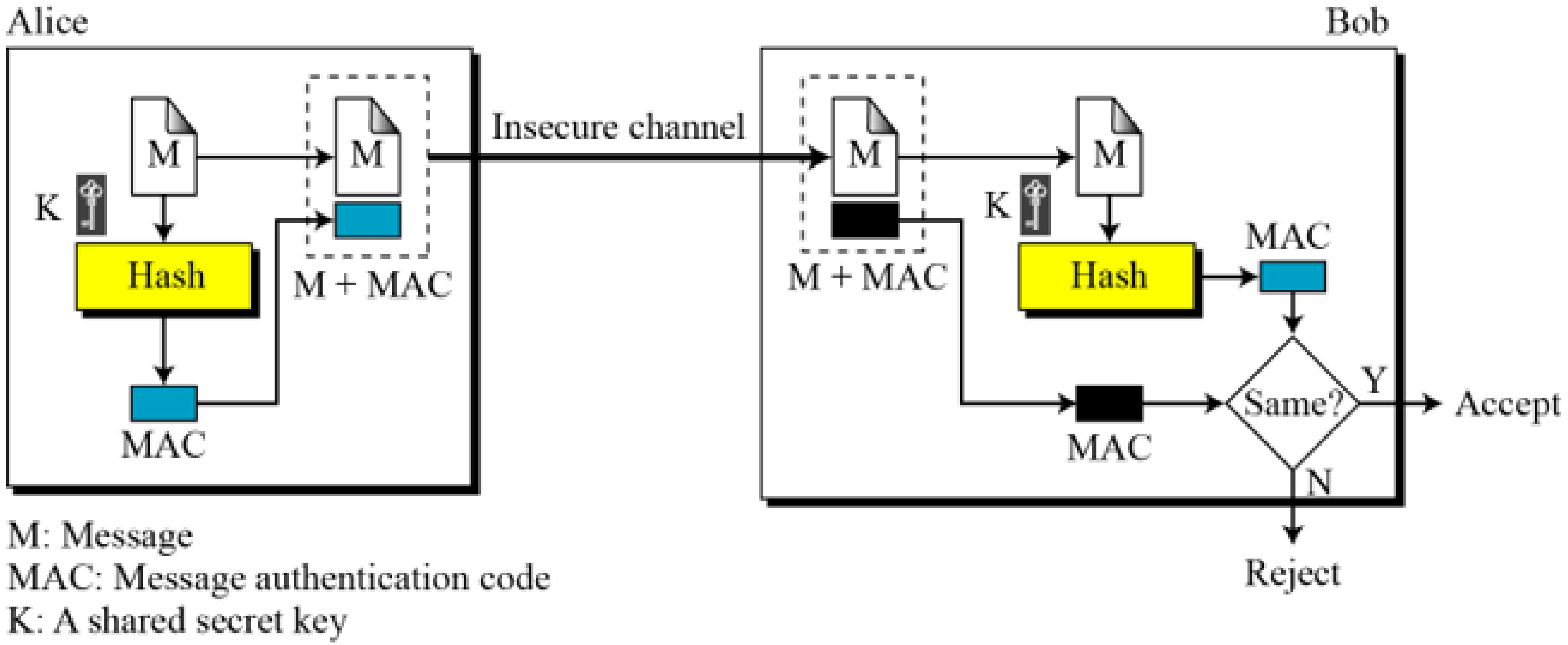
Found Collision!!

```
usr = '20' * 51 + m0
usr1 = '20' * 51 + m1
```

Message Authentication Code (MAC)

MAC

- A cryptographic checksum on data that uses a symmetric key to detect both accidental and intentional modifications of the data.



Simple MAC

- $MAC(M) = H(M \mid\mid K)$: Internal collision
  - $H(M_1) = H(M_2) \rightarrow H(M_1 \mid\mid K) = H(M_2 \mid\mid K)$
- $MAC(M) = H(K \mid\mid M)$: Length extension attack
  - $M(M_1 \mid\mid P \mid\mid M_2) = H(K \mid\mid M_1 \mid\mid P \mid\mid M_2, IV=0) = H(M_2, IV=H(K \mid\mid M_1, IV=0))$

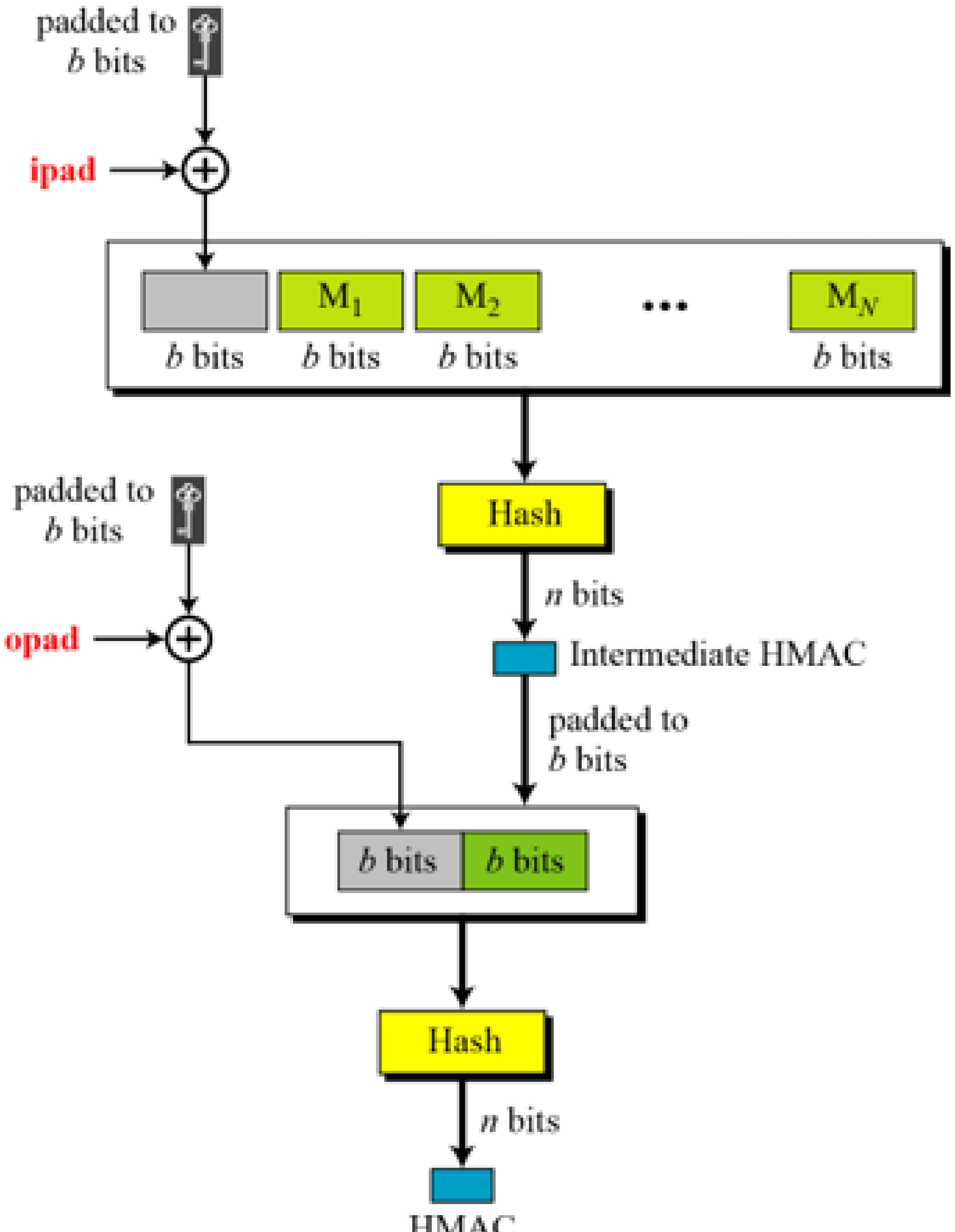
HMAC

$$\mathrm{HMAC}(K,m) = \boxed{\mathrm{H}\left((K' \oplus opad) \parallel \boxed{\mathrm{H}\left((K' \oplus ipad) \parallel m\right)}\right)}$$

$$K' = \begin{cases} \mathrm{H}(K) & K \text{ is larger than block size} \\ K & \text{otherwise} \end{cases}$$

opad: the block-sized padding of repeated bytes “0x5c5c5c….”

ipad: the block-sized padding of repeated bytes “0x363636….”



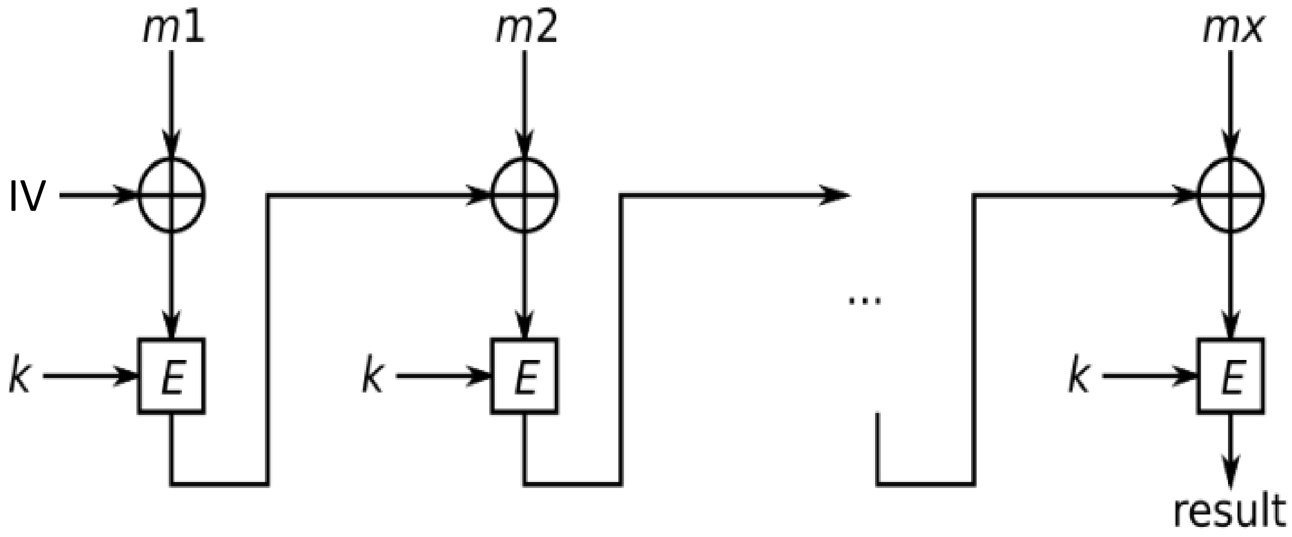
CBC-MAC

- Suffer from padding oracle

Secure only for fixed-length messages

Solution for variable-length messages:

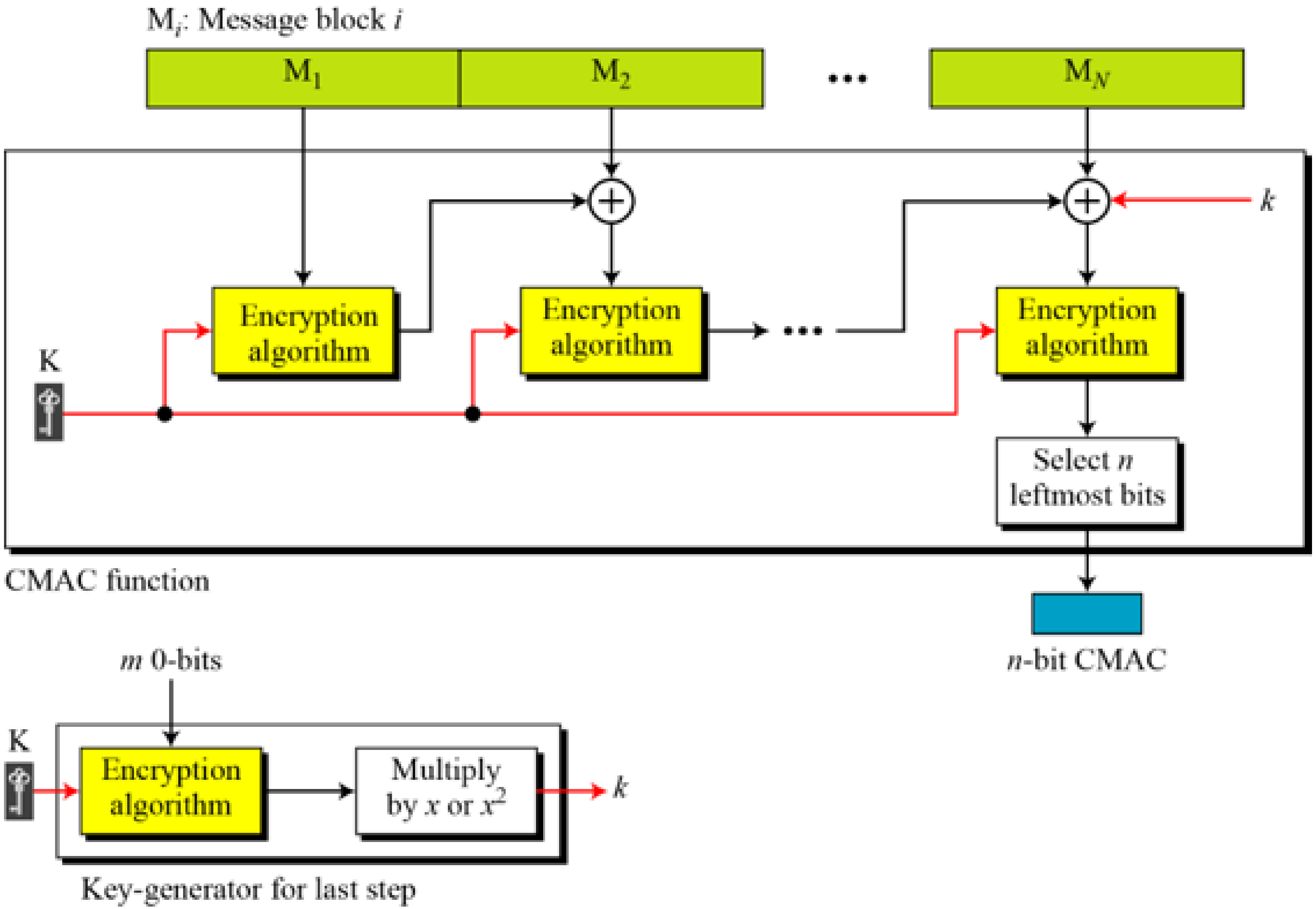
- Length prepending
- Encrypt last block with another key




```
| 12 34 56 78 | 04 04 04 04 | ↔ | 9e 42 7b a0 | f9 08 2c d5 |  :  OK
| 04 cd 72 b9 | 04 04 04 05 | ↔ | 9e 42 7b a1 | f9 08 2c d5 |  :  Invalid padding
| 11 cf e6 95 | 04 04 04 01 | ↔ | 9e 42 7b a5 | f9 08 2c d5 |  :  Corrupted data
| 25 64 b6 f9 | 04 04 05 02 | ↔ | 9e 42 7a a6 | f9 08 2c d5 |  :  Invalid padding
| 70 72 df bc | 04 04 02 02 | ↔ | 9e 42 7d a6 | f9 08 2c d5 |  :  Corrupted data
```

CMAC

- One-key CBC-MAC



Reference

- Forouzan, Behrouz A. ,Cryptography and Network Security, McGraw-Hall International Edition, 2008.
- Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman, An Introduction to Mathematical Cryptography

Thanks For Your Listening!