Internet Security

One-way Hash Function

A Game With Students

Me	You
X	Y

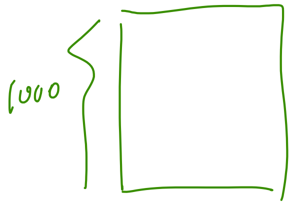
$$X+Y \begin{cases} \text{even: I win} \\ \text{odd: You win} \end{cases}$$

rule: You show real number Y first

Concept

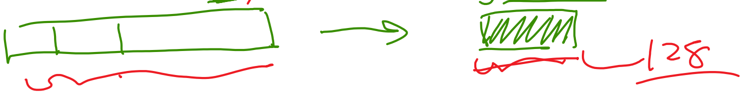
Hash function.

$n \% 1000 \rightarrow h$. (hash value) 5



Many → one

fixed size



2^{128}

one-way property. $hash(M) \rightarrow h$. $\xleftarrow{\times}$

$\sqrt{2^{128}}$

collision-resistance property: $hash(M_1) = hash(M_2)$

MD5

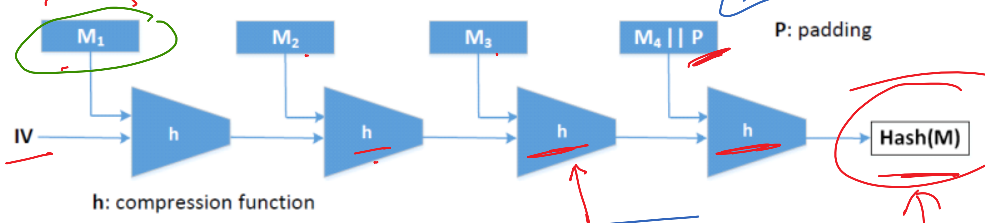
SHA1

Q: give M_1, you can find M_2, s.t $h(M_1) = h(M_2)$

$h(M_1)$

Algorithms

128 bit



MD : Message Digest

SHA . Secure Hash Algorithm

MD1 MD2 . . MD4 MD5

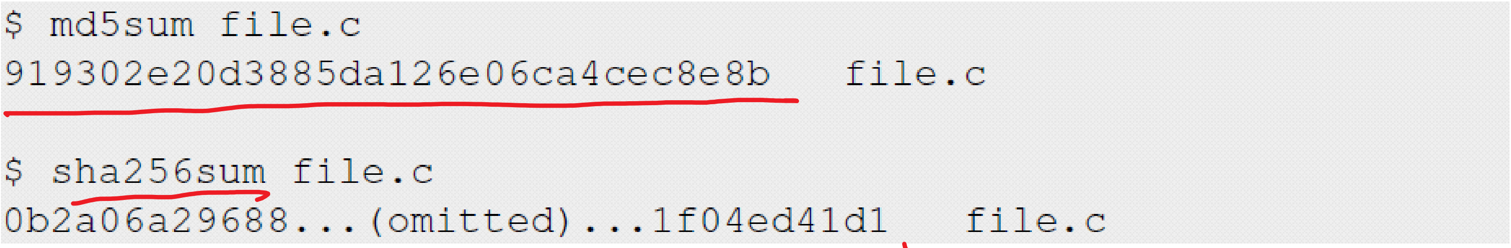
SHA 0 X

SHA1 X

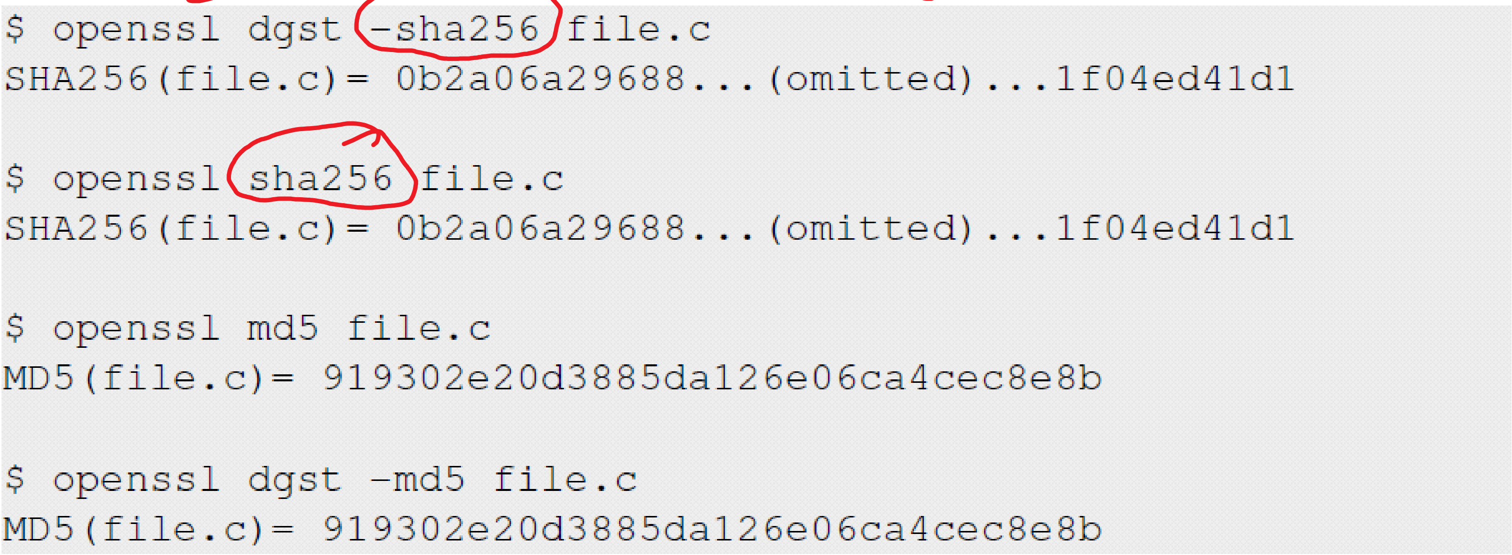
SHA2 — SHA256 ✓
SHA512 ✓

SHA3

Programs

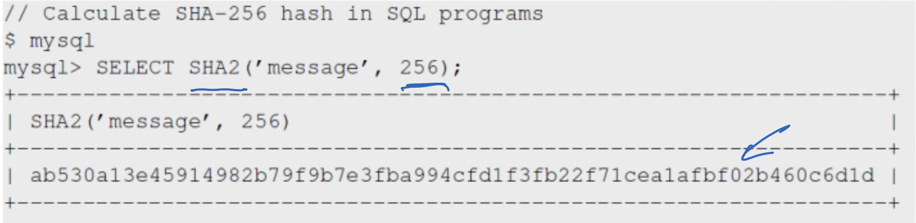


```
$ md5sum file.c
919302e20d3885da126e06ca4cec8e8b   file.c

$ sha256sum file.c
0b2a06a29688...(omitted)...1f04ed41d1   file.c
```

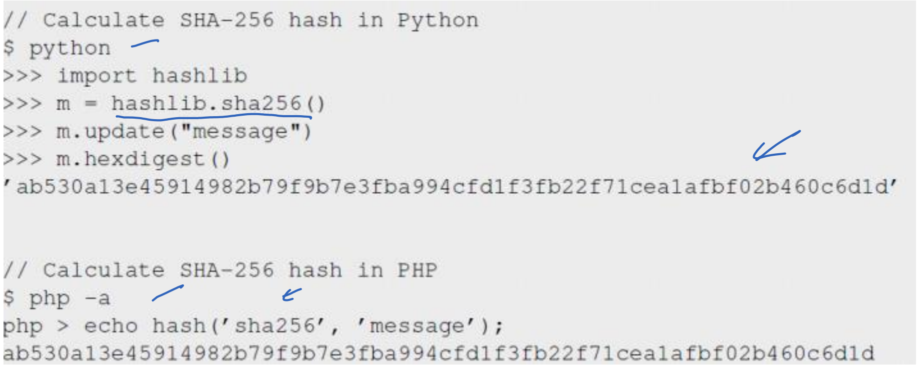


```
$ openssl dgst -sha256 file.c
SHA256(file.c)= 0b2a06a29688...(omitted)...1f04ed41d1

$ openssl sha256 file.c
SHA256(file.c)= 0b2a06a29688...(omitted)...1f04ed41d1

$ openssl md5 file.c
MD5(file.c)= 919302e20d3885da126e06ca4cec8e8b

$ openssl dgst -md5 file.c
MD5(file.c)= 919302e20d3885da126e06ca4cec8e8b
```

Coding

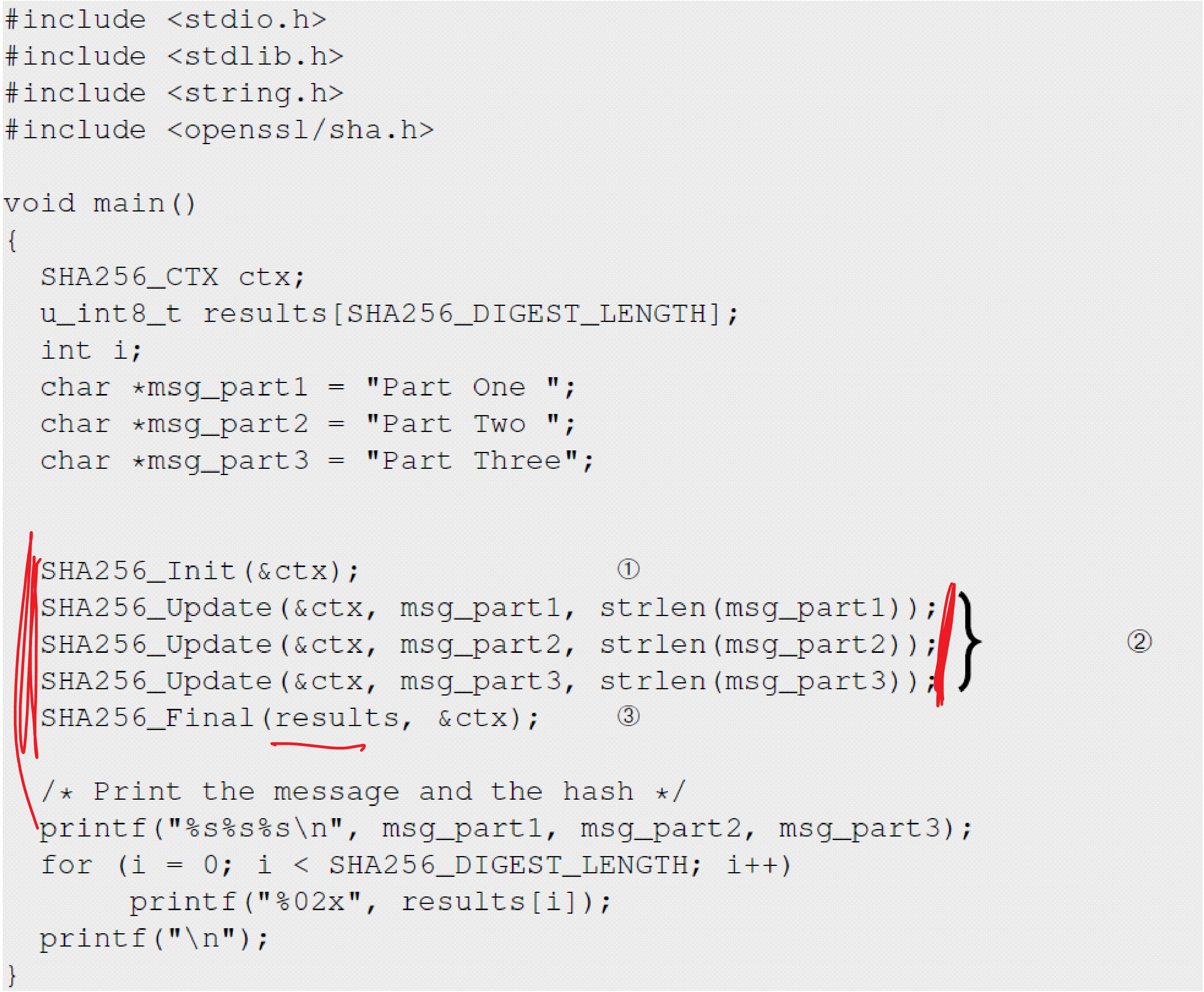


```
// Calculate SHA-256 hash in SQL programs
$ mysql
mysql> SELECT SHA2(’message’, 256);
+------------------------------------------------------------------+
| SHA2(’message’, 256)                                             |
+------------------------------------------------------------------+
| ab530a13e45914982b79f9b7e3fba994cfd1f3fb22f71cea1afbf02b460c6d1d |
+------------------------------------------------------------------+
```



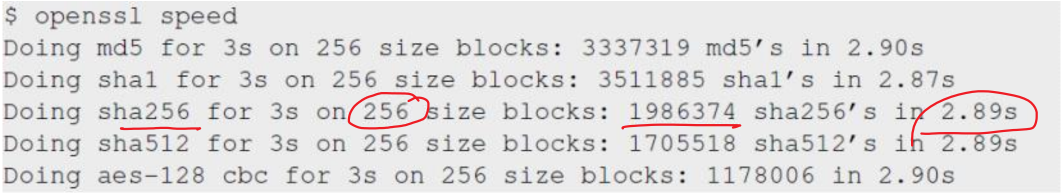
```
// Calculate SHA-256 hash in Python
$ python
>>> import hashlib
>>> m = hashlib.sha256()
>>> m.update("message")
>>> m.hexdigest()
’ab530a13e45914982b79f9b7e3fba994cfd1f3fb22f71cea1afbf02b460c6d1d’


// Calculate SHA-256 hash in PHP
$ php -a
php > echo hash(’sha256’, ’message’);
ab530a13e45914982b79f9b7e3fba994cfd1f3fb22f71cea1afbf02b460c6d1d
```

Coding: C Program



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/sha.h>

void main()
{
  SHA256_CTX ctx;
  u_int8_t results[SHA256_DIGEST_LENGTH];
  int i;
  char *msg_part1 = "Part One ";
  char *msg_part2 = "Part Two ";
  char *msg_part3 = "Part Three";


  SHA256_Init(&ctx);                     ①
  SHA256_Update(&ctx, msg_part1, strlen(msg_part1));  }
  SHA256_Update(&ctx, msg_part2, strlen(msg_part2));  }        ②
  SHA256_Update(&ctx, msg_part3, strlen(msg_part3));  }
  SHA256_Final(results, &ctx);      ③

  /* Print the message and the hash */
  printf("%s%s%s\n", msg_part1, msg_part2, msg_part3);
  for (i = 0; i < SHA256_DIGEST_LENGTH; i++)
       printf("%02x", results[i]);
  printf("\n");
}
```

Performance



```
$ openssl speed
Doing md5 for 3s on 256 size blocks: 3337319 md5's in 2.90s
Doing sha1 for 3s on 256 size blocks: 3511885 sha1's in 2.87s
Doing sha256 for 3s on 256 size blocks: 1986374 sha256's in 2.89s
Doing sha512 for 3s on 256 size blocks: 1705518 sha512's in 2.89s
Doing aes-128 cbc for 3s on 256 size blocks: 1178006 in 2.90s
```

256 × 1,986,374 / 3

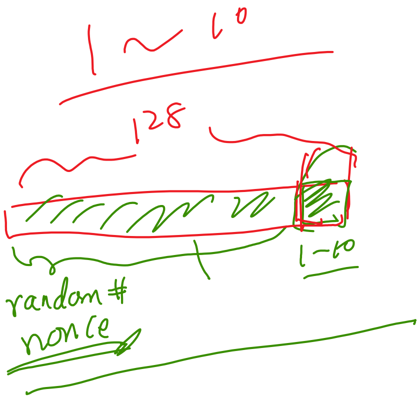
160 M bytes

Question: Play the Game Again

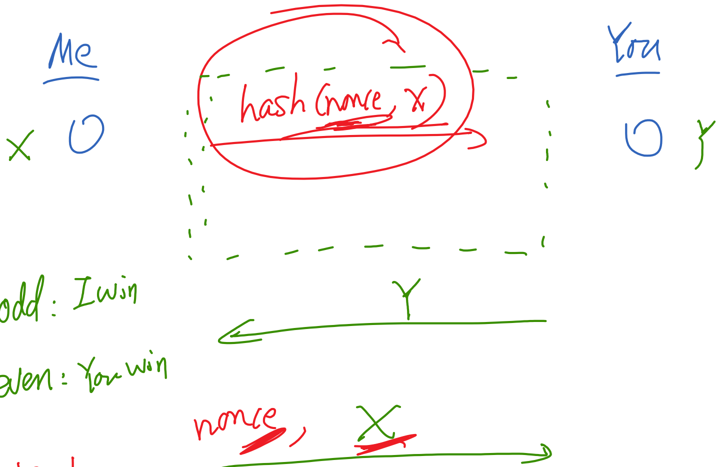
Let's play the game again, this time using one-way hash function. Please describe how you would make the game fair for both sides.

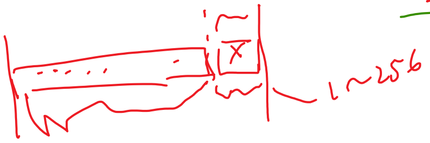
$hash(x) = hash(x')$ ✗

- One-way property
- Collision - resistance



$x + y$
- odd: I win
- even: You win





Collision-Resistance

You can't find M_1 and M_2

s.t. $h(M_1) = h(M_2)$

$$\text{hash}(M) = \boxed{h}$$

$\sqrt{365}$

$O(\sqrt{n})$

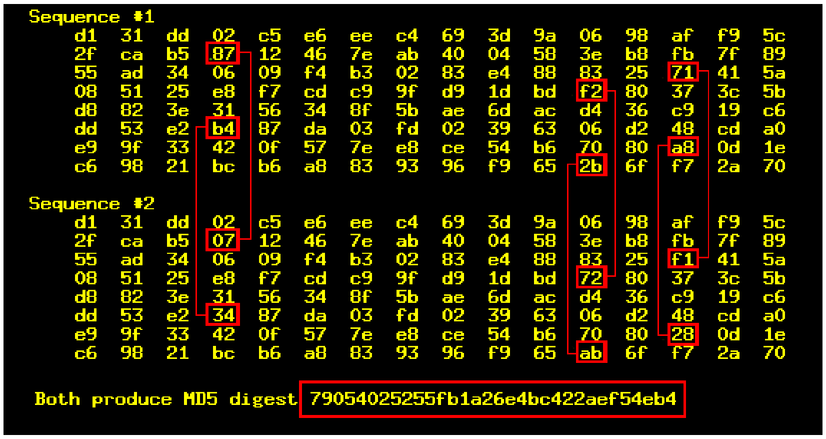
$O(n)$

2^{128}

Collision Attack Against MD5 (2004, Xiaoyun Wang)

128

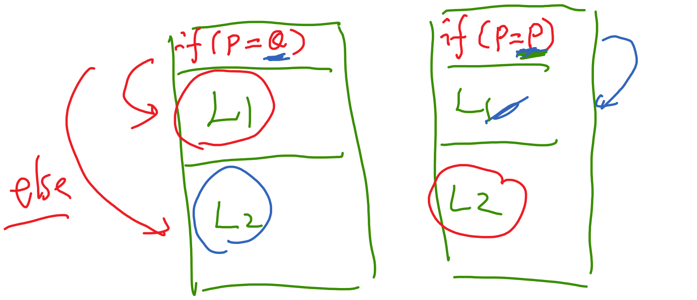
$\sqrt{n} \approx 64$-bit



```
  Sequence #1
      d1   31   dd   02   c5   e6   ee   c4   69   3d   9a   06   98   af   f9   5c
      2f   ca   b5   87   12   46   7e   ab   40   04   58   3e   b8   fb   7f   89
      55   ad   34   06   09   f4   b3   02   83   e4   88   83   25   71   41   5a
      08   51   25   e8   f7   cd   c9   9f   d9   1d   bd   f2   80   37   3c   5b
      d8   82   3e   31   56   34   8f   5b   ae   6d   ac   d4   36   c9   19   c6
      dd   53   e2   b4   87   da   03   fd   02   39   63   06   d2   48   cd   a0
      e9   9f   33   42   0f   57   7e   e8   ce   54   b6   70   80   a8   0d   1e
      c6   98   21   bc   b6   a8   83   93   96   f9   65   2b   6f   f7   2a   70

  Sequence #2
      d1   31   dd   02   c5   e6   ee   c4   69   3d   9a   06   98   af   f9   5c
      2f   ca   b5   07   12   46   7e   ab   40   04   58   3e   b8   fb   7f   89
      55   ad   34   06   09   f4   b3   02   83   e4   88   83   25   f1   41   5a
      08   51   25   e8   f7   cd   c9   9f   d9   1d   bd   72   80   37   3c   5b
      d8   82   3e   31   56   34   8f   5b   ae   6d   ac   d4   36   c9   19   c6
      dd   53   e2   34   87   da   03   fd   02   39   63   06   d2   48   cd   a0
      e9   9f   33   42   0f   57   7e   e8   ce   54   b6   70   80   28   0d   1e
      c6   98   21   bc   b6   a8   83   93   96   f9   65   ab   6f   f7   2a   70

   Both produce MD5 digest 79054025255fb1a26e4bc422aef54eb4
```

Postscript

.ps

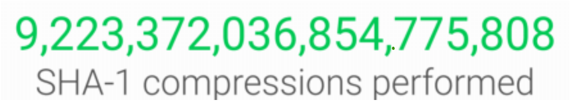


Collision Attack Against SHA-1

160

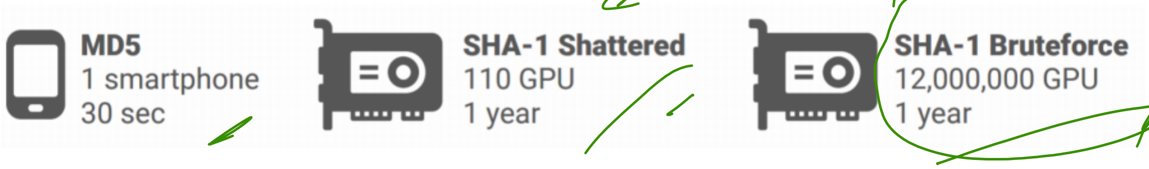
- On February 23, 2017, CWI Amsterdam and Google announced they had performed a collision attack against SHA-1.

- Computation complexity



9,223,372,036,854,775,808
SHA-1 compressions performed

- Compared to other collision attacks



MD5
1 smartphone
30 sec

SHA-1 Shattered
110 GPU
1 year

SHA-1 Bruteforce
12,000,000 GPU
1 year

- Impact

SHAttered
The first concrete collision attack against SHA-1
https://shattered.io

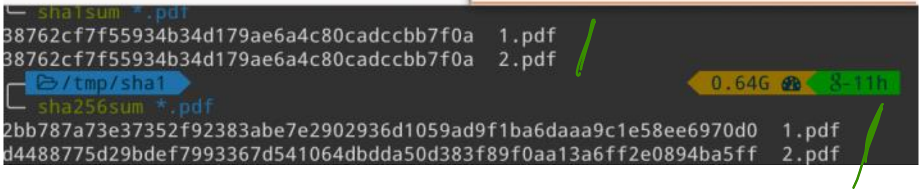
CWI
Marc Stevens
Pierre Karpman

Google
Elie Bursztein
Ange Albertini
Yarik Markov

SHAttered
The first concrete collision attack against SHA-1
https://shattered.io

CWI
Marc Stevens
Pierre Karpman

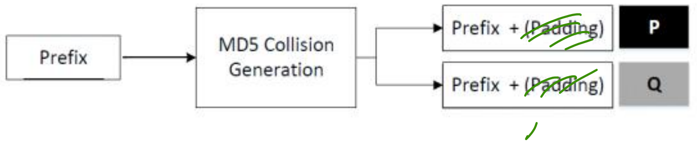
Google
Elie Bursztein
Ange Albertini
Yarik Markov

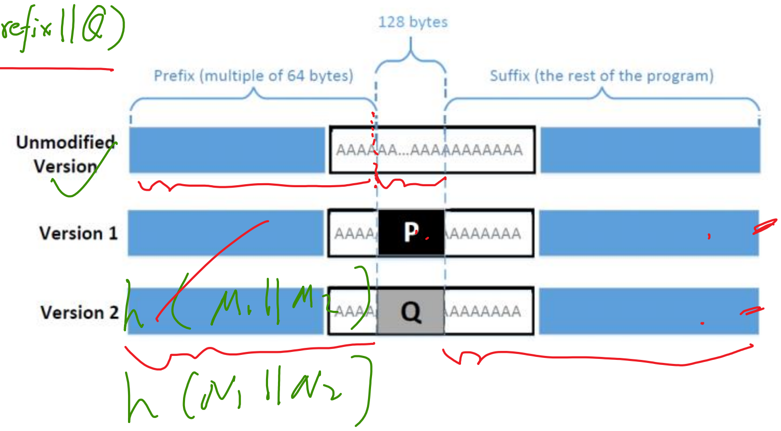


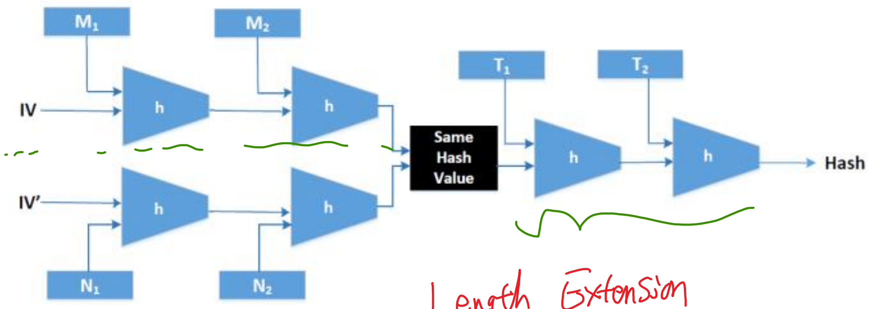
```
└─ sha1sum *.pdf
38762cf7f55934b34d179ae6a4c80cadccbb7f0a  1.pdf
38762cf7f55934b34d179ae6a4c80cadccbb7f0a  2.pdf
 🗁/tmp/sha1                                      0.64G ⚙  8-11h
└─ sha256sum *.pdf
2bb787a73e37352f92383abe7e2902936d1059ad9f1ba6daaa9c1e58ee6970d0  1.pdf
d4488775d29bdef7993367d541064dbdda50d383f89f0aa13a6ff2e0894ba5ff  2.pdf
```

Create MD5 Collisions

$hash(prefix || P) = hash(prefix || Q)$





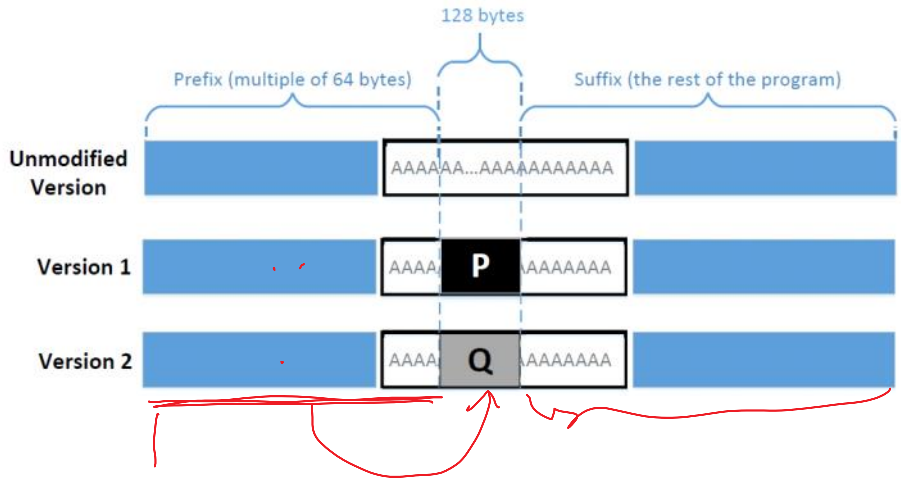


Length Extension

$$h(M_1 || M_2 || T_1 || T_2)$$

$$= h(N_1 || N_2 || T_1 || T_2)$$

Create Two Different Programs with the Same MD5 Hash



Hint

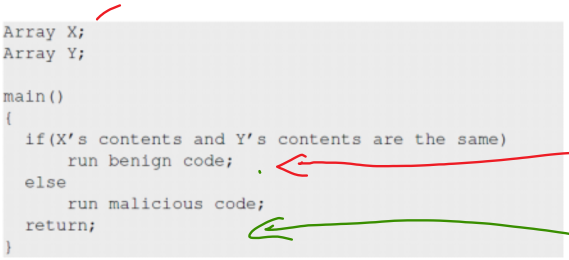
- if ( )

"Hello world"

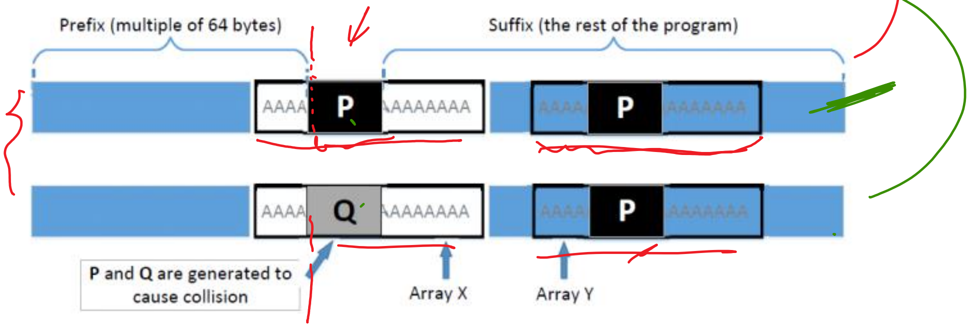
else

"Hello Universe"

A Solution



```
Array X;
Array Y;

main()
{
  if(X's contents and Y's contents are the same)
      run benign code;
  else
      run malicious code;
  return;
}
```



Prefix (multiple of 64 bytes)
Suffix (the rest of the program)
AAAA
P
AAAAAAA
AAAA
P
AAAAAAA
AAAA
Q
AAAAAAA
AAAA
P
AAAAAAA
P and Q are generated to cause collision
Array X
Array Y

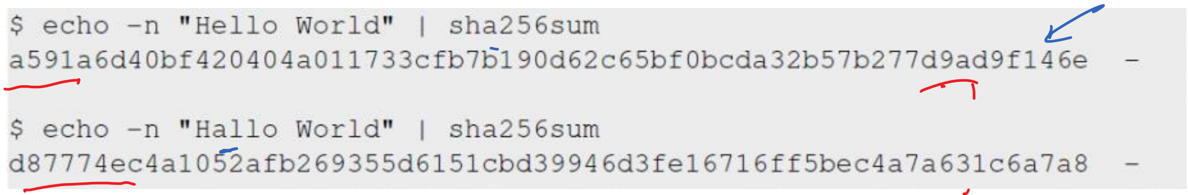
Sign (Hash)

prefix1 P

prefix2 Q

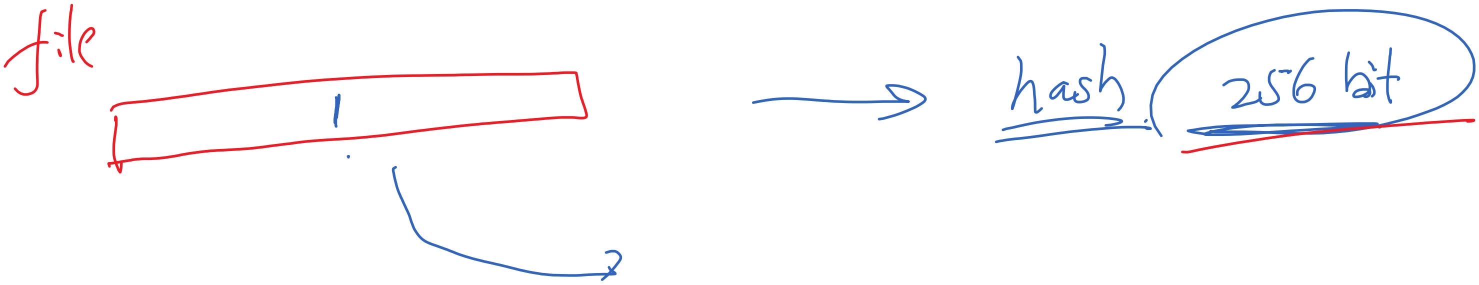
Application: Integrity Verification

0x65 0101



```
$ echo -n "Hello World" | sha256sum
a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b57b277d9ad9f146e  -

$ echo -n "Hallo World" | sha256sum
d87774ec4a1052afb269355d6151cbd39946d3fe16716ff5bec4a7a631c6a7a8  -
```

0x61 0001



Application: Password Authentication

hash (salt || password)

Internet Worm

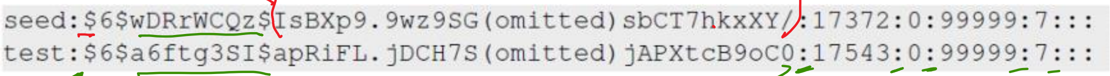
password ⇒ hash

SHA512

salt

shadow file

hash

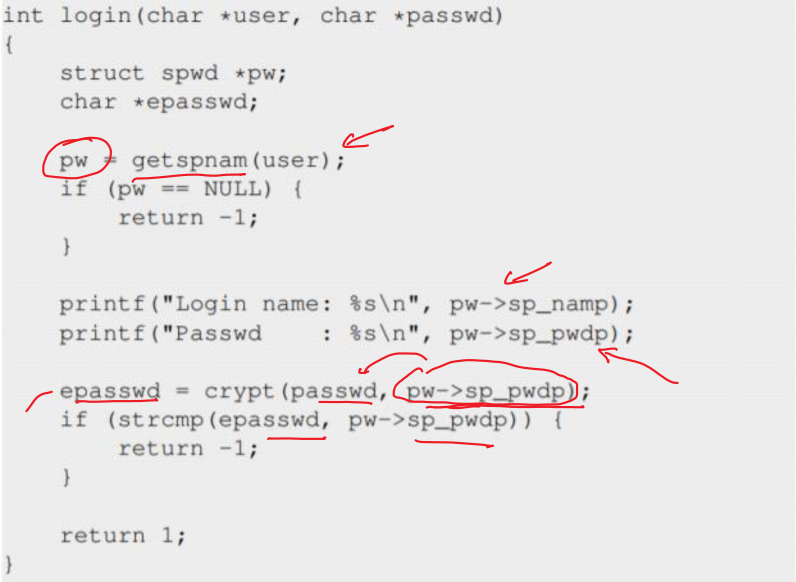


```
seed:$6$wDRrWCQz$IsBXp9.9wz9SG(omitted)sbCT7hkxXY/:17372:0:99999:7:::
test:$6$a6ftg3SI$apRiFL.jDCH7S(omitted)jAPXtcB9oC0:17543:0:99999:7:::
```

root

password entry

Password Authentication: Code

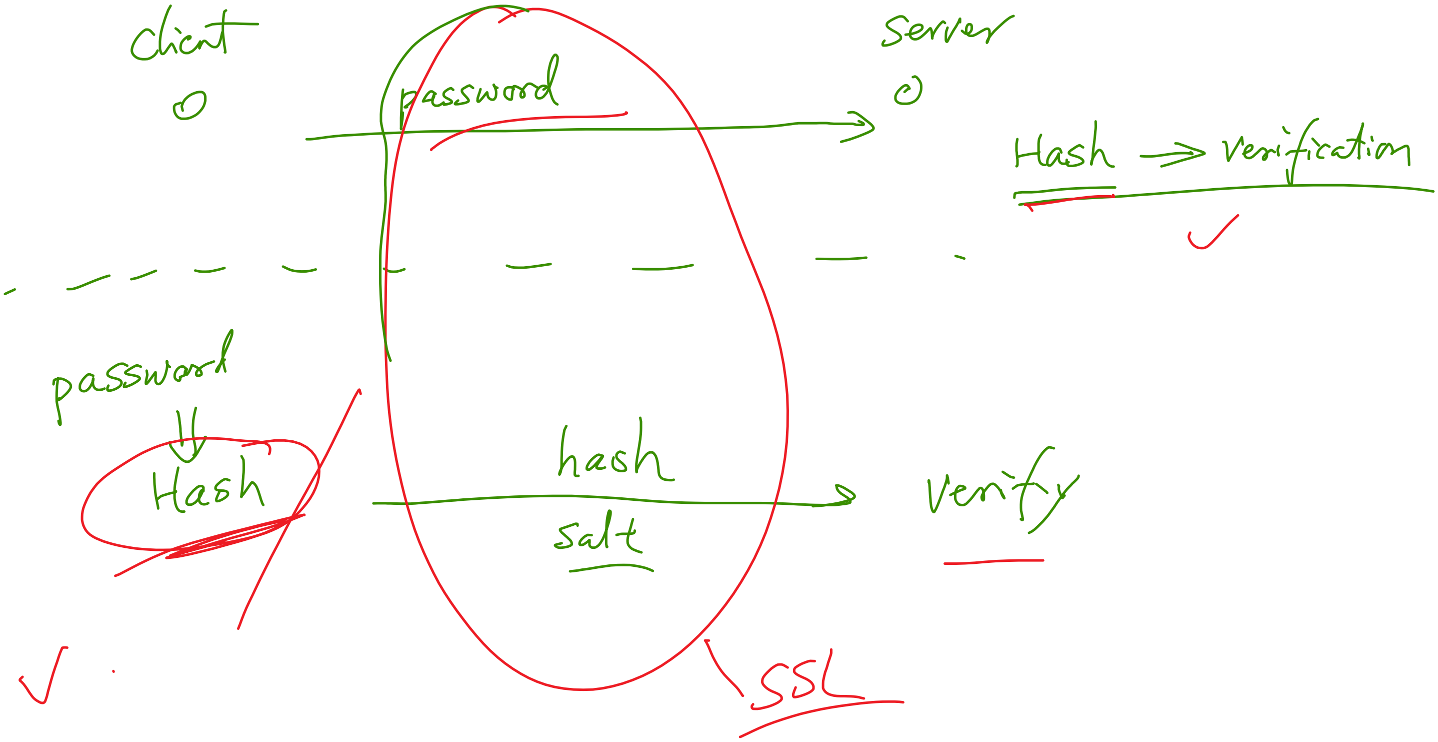


```
int login(char *user, char *passwd)
{
    struct spwd *pw;
    char *epasswd;

    pw = getspnam(user);
    if (pw == NULL) {
        return -1;
    }

    printf("Login name: %s\n", pw->sp_namp);
    printf("Passwd     : %s\n", pw->sp_pwdp);

    epasswd = crypt(passwd, pw->sp_pwdp);
    if (strcmp(epasswd, pw->sp_pwdp)) {
        return -1;
    }

    return 1;
}
```

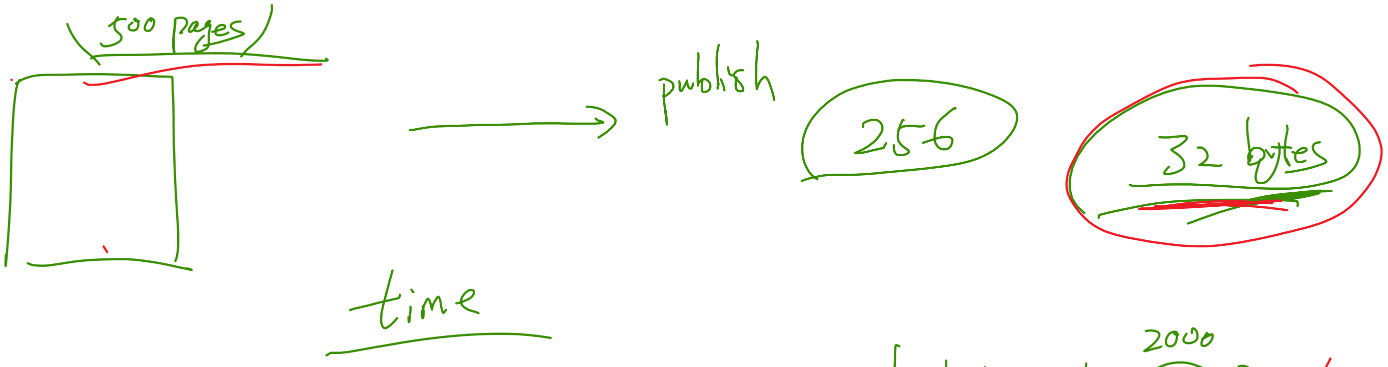
VPN

Client
passwd
Server

Password Authentication: Where do we do hash?



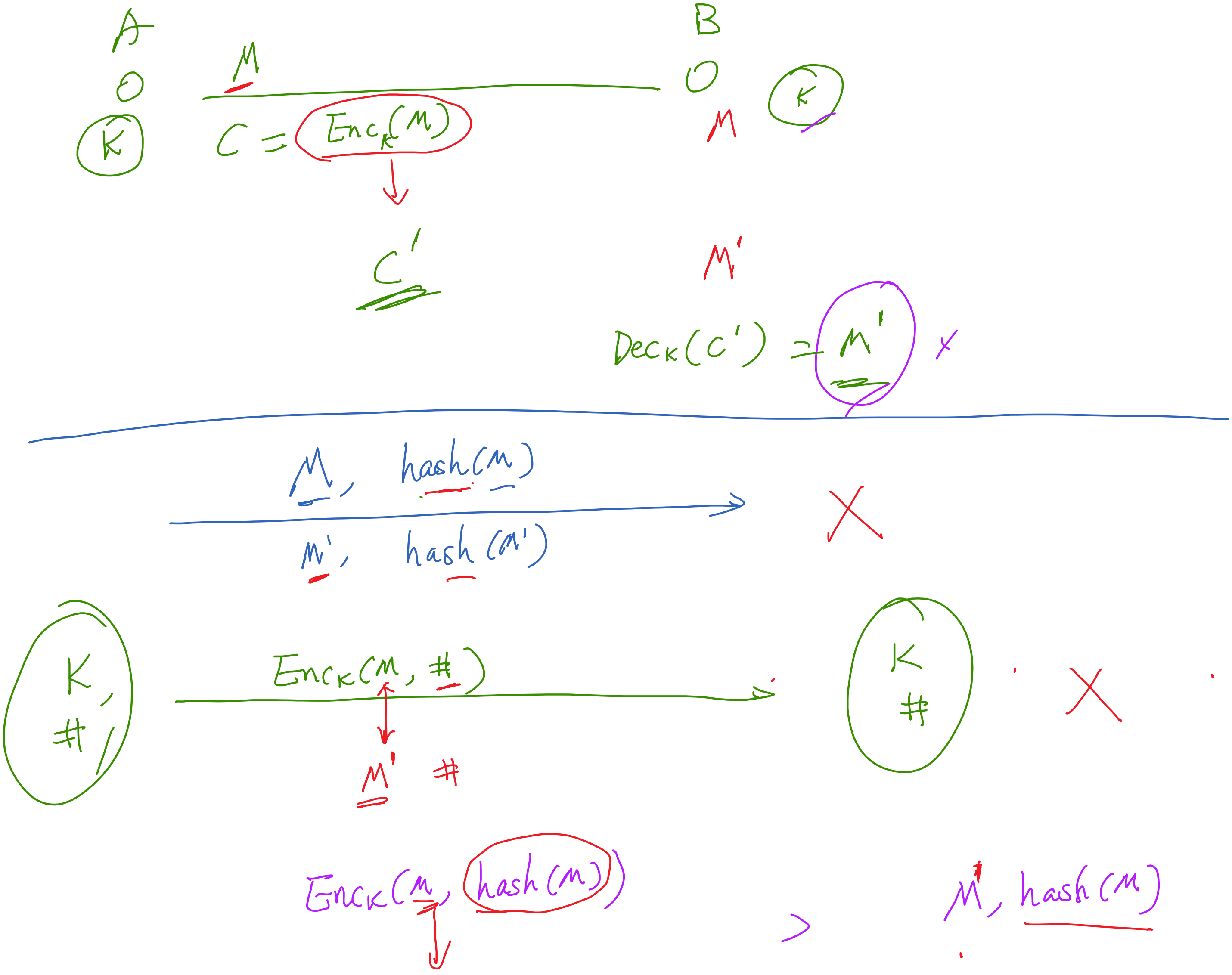
Application: Time Stamping



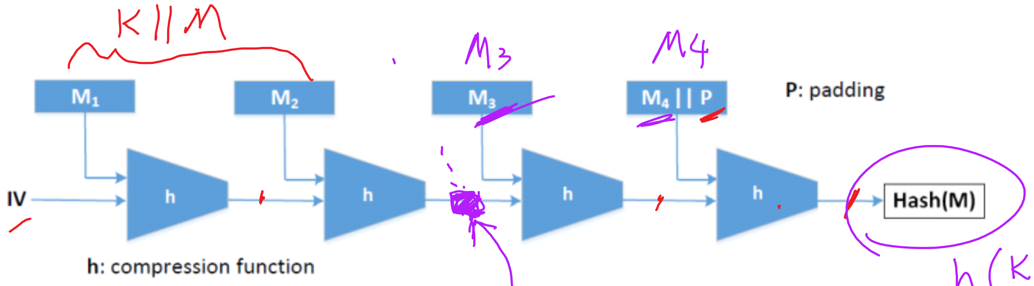
- hash(book . ~~time~~ 2000)

- Trusted 3rd Party

MAC: Message Authentication Code



Length Extension Attack: Discussion



64 bytes

$h(K||M||M_3||M_4||P)$

APIs

A — K

M

hash(K, M)

B — K

M'

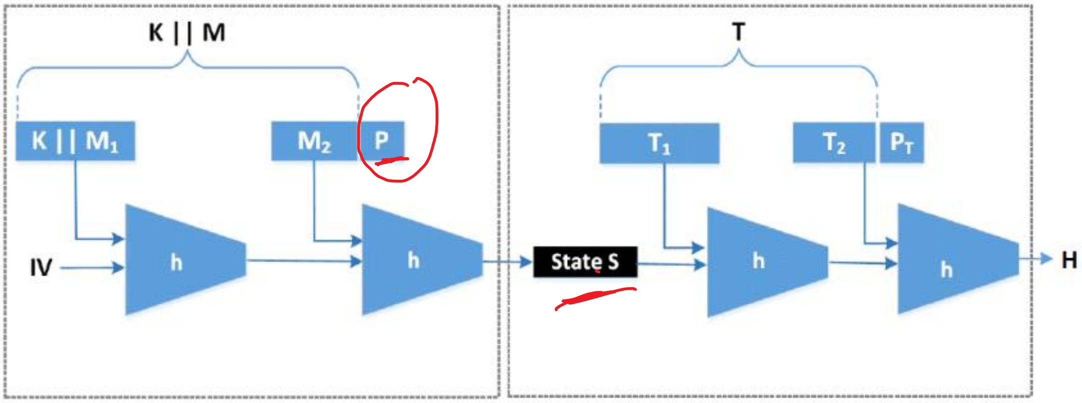
M', hash(K || M')

hash(K || M) ✗

hash(M || K) ✓

Length Extension Attack

K || m || P || m'



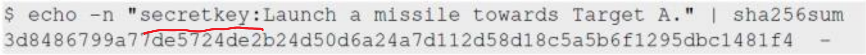
S can be derived from Hash(K || M)

H = Hash(K || M || P || T)

Length Extension Attack

M, hash(K||M) MAC

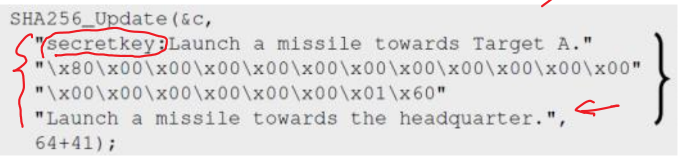
Original Message



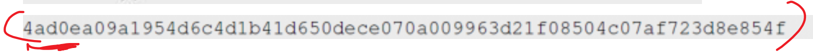
```
$ echo -n "secretkey:Launch a missile towards Target A." | sha256sum
3d8486799a77de5724de2b24d50d6a24a7d112d58d18c5a5b6f1295dbc1481f4  -
```

New Message

hash

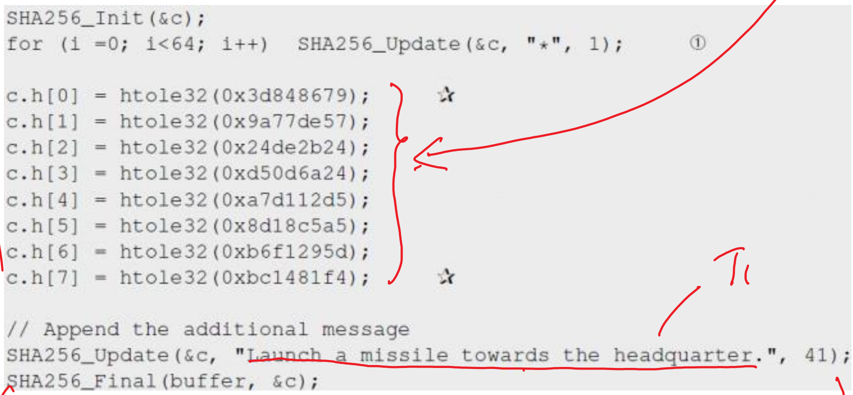


```
SHA256_Update(&c,
  "secretkey:Launch a missile towards Target A."
  "\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
  "\x00\x00\x00\x00\x00\x00\x01\x60"
  "Launch a missile towards the headquarter.",
  64+41);
```



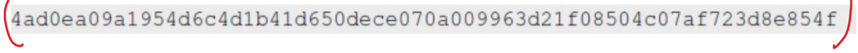
```
4ad0ea09a1954d6c4d1b41d650dece070a009963d21f08504c07af723d8e854f
```

Length Extension Attack

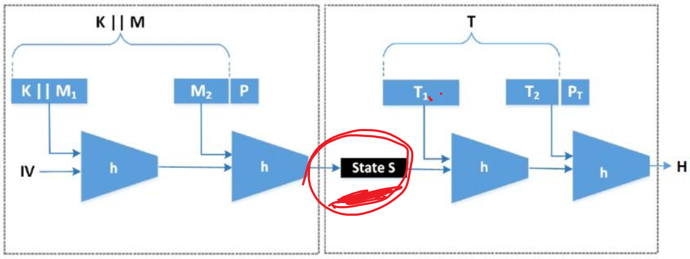


```
SHA256_Init(&c);
for (i =0; i<64; i++)  SHA256_Update(&c, "*", 1);      ①

c.h[0] = htole32(0x3d848679);    ☆
c.h[1] = htole32(0x9a77de57);
c.h[2] = htole32(0x24de2b24);
c.h[3] = htole32(0xd50d6a24);
c.h[4] = htole32(0xa7d112d5);
c.h[5] = htole32(0x8d18c5a5);
c.h[6] = htole32(0xb6f1295d);
c.h[7] = htole32(0xbc1481f4);    ☆

// Append the additional message
SHA256_Update(&c, "Launch a missile towards the headquarter.", 41);
SHA256_Final(buffer, &c);
```

T1



```
4ad0ea09a1954d6c4d1b41d650dece070a009963d21f08504c07af723d8e854f
```



S can be derived from Hash(K || M)

H = Hash(K || M || P || T)

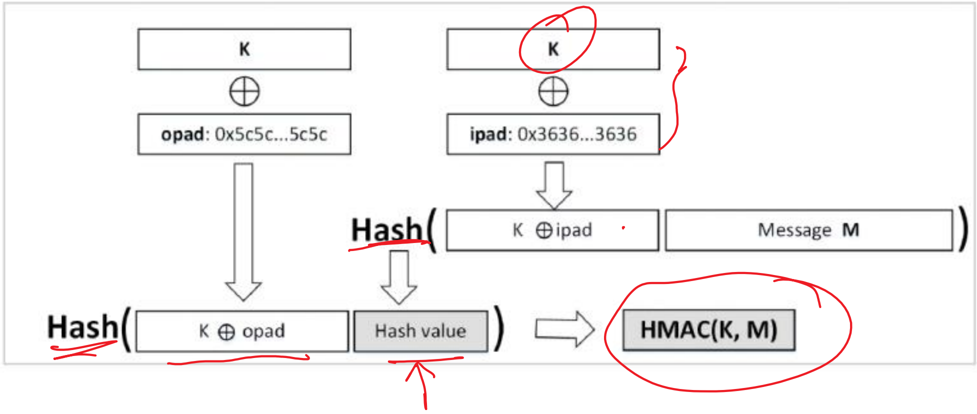
(K||M) X

(M||K) ✓

HMAC

$$HMAC_K(m) = h((K \oplus opad) \; || \; h((K \oplus ipad)||m))$$

Keyed Hash



HMAC-MD5

HMAC-SHA256

:

$Enc_K(M)$, $HMAC(K', M)$

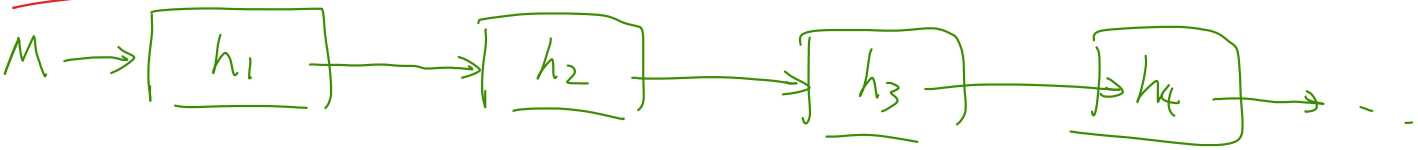
Confidentiality

Integrity

Application: Blockchain

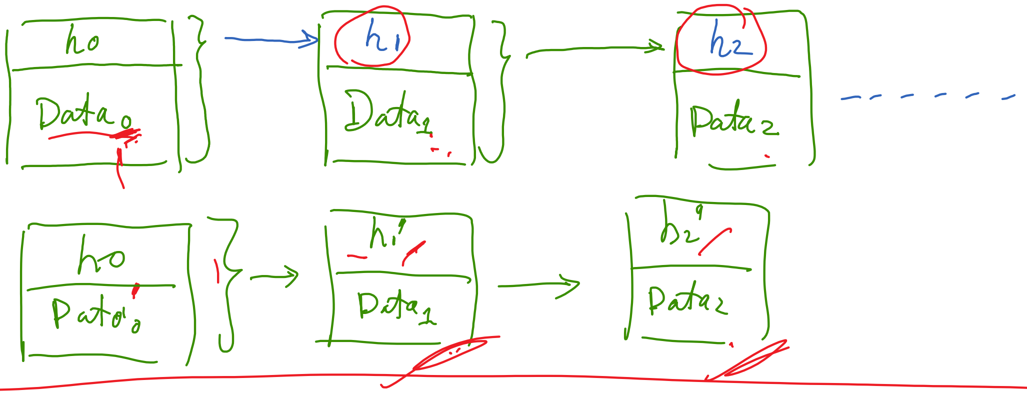
hash chain

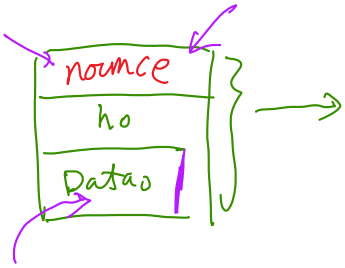
$h_2 = hash(h_1)$



Block chain

$h_1 = hash(h_0, Data_0)$

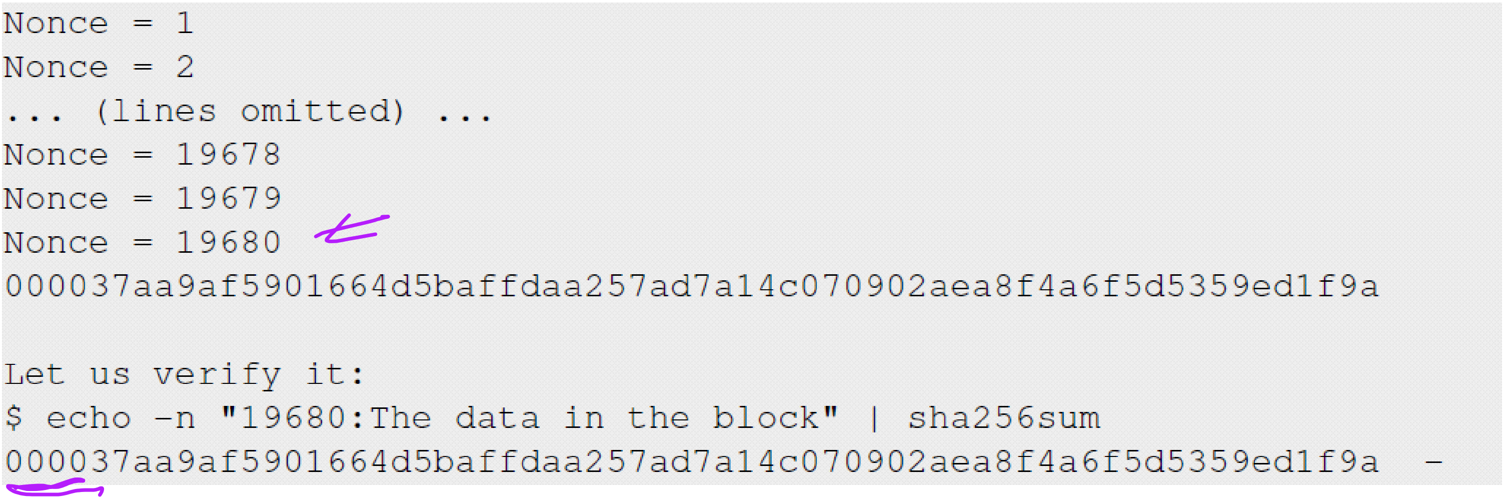




h_1 : has 10 leading zeros

Mining

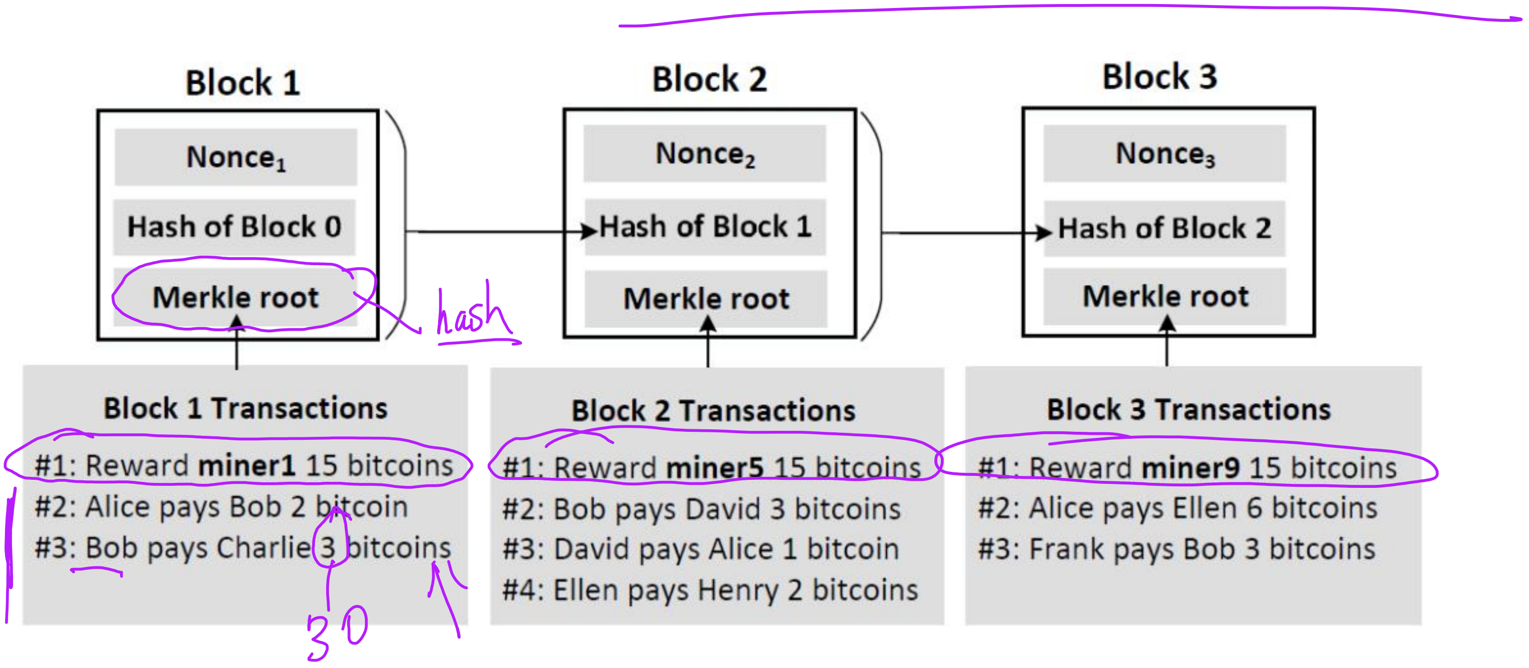
Requirement: 16 bits of leading zeros in the hash.



```
Nonce = 1
Nonce = 2
... (lines omitted) ...
Nonce = 19678
Nonce = 19679
Nonce = 19680
000037aa9af5901664d5baffdaa257ad7a14c070902aea8f4a6f5d5359ed1f9a

Let us verify it:
$ echo -n "19680:The data in the block" | sha256sum
000037aa9af5901664d5baffdaa257ad7a14c070902aea8f4a6f5d5359ed1f9a  -
```

Blockchain: An Actual Bitcoin Block

```
Block #506288 (Jan 26, 2018 9:35:08 PM)
BlockHash: 0000000000000000004dc9e28(omitted)bbb80ef5a707e023
Nonce: 699100228
```



Block 1

Nonce₁

Hash of Block 0

Merkle root

Block 1 Transactions

#1: Reward **miner1** 15 bitcoins

#2: Alice pays Bob 2 bitcoin

#3: Bob pays Charlie 3 bitcoins

Block 2

Nonce₂

Hash of Block 1

Merkle root

Block 2 Transactions

#1: Reward **miner5** 15 bitcoins

#2: Bob pays David 3 bitcoins

#3: David pays Alice 1 bitcoin

#4: Ellen pays Henry 2 bitcoins

Block 3

Nonce₃

Hash of Block 2

Merkle root

Block 3 Transactions

#1: Reward **miner9** 15 bitcoins

#2: Alice pays Ellen 6 bitcoins

#3: Frank pays Bob 3 bitcoins

Summary

- One-way hash function
  - One-way property
  - Collision-free property
- Algorithms
- Collision Attack
- Applications
  - Online game
  - Password authentication
  - Time stamping
  - Message authentication code
  - HMAC and length-extension attack
  - Blockchain