Cryptography

One-Way Hash Functions

Professor Travis Peters
CSCI 476 - Computer Security
Spring 2020

Some slides and figures adapted from Wenliang (Kevin) Du's
Computer & Internet Security: A Hands-on Approach (2nd Edition).
Thank you Kevin and all of the others that have contributed to the SEED resources!

Introduction to One-way Hash Functions (Part I)

This Video Covers:

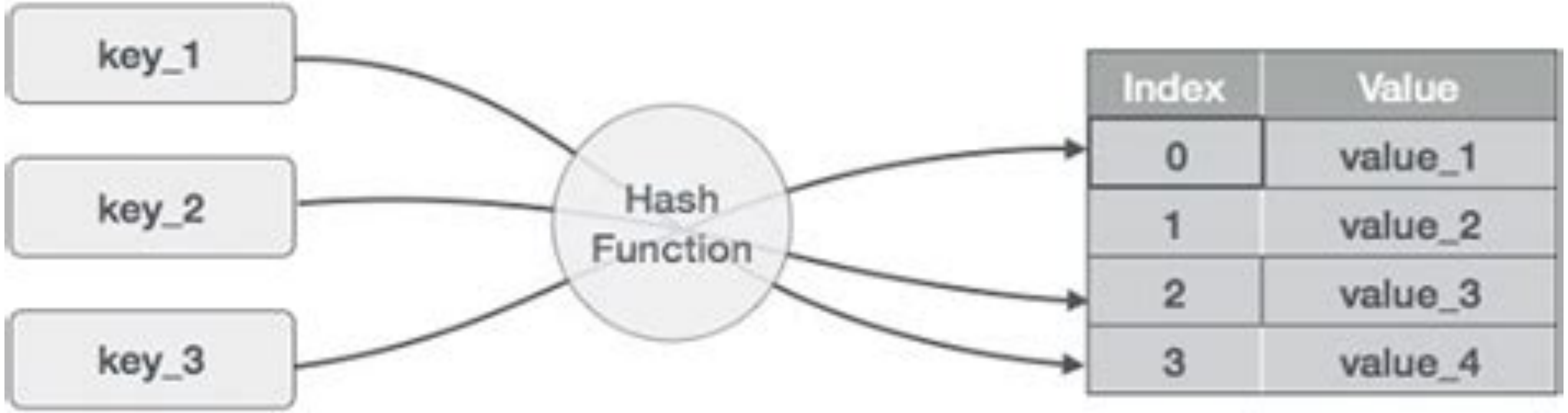
- Overview & Properties of Hash Functions

Overview of One-way Hash Functions

- One-way Hash Functions are an *essential* building block in cryptography, with desirable ***practical*** and ***security*** properties.
- Applications — *integrity verification, password authentication, commitments, etc.*
- Possible Attacks — *collision attacks, length extension attacks*

Hash Functions (and *Hash Tables*)

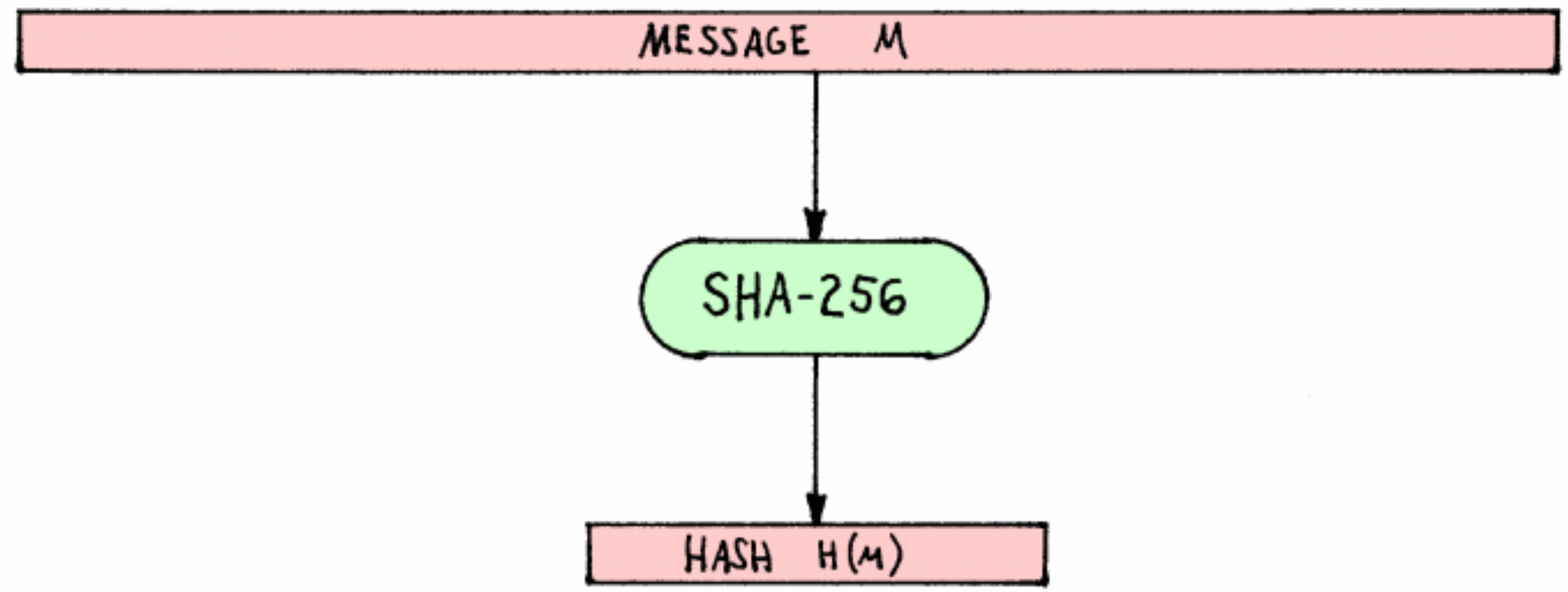
- Difference from "Normal" Hash Function
  - **Hash function:** maps arbitrary size data to data of fixed size
  - **Example:** $f(x) = x \bmod 1000$



Collisions happen…
Use your favorite collision resolution technique
(open addressing, chaining, etc.)

Practical Properties of ***One-Way*** Hash Functions

- **Compression:** compress *arbitrarily long inputs* into *fixed-length outputs*

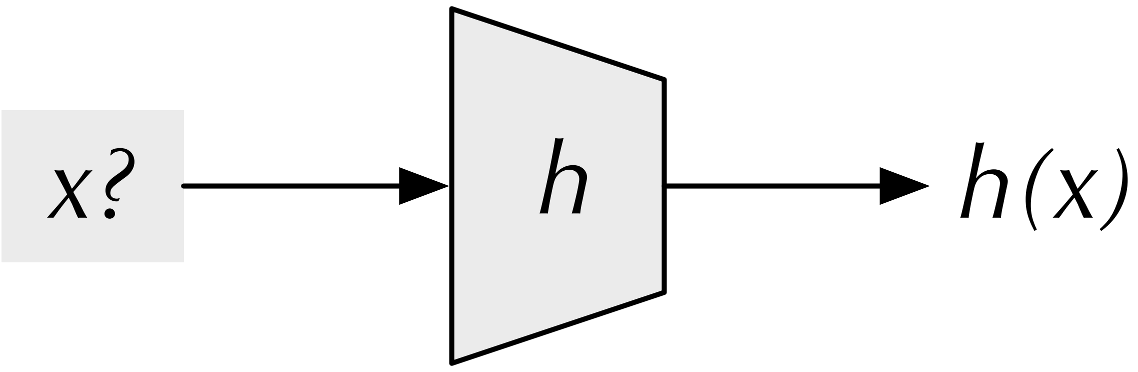


- **Easy to compute:** fast and easy (speed + efficiency) to compute

```
$ openssl speed
Doing md5 for 3s on 256 size blocks: 5123210 md5's in 3.00s
Doing hmac(md5) for 3s on 256 size blocks: 4907417 hmac(md5)'s in 3.00s
Doing sha1 for 3s on 256 size blocks: 5720106 sha1's in 2.99s
Doing sha256 for 3s on 256 size blocks: 3289471 sha256's in 3.00s
Doing sha512 for 3s on 256 size blocks: 2248701 sha512's in 3.00s
```

Security Properties of ***One-Way*** Hash Functions

- **Preimage Resistance ("One-Way")**
  Given $\mathtt{h}(x) = z$, hard to find x
  *(or **any** input that hashes to z for that matter)*



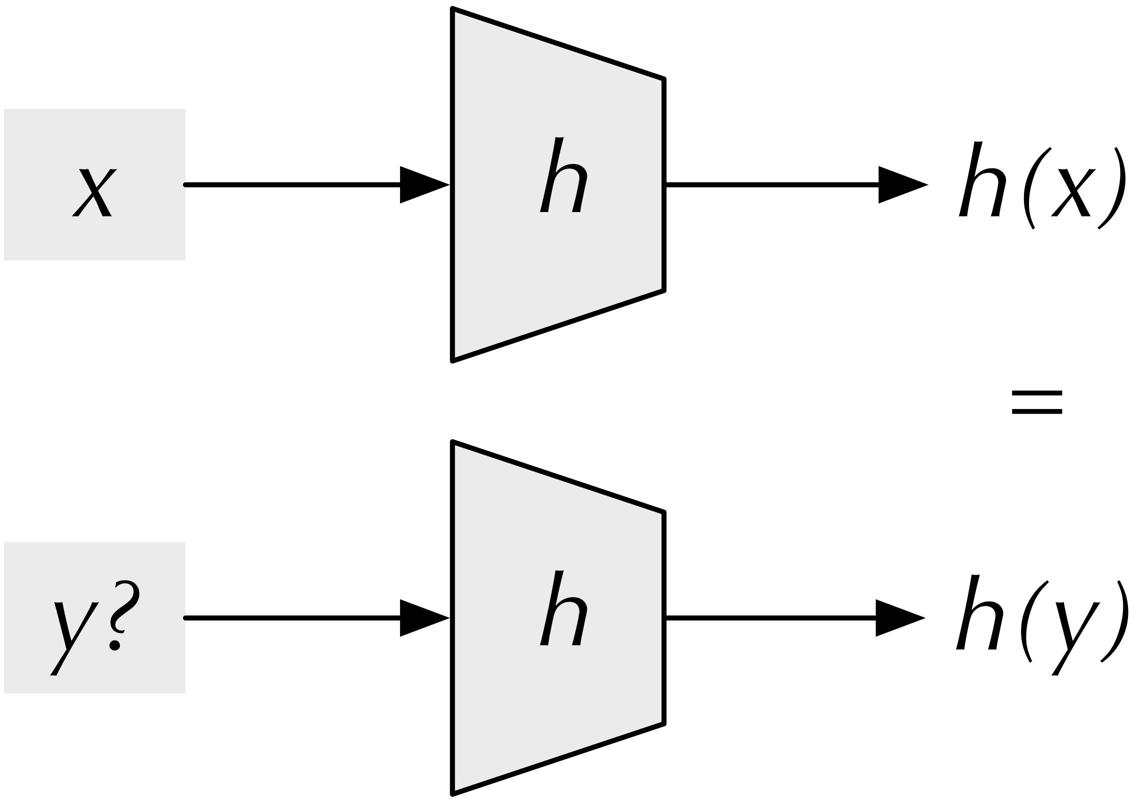
- **Second Preimage Resistance**
  Given x and $\mathtt{h}(x)$, hard to find y s.t., $\mathtt{h}(x) = \mathtt{h}(y)$

- **Collision Resistance (or, ideally, "Collision *Free*")**
  Difficult to find x and y s.t. $\mathtt{hash}(x) = \mathtt{hash}(y)$

Definitions adopted from ***Everyday Cryptography*** by Keith M. Martin (pg. 212-217)

Security Properties of ***One-Way*** Hash Functions

- **Preimage Resistance ("One-Way")**
  Given $\mathtt{h}(x) = z$, hard to find x
  *(or **any** input that hashes to z for that matter)*



- **Second Preimage Resistance**
  Given x and $\mathtt{h}(x)$, hard to find y s.t., $\mathtt{h}(x) = \mathtt{h}(y)$

- **Collision Resistance (or, ideally, "Collision *Free*")**
  Difficult to find x and y s.t. $\mathtt{hash}(x) = \mathtt{hash}(y)$

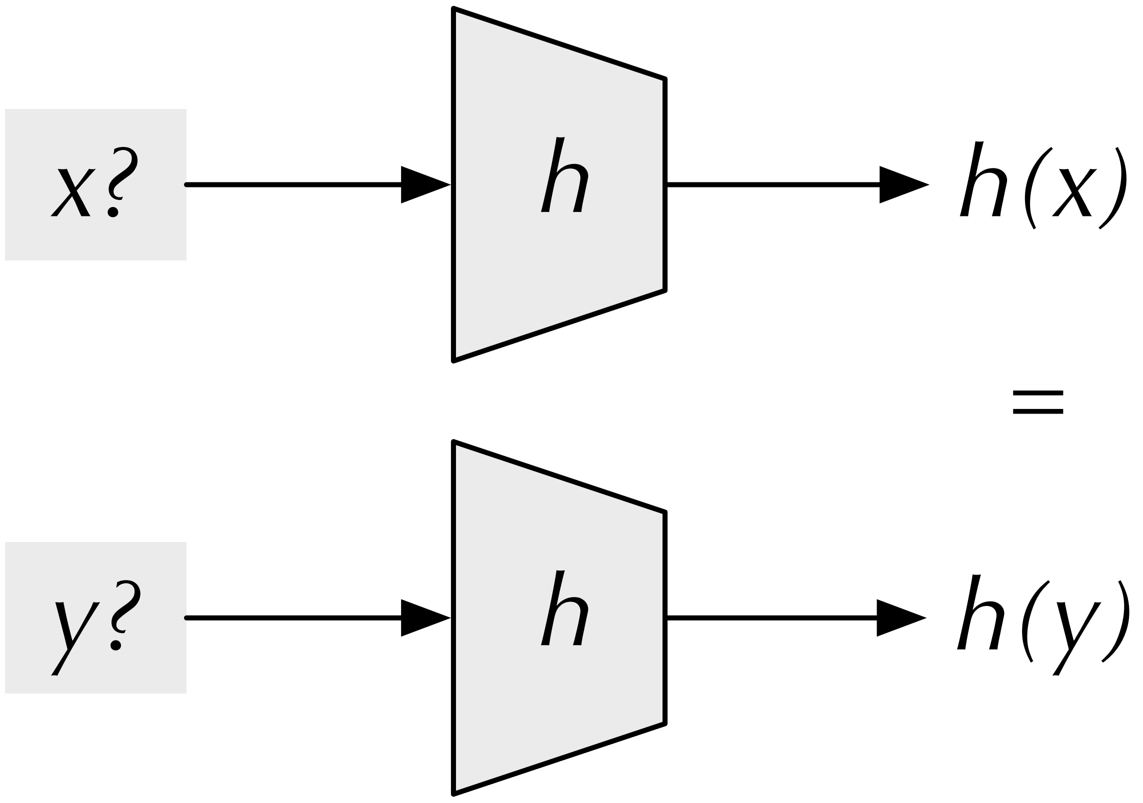
Definitions adopted from ***Everyday Cryptography*** by Keith M. Martin (pg. 212-217)

Security Properties of ***One-Way*** Hash Functions

- **Preimage Resistance ("One-Way")**
  Given $\mathtt{h}(x) = z$, hard to find x
  *(or **any** input that hashes to z for that matter)*

- **Second Preimage Resistance**
  Given x and $\mathtt{h}(x)$, hard to find y s.t., $\mathtt{h}(x) = \mathtt{h}(y)$

- **Collision Resistance (or, ideally, "Collision *Free*")**
  Difficult to find x and y s.t. $\mathtt{hash}(x) = \mathtt{hash}(y)$



Definitions adopted from ***Everyday Cryptography*** by Keith M. Martin (pg. 212-217)

Introduction to One-way Hash Functions (Part II)

This Video Covers:

- Common Hash Function Families
- Hash Function Construction
- Introduce Linux Hash Commands

The **MD** One-Way Hash Functions

- ***Message Digest***
  - Developed by Ron Rivest
  - Produces 128-bit hashes
  - Includes MD2, MD4, ***MD5***, and MD6

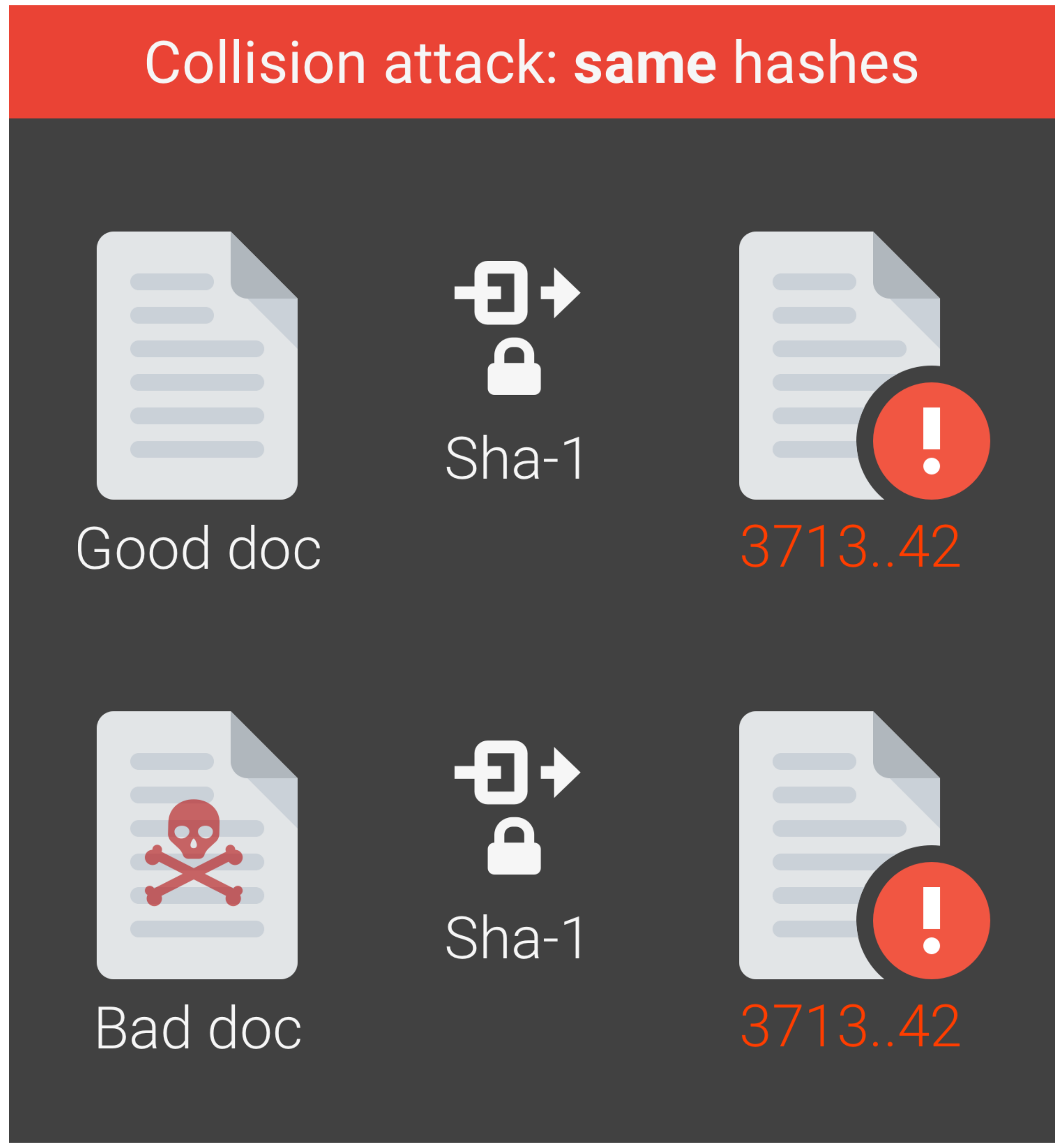
- Status of Algorithms:
  - MD2, MD4 - severely broken (obsolete)
  - **MD5** - collision resistance property broken, one-way property *not* broken
    - Often used for *file integrity checking*
    - *No longer recommended for use!*
  - MD6 - developed in response to proposal by NIST *(not widely used…)*

The **SHA** One-Way Hash Functions

- ***Secure Hash Algorithm***
  - Published by NIST
  - Includes SHA-0, SHA-1, SHA-2, and SHA-3

- Status of Algorithms:
  - SHA-0: withdrawn due to flaw
  - SHA-1: Designed by NSA → *Collision attack found in 2017*
  - SHA-2: Designed by NSA
    - *Includes SHA-256 and SHA-512 + other truncated versions;*
    - *No significant attack found yet…*
  - SHA-3: *Not* Designed by NSA
    - *Released in 2015; not a replacement to SHA-2, but meant to be a genuine alternative*
    - *Has <u>different construction structure</u> (compared to SHA-1 and SHA-2)*

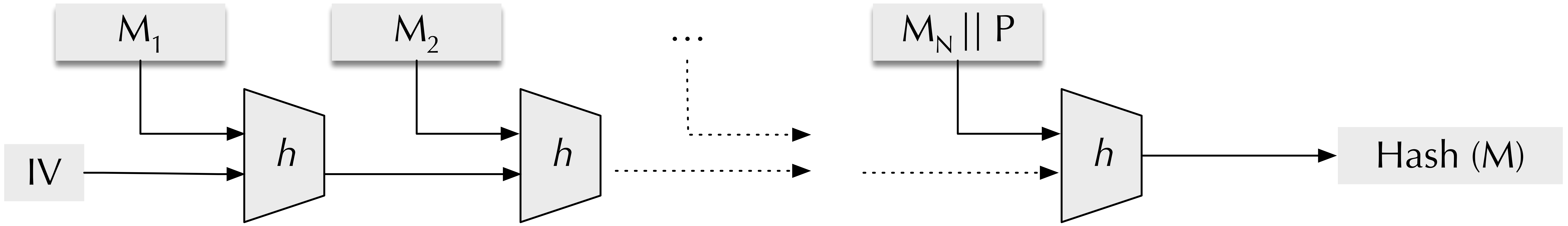
https://en.wikipedia.org/wiki/Sponge_function



https://shattered.it

How (Most) One-Way Hash Algorithms Work

Most hash algorithms *(e.g., MD5, SHA-1, SHA-2)*
use a **Merkle–Damgard** construction:



Given message M and initialization vector IV, produce Hash(M)
where
h = compression function, P = padding

Davies-Meyer *compression function uses a* ***block cipher*** *to construct a compression function*
(e.g., ***SHA family*** *uses this compression function)*
Others are possible…

One-Way Hash Commands

Linux utility programs

- e.g., `md5sum, sha256sum, sha512sum, openssl *`

```
$ md5sum print_array.c
aef3a2cac2b4153b9b5a9ff702892e12  print_array.c
$ sha256sum print_array.c
d7653b35b8c37423c6a70852dc373a3e3b2873feab6d19d9d8899eb0e2b5fce0  print_array.c

$ openssl dgst -sha256 print_array.c
SHA256(print_array.c)= d7653b35b8c37423c6a70852dc373a3e3b2873feab6d19d9d8899eb0e2b5fce0
$ openssl sha256 print_array.c
SHA256(print_array.c)= d7653b35b8c37423c6a70852dc373a3e3b2873feab6d19d9d8899eb0e2b5fce0

$ openssl dgst -md5 print_array.c
MD5(print_array.c)= aef3a2cac2b4153b9b5a9ff702892e12
$ openssl md5 print_array.c
MD5(print_array.c)= aef3a2cac2b4153b9b5a9ff702892e12
```

There is also support for hashing commands in C (openssl/sha.h), C++, Python, SQL, PHP, etc.

```
$ python -c "import hashlib; print hashlib.md5('hello').hexdigest();"
```

Applications of One-Way Hash Functions

This Video Covers:

- **Integrity Verification** — *Detecting when data has been altered*
- Commitments — *Committing a secret without telling it*
- Password Verification — *Verifying a password without storing the plaintext*

Integrity Verification

- Changing one bit of the original data changes the hash value

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

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

- Examples:
  - Detect changes in system files
  - Detect if file downloaded from website is corrupted *(e.g., SEED VM!)*

```
$ md5sum SEEDUbuntu-16.04-32bit-15-31-57-662.zip
12c48542c29c233580a23589b72b71b8  SEEDUbuntu-16.04-32bit-15-31-57-662.zip
```

VM was built in June 2019
the following servers:

- Google Drive: SEEDUbuntu-16.04-32bit.zip
- DigitalOcean: SEEDUbuntu-16.04-32bit.zip
- Cybersecurty.com: SEEDUbuntu-16.04-32bit.zip
- Syracuse University (New York, US): SEEDUbuntu-16
- Zhejiang University (Zhejiang, China): SEEDUbuntu-
- MD5 value: 12c48542c29c233580a23589b72b71b8

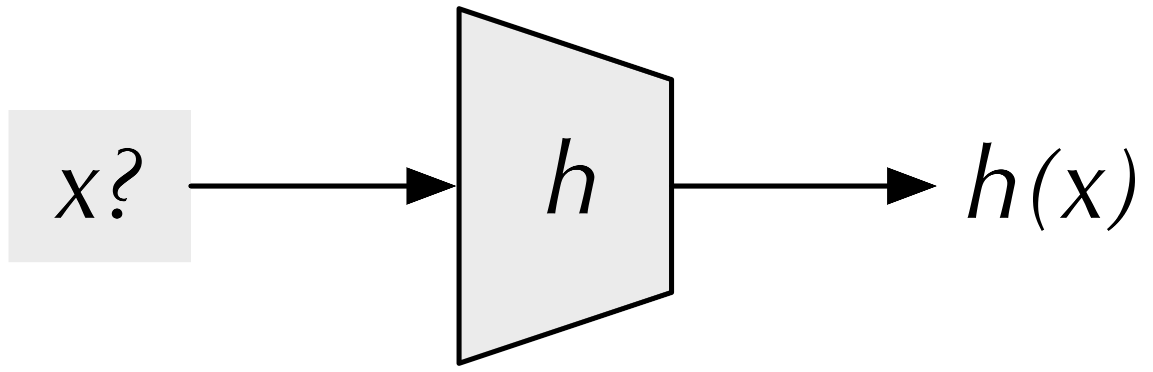
Applications of One-Way Hash Functions

This Video Covers:

- Integrity Verification — *Detecting when data has been altered*
- **Commitments** — *Committing a secret without telling it*
- Password Verification — *Verifying a password without storing the plaintext*

Commitments — *Committing a Secret Without Telling It*

- One-way property
  - Disclosing the hash does not disclose the original message
  - Useful to commit secret without disclosing the secret itself



Given $\mathtt{h}(x) = z$, hard to find x
*(or **any** input that hashes to z for that matter)*

Commitments — *Committing a Secret Without Telling It*

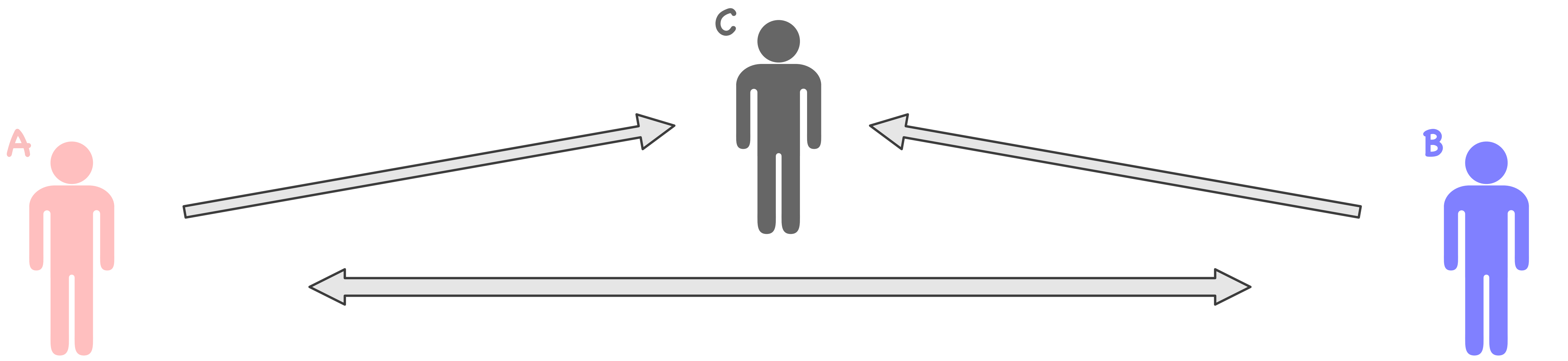
- One-way property
  - Disclosing the hash does not disclose the original message
  - Useful to commit secret without disclosing the secret itself

- Example:
  Fair Games

Commitments — *Committing a Secret Without Telling It*

- One-way property
  - Disclosing the hash does not disclose the original message
  - Useful to commit secret without disclosing the secret itself

- Example:
  Fair Games / Contract Bids



Send Commitments

h(A's bid) = 2f9a5...

h(B's bid) = c1558...

Reveal Bids

A's bid

B's bid

Verify Bids

h(B's bid) = c1558...??

h(A's bid) = 2f9a5...??

Applications of One-Way Hash Functions

This Video Covers:

- Integrity Verification — *Detecting when data has been altered*
- Commitments — *Committing a secret without telling it*
- **Password Verification** — *Verifying a password without storing the plaintext*

Password Verification

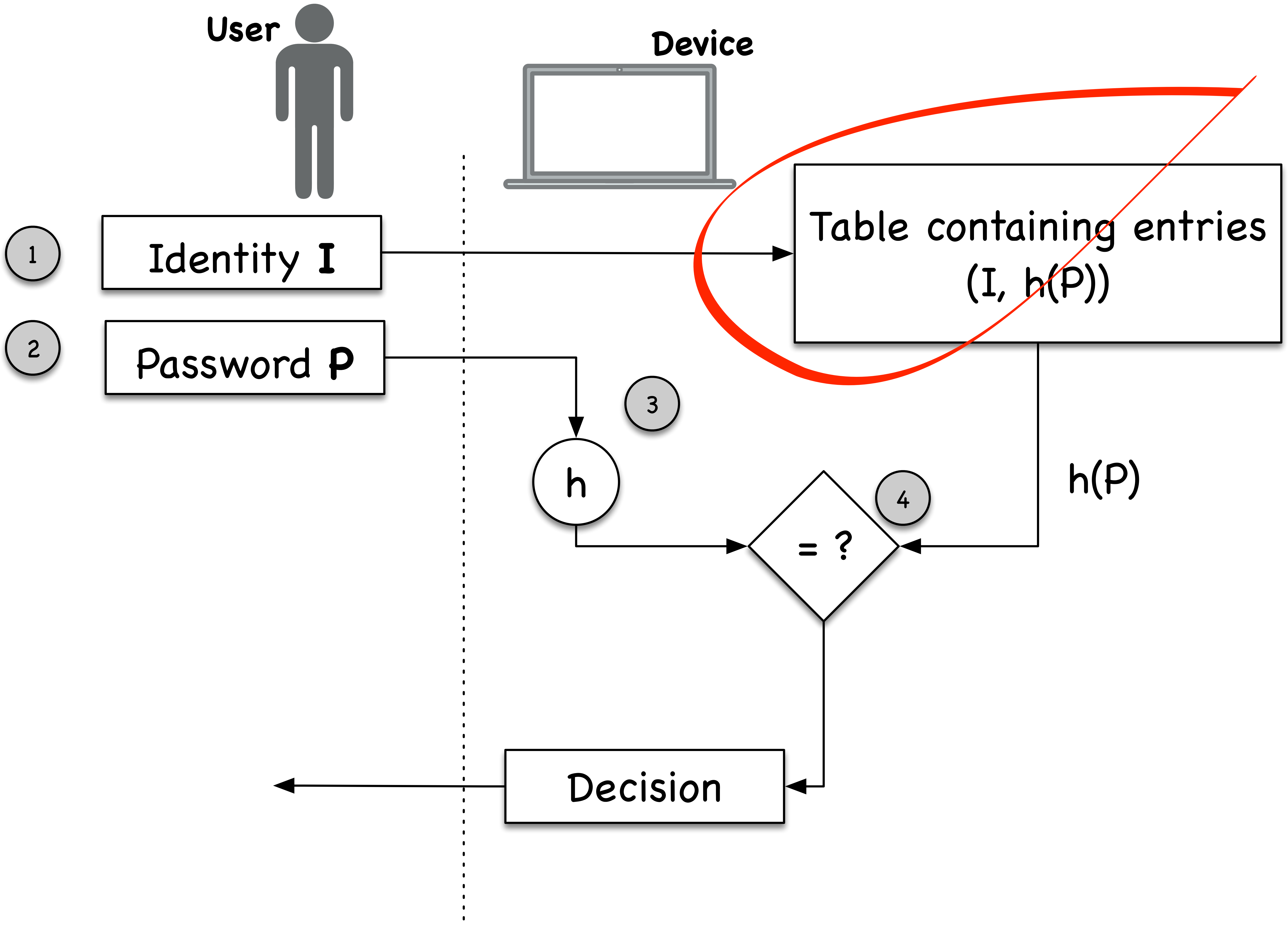
- To login into account, user needs to know the secret (password)
- Should ***not*** store the secrets in their plaintext form
- **Requirements:**
  - Password storage where nobody can know what the password is
  - If provided with a password, it verified against the stored password

- **Solution:** store *hash of password* using one-way hash function

Example: Linux stores passwords in the `/etc/shadow` file

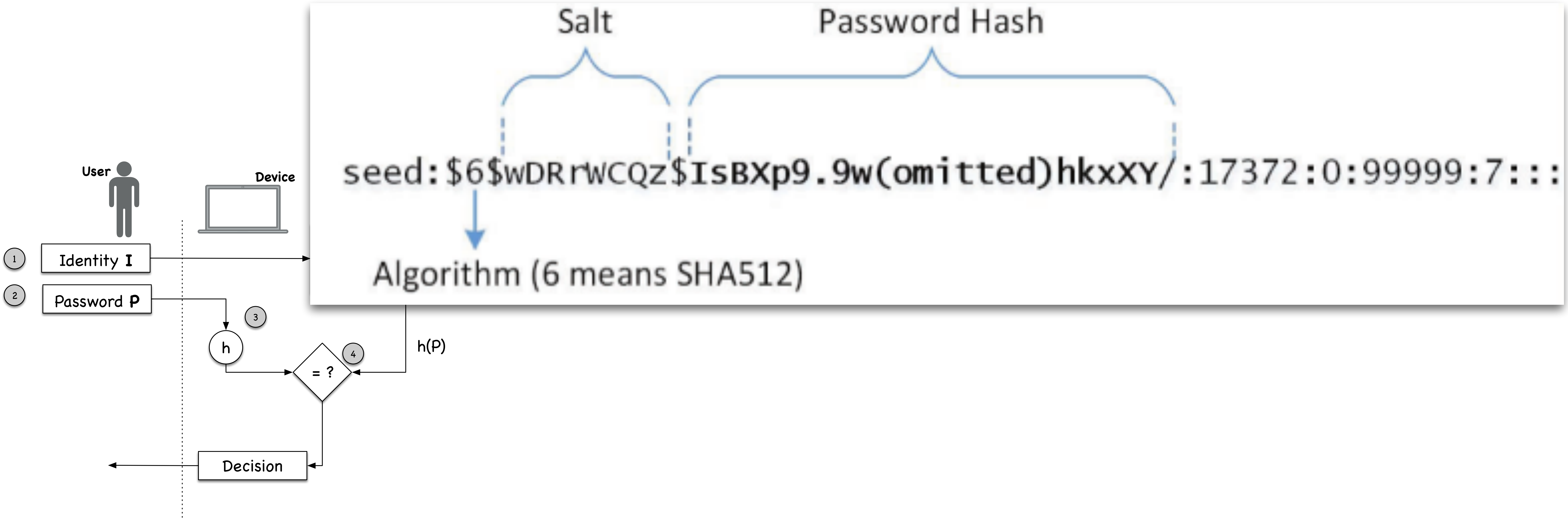
```
$ sudo cat /etc/shadow
root:$6$NrF46O1p$.vDnKEtVFC2bXsl ...(omitted)... spr/kqzAqtcu.:17400:0:99999:7:::
...
seed:$6$wDRrWCQz$IsBXp9.9wz9SGrF ...(omitted)... J8sbCT7hkxXY/:17372:0:99999:7:::
john:$6$6MiP8itO$uFVUFX8qZnxcIUD ...(omitted)... Fz/biD8mR7an.:18290:0:99999:7:::
newseed:$6$ZPwHFy.m$tKETCWrzE6WL ...(omitted)... cDsSgSm4TNRrf:18290:0:99999:7:::
```

Password Verification (High-Level Overview)



Case Study: Linux Shadow File

- Password field has 3 parts: the ***algorithm*** used, ***salt***, ***password hash***
- Salt and password hash are encoded into printable characters *(e.g., base64)*
- Multiple rounds of hash function → *slow down brute-force attack*



Purpose of Salt

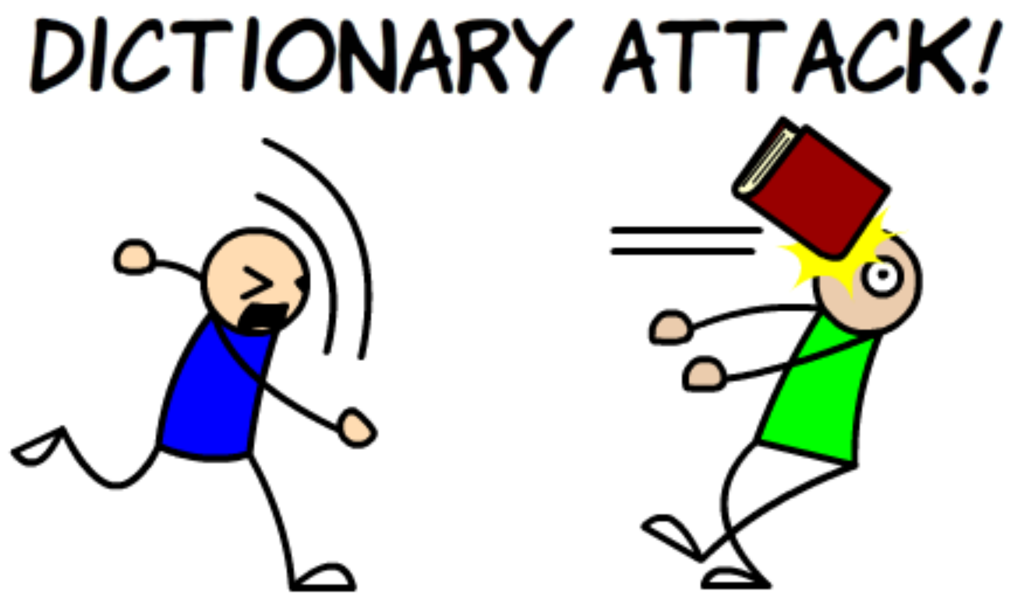
So what is the purpose of a "salt"?

- Salt is nothing more than a random value (string)
- Using salt, the same input can result in different hashes
- Password hash = one-way hash rounds (password || random string)

```
$ python
>>> import crypt
>>> print crypt.crypt('dees', '$6$wDRrWCQz')
$6$wDRrWCQz$IsBXp9.9wz9SGrF  ...(omitted)... J8sbCT7hkxXY/
```

```
$ sudo cat /etc/shadow
...
seed:$6$wDRrWCQz$IsBXp9.9wz9SGrF ...(omitted)... J8sbCT7hkxXY/:17372:0:99999:7:::
...
```

Attacks Prevented by Salt



- ***Dictionary Attack***
  - Put candidate words in a dictionary
  - Try each against the targeted password hash to find a match
- ***Rainbow Table Attack***
  - Precomputed table for reversing cryptographic hash functions
- ***How Does A Salt Prevent These Attacks?***
  - If target password is same as precomputed data, the hash will be the same
  - If this property does not hold, all the precomputed data are useless
  - Salt destroys that property

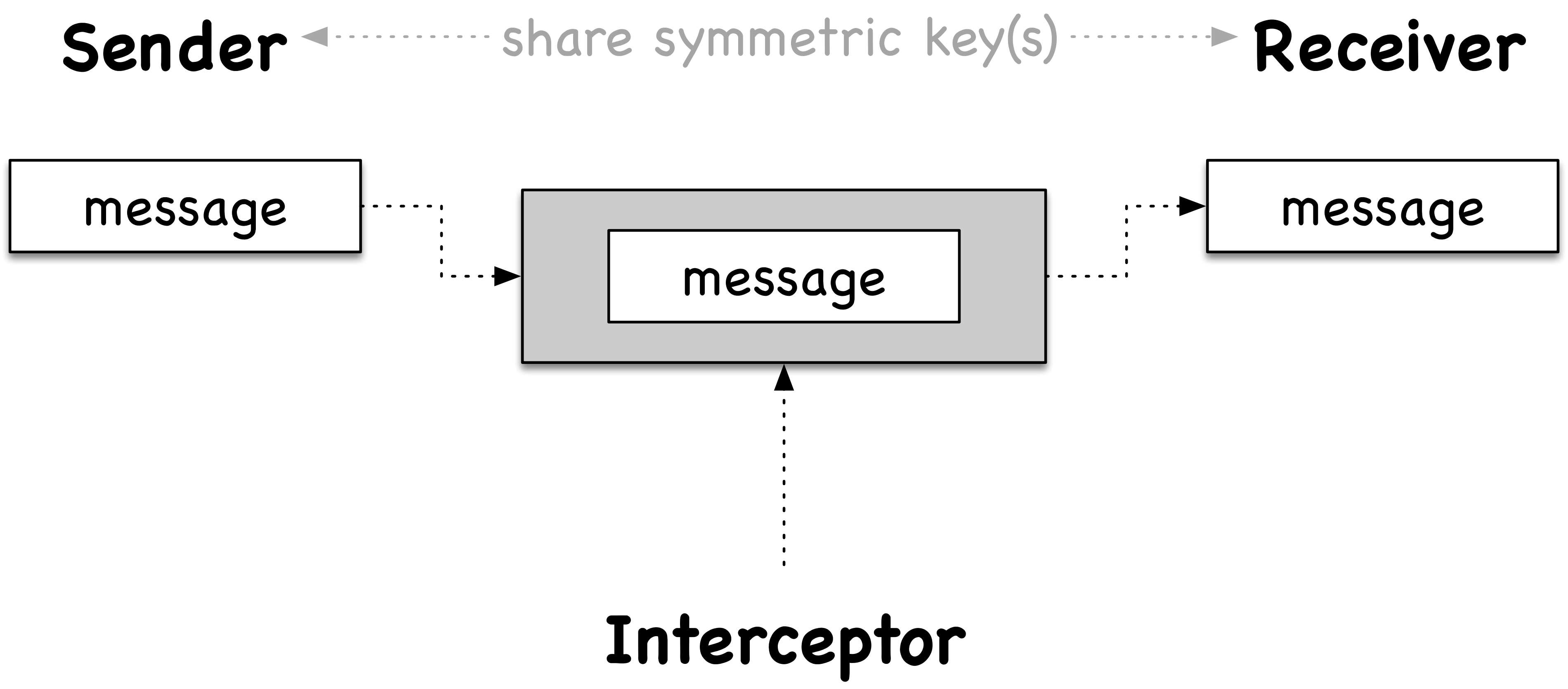
Message Authentication Code (MAC)

This Video Covers:

- MACs — what are they and how do they work?

Message Authentication Code (MAC)

Problem:
- MITM attacks possible on network communication
- MITM can *intercept* and *modify* data
- Receiver needs to verify integrity of data

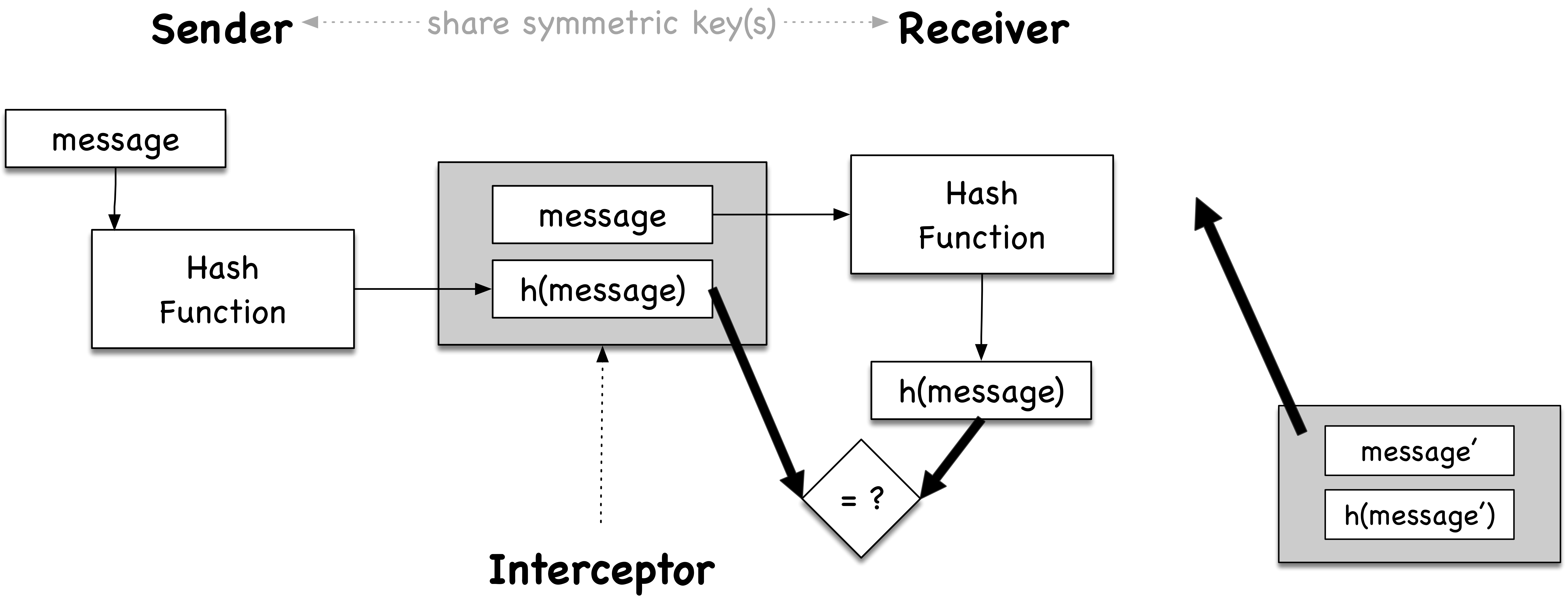


Solution: Attach a ***tag*** to data
- ***Don't*** use *(only…)* a one-way hash as tag *(MITM can recompute hash!)*
- ***Do*** use a shared secret (key) between sender and receiver in the hash
- MITM cannot compute hash without secret key

→ ***(Keyed) "Hash-based MAC" (HMAC)***

Using Only a One-Way Hash Function…

Why should we not just use a one-way hash function to compute the *tag*?
→ *MITM can generate a new message (re)compute its hash!*

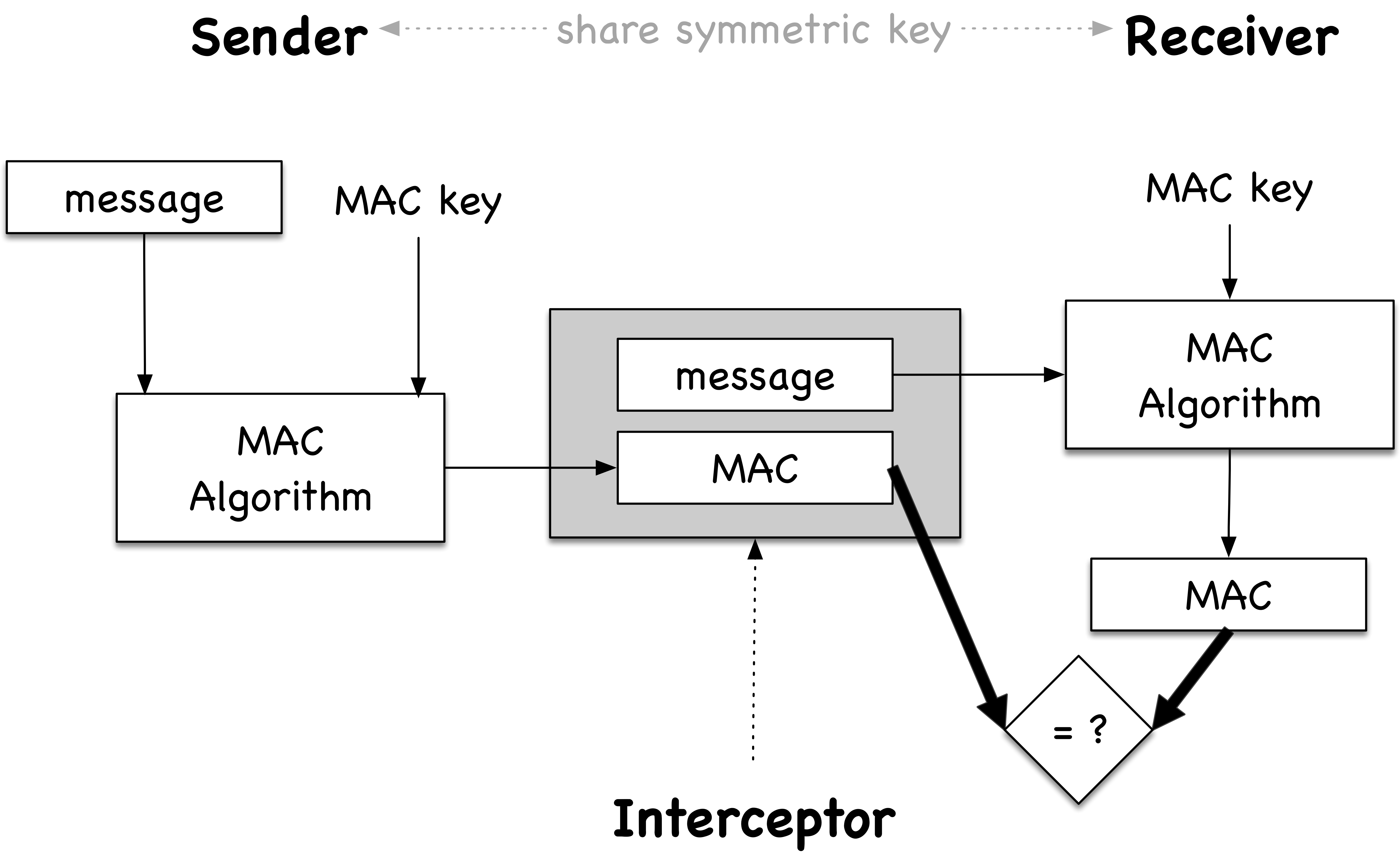


Without a KEY, attacker can generate their own message' and valid hash — h(message')

Message Authentication Code (MAC)

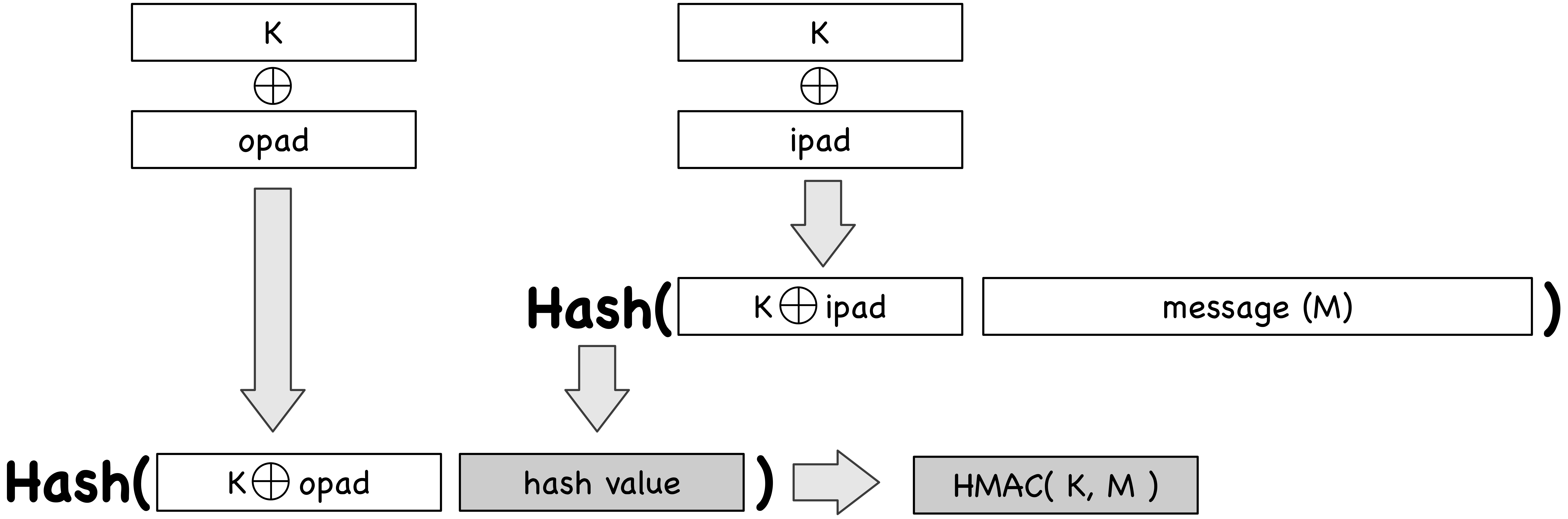
Why use a MAC algorithm to generate the *MAC?*

→ *MITM cannot generate a new message/MAC pair without knowledge of the key!*



Hash-based Message Authentication Code (HMAC)

- Different approaches for building MAC algorithm
  - Based on block cipher (e.g., CBC-MAC)
  - Based on cryptographic hash function (e.g., HMAC) $\rightarrow h(K_1 \,||\, h(K_2 \,||\, M))$



Hash Collision Attacks

This Video Covers:

- Security Impact of Collision Attacks
- Generating Two Different Files with the Same MD5 Hash
- Generating Two Programs with the Same MD5 Hash

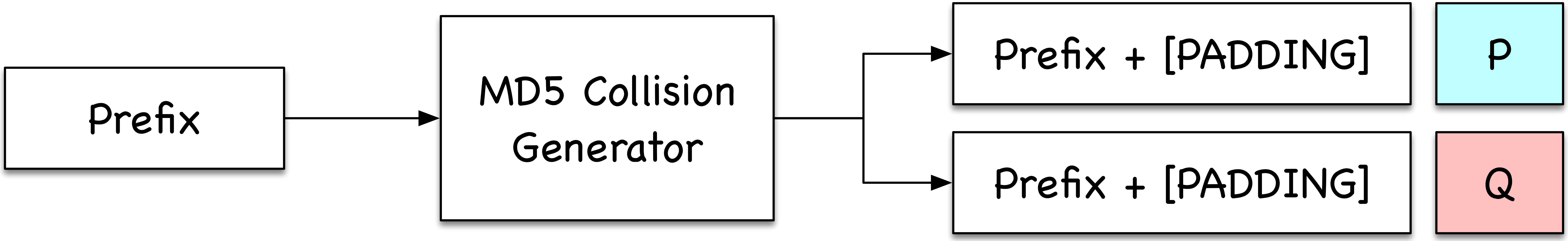
Security Impact of Collision Attacks

- **Forging public-key certificates**
  - Assume two certificate requests for www.example.com and www.attacker.com have same hash due to a collision
  - CA signing of either request would be equivalent
  - Attacker can get certificate signed for www.example.com without owning it!

- **Integrity of Programs**
  - Ask CA to sign a legitimate program's hash
  - Attacker creates a malicious program with same hash
  - The certificate for legitimate program is also valid for malicious version

Generating Two ***Different Files*** w/ ***Same MD5 Hash***

`md5collgen` *tool generates two files with same prefix*

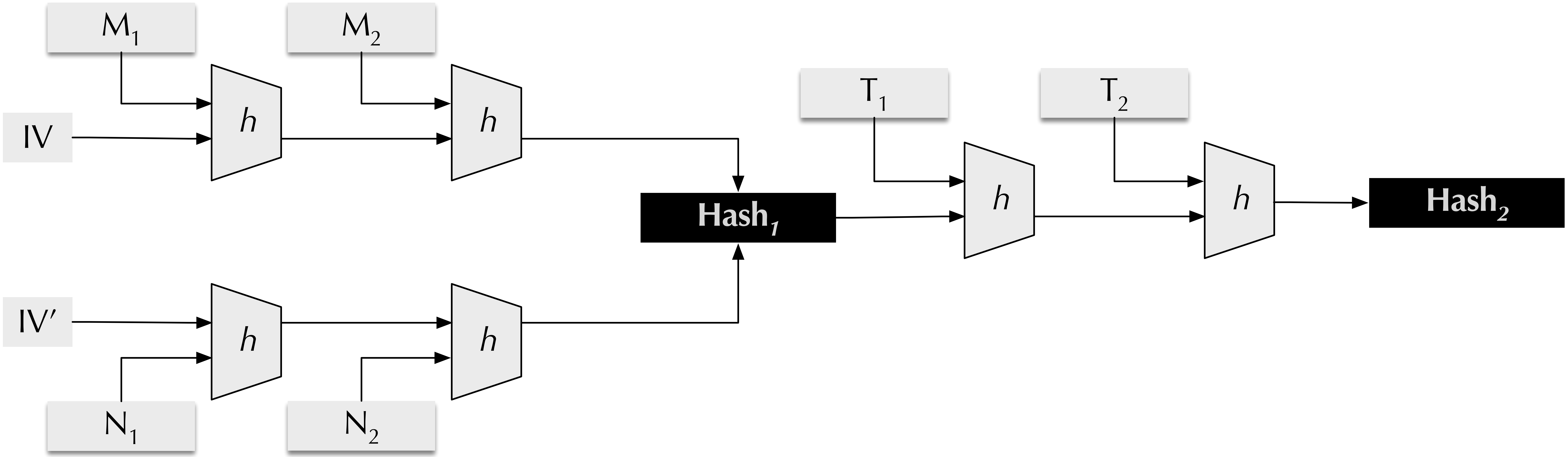



```
$ echo "Message prefix" > prefix.txt
$ md5collgen -p prefix.txt -o out1.bin out2.bin
...
```

```
$ md5sum out1.bin
f53f8e097ffe4fd3710aad0fbac17123  out1.bin
$ md5sum out2.bin
f53f8e097ffe4fd3710aad0fbac17123  out2.bin
```

Length Extension

- Generate two files with same prefix and same suffix
  - Focus on MD5, SHA-1, SHA-2 using Merkle-Damgard construction
  - If $\texttt{hash}(M) = \texttt{hash}(N)$, then for any input T, $\texttt{hash}(M \,||\, T) = \texttt{hash}(N \,||\, T)$,



Length Extension (cont.)

Example using `out1.bin` *and* `out2.bin` *generated by* `md5collgen`

```
$ echo "Message suffix" > suffix.txt
$ cat out1.bin suffix.txt > out1_long.bin
$ cat out2.bin suffix.txt > out2_long.bin
```

```
$ diff out1_long.bin out2_long.bin
Binary files out1_long.bin and out2_long.bin differ
```

```
$ md5sum out1_long.bin
0fbe0c2e0fc197a0f053b0640c7fd2d5  out1_long.bin
$ md5sum out2_long.bin
0fbe0c2e0fc197a0f053b0640c7fd2d5  out2_long.bin
```

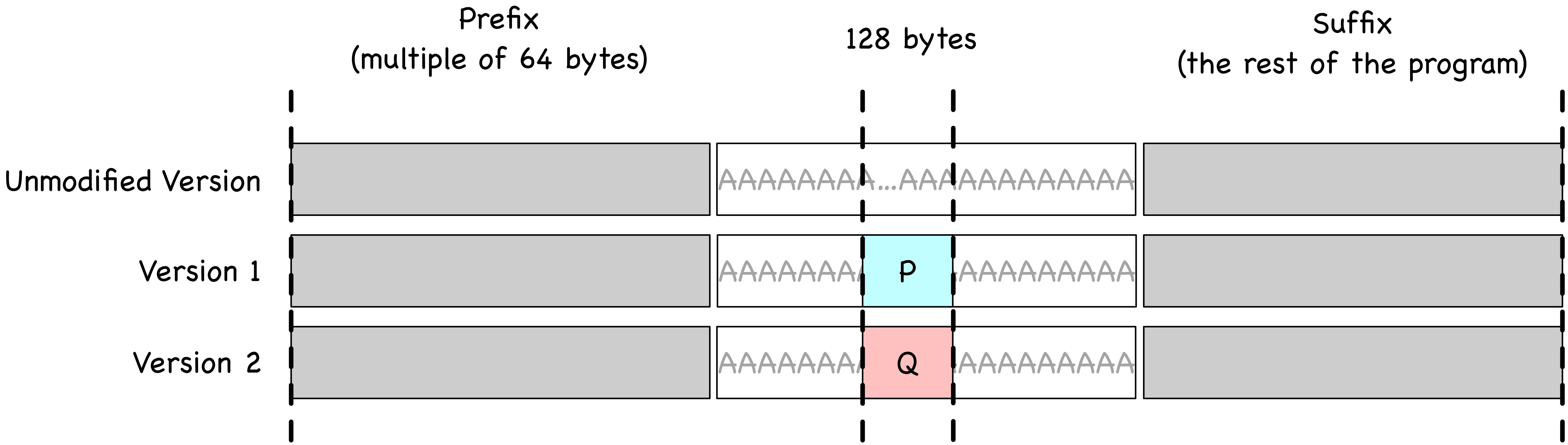
Generating Two ***Different Programs*** w/ ***Same MD5 Hash***

Create two versions of a program with different values for the array `xyz`

```
$ cat print_array.c
#include <stdio.h>

unsigned char xyz[200] = { /* The contents of this array are set by you */ }

int main()
{
  int i;
  for (i=0; i<200; i++){
    printf("%x", xyz[i]);
  }
  printf("\n");
}
```

Generating Two ***Different Programs*** w/ ***Same MD5 Hash*** (cont.)

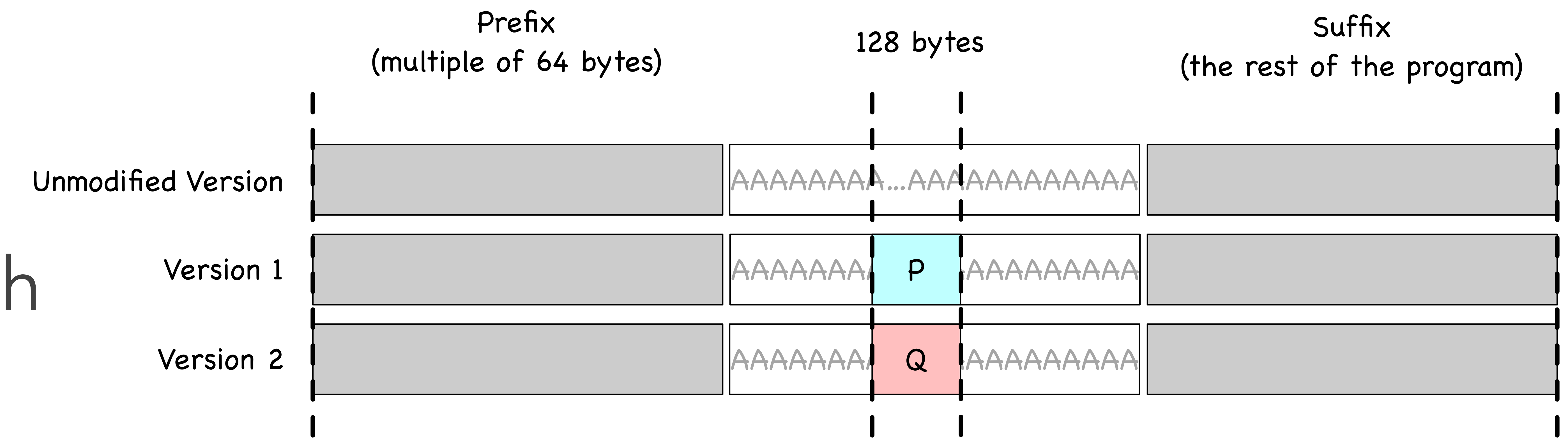
- Program will be compiled into binary (***tip:*** *fill `xyz` with fixed value)*
- Portion of binary containing `xyz` will be divided into three parts



Generating Two ***Different Programs*** w/ ***Same MD5 Hash*** (cont.)

Use `md5collgen` on prefix:

- generate two files with same hash
- last 128 bits of each generated file is P and Q



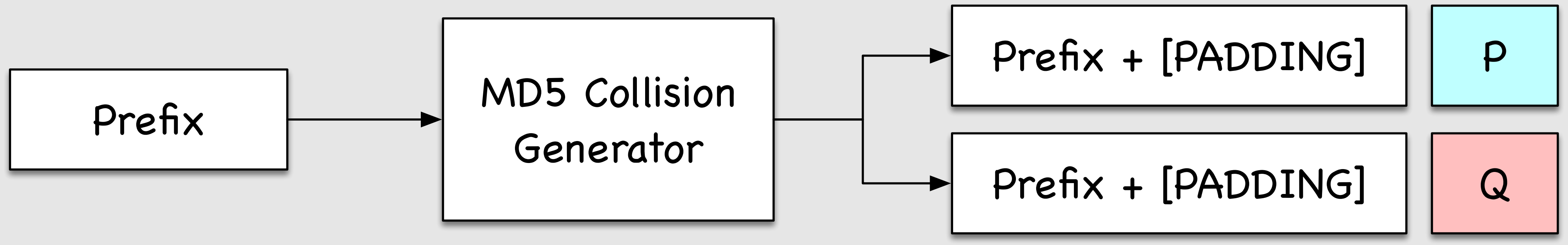
$$\mathtt{md5}(\mathtt{prefix} \parallel P) = \mathtt{md5}(\mathtt{prefix} \parallel Q)$$

$$\Downarrow$$

$$\mathtt{md5}(\mathtt{prefix} \parallel P \parallel \mathtt{suffix}) = \mathtt{md5}(\mathtt{prefix} \parallel Q \parallel \mathtt{suffix})$$

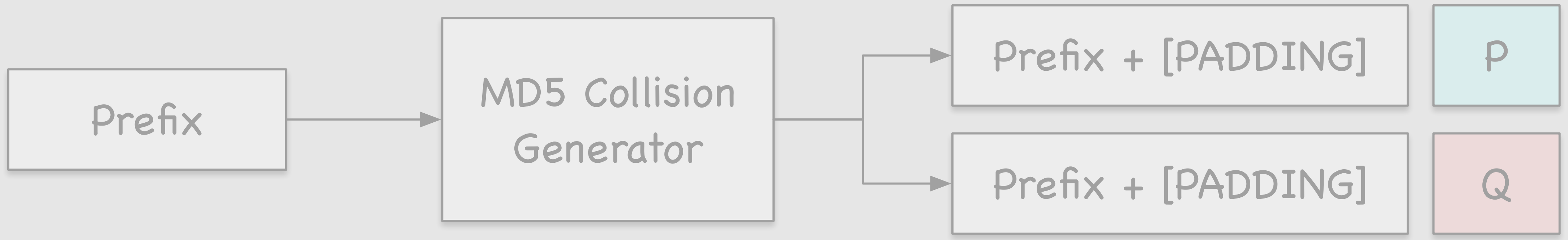
Generating Two ***Different Programs*** w/ ***Same MD5 Hash*** (cont.)

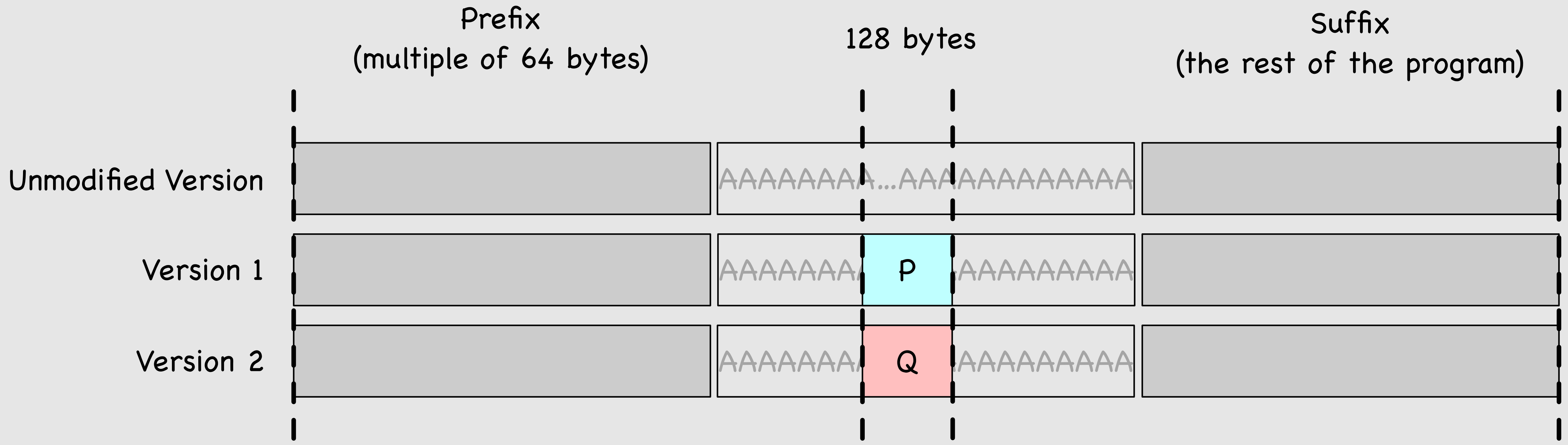
```
$ gcc print_array.c -o pa
$ ghex pa # confirm offset of start of array xyz — I see 4160
$ head -c  4160 pa > prefix
$ tail -c +4288 pa > suffix # 4160+128=4288

$ md5collgen -p prefix -o out1.bin out2.bin
$ tail -c 128 out1.bin > P
$ tail -c 128 out2.bin > Q

$ cat prefix P suffix > a1.out
$ cat prefix Q suffix > a2.out
$ chmod a+x a1.out a2.out

$ diff a1.out a2.out
Binary files a1.out and a2.out differ
$ md5sum a1.out
c09b82f44e37f7d3d32919fa7878d660  a1.out
$ md5sum a2.out
c09b82f44e37f7d3d32919fa7878d660  a2.out

$ vimdiff <(./a1.out) <(./a2.out)# can you spot the difference?!
```



Generating Two *Different Programs* w/ *Same MD5 Hash* (cont.)

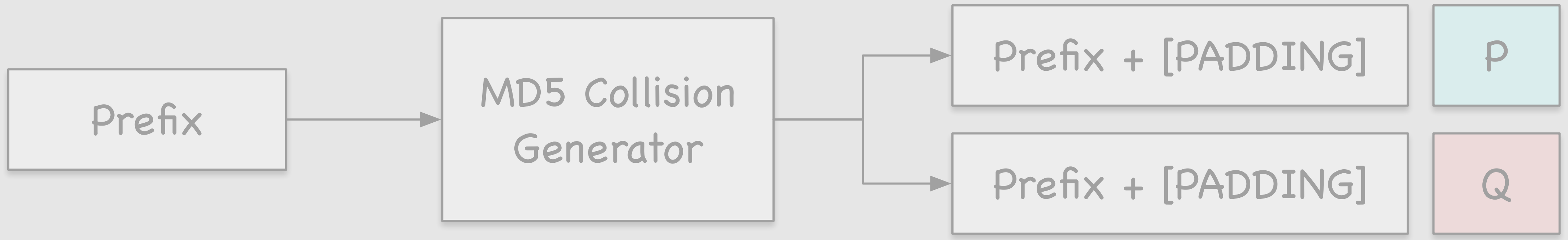
```
$ gcc print_array.c -o pa
$ ghex pa # confirm offset of start of array xyz — I see 4160
$ head -c  4160 pa > prefix
$ tail -c +4288 pa > suffix # 4160+128=4288

$ md5collgen -p prefix -o out1.bin out2.bin
$ tail -c 128 out1.bin > P
$ tail -c 128 out2.bin > Q

$ cat prefix P suffix > a1.out
$ cat prefix Q suffix > a2.out
$ chmod a+x a1.out a2.out

$ diff a1.out a2.out
Binary files a1.out and a2.out differ
$ md5sum a1.out
c09b82f44e37f7d3d32919fa7878d660  a1.out
$ md5sum a2.out
c09b82f44e37f7d3d32919fa7878d660  a2.out

$ vimdiff <(./a1.out) <(./a2.out)# can you spot the difference?!
```

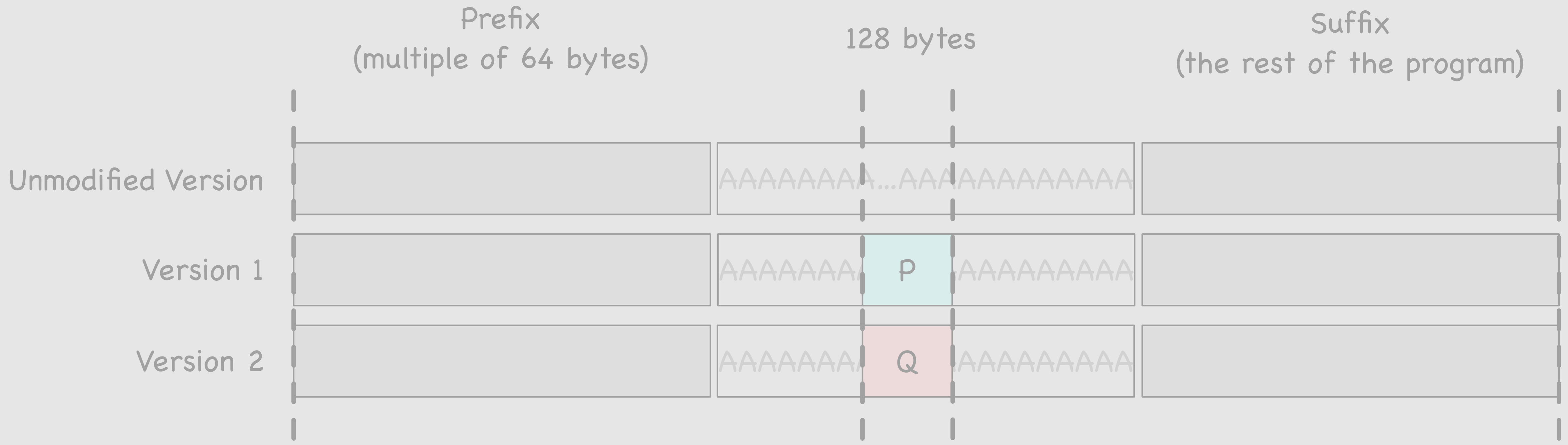




Generating Two ***Different Programs*** w/ ***Same MD5 Hash*** (cont.)

```
$ gcc print_array.c -o pa
$ ghex pa # confirm offset of start of array xyz — I see 4160
$ head -c  4160 pa > prefix
$ tail -c +4288 pa > suffix # 4160+128=4288

$ md5collgen -p prefix -o out1.bin out2.bin
$ tail -c 128 out1.bin > P
$ tail -c 128 out2.bin > Q

$ cat prefix P suffix > a1.out
$ cat prefix Q suffix > a2.out
$ chmod a+x a1.out a2.out

$ diff a1.out a2.out
Binary files a1.out and a2.out differ
$ md5sum a1.out
c09b82f44e37f7d3d32919fa7878d660  a1.out
$ md5sum a2.out
c09b82f44e37f7d3d32919fa7878d660  a2.out

$ vimdiff <(./a1.out) <(./a2.out)# can you spot the difference?!
```





In the lab, you'll try this and even take it one step further :-)

Summary: *One-Way Hash Functions*

- One-way hash functions are an essential building block in cryptography
- Important Properties: *one-way* and *collision resistant*
- Applications:
  - File integrity
  - Commitments
  - Password authentication
  - MAC used to preserve integrity of communication
- Attacks on one-way hashes

  *(length extension attacks and **collision attacks**)*