



CS 553

CRYPTOGRAPHY

Lecture 18

Hash Functions

Instructor
Dr. Dhiman Saha

Cryptographic Hash Functions

HASH, x. There is no definition for this word — nobody knows what hash is.

Ambrose Bierce, The Devils Dictionary

A many-to-one mapping



The Cryptographers Swiss Army Knife

Cryptographic Hash Functions used in

- ▶ Digital signatures
- ▶ Public-key encryption
- ▶ Integrity verification
- ▶ Message authentication
- ▶ Password protection
- ▶ Key agreement protocols
- ▶ And many other cryptographic protocols.

Hash functions are by far the most versatile and ubiquitous of all crypto algorithms.

The Cryptographers Swiss Army Knife

Cryptographic Hash Functions used in

- ▶ Digital signatures
- ▶ Public-key encryption
- ▶ Integrity verification
- ▶ Message authentication
- ▶ Password protection
- ▶ Key agreement protocols
- ▶ And many other cryptographic protocols.

Hash functions are by far the most versatile and ubiquitous of all crypto algorithms.

The Cryptographers Swiss Army Knife

Cryptographic Hash Functions used in

- ▶ Digital signatures
- ▶ Public-key encryption
- ▶ Integrity verification
- ▶ Message authentication
- ▶ Password protection
- ▶ Key agreement protocols
- ▶ And many other cryptographic protocols.

Hash functions are by far the most versatile and ubiquitous of all crypto algorithms.

The Cryptographers Swiss Army Knife

Cryptographic Hash Functions used in

- ▶ Digital signatures
- ▶ Public-key encryption
- ▶ Integrity verification
- ▶ Message authentication
- ▶ Password protection
- ▶ Key agreement protocols
- ▶ And many other cryptographic protocols.

Hash functions are by far the most versatile and ubiquitous of all crypto algorithms.

The Cryptographers Swiss Army Knife

Cryptographic Hash Functions used in

- ▶ Digital signatures
- ▶ Public-key encryption
- ▶ Integrity verification
- ▶ Message authentication
- ▶ Password protection
- ▶ Key agreement protocols
- ▶ And many other cryptographic protocols.

Hash functions are by far the most versatile and ubiquitous of all crypto algorithms.

The Cryptographers Swiss Army Knife

Cryptographic Hash Functions used in

- ▶ Digital signatures
- ▶ Public-key encryption
- ▶ Integrity verification
- ▶ Message authentication
- ▶ Password protection
- ▶ Key agreement protocols
- ▶ And many other cryptographic protocols.

Hash functions are by far the most versatile and ubiquitous of all crypto algorithms.

The Cryptographers Swiss Army Knife

Cryptographic Hash Functions used in

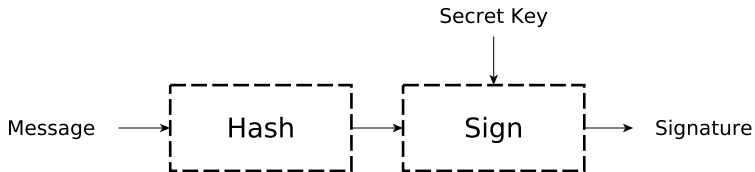
- ▶ Digital signatures
- ▶ Public-key encryption
- ▶ Integrity verification
- ▶ Message authentication
- ▶ Password protection
- ▶ Key agreement protocols
- ▶ And many other cryptographic protocols.

Hash functions are by far the most versatile and ubiquitous of all crypto algorithms.

Undoubtedly Yes!

- ▶ Connecting to a HTTPS website
- ▶ Remote connection using SSH/IPSec
- ▶ Intrusion Detection Systems
- ▶ Forensic Analysis
- ▶ Bitcoin in proof-of-work
- ▶ Revision control systems like GIT
- ▶ And so on.

Use in Digital Signatures



The hash acts as an identifier for the message.

Do not confuse cryptographic hash functions with **non-cryptographic** ones.

Example (Noncryptographic Hash)

Hash functions are used in

- ▶ Data structures such as hash tables
- ▶ Cyclic redundancy checks (CRCs) to detect accidental modifications of a file

No notion of security for non-cryptographic hashes

Do not confuse cryptographic hash functions with **non-cryptographic** ones.

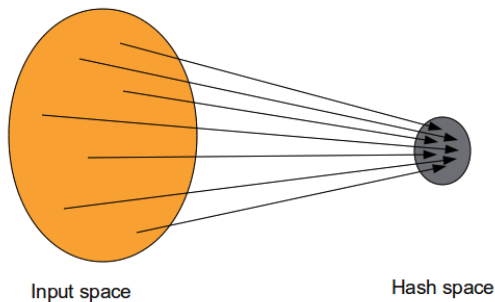
Example (Noncryptographic Hash)

Hash functions are used in

- ▶ Data structures such as hash tables
- ▶ Cyclic redundancy checks (CRCs) to detect accidental modifications of a file

No notion of security for non-cryptographic hashes

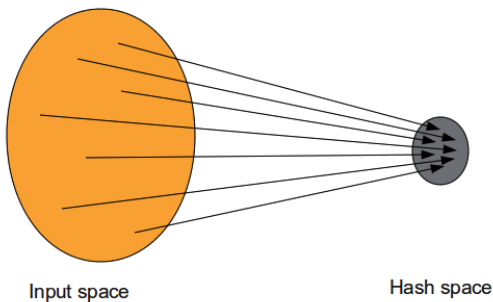
- ▶ A symmetric-key primitive
- ▶ **Variable length** input
- ▶ **Fixed** length output called **hash** or **digest**



Recently

Notion of **variable length** output

- ▶ A symmetric-key primitive
- ▶ **Variable length** input
- ▶ **Fixed** length output called **hash** or **digest**



Recently

Notion of **variable length** output

Unpredictability of Hash output

$$\text{SHA-256 ("a")} = \begin{cases} 87428fc522803d31065e7bce3cf03fe4 \\ 75096631e5e07bbd7a0fde60c4cf25c7 \end{cases}$$
$$\text{SHA-256 ("b")} = \begin{cases} a63d8014dba891345b30174df2b2a57e \\ fbb65b4f9f09b98f245d1b3192277ece \end{cases}$$

The avalanche effect

<https://passwordsgenerator.net/sha256-hash-generator/>

A secure hash function should behave like a **truly** random function

- ▶ Sometimes called a **random oracle**

More precisely

A secure hash function should **not** have any property or pattern that a random function would **not** have.

A secure hash function should behave like a **truly** random function

- ▶ Sometimes called a **random oracle**

More precisely

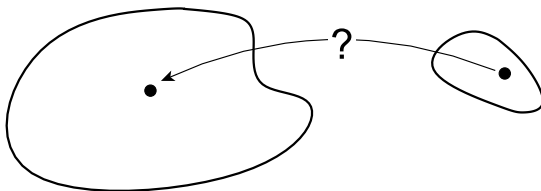
A secure hash function should **not** have any property or pattern that a random function would **not** have.

Analyzing Hash Functions

More Specific Notions

Preimage · 2nd-Preimage · Collision · Distinguishers

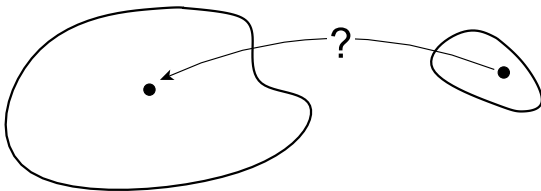
- ▶ Finding preimage
 - ▶ Get x from $h(x)$



Points to Ponder

- ▶ How many pre-images on average?
- ▶ How can one be sure?

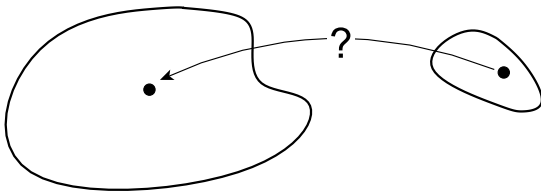
- ▶ Finding preimage
 - ▶ Get x from $h(x)$



Points to Ponder

- ▶ How many pre-images on average?
- ▶ How can one be sure?

- ▶ Finding preimage
 - ▶ Get x from $h(x)$



Points to Ponder

- ▶ How many pre-images on average?
- ▶ How can one be sure?

The optimal preimage search algorithm

```
find-preimage(H) {  
  repeat {  
    M = random_message  
    if Hash(M) == H then return M  
  }  
}
```

Complexity

2^n

The optimal preimage search algorithm

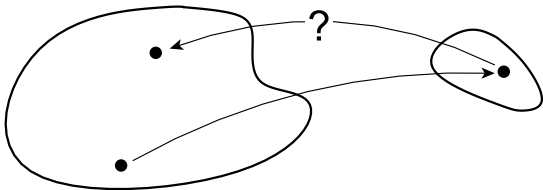
```
find-preimage(H) {  
  repeat {  
    M = random_message  
    if Hash(M) == H then return M  
  }  
}
```

Complexity

2^n

Second Pre-image Resistance

- ▶ Finding second-preimage
 - ▶ Given x , find $x' \neq x$ such that $h(x) = h(x')$



Points to Ponder

- ▶ Why Second-Preimage Resistance Is Weaker Than Preimage Resistance?

Finding Second-Preimage

For the same hash function, if you can find first preimages, you can find second preimages as well

```
solve-second-preimage(M) {  
    H = Hash(M)  
    return solve-preimage(H)  
}
```

Why?

Any second-preimage resistant hash function is also preimage resistant.

► Complexity?

For the same hash function, if you can find first preimages, you can find second preimages as well

```
solve-second-preimage(M) {  
    H = Hash(M)  
    return solve-preimage(H)  
}
```

Why?

Any second-preimage resistant hash function is also preimage resistant.

► Complexity?

For the same hash function, if you can find first preimages, you can find second preimages as well

```
solve-second-preimage(M) {  
    H = Hash(M)  
    return solve-preimage(H)  
}
```

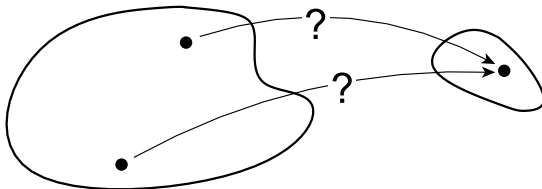
Why?

Any second-preimage resistant hash function is also preimage resistant.

► Complexity?

- Finding collisions

- Get $x \neq x'$ such that $h(x) = h(x')$

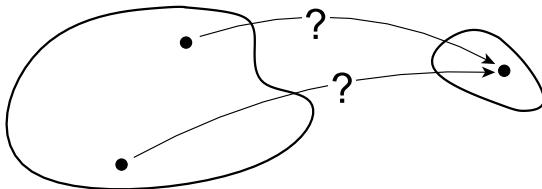


Points to Ponder

- Will collisions always exist?
- Recall the pigeonhole principle

- Finding collisions

- Get $x \neq x'$ such that $h(x) = h(x')$



Points to Ponder

- Will collisions always exist?
- Recall the pigeonhole principle

Relation with Second Preimage Resistance

If you can find second preimages for a hash function, you can also find collisions

```
solve-collision() {  
    M = random_message()  
    return (M, solve-second-preimage(M))  
}
```

Why?

Any collision-resistant hash is also second preimage resistant.

► Complexity of naive collision search algorithm?

Relation with Second Preimage Resistance

If you can find second preimages for a hash function, you can also find collisions

```
solve-collision() {  
    M = random_message()  
    return (M, solve-second-preimage(M))  
}
```

Why?

Any collision-resistant hash is also second preimage resistant.

► Complexity of naive collision search algorithm?

Relation with Second Preimage Resistance

If you can find second preimages for a hash function, you can also find collisions

```
solve-collision() {  
    M = random_message()  
    return (M, solve-second-preimage(M))  
}
```

Why?

Any collision-resistant hash is also second preimage resistant.

- Complexity of naive collision search algorithm?

Will be discussed in next class

The Birthday Paradox

- ▶ Finding preimage
 - ▶ Get x from $h(x)$
- ▶ Finding second-preimage
 - ▶ Given x , find $x' \neq x$ such that $h(x) = h(x')$
- ▶ Finding collisions
 - ▶ Get $x \neq x'$ such that $h(x) = h(x')$
- ▶ Devising distinguishers
 - ▶ Exhibit non-random behavior of $h(\cdot)$ or internal transformation

Above tasks should be **hard** for a good hash function

Lets design a simple hash function and then try to generate a pair of colliding messages