CISC 468: CRYPTOGRAPHY

LESSON 15: HASH FUNCTIONS

Furkan Alaca

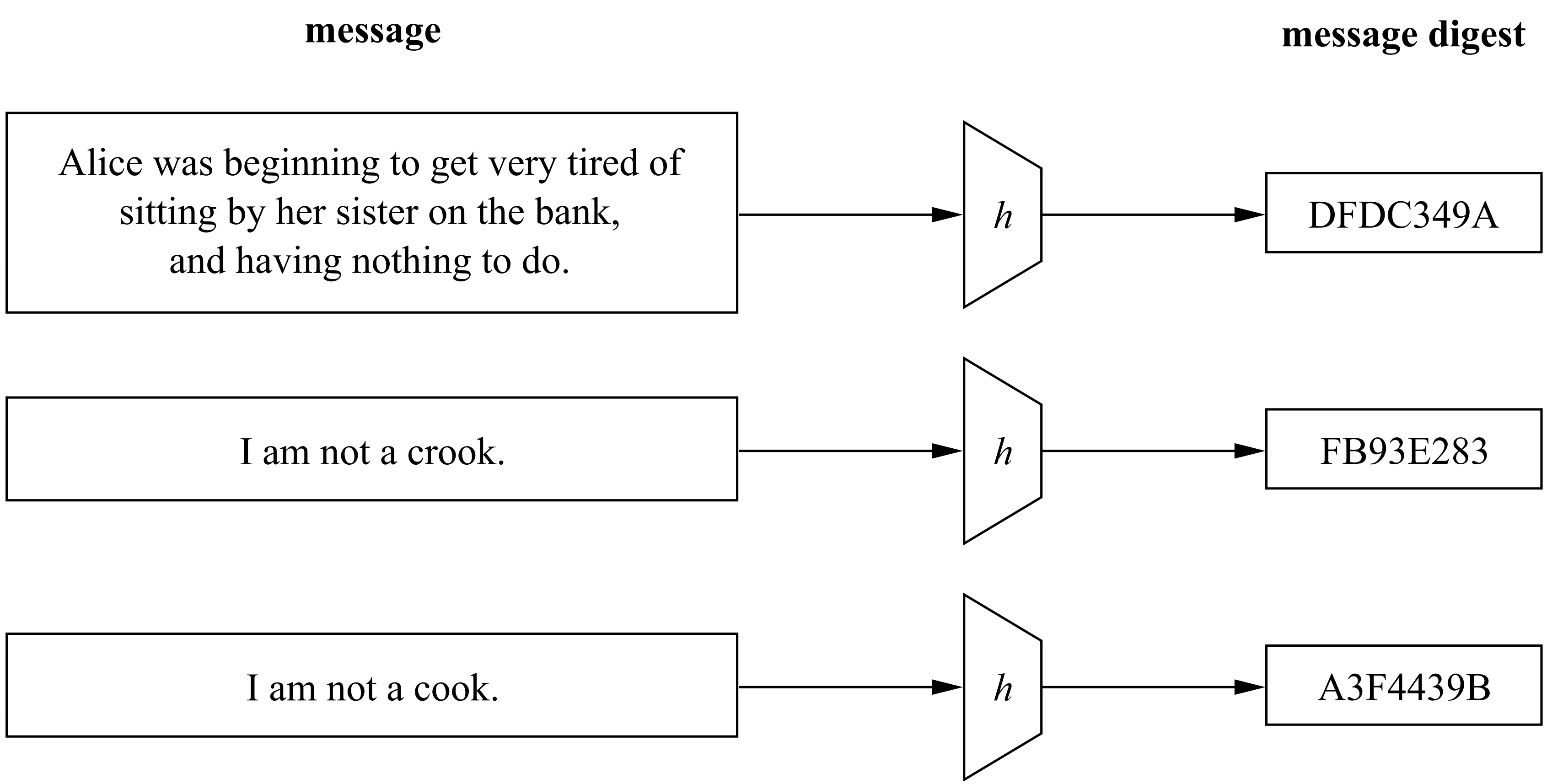
READINGS

- Section 11.1: Motivation: Signing Long Messages, Paar & Pelzl
- Section 11.2: Security Requirements of Hash Functions, Paar & Pelzl
- Section 11.3: Overview of Hash Algorithms, Paar & Pelzl

INTRODUCTION

- *Hash functions* take an input message of any size and output a short, fixed-length output called a *message digest*
  - Can think of it as a kind of compression function
  - e.g., for a 256-bit hash function, any input message regardless of its input length would be mapped to a 256-bit output
- *Cryptographic hash functions* have special properties that non-cryptographic hash functions do not have
  - These are essential for many security applications e.g., digital signatures, message authentication codes, key derivation, password storage

HASH FUNCTION BEHAVIOUR



MOTIVATION: SIGNING LONG MESSAGES

- We learned that digital signature algorithms have limitations on the message length, e.g., in RSA the message cannot be larger than the modulus
- An intuitive solution would be to design a mechanism to split up the message into chunks and sign one chunk at a time (analogous to block cipher modes of operation)
  - But this would be very slow, and the digital signature would be very large
- By signing the hash of a message instead of the original message itself, we can quickly generate a signature for an input message of any length

DIGITAL SIGNATURES WITH A HASH FUNCTION

- Bob computes the hash of the message $h(x) = z$ and signs it
- Bob then sends the message x and the signature to Alice
- Alice computes the hash $h(x) = z$ and validates the signature

Alice **Bob**

$$\xleftarrow{\quad k_{pub,B} \quad}$$

$$z = h(x)$$
$$s = \mathrm{sig}_{k_{pr,B}}(z)$$

$$\xleftarrow{\quad (x,s) \quad}$$

$$z' = h(x)$$
$$\mathrm{ver}_{k_{pub,B}}(s, z') = \text{true/false}$$

DESIRABLE HASH FUNCTION BEHAVIOUR

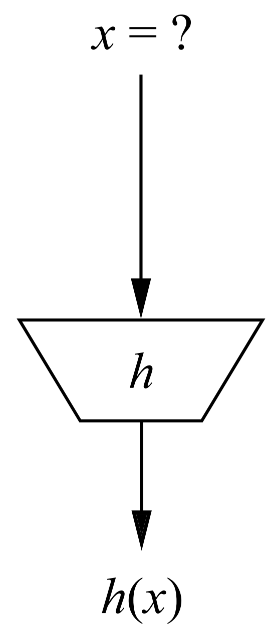
- The hash function should accept input of any size
- Computing the hash of a message should be fast, even for large messages
- The output of a hash function should be fixed
- The computed hash should be highly sensitive to all input bits, i.e., making a minor modification to the input message should result in a very diferent hash

SECURITY PROPERTIES OF HASH FUNCTIONS

- The pigeonhole principle tells us there will be infinitely many messages that share the same hash value
  - The question that matters from a security perspective is: How difficult is it to find such messages?
- This necessitates some special properties to be fulfilled for a hash function to be suitable for security applications

PREIMAGE RESISTANCE

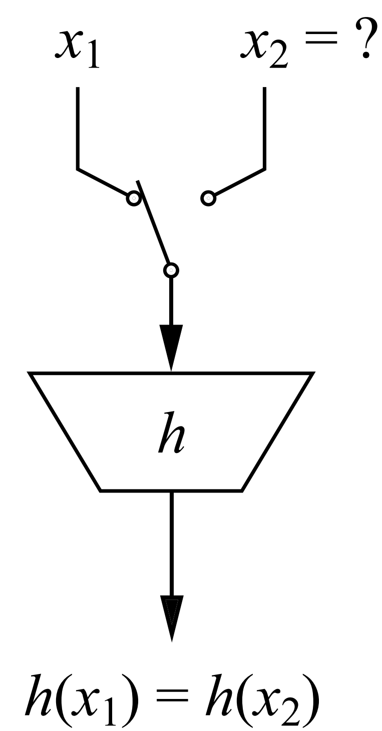
- Also called the *one-way* property
- Given a hash output $z = h(x)$, it must be computationally infeasible to recover the original input message x



preimage resistance

SECOND PREIMAGE RESISTANCE

- Also called *weak collision resistance*
- Given a message x_1 and its hash $h(x_1)$, it should be computationally infeasible to consruct another message $x_2 \neq x_1$ such that $h(x_1) = h(x_2)$



second preimage resistance

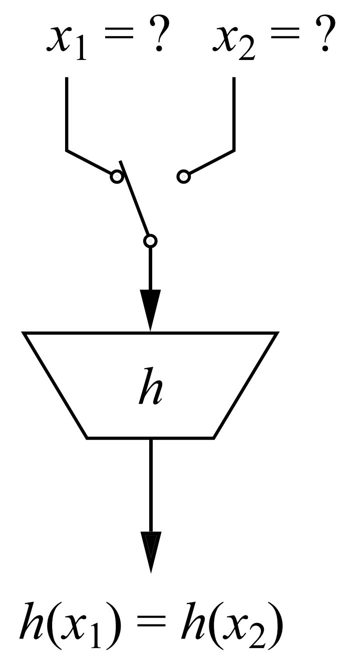
PREIMAGE ATTACK

- Suppose Bob hashes and signs a message x_1
- If Oscar can find another message x_2 such that $h(x_1) = h(x_2)$, he can perform this substitution attack:

Alice		**Oscar**		**Bob**
			$\xleftarrow{k_{pub,B}}$	
				$z = h(x_1)$ $s = \mathrm{sig}_{k_{pr,B}}(z)$
	$\xleftarrow{(x_2, s)}$	$\lightning$ substitute	$\xleftarrow{(x_1, s)}$	
$z = h(x_2)$ $\mathrm{ver}_{k_{pub,B}}(s, z) = \mathrm{true}$				

COLLISION RESISTANCE

- Also called *strong collision resistance*
- It should be computationally infeasible to consruct two different messages $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$



collision resistance

COLLISION ATTACKS

- Oscar, constructs two sales contracts x_1 and x_2, where x_1 charges \$10 and x_2 charges \$10,000 to the customer
- Oscar sends x_1 to the customer, who signs it
- Oscar then sends x_2 along with the customer's signature to the bank, which executes the money transfer

Bank		**Oscar**		**Customer**
			$\xleftarrow{k_{pub,B}}$	
			$\xrightarrow{x_1}$	
				$z = h(x_1)$ $s = \mathrm{sig}_{k_{pr,B}}(z)$
	$\xleftarrow{(x_2,s)}$	$\lightning$ substitute	$\xleftarrow{(x_1,s)}$	
$z = h(x_2)$ $\mathrm{ver}_{k_{pub,B}}(s,z) = \mathrm{true}$				

SECOND PREIMAGE VS. COLLISION RESISTANCE

- In a *preimage attack*, the attacker has one degree of freedom
  - Needs to construct **one** message with a **specific** hash value
- In a *collision attack*, the attacker has two degrees of freedom
  - Needs to construct **two** messages with the **same** hash value
- A collision attack is thus easier to carry out, meaning that it requires stronger protection to defend against
- Collision resistance implies second preimage resistance

PREIMAGE ATTACKS

To protect against the following attack, we require a hash function that is second preimage resistant, but it does not necessarily need to be collision resistant.

Alice		**Oscar**		**Bob**
			$\xleftarrow{\quad k_{pub,B} \quad}$	
				$z = h(x_1)$ $s = \mathrm{sig}_{k_{pr,B}}(z)$
	$\xleftarrow{\quad (x_2,s) \quad}$	$\lightning$ substitute	$\xleftarrow{\quad (x_1,s) \quad}$	
$z = h(x_2)$ $\mathrm{ver}_{k_{pub,B}}(s,z) = \mathrm{true}$				

COLLISION ATTACKS

To protect against the following attack, we require a hash function that is collision resistant.

Bank		Oscar		Customer
			$\xleftarrow{k_{pub,B}}$	
			$\xrightarrow{x_1}$	
				$z = h(x_1)$ $s = \mathrm{sig}_{k_{pr,B}}(z)$
	$\xleftarrow{(x_2,s)}$	$\lightning$ substitute	$\xleftarrow{(x_1,s)}$	
$z = h(x_2)$ $\mathrm{ver}_{k_{pub,B}}(s,z) = \mathrm{true}$				

MEASURING SECOND PREIMAGE RESISTANCE

- Given an input x_1, if the amount of work required is 2^N to find an x_2 such that $h(x_2) = h(x_1)$, then the *second preimage resistance* is N bits
- In the absence of analytical attacks, the expected second preimage resistance is equivalent to the output length of the hash function
  - This means that the best an attacker can do, given x_1, is to repeatedly select random inputs for x_2 and computes the hash until it finds that $h(x_1) = h(x_2)$
  - e.g., SHA-256 outputs a 256-bit value, and its expected second preimage resistance is 256 bits

MEASURING COLLISION RESISTANCE

- If the amount of work required is 2^N to find an x_1 and x_2 such that $h(x_2) = h(x_1)$, then the *collision resistance* is N bits
- In the absence of analytical attacks, the expected collision resistance is equivalent to half the output length of the hash function
  - e.g., SHA-256 outputs a 256-bit value, and its expected collision resistance is 128 bits

THE BIRTHDAY PARADOX

- The *birthday paradox* is named after the observation that in a group of 23 people, there is a 50% probability that two people share the same birthday
  - There is roughly a square-root relation between the number of days n and the number of people in the group m
  - See Wikipedia article for more details

COLLISION RESISTANCE AND THE BIRTHDAY PARADOX

- Similarly, in a *birthday attack* against a hash function with 2^n possible output values, the number of input messages that need to be hashed to find a collision is roughly

$$2^{(n+1)/2}\sqrt{\ln \frac{1}{1-\lambda}},$$

where n the hash output length and λ is the desired probability of success

COLLISION RESISTANCE AND THE BIRTHDAY PARADOX (2)

Using the approximate formula, for a 256-bit hash function (e.g., SHA-256), the amount of work required is:

Roughly $2^{256/2}\sqrt{\ln \frac{1}{1-0.5}} \approx 2^{129}$, for a 50% chance of success

Roughly $2^{256/2}\sqrt{\ln \frac{1}{1-0.9}} \approx 2^{130}$, for a 90% chance of success

This illustrates why the expected collision resistance of a 256-bit hash function is measured at 128 bits.

COMING UP NEXT...

- We have studied the required properties of hash functions
- Next, we will study the construction of hash functions