CS 45, Lecture 14
Cryptography

Spring 2023

Akshay Srivatsan, Ayelet Drazen, Jonathan Kula

Administrativa

A few administrative announcements:

- Assignment 5 due tonight!
- Final project released soon! Due Thursday, June 8th. Optional student presentations during last week of class
- Assignment 7 released tonight/tomorrow

Lecture Overview

In today's lecture, we will explore how cryptography motivates solutions to the security concerns we discussed in Monday's lecture. To do so, we will discuss:

- Entropy
- Hash functions
- Key derivation functions
- Encryption/Decryption

Lecture Recap

We can divide computer security into different goals:

1. Availability
2. Authentication
3. Confidentiality
4. Integrity

What is Cryptography?

Cryptography is the study of secure communication techniques in the presence of an adversary.

Cryptography requires some way for the intended recipient to be able to understand the encrypted (a.k.a secret) message while preventing others from understanding that same message.

Authentication

Authentication is used to verify that a user is who they say they are.

> **Problem:** we want to prevent unauthorized users from gaining access to our systems

Authentication

Longer passwords increase security.

A password like `horsebatterystaple` is more secure than a password like `$ecretW0rd!`

But why?

Entropy

Entropy is a measure of randomness, measured in bits of random.

Used when considering choosing from a set of possible outcomes uniformly at random.

Entropy

Entropy is a measure of randomness, measured in bits of random.

Used when considering choosing from a set of possible outcomes uniformly at random.

Calculated as:

$$\log_2(\# \text{ of possibilities})$$

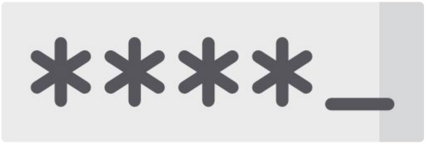
Entropy

Entropy describes bits of randomness.

The more bits of randomness we have, the better off we are from a security standpoint.

A coin has 1 bit of entropy

A die has 2.58 bits of entropy



A 5 digit password has 16.61 bits of entropy

Authentication

In addition to high entropy passwords, secure authentication requires securely storing user credentials.

What's the best way to do that?

Hash Functions

Cryptographic hash functions are used to convert some variable length input (e.g. `HelloWorldILoveYou`) into a fixed length output.

The values returned by hash functions are known as *hashes*.

There are many different hash functions out there:
`MD5`, SHA-1, SHA-2, NTLM, LANMAN

Hash Functions

Hash functions have two important properties:

They are non-invertible.

Hash Functions

Hash functions have two important properties:

They are non-invertible.

Given the output of a hash function (e.g. `68e109f0f40ca72a15e05cc2`), no one should be able to find the input (e.g. `HelloWorld`). The function cannot be "inverted".

Hash Functions

Hash functions have two important properties:

They are non-invertible.

`HelloWorld` → `68e109f0f40ca72a15e05cc2`

`68e109f0f40ca72a15e05cc2` → ❓

Hash Functions

Hash functions have two important properties:

They are non-invertible.

Hash Functions

Hash functions have two important properties:

They are non-invertible.

They are collision resistant.

Hash Functions

Hash functions have two important properties:

They are non-invertible.

They are collision resistant.

If there is some input (e.g. `HelloWorld`) which hashes to some output (e.g. `68e109f0f40ca72a15e05cc2`), it should be impossible to find another input that hashes to the same output (e.g. `68e109f0f40ca72a15e05cc2`).

Hash Functions

Hash functions have two important properties:

They are non-invertible.

They are collision resistant.

`HelloWorld` → `68e109f0f40ca72a15e05cc2`

`GoodbyeWorld` → `68e109f0f40ca72a15e05cc2`

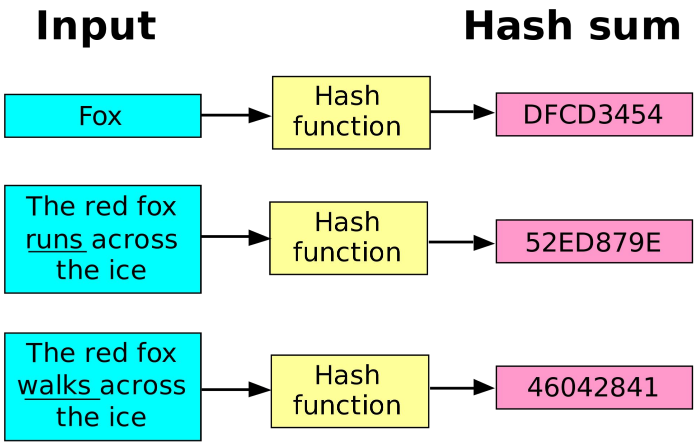
Hash Functions

Hash functions have two important properties:

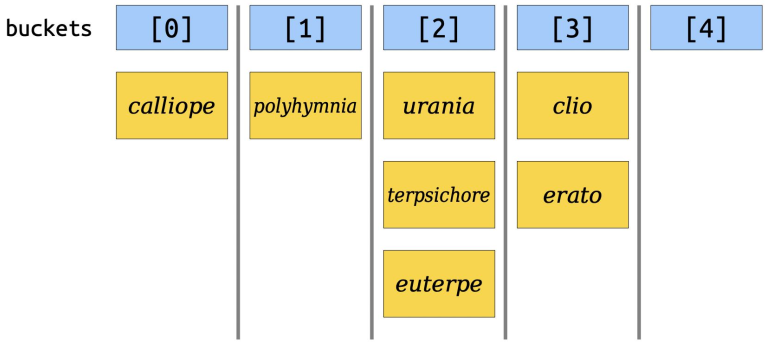
They are non-invertible.

They are collision resistant.

Hash Functions



VS



Hash Functions

`openssl` is a cryptography toolkit that allows you to:

- Create private and public keys
- Handle of S/MIME signed or encrypted email
- Encrypt and decrypt with ciphers

Hash Functions

`openssl` is a cryptography toolkit that allows you to:

- Create private and public keys
- Handle of S/MIME signed or encrypted email
- Encrypt and decrypt with ciphers

To use the `sha256` hash function, run the following:

```
openssl  sha256 test.txt
```

- or -

```
printf 'Hello' | openssl sha256
```

Hash Functions

There are many possible use cases for hash functions!

Hash Functions

There are many possible use cases for hash functions:

- **Commitment schemes:** commit to a value without revealing the value.

Hash Functions

There are many possible use cases for hash functions:

- **Commitment schemes:** commit to a value without revealing the value.
- **Git file hashes:** store file hashes so that you can verify that you are using the correct file.

Hash Functions

```
commit 58fa56e4124b2eb362e61fe0268f660f2c01b8bc
Author: adrazen <adrazen@stanford.edu>
Date:   Fri Feb 17 10:14:40 2023 -0800

    Add assign5 to course website.
```

The commit hash is the SHA-1 hash of the state of the Git repository at the time of the commit.

Why do we need a cryptographic hash function for this?

Hash Functions

There are many possible use cases for hash functions:

- **Commitment schemes:** commit to a value without revealing the value.
- **Git file hashes:** store file hashes so that you can verify that you are using the correct file.
- **Storing passwords:** never store passwords in plaintext and instead store hashes of passwords.

Hash Functions

Before hashing:

AD123AD5678	Paris2024	Paris2024	AD123AD5678
LittleLemon123	AD123AD5678	AD123AD5678	WFH14home
EatVeggies!	LittleLemon123	Int45678	Love2Code
Love2Code	NewAccountPassword	sunshine	LittleLemon123
HorseBatteryStaple	SecurityPwd	rainbows	HorseBatteryStaple
Int45678	HorseBatteryStaple	Love2Code	AppleWater

Hash Functions

After hashing:

5f4dcc3b5aa765d61d83	5ebe2294ecd0e0f08eab	5ebe2294ecd0e0f08eab	5f4dcc3b5aa765d61d83
25d55ad283aa400af464	5f4dcc3b5aa765d61d83	5f4dcc3b5aa765d61d83	f25a2fc72690b780b2a1
f25a2fc72690b780b2a1	25d55ad283aa400af464	9111629eb0309ee7c581	aat345sgdk7571123ght
d87da732b86c41782756	87bfh294ecd0e0f08eab	25d55ad283aa400af464	25d55ad283aa400af464
88099a55dd26bd54af1c	f25a2fc72690b780b2a1	f25a2fc72690b780b2a1	88099a55dd26bd54af1c
9111629eb0309ee7c581	88099a55dd26bd54af1c	d87da732b86c41782756	rhl345sgdk7571123ght

Hash Functions

After hashing:

5f4dcc3b5aa765d61d83	5ebe2294ecd0e0f08eab	5ebe2294ecd0e0f08eab	5f4dcc3b5aa765d61d83
25d55ad283aa400af464	5f4dcc3b5aa765d61d83	5f4dcc3b5aa765d61d83	f25a2fc72690b780b2a1
f25a2fc72690b780b2a1	25d55ad283aa400af464	9111629eb0309ee7c581	aat345sgdk7571123ght
d87da732b86c41782756	87bfh294ecd0e0f08eab	25d55ad283aa400af464	25d55ad283aa400af464
88099a55dd26bd54af1c	f25a2fc72690b780b2a1	f25a2fc72690b780b2a1	88099a55dd26bd54af1c
9111629eb0309ee7c581	88099a55dd26bd54af1c	d87da732b86c41782756	rhl345sgdk7571123ght

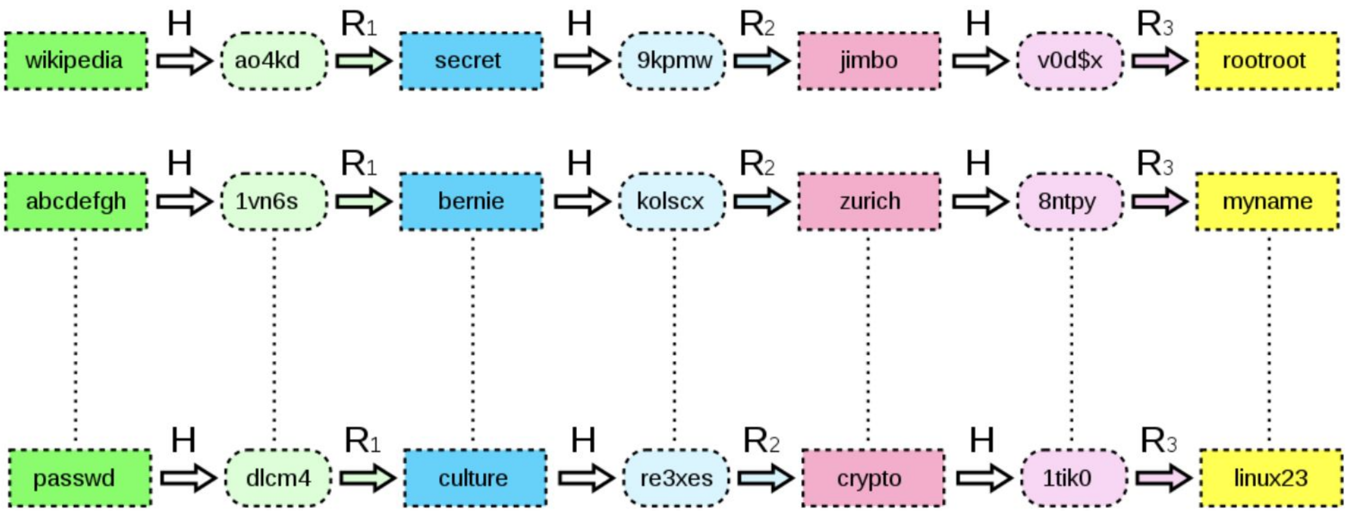
Cryptographic Salts

A **cryptographic salt** is random data that is used as additional input to a hashing function.

Let's return to an earlier fact: we never want to store passwords in plaintext. Hashed passwords are better, but still pose some problems.

Cryptographic Salts

Many users reuse the same passwords on multiple sites, and many sites use the same hashing algorithms. Attackers create **rainbow tables** of common passwords and their hashed equivalent.



Cryptographic Salts

Before a salt:

Google

BANK OF AMERICA

5f4dcc3b5aa765d61d83	5ebe2294ecd0e0f08eab	5ebe2294ecd0e0f08eab	5f4dcc3b5aa765d61d83
25d55ad283aa400af464	5f4dcc3b5aa765d61d83	5f4dcc3b5aa765d61d83	f25a2fc72690b780b2a1
f25a2fc72690b780b2a1	25d55ad283aa400af464	9111629eb0309ee7c581	aat345sgdk7571123ght
d87da732b86c41782756	87bfh294ecd0e0f08eab	25d55ad283aa400af464	25d55ad283aa400af464
88099a55dd26bd54af1c	f25a2fc72690b780b2a1	f25a2fc72690b780b2a1	88099a55dd26bd54af1c
9111629eb0309ee7c581	88099a55dd26bd54af1c	d87da732b86c41782756	rhl345sgdk7571123ght

Cryptographic Salts

Before a salt:

5f4dcc3b5aa765d61d83	5ebe2294ecd0e0f08eab	5ebe2294ecd0e0f08eab	5f4dcc3b5aa765d61d83
25d55ad283aa400af464	5f4dcc3b5aa765d61d83	5f4dcc3b5aa765d61d83	f25a2fc72690b780b2a1
f25a2fc72690b780b2a1	25d55ad283aa400af464	9111629eb0309ee7c581	aat345sgdk7571123ght
d87da732b86c41782756	87bfh294ecd0e0f08eab	25d55ad283aa400af464	25d55ad283aa400af464
88099a55dd26bd54af1c	f25a2fc72690b780b2a1	f25a2fc72690b780b2a1	88099a55dd26bd54af1c
9111629eb0309ee7c581	88099a55dd26bd54af1c	d87da732b86c41782756	rhl345sgdk7571123ght

Cryptographic Salts

A cryptographic salt helps prevent a rainbow table attack. Each database uses a different salt to ensure the hashes are different.

Attackers can no longer precompute hashes to common passwords.

Cryptographic Salts

With a salt:

Google aa56g47th34f6	b1g5ab18h22a8	23f88qw12a8p0	BANK OF AMERICA 56hq19trw89wq
1e70537a50e140b910db	6h8kl4fb18c7lb7e9h11	5ebe2294ecd0e0f08eab	d837hw56lp01n7hhq8zx
25d55ad283aa400af464	5f4dcc3b5aa765d61d83	674dlc3b5ar765d61d83	f25a2fc72690b780b2a1
f25a2fc72690b780b2a1	25d55ad283aa400af464	9111629eb0309ee7c581	aat345sgdk7571123ght
d87da732b86c41782756	87bfh294ecd0e0f08eab	25d55ad283aa400af464	25d55ad283aa400af464
88099a55dd26bd54af1c	f25a2fc72690b780b2a1	f25a2fc72690b780b2a1	88099a55dd26bd54af1c
9111629eb0309ee7c581	88099a55dd26bd54af1c	d87da732b86c41782756	rhl345sgdk7571123ght

Cryptographic Salts

With a salt:

Google	Facebook	Stanford	Bank of America
aa56g47th34f6	b1g5ab18h22a8	23f88qw12a8p0	56hq19trw89wq
1e70537a50e140b910db	6h8kl4fb18c7lb7e9h11	5ebe2294ecd0e0f08eab	d837hw56lp01n7hhq8zx
25d55ad283aa400af464	5f4dcc3b5aa765d61d83	674dlc3b5ar765d61d83	f25a2fc72690b780b2a1
f25a2fc72690b780b2a1	25d55ad283aa400af464	9111629eb0309ee7c581	aat345sgdk7571123ght
d87da732b86c41782756	87bfh294ecd0e0f08eab	25d55ad283aa400af464	25d55ad283aa400af464
88099a55dd26bd54af1c	f25a2fc72690b780b2a1	f25a2fc72690b780b2a1	88099a55dd26bd54af1c
9111629eb0309ee7c581	88099a55dd26bd54af1c	d87da732b86c41782756	rhl345sgdk7571123ght

Authentication

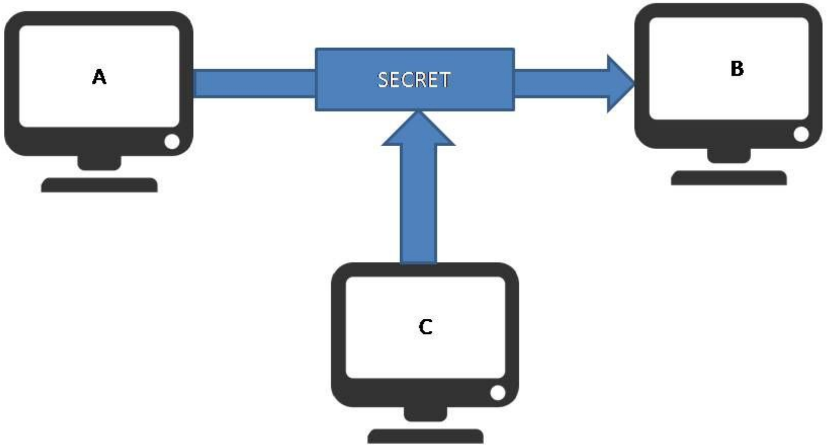
Secure authentication requires:

1. High entropy passwords
2. Secure storage using cryptographic hashes and salts

Confidentiality

Confidentiality: only intended users should be able to read our data or information.

Problem: we want to prevent unintended users from reading information we send or that is stored on our systems



Confidentiality

HTTPS is a secure (encrypted) networking protocol.

HTTPS draws on the idea of encryption.

Key Derivation Functions

Key Derivation Functions (KDFs) have the same properties as hash functions except they are *slow* to compute.

Key Derivation Functions

Key Derivation Functions (KDFs) have the same properties as hash functions except they are *slow* to compute.

Why is slow good?

Key Derivation Functions

Key Derivation Functions (KDFs) have the same properties as hash functions except they are *slow* to compute.

> The property of being slow is actually useful in certain cases as it prevents and attacker from being able to brute force and guess the key.

Examples: PBKDF2, Argon2, Scrypt

Encryption and Decryption

An encryption system includes three different functions:

Key generation: generates a key in order to prevent having to memorize password

Encrypt: takes in plaintext and the key, returns ciphertext

Decrypt: takes in ciphertext and the key, returns plaintext

Encryption and Decryption

Encryption using KDF:

```
openssl  aes-256-cbc -in file.txt -out file.txt.enc
```

The (symmetric)
encryption algorithm
which uses the KDF

This will prompt for a password that gets run through a KDF, which then generates a key for encryption.

Encryption and Decryption

Decryption using KDF:

```
openssl  aes-256-cbc -d -in file.txt.enc -out file.txt.dec
```

The (symmetric) encryption algorithm which uses the KDF

This will prompt for a password that gets run through a KDF, which then generates a key for encryption.

Encryption and Decryption

Properties of Encryption/Decryption:

- Ciphertext does not reveal anything about the plaintext
- If you take a message `m` and encrypt with a key `k` and produce a ciphertext, you should be able to decrypt that ciphertext with `k` and get the original message `m`

Encryption and Decryption

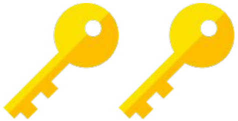
Use Cases of Symmetric Encryption:

- Store a file on a cloud server. You may want to encrypt the file on the server because you can't trust the cloud provider.

Symmetric vs Asymmetric

The key difference between **symmetric** and **asymmetric** cryptography is the number of keys.

Symmetric cryptography uses 1 key 

Asymmetric cryptography uses 2 keys 

Asymmetric Cryptography

In asymmetric cryptography, we'll use 2 keys. 🔑🔑

Asymmetric Cryptography

In asymmetric cryptography, we'll use 2 keys. 🔑🔑

Our first key is a **private key** which is never revealed to any other person, party, or system.

Asymmetric Cryptography

In asymmetric cryptography, we'll use 2 keys. 🔑🔑

Our first key is a **private key** which is never revealed to any other person, party, or system.

Our second key is a **public key** which can be shared freely with anyone. (A secure asymmetric cryptographic system makes it so that sharing the public key can't compromise the integrity of the system).

Asymmetric Cryptography

An asymmetric encryption system includes three different functions:

Key generation: generates a pair of keys `<private_key, public_key>`

Encrypt: takes in plaintext and the **public** key, returns ciphertext

Decrypt: takes in ciphertext and the **private** key, returns plaintext

Asymmetric Cryptography

Asymmetric Cryptography

An asymmetric encryption system also allows for signing and verifying:

Sign: takes in message and the **private** key, returns signature

Verify: takes in message and the **public** key, returns Accept/Reject

Encryption and Decryption

Use Cases of Symmetric Encryption:

- Email encryption
- Secure online communication using HTTPS
- Private messaging
- Sign software releases
- Signing Git commits

Confidentiality

TLS (which underlies HTTPS) uses public key cryptography to ensure that we always maintain a secure channel of communication.

1. An initial "handshake" that verifies the server's identity (e.g. that Facebook is Facebook)
2. Generate session keys to use symmetric cryptography after the initial handshake

End-to-End Encryption

End-to-End Encryption (E2EE) is used for secure communication when data is transferred from one device to another.

- In E2EE, the data is encrypted on the sender's system or device and only the intended recipient can decrypt it.
- E2EE uses public key encryption.
- Data is protected from all possible eavesdroppers

Signal

Encryption At Rest vs Encryption in Transit

Encryption at Rest refers to data that is stored on hard drives, laptops, flash drives, or cloud storage.

Encryption in Transit refers to data that is moving between devices and networks.

Integrity

Integrity: only authorized users should be able to modify data or information.

Problem: we want to prevent unauthorized users from modifying information that we send or that is stored on our systems

Integrity

Integrity requires verifying that the contents of a given file or system haven't changed.

How can we use cryptographic primitives to achieve this?

Integrity

- Cryptographic hash functions allow us to verify the contents of files

```
openssl sha256 file.txt
```

Integrity

- Cryptographic hash functions allow us to verify the contents of files
- Encryption allows us to upload data to untrusted providers while maintaining integrity

Assignment 7

Exciting assignment but use techniques responsibly!