CS 45, Lecture 14
Cryptography

Spring 2023
Akshay Srivatsan, Ayelet Drazen, Jonathan Kula
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
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
We can divide computer security into different goals:

1. Availability
2. Authentication
3. Confidentiality
4. Integrity
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 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 is a measure of randomness, measured in bits of random.

Used when considering choosing from a set of possible outcomes uniformly at random.
**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 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
In addition to high entropy passwords, secure authentication requires securely storing user credentials.

What's the best way to do that?
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 have two important properties:

They are non-invertible.
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 $\rightarrow$ 68e109f0f40ca72a15e05cc2

68e109f0f40ca72a15e05cc2 $\rightarrow$ ?
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 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 $\rightarrow$ 68e109f0f40ca72a15e05cc2

GoodbyeWorld $\rightarrow$ 68e109f0f40ca72a15e05cc2
Hash functions have two important properties:

- They are non-invertible.
- They are collision resistant.
Hash Functions

Input

- Fox
- The red fox runs across the ice
- The red fox walks across the ice

Hash sum

- DFCD3454
- 52ED879E
- 46042841

VS

<table>
<thead>
<tr>
<th>buckets</th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>calliope</td>
<td>polyhymnia</td>
<td>urania</td>
<td>clio</td>
<td>erato</td>
</tr>
<tr>
<td></td>
<td>euterpe</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

21
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
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 \texttt{sha256} hash function, run the following:

\begin{verbatim}
openssl sha256 test.txt
\end{verbatim}

\texttt{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.
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.
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?**
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:

<table>
<thead>
<tr>
<th>Before Hashing</th>
<th>After Hashing 1</th>
<th>After Hashing 2</th>
<th>After Hashing 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD123AD5678</td>
<td>Paris2024</td>
<td>Paris2024</td>
<td>AD123AD5678</td>
</tr>
<tr>
<td>LittleLemon123</td>
<td>AD123AD5678</td>
<td>AD123AD5678</td>
<td>WFH14home</td>
</tr>
<tr>
<td>EatVeggies!</td>
<td>LittleLemon123</td>
<td>Int45678</td>
<td>Love2Code</td>
</tr>
<tr>
<td>Love2Code</td>
<td>NewAccountPassword</td>
<td>sunshine</td>
<td>LittleLemon123</td>
</tr>
<tr>
<td>HorseBatteryStaple</td>
<td>SecurityPwd</td>
<td>rainbows</td>
<td>HorseBatteryStaple</td>
</tr>
<tr>
<td>Int45678</td>
<td>HorseBatteryStaple</td>
<td>Love2Code</td>
<td>AppleWater</td>
</tr>
</tbody>
</table>
# Hash Functions

After hashing:

<table>
<thead>
<tr>
<th>Google</th>
<th>Facebook</th>
<th>Stanford</th>
<th>Bank of America</th>
</tr>
</thead>
<tbody>
<tr>
<td>5f4dcc3b5aa765d61d83</td>
<td>5ebe2294ecd0e0f08eab</td>
<td>5ebe2294ecd0e0f08eab</td>
<td>5f4dcc3b5aa765d61d83</td>
</tr>
<tr>
<td>25d55ad283aa400af464</td>
<td>5f4dcc3b5aa765d61d83</td>
<td>5f4dcc3b5aa765d61d83</td>
<td>f25a2fc72690b780b2a1</td>
</tr>
<tr>
<td>f25a2fc72690b780b2a1</td>
<td>25d55ad283aa400af464</td>
<td>9111629eb0309ee7c581</td>
<td>aat345sgdk7571123ght</td>
</tr>
<tr>
<td>d87da732b86c41782756</td>
<td>87bfh294ecd0e0f08eab</td>
<td>25d55ad283aa400af464</td>
<td>25d55ad283aa400af464</td>
</tr>
<tr>
<td>88099a55dd26bd54af1c</td>
<td>f25a2fc72690b780b2a1</td>
<td>f25a2fc72690b780b2a1</td>
<td>88099a55dd26bd54af1c</td>
</tr>
<tr>
<td>9111629eb0309ee7c581</td>
<td>88099a55dd26bd54af1c</td>
<td>d87da732b86c41782756</td>
<td>rhl345sgdk7571123ght</td>
</tr>
</tbody>
</table>
# Hash Functions

After hashing:

<table>
<thead>
<tr>
<th>Google</th>
<th>Facebook</th>
<th>Stanford</th>
<th>Bank of America</th>
</tr>
</thead>
<tbody>
<tr>
<td>5f4dccc3b5aa765d61d83</td>
<td>5ebe2294ecd0e0f08eab</td>
<td>5ebe2294ecd0e0f08eab</td>
<td>5f4dccc3b5aa765d61d83</td>
</tr>
<tr>
<td>25d55ad283aa400af464</td>
<td>5f4dccc3b5aa765d61d83</td>
<td>5f4dccc3b5aa765d61d83</td>
<td>f25a2fc72690b780b2a1</td>
</tr>
<tr>
<td>f25a2fc72690b780b2a1</td>
<td>25d55ad283aa400af464</td>
<td>9111629eb0309ee7c581</td>
<td>aat345sgdk7571123ght</td>
</tr>
<tr>
<td>d87da732b86c41782756</td>
<td>87bfh294ecd0e0f08eab</td>
<td>25d55ad283aa400af464</td>
<td>25d55ad283aa400af464</td>
</tr>
<tr>
<td>88099a55dd26bd54af1c</td>
<td>f25a2fc72690b780b2a1</td>
<td>f25a2fc72690b780b2a1</td>
<td>88099a55dd26bd54af1c</td>
</tr>
<tr>
<td>9111629eb0309ee7c581</td>
<td>88099a55dd26bd54af1c</td>
<td>d87da732b86c41782756</td>
<td>rhl345sgdk7571123ght</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Google</th>
<th>Facebook</th>
<th>Stanford</th>
<th>Bank of America</th>
</tr>
</thead>
<tbody>
<tr>
<td>5f4dcc3b5aa765d61d83</td>
<td>5ebe2294ecd0e0f08eab</td>
<td>5f4dcc3b5aa765d61d83</td>
<td>5f4dcc3b5aa765d61d83</td>
</tr>
<tr>
<td>25d55ad283aa400af464</td>
<td>5f4dcc3b5aa765d61d83</td>
<td>5f4dcc3b5aa765d61d83</td>
<td>f25a2fc72690b780b2a1</td>
</tr>
<tr>
<td>f25a2fc72690b780b2a1</td>
<td>25d55ad283aa400af464</td>
<td>9111629eb0309ee7c581</td>
<td>aat345sgdk7571123ght</td>
</tr>
<tr>
<td>d87da732b86c41782756</td>
<td>87bfh294ecd0e0f08eab</td>
<td>25d55ad283aa400af464</td>
<td>25d55ad283aa400af464</td>
</tr>
<tr>
<td>88099a55dd26bd54af1c</td>
<td>f25a2fc72690b780b2a1</td>
<td>f25a2fc72690b780b2a1</td>
<td>88099a55dd26bd54af1c</td>
</tr>
<tr>
<td>9111629eb0309ee7c581</td>
<td>88099a55dd26bd54af1c</td>
<td>d87da732b86c41782756</td>
<td>rhl345sgdk7571123ght</td>
</tr>
</tbody>
</table>
## Cryptographic Salts

### Before a salt:

<table>
<thead>
<tr>
<th>Company</th>
<th>Salt 1</th>
<th>Salt 2</th>
<th>Salt 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>5f4dcc3b5aa765d61d83</td>
<td>5ebe2294ecd0e0f08eab</td>
<td>5ebe2294ecd0e0f08eab</td>
</tr>
<tr>
<td>Facebook</td>
<td>25d55ad283aa400af464</td>
<td>5f4dcc3b5aa765d61d83</td>
<td>5ebe2294ecd0e0f08eab</td>
</tr>
<tr>
<td>Stanford</td>
<td>f25a2fc72690b780b2a1</td>
<td>25d55ad283aa400af464</td>
<td>25d55ad283aa400af464</td>
</tr>
<tr>
<td>Bank of America</td>
<td>d87da732b86c41782756</td>
<td>25d55ad283aa400af464</td>
<td>25d55ad283aa400af464</td>
</tr>
<tr>
<td></td>
<td>9111629eb0309ee7c581</td>
<td>25d55ad283aa400af464</td>
<td>88099a55dd26bd54af1c</td>
</tr>
<tr>
<td></td>
<td>88099a55dd26bd54af1c</td>
<td>9111629eb0309ee7c581</td>
<td>88099a55dd26bd54af1c</td>
</tr>
<tr>
<td></td>
<td>d87da732b86c41782756</td>
<td>88099a55dd26bd54af1c</td>
<td>rhl345sgdk7571123ght</td>
</tr>
</tbody>
</table>
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:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1e70537a50e140b910db</td>
<td>6h8kl4fb18c71b7e9h11</td>
<td>5ebe2294ecd0e0f08eab</td>
<td>d837hw561p01n7hhq8zx</td>
</tr>
<tr>
<td>25d55ad283aa400af464</td>
<td>5f4dcc3b5aa765d61d83</td>
<td>674d1c3b5ar765d61d83</td>
<td>f25a2fc72690b780b2a1</td>
</tr>
<tr>
<td>f25a2fc72690b780b2a1</td>
<td>25d55ad283aa400af464</td>
<td>9111629eb0309ee7c581</td>
<td>aat345sgdk7571123ght</td>
</tr>
<tr>
<td>d87da732b86c41782756</td>
<td>87bfh294ecd0e0f08eab</td>
<td>25d55ad283aa400af464</td>
<td>25d55ad283aa400af464</td>
</tr>
<tr>
<td>88099a55dd26bd54af1c</td>
<td>f25a2fc72690b780b2a1</td>
<td>f25a2fc72690b780b2a1</td>
<td>88099a55dd26bd54af1c</td>
</tr>
<tr>
<td>9111629eb0309ee7c581</td>
<td>88099a55dd26bd54af1c</td>
<td>d87da732b86c41782756</td>
<td>rhl345sgdk7571123ght</td>
</tr>
</tbody>
</table>
Cryptographic Salts

With a salt:

<table>
<thead>
<tr>
<th>Salt 1</th>
<th>Salt 2</th>
<th>Salt 3</th>
<th>Salt 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1e70537a50e140b910db</td>
<td>6h8k14fb18c71b7e9h11</td>
<td>5ebe2294ecd0e0f08eab</td>
<td>d837hw56lp01n7hhq8zx</td>
</tr>
<tr>
<td>25d55ad283aa400af464</td>
<td>5f4dcc3b5aa765d61d83</td>
<td>674d1c3b5ar765d61d83</td>
<td>f25a2fc72690b780b2a1</td>
</tr>
<tr>
<td>f25a2fc72690b780b2a1</td>
<td>25d55ad283aa400af464</td>
<td>9111629eb0309ee7c581</td>
<td>aat345sgdk7571123ght</td>
</tr>
<tr>
<td>d87da732b86c41782756</td>
<td>87bfh294ecd0e0f08eab</td>
<td>25d55ad283aa400af464</td>
<td>25d55ad283aa400af464</td>
</tr>
<tr>
<td>88099a55dd26bd54af1c</td>
<td>f25a2fc72690b780b2a1</td>
<td>f25a2fc72690b780b2a1</td>
<td>88099a55dd26bd54af1c</td>
</tr>
<tr>
<td>9111629eb0309ee7c581</td>
<td>88099a55dd26bd54af1c</td>
<td>d87da732b86c41782756</td>
<td>rhl345sgdk7571123ght</td>
</tr>
</tbody>
</table>
Secure authentication requires:

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

https://crypto.stanford.edu/~dabo/cs255/
Key Derivation Functions (KDFs) have the same properties as hash functions except they are slow to compute.
Key Derivation Functions (KDFs) have the same properties as hash functions except they are **slow** to compute.

**Why is slow good?**
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 an attacker from being able to brute force and guess the key.

Examples: PBKDF2, Argon2, Scrypt
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.
Decryption using KDF:

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

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 \( m \)
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 $\langle$private_key, public_key$\rangle$
- **Encrypt**: takes in plaintext and the **public** key, returns ciphertext
- **Decrypt**: takes in ciphertext and the **private** key, returns plaintext
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
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 (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.
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: 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 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

```bash
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!