Lecture 14
Hash Function and Digital Signature

ECE 422: Reliable and Secure Systems Design

UNIVERSITY OF ALBERTA

Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
- Digital Signature
  - Irreversibility of hash function
  - Why is hash collision a problem
- Hash function: SHA256
  - Hash collision
  - Applications in practice
- Next class: salting
- TODOs

Birthday problem

Let Q = (1 - P), the probability that two people have the same birthday, then:

- P_{10} = 88.31% Q = 1 - P = 11.69%
- P_{25} = 43.13% Q = 1 - P = 56.87%
- P_{50} = 2.96% Q = 1 - P = 97.04%
- P_{78} = 00.01% Q = 1 - P = 99.99%

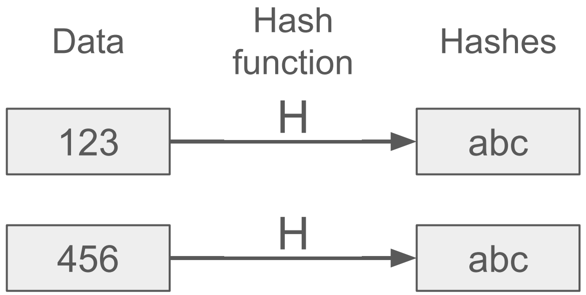
Hash collision

Hash function is unaware of its set of inputs.

- Performs arithmetic operations on the input passed to it
- Produces hashes of a fixed size

Problem? Hash collision is likely to happen.

- Hash collision happens when two pieces of data in a hash table share the same hash value
- Similar to the idea of birthday collision



Digital signature

Digital signatures verify the authenticity

- Detect the identity of the sender/signer

Digital signatures check the integrity

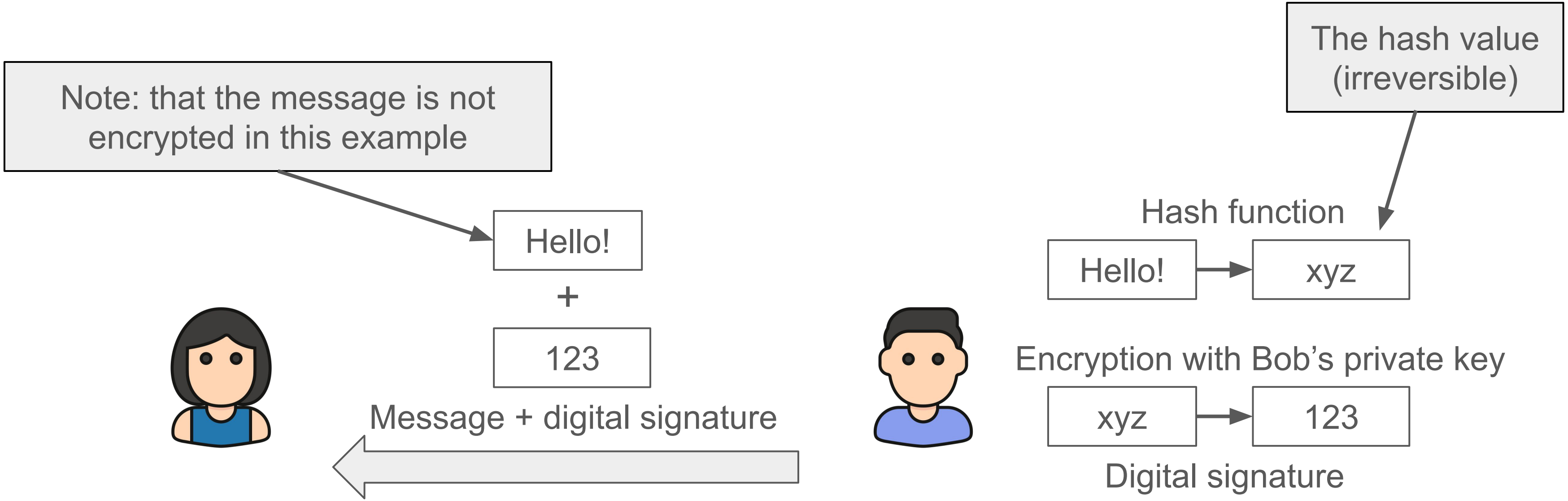
- Verify that the message was not changed

Digital signatures ensure non-repudiation

- Verify that the signature is not fake

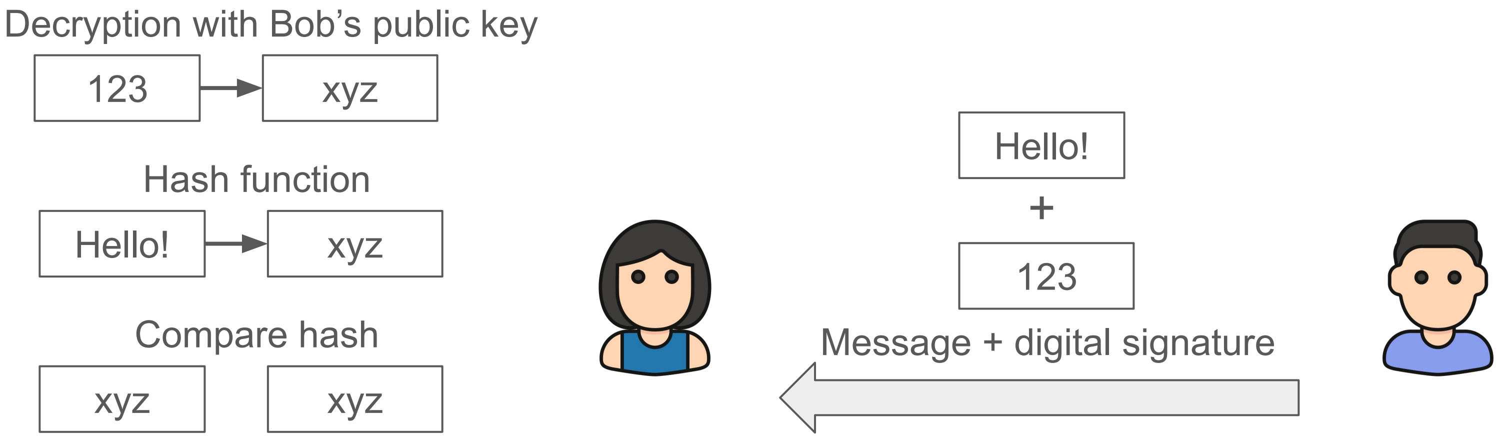
With digital signature

- Bob converts the message “Hello” into a hash “xyz”.
- Bob encrypts the hash with his private key, and sends it back together with the message “Hello” to Alice.



With digital signature

- Alice uses Bob’s public key to decrypt the message.
- Alice create a hash of the message by herself.
- Alice verifies whether the hash matches what Bob sends.



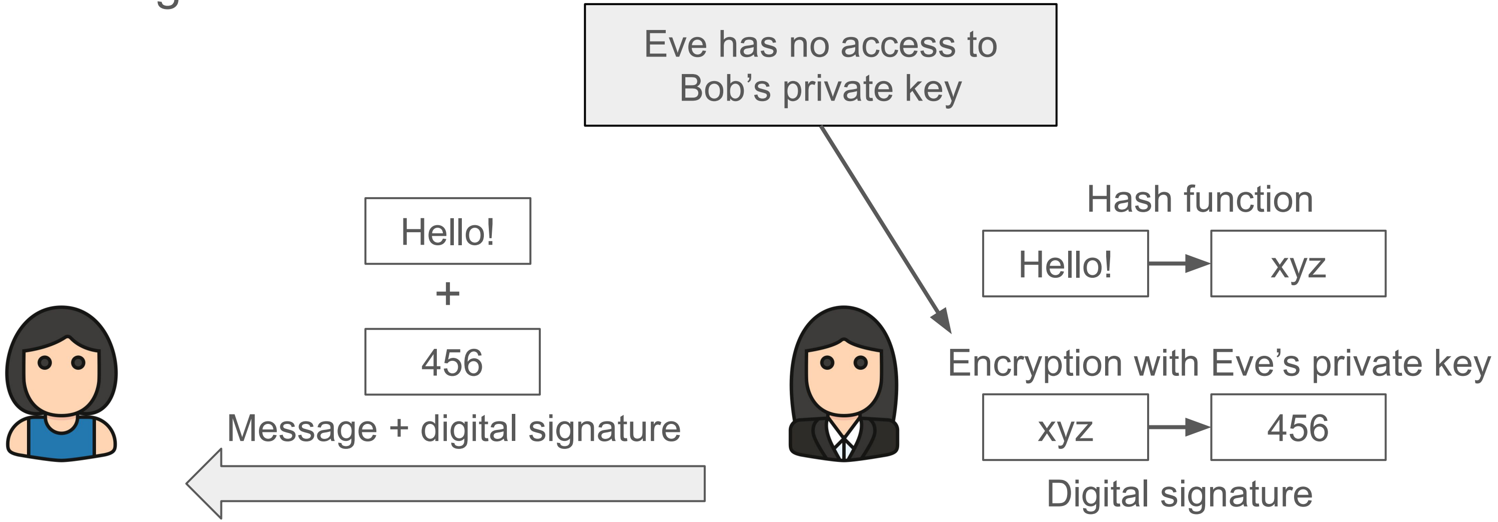
Digital signature

What happens if Eve pretends to be Bob?

Digital signature

What happens if Eve pretends to be Bob?

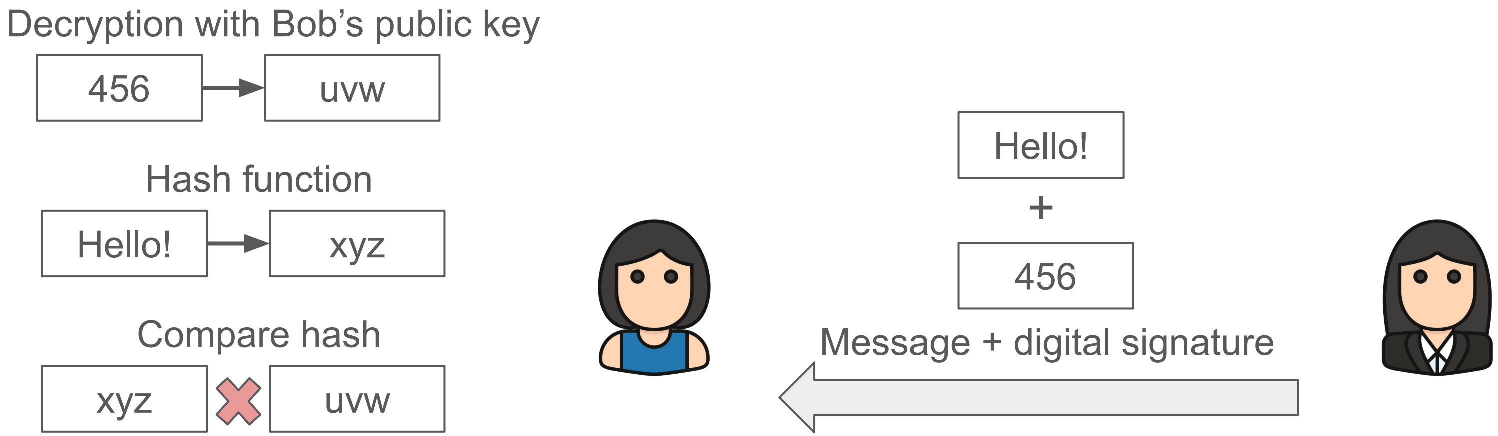
- Eve converts the message “Hello!” into the hash “xyz”.
- Eve encrypts the hash with her private key, and sends it back together with the message to Alice.



Digital signature

What happens if Eve pretends to be Bob?

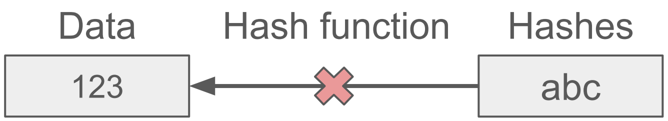
- Alice uses Bob’s public key to decrypt the message.
- At the same time, Alice also converts the message “Hello!” into its hash.
- Alice verifies whether the hash values match.



Irreversibility

Are hash functions still secure if they are publicly available?

- Publicly available as in:
  - Which hash function is used?
  - How did we implement it?
- Yes, because hash functions are irreversible.



The irreversibility is similar to the birthday problem:

- Easy to guess someone’s birthday
- Hard to tell who is having the birthday

Analogous to jigsaw puzzle

Analogy to jigsaw puzzles

- Data = a blank sheet of paper
- Hash function = cutting the paper into one million pieces of jigsaw puzzle and shuffling it
- Hash = pieces of jigsaw puzzle

We can tell how the hash function works, but it is impossible to transform the hash value back to the data.

Why is hash collision a problem?

Hash function is unaware of its set of inputs.

- Produces hashes of a fixed size
- Problem? Hash collision is likely to happen

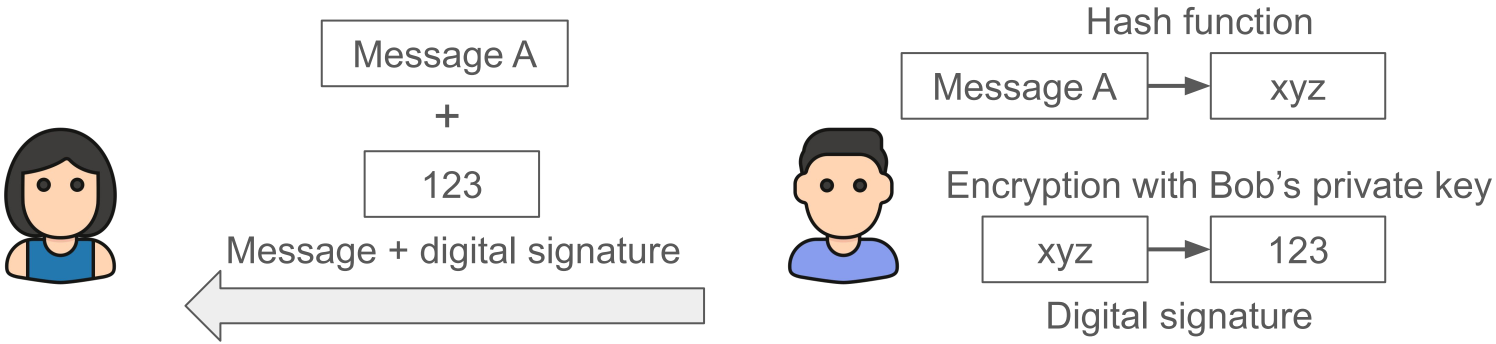
Hash collision makes systems vulnerable to collision attack.

- Collision attack tries to find two inputs that produce the same hash value
- Applications of collision attack
  - Digital signature: two messages with the same hash value
  - File integrity checks: two files with the same hash value

Example of collision attack: digital signature

Collision attack, scenario 1:

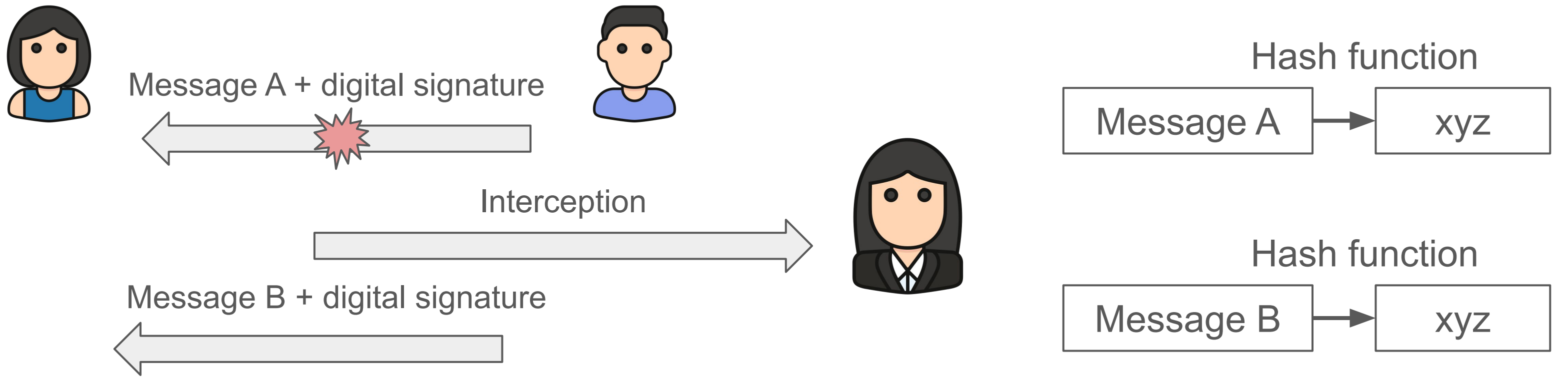
- Bob sends a message A to Alice with a digital signature.



Example of collision attack: digital signature

Collision attack, scenario 1:

- Eve intercepts the message.
- Eve changes message A into message B that produces the same hash value
- Eve sends the changed message together with Bob’s signature to Alice.

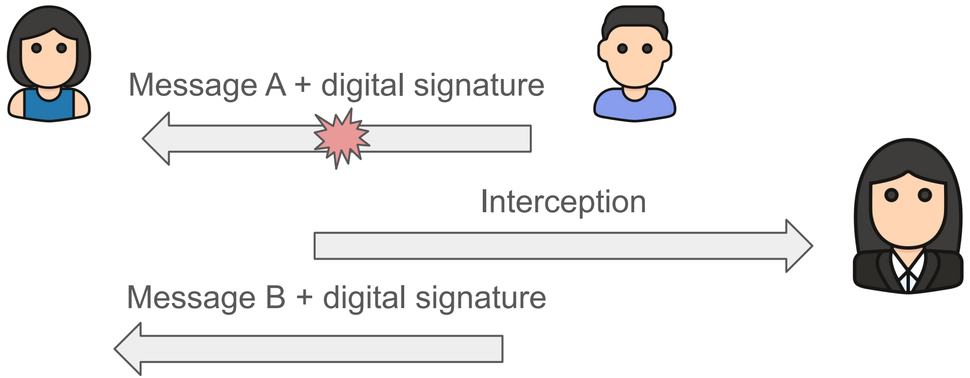


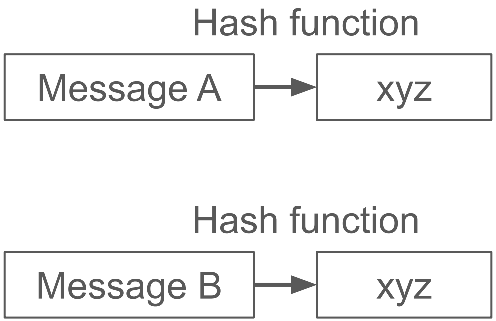
Example of collision attack: digital signature

Collision attack, scenario 1:

- Eve intercepts the message.
- Eve changes message A into message B that produces the same hash value
- Eve sends the changed message together with Bob's signature to Alice.

What problem can this cause?

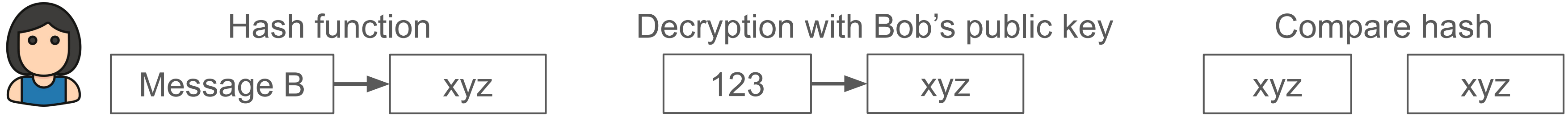




Example of collision attack: digital signature

Collision attack, scenario 1:

- Alice receives a message B with Bob’s digital signature.
- Alice: “The hashes match, it must be coming from Bob”.



Hash collision presents a threat to digital signature.

- Despite the message was changed, its hash value matches
- Integrity of the message is broken: changed message

Example of collision attack: integrity checks

Collision attack, scenario 2:

- Eve shares a file A online that has the same hash as another malicious program B.
- Before downloading the file, Bob asks for the hash to verify the file's integrity.
- Eve provides the malicious file B instead.
- Bob: "The hashes match, it must be the same file".

Hash collision presents a threat to file integrity checks.

- Despite the files are different, their hashes match
- Integrity of the file is broken: changed file

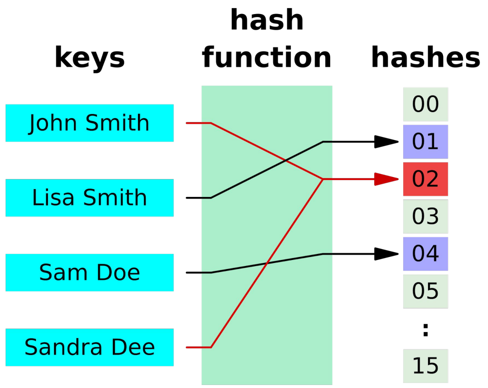
Hash collisions are unavoidable

To note that it is impossible to design a hash function that avoid collisions.

- Hash function: converts an input from a large domain to a smaller domain
- Hash collisions are unavoidable by nature
- The goal is to minimize collisions, not eliminate them

A good hash function must satisfies two properties:

- Fast hashing
- Minimal collisions



Schedule for today

- Key concepts from last class
- Digital Signature
  - Irreversibility of hash function
  - Why is hash collision a problem
- Hash function: SHA256
  - Hash collision
  - Applications in practice
- Next class: salting
- TODOs

Example of hash function: SHA256

SHA256 hash is hash function that can generates unique 256-bit hashes.

- Developed by US government's National Security Agency (NSA) in 2001
- Part of the SHA-2 (Secure Hash Algorithm 2)
- Produces hash of 64 hexadecimal characters / 256 bits
    - The output is in fact in binary, but hexadecimal is used to visually represent the hash value.

6 8 e 6 5 6 …

⇩ ⇩

110 1000 1110 0110 0101 0110 …

Decimal	HEX	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	a	1010
11	b	1011
12	c	1100
13	d	1101
14	e	1110
15	f	1111

Example of SHA256

Input 1: “Hi ECE 422”

SHA256 hash:

911fe59b33e0cf049ba953138c05178c9ffd4e57a0bd43be4c88cbf39dd7959a

Input 2: “Hi ECE422”

SHA256 hash:

f7046b0935cab27f82cd40e4c2ff6a422239d11e6f7c60da9f3d82f1b13099d0

…try it by yourself: [SHA256 online generation tool]

Hash collision in SHA256

Theory: Given a hash function that produces *n* hashes, it requires $1.2\sqrt{n}$ distinct values for the probability of a hash collision to be larger than 50%.

In practice: SHA256 produces 2^{256} hashes.

- 256 bits, each bit presents the possibility between 0 and 1.
- $2^{256} = (2^{32})^8 = (4.3 \text{ billions})^8 = (1.16\text{x}10^{77})$
- The probability of a hash collision in SHA256 is 1 out of $(4.3 \text{ billions})^8$

We need to test at least $1.2*(4.3 \text{ billions})^4$ hashes for a 50% chance of a collision.

SHA256 in practice

In practice, SHA256 has many different applications.

Example 1: Verify the downloaded file

- E.g., sha256sum for Linux distributions
- Compare the hashes for integrity
- Check digital signature for authenticity and non-repudiation properties

On reliability: Data loss during transmission may be another reason for file integrity check

First open a terminal and go to the correct directory to check a downloaded **iso** file:

```
cd download_directory
```

Then run the following command from within the download directory.

```
sha256sum ubuntu-9.10-dvd-i386.iso
```

sha256sum should then print out a single line after calculating the hash:

```
c01b39c7a35ccc3b081a3e83d2c71fa9a767ebfeb45c69f08e17dfe3ef375a7b *ubuntu-9.10-dvd-i386.iso
```

Compare the hash (the alphanumeric string on left) that your machine calculated with the corresponding hash in the SHA256SUMS file.

SHA256 in practice

Example 2: Password storage

- E.g., storing password as a (salted) hash in database systems
- When a user tries to login with the password, we compare the hashes
- The actual password is never stored anywhere

Next class: salting

Salting is a technique to protect passwords stored in databases by adding characters before hashing.

- Every password gets a random salt
- Extends the length of the original password
- Every hash value is different
- The salt is stored with the password

While salting does not stop the reverse-engineering, it slows down the brute force process.

TODOs

- Final report due Friday, February 9, 23:59 MST
  - One submission per team on eClass
  - Make sure to commit all your files on GitHub before the final report deadline
- Sign up for the demo time slots before February 10 (19 teams signed up)
  - Each demo is expected to be 10-15 minutes
  - All group members must attend
  - One booking per team on eClass
- Demo sessions will be held in DICE 11-251
  - Review the demo guide
  - Please be on time