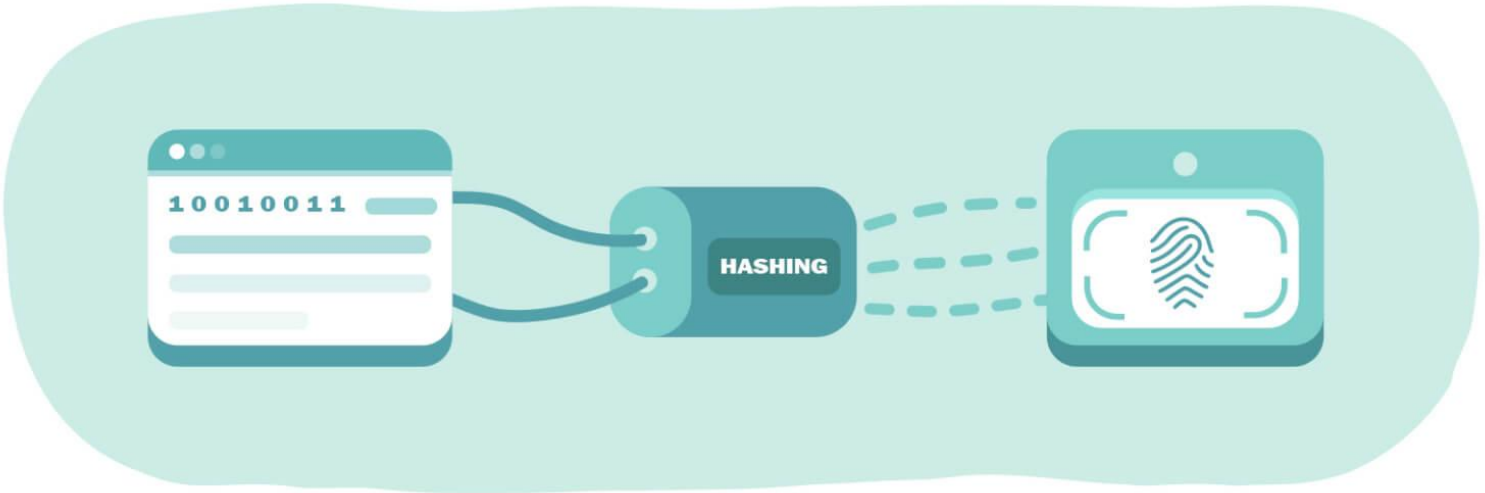
Cryptographic Hashing



We rely on cryptography to ensure **integrity, authentication and confidentiality**

of data so that information remains protected from unauthorized access or tampering.

These three pillars work together to guarantee that only the right people can read the data,

verify it hasn’t been altered, and confirm where it came from.

Encryption

- Encryption is used to provide Confidentiality
  - Confidentiality: Only intended recipient can interpret the Data

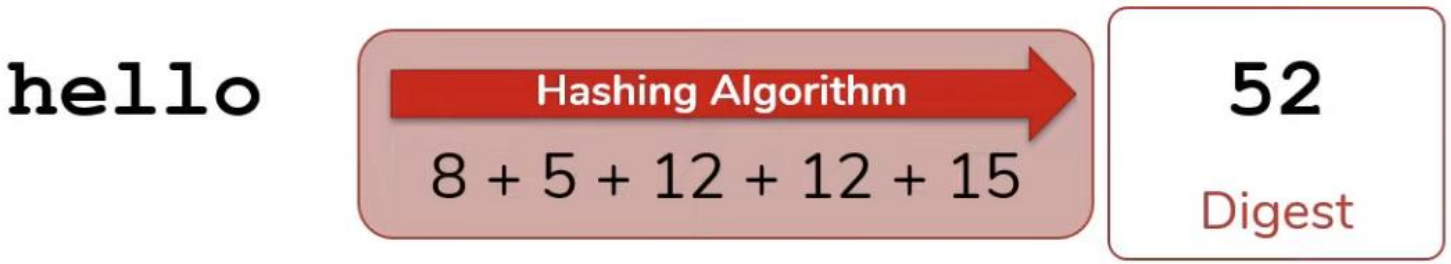
  - Plain Text: Data before encryption
  - Cipher Text: Data while encrypted

What is Hashing?

- Hashing is an algorithm that calculates a fixed-size bit string value from a file. A file basically contains blocks of data.
- Hashing transforms this data into a far shorter fixed-length value or key which represents the original string.
- The hash value can be considered the distilled summary of everything within that file.

Hashing

- Algorithm which takes as input a message of arbitrary length and produces as output a “fingerprint” of the original message



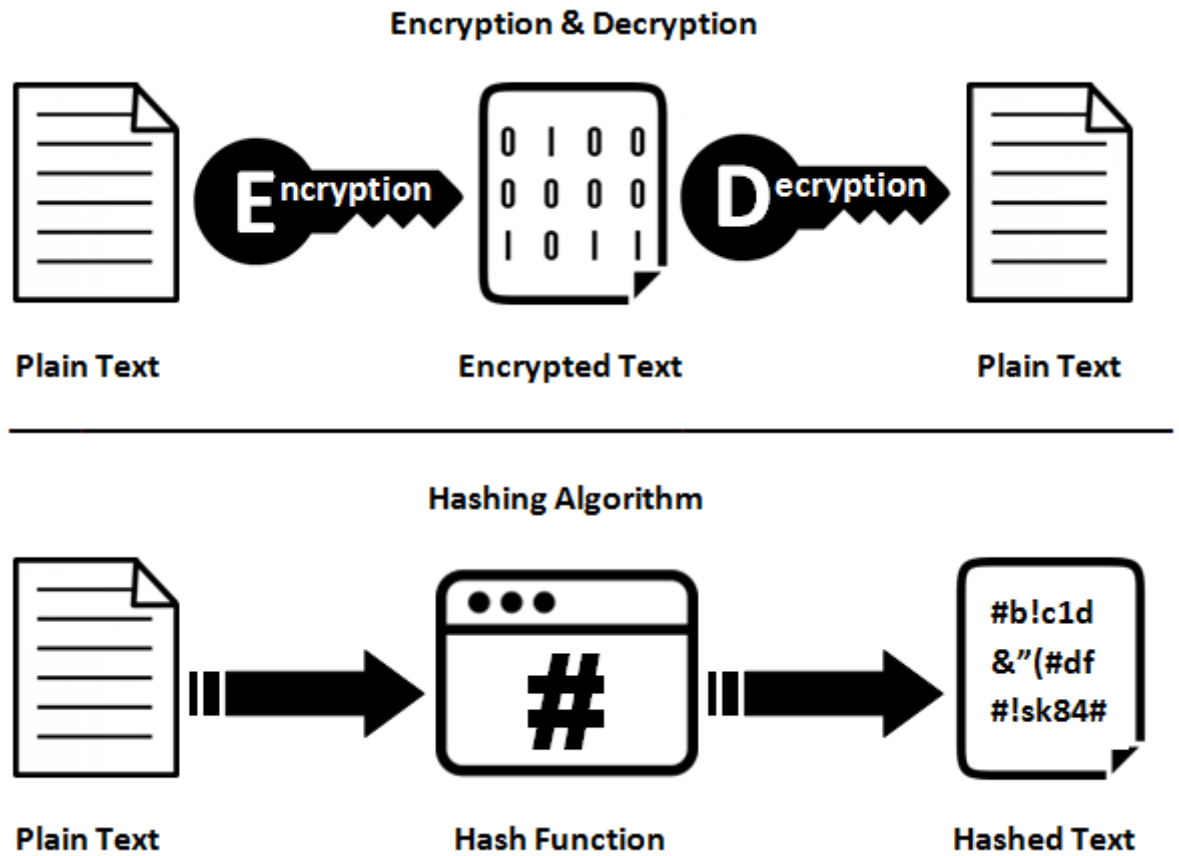
  - Result of the Hashing Algorithm is called a **Digest**
    - Also called: *Checksum*, *Fingerprint*, *Hash*, *CRC*, etc…

Rules for Hashing

- Real world Hashing Algorithms must satisfy four requirements:
  - Infeasible to produce a given digest
  - Impossible to extract original message
  - Slight changes produce drastic differences
  - Resulting digest is fixed width (length)

```
echo "hello" |md5sum
b1946ac92492d2347c6235b4d2611184
```

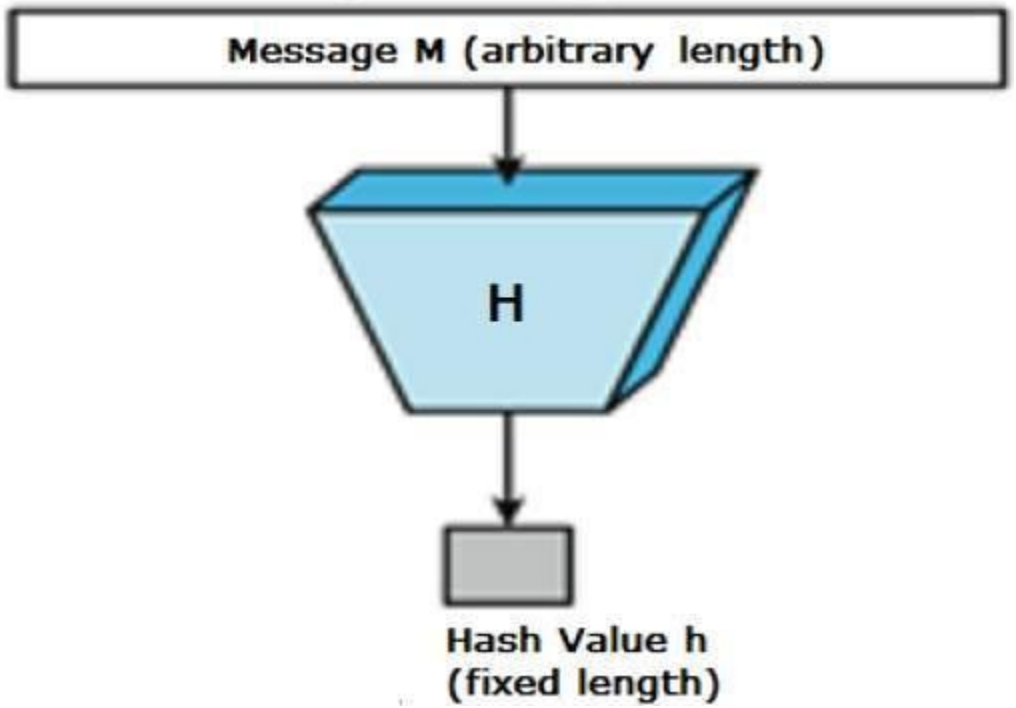
```
echo "hellow world , this is the networking class" | md5sum
08402369a3c5ed69ef435bd2d50e1ade
```


Encryption & Decryption
Encryption
Decryption
Plain Text
Encrypted Text
Plain Text
Hashing Algorithm
#b!c1d
&"(#df
#!sk84#
Plain Text
Hash Function
Hashed Text

- A good hashing algorithm would exhibit a property called the avalanche effect, where the resulting hash output would change significantly or entirely even when a single bit or byte of data within a file is changed.
- A hash function that does not do this is considered to have poor randomization, which would be easy to break by hackers.

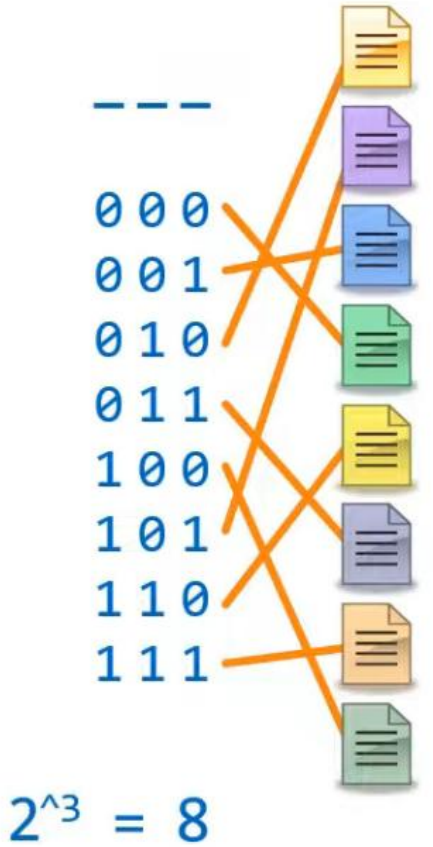
- A hash is usually a hexadecimal string of several characters. Hashing is also a **unidirectional** process so you can never work backwards to get back the original data.
- A good hash algorithm should be complex enough such that it does not produce the same hash value from two different inputs. If it does, this is known as a hash **collision**. A hash algorithm can only be considered good and acceptable if it can offer a very low chance of collision.

Summary



After hashing , we named the hashed file as digest

- Imaginary Hashing algorithm:
  - PRAC-HASH – 3 bit digest
    - “bit” is either 0 or 1
    - 3 bits means eight possible digests



Properties of Hash Functions

- Pre-image Resistance
  - Computationally unfeasible to reverse a hash function.
  - Inability to find the input value "x" that produced hash "z".
- 2nd Pre-image Resistance
  - Given an input and a hash, it should be hard to find a different input that produces the same hash.
  - In other words, if a hash function h for an input x produces hash value h(x), then it should be difficult to find any other input value y such that h(y) = h(x).
- Collision Resistance
  - This property means it should be hard to find two different inputs of any length that result in the same hash. This property is also referred to as collision free hash function.
  - In other words, for a hash function h, it is hard to find any two different inputs x and y such that h(x) = h(y).

Preimage Resistance

- Given a hash value h, it should be hard to find any message m such that h = hash(k, m).

Uses for Hashing

Password Verification

- Storing passwords in a regular text file is dangerous, so nearly all sites store passwords as hashes. When a user inputs their password, it is hashed and the result is compared to the list of hashed values stored on the company's servers. This is not a fool-proof practice, however, as the Collection #1 trove of 21 million stolen passwords, discovered in 2019, demonstrates.2

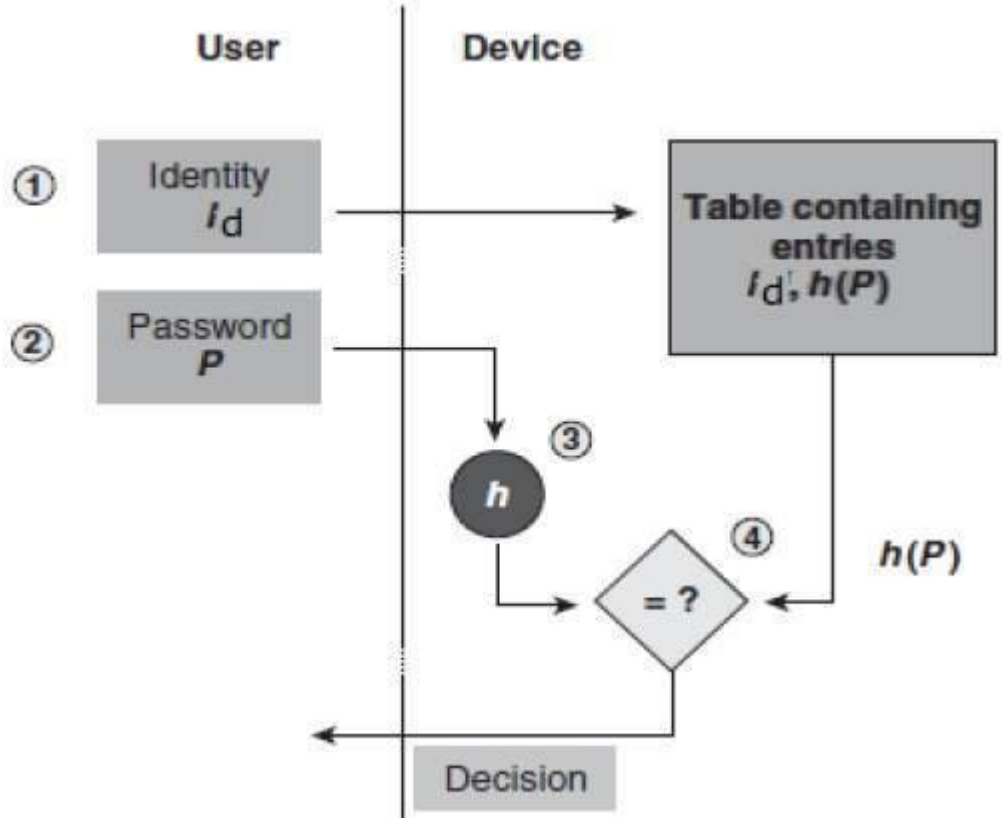
Signature Generation and Verification

- Verifying signatures is a mathematical process used to verify the authenticity of digital documents or messages. A valid digital signature, where the prerequisites are satisfied, gives its receiver strong proof that the message was created by a known sender and that the message was not altered in transit. A digital signature scheme typically consists of three algorithms: a key generation algorithm; a signing algorithm that, given a message and a private key, produces a signature; and a signature verifying algorithm. Merkle Trees, a technology used in cryptocurrencies, is a kind of digital signature.

Verifying File and Message Integrity

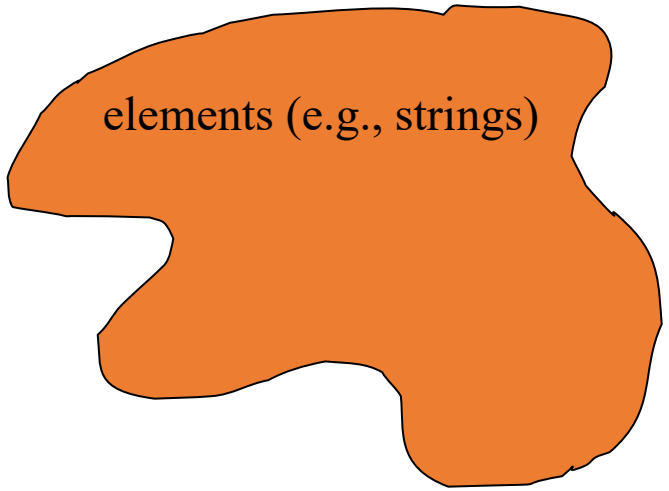
- Hashes can be used to make sure messages and files transmitted from sender to receiver are not tampered with during transit. The practice builds a "chain of trust." For example, a user might publish a hashed version of their data and the key so that recipients can compare the hash value they compute to the published value to make sure they align.

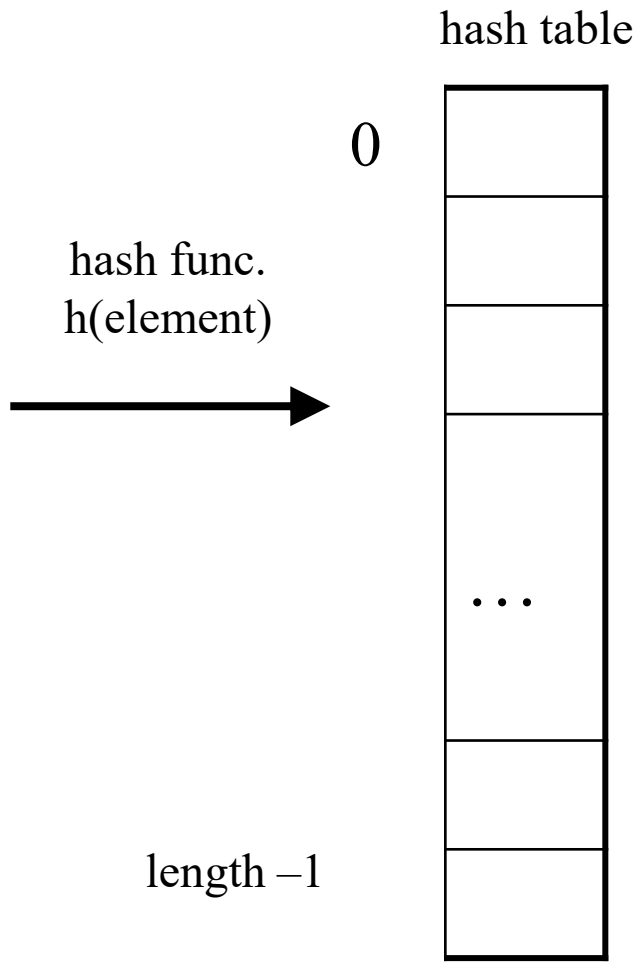
Password Validation



Hash Tables

- **hash table**: an array of some fixed size, that positions elements according to an algorithm called a **hash function**





Types of Collision Resistance

- WEAK
  - The hash has very little chance of colliding with the subset of hashes you generate in your own subset of hashes (your database).
- STRONG
  - The hash has a very little chance of colliding with ALL instances of the hash, in all existence.

Common Tools Used for Hashing (Windows)

- DOS
  - certutil -hashfile raw.txt SHA512
  - certutil -hashfile raw.txt MD5
- PowerShell
  - Get-FileHash -Path z:\desktop\raw.txt -Algorithm SHA256
- Linux
  - md5sum filename

- Common Hashing Algorithms:
  - MD5 128 bits
  - SHA / SHA1 160 bits
  - SHA2 family:
    - SHA-224 224 bits
    - SHA-256 256 bits
    - SHA-384 384 bits
    - SHA-512 512 bits

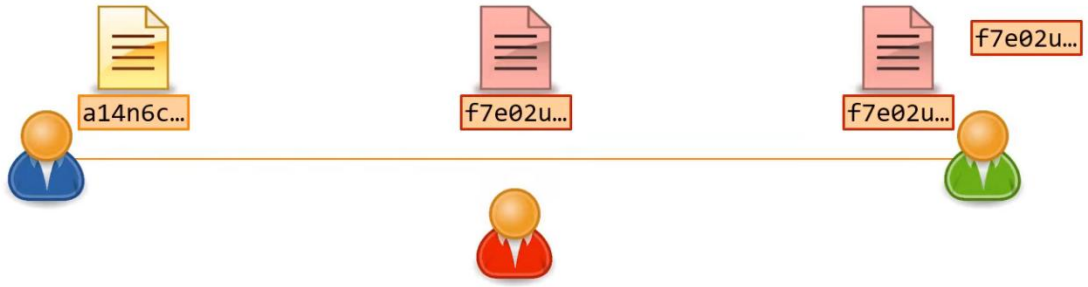
Data Integrity - How Hashing is used to ensure data isn't modified - HMAC

Data Integrity

- Hashing is used to provide Integrity
  - Sender calculates Digest from the Message
  - Sender sends Message and Digest
  - Receiver calculates Digest from received Message
  - Receiver compares both digests
    - If digests are identical, the message was not modified in transit

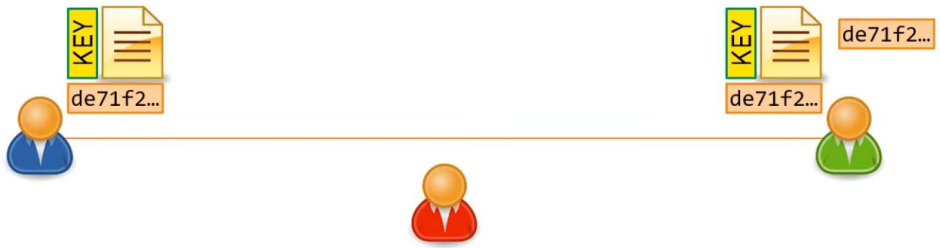
Hashing is not enough for data integrity

- Simply sending the Message's Digest is not sufficient
  - Something else must be done...



Assume that both parties can share a mutual **Secret key** (We will discuss later how this process happens)

- Sender combines Message + Secret Key to create Digest
- Receiver verifies by calculating hash of Message + Secret Key
  - Message was not modified in transit **Integrity**
  - Sender had the identical Secret Key **Authentication**



- **Message Authentication Code** (MAC)
  - Concept combining Message + Secret Key when calculating Digest
  - Provides **Integrity** and **Authentication** for Bulk Data transfer

Industry has a specific standard method to implement Message Authentication Code (MAC) v

Hash Based Message Authentication Code (RFC 2104)

HMAC defines how we can combine message with and secret key.

If you remember , for networking we talk about 3-way handshake for TLS connection. One of the part Of encryption is hashing for Data integrity and Authentication.

In a nutshell

Data Integrity

- **Hashing Algorithm**
  - INPUT: Message
  - OUTPUT: Digest
  - Example: MD5, SHA1, etc...

- **MAC – Message Authentication Code**
  - INPUT: Message + Secret Key
  - OUTPUT: Digest
  - Example: HMAC (Hash Based Message Authentication Code)