Securing Passwords

Securing Passwords

1. Hashing
2. Rainbow tables
3. Salts
4. Password cracking tools

Number Game

Students	vs.	Professor
Pick a number **A**		Pick a number **B**
Students win if **(A + B)** is odd		Professor wins if **(A + B)** is even

Securing Passwords

1. Hashing
2. Rainbow tables
3. Salts
4. Password cracking tools

/etc/passwd

```
alice:P@ssword123!:1020:1003::/home/alice:/bin/bash
bob:Setec@str0n0my:1021:1003::/home/bob:/bin/bash
```

/etc/passwd

```
alice:$y$j9T$xwEl…:1020:1003::/home/alice:/bin/bash
bob:$y$j9T$q6RlJa…:1021:1003::/home/bob:/bin/bash
```

hashed password

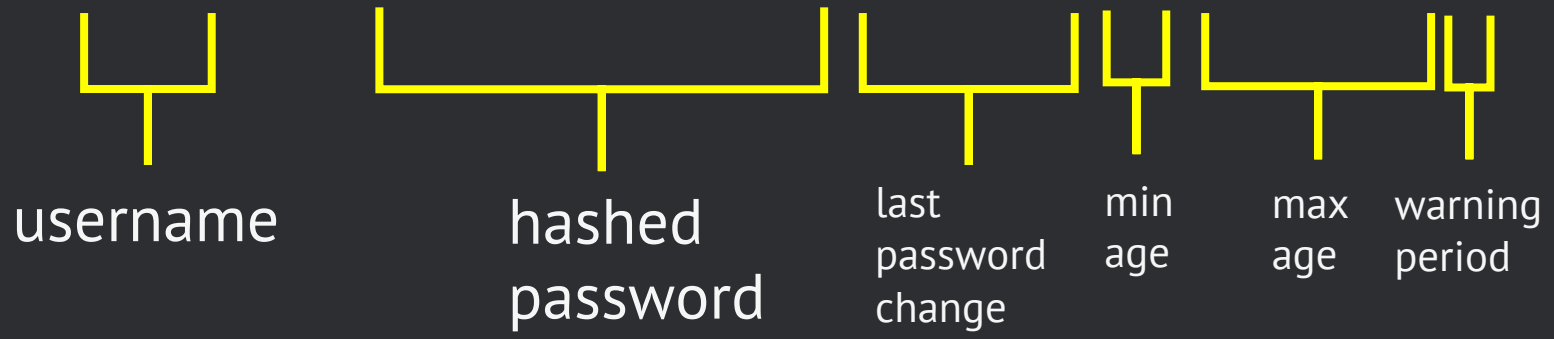
/etc/passwd

```
alice:x:1020:1003::/home/alice:/bin/bash
bob:x:1021:1003::/home/bob:/bin/bash
```

hashed password is stored in the file **/etc/shadow**

/etc/shadow

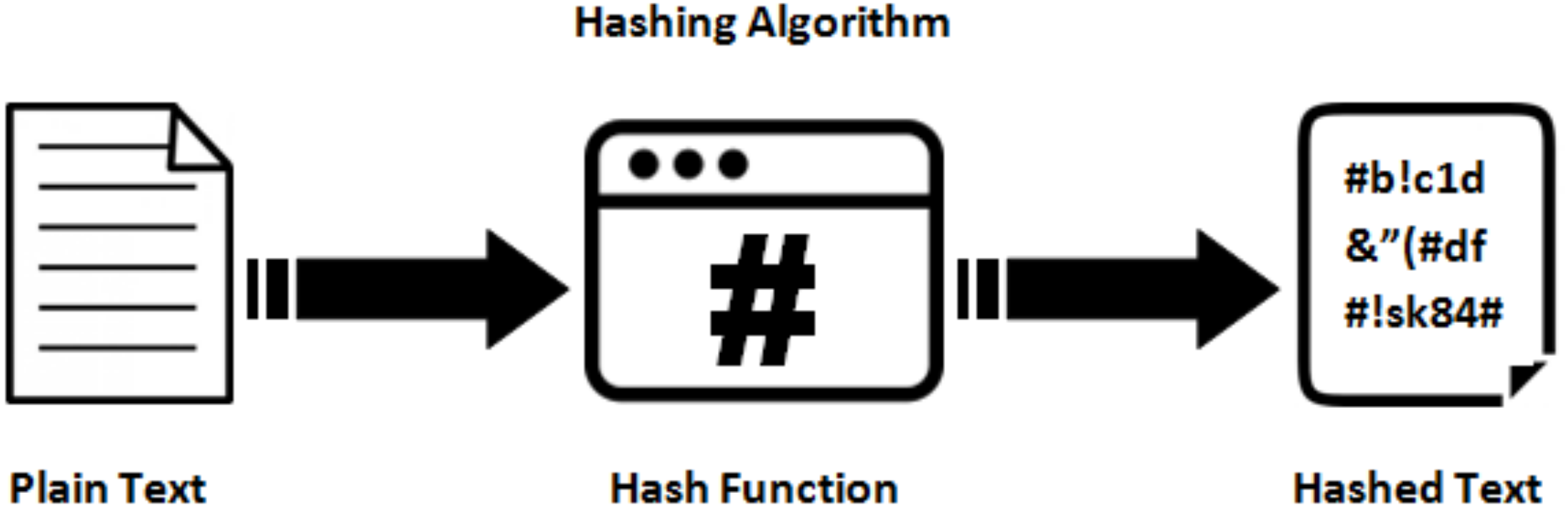
```
alice:$y$j9T$xwEl…:19741:0:99999:7:::
bob:$y$j9T$q6RlJa…:19741:0:99999:7:::
```



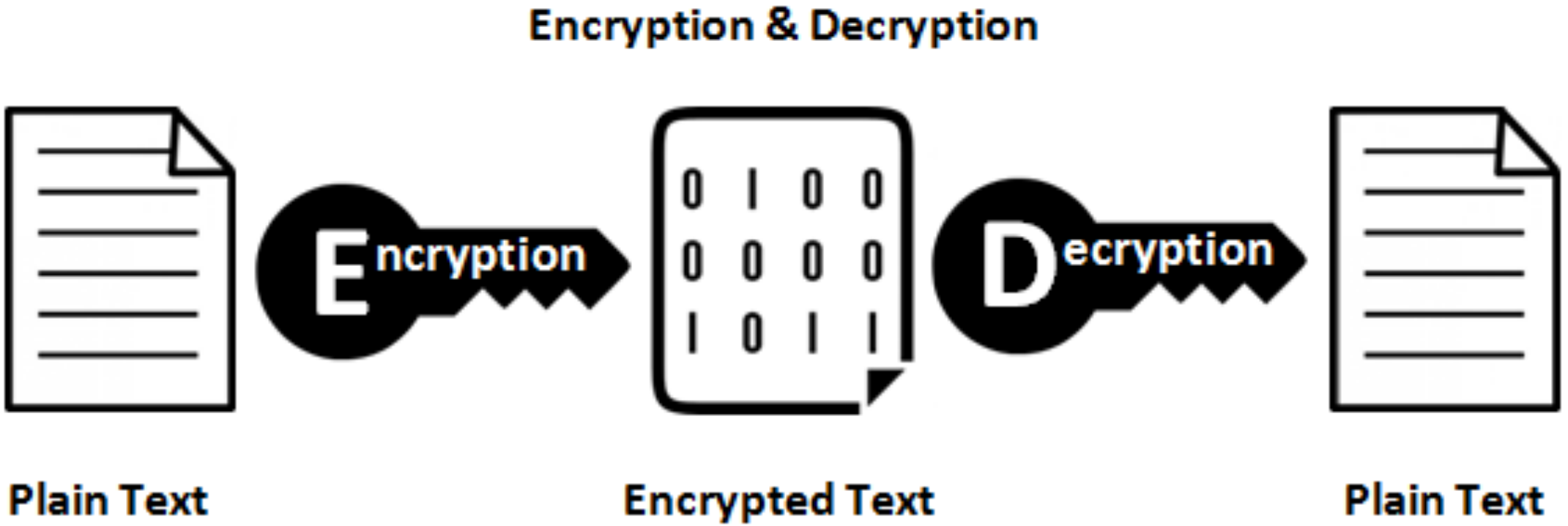
/etc/shadow

```
alice:$y$j9T$xwEl…:19741:0:99999:7:::
bob:$y$j9T$q6RlJa…:19741:0:99999:7:::
```

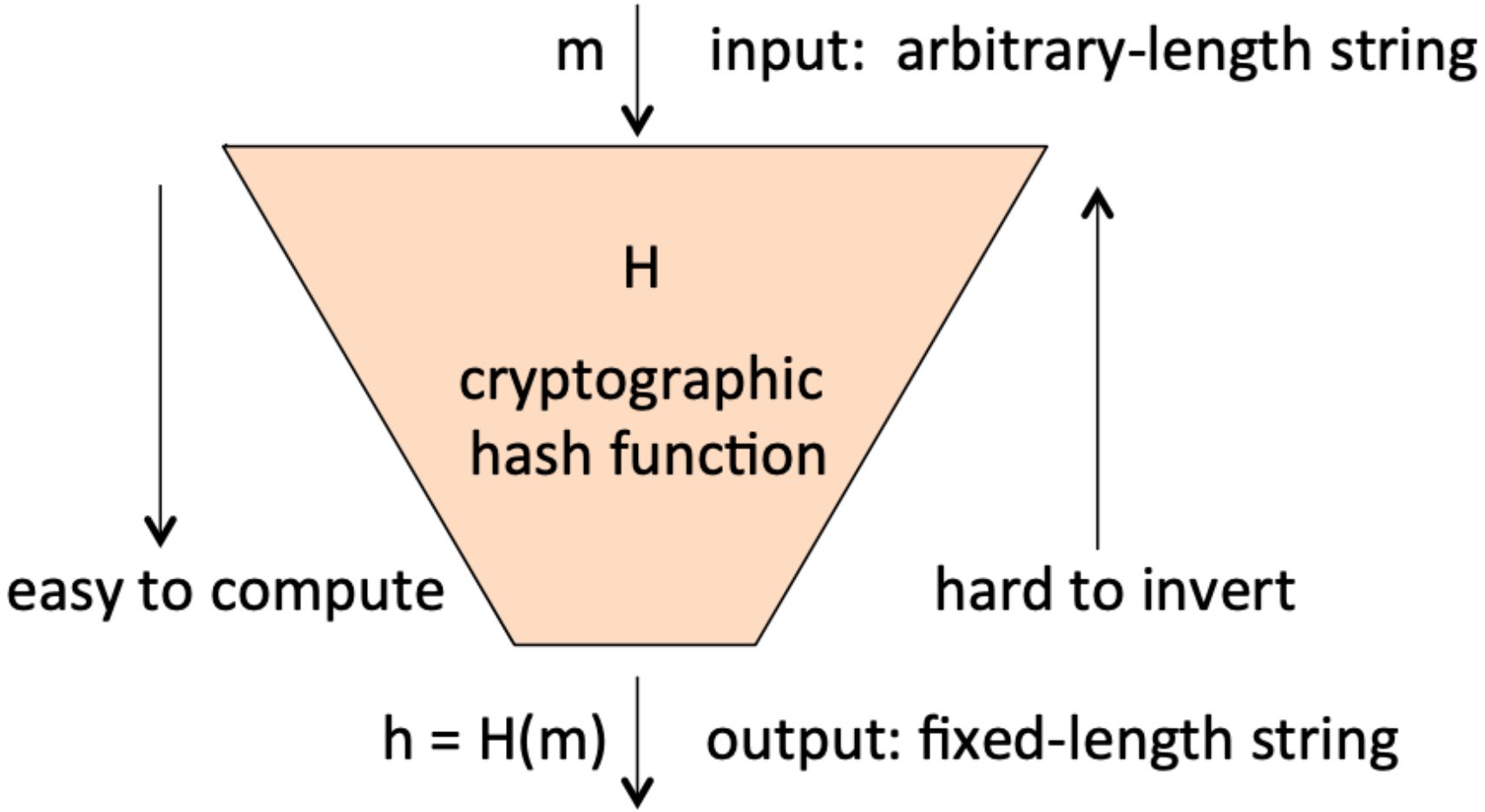
Hashing


Hashing Algorithm
#
#b!c1d
&”(#df
#!sk84#
Plain Text
Hash Function
Hashed Text

Encryption


Encryption & Decryption
Encryption
Decryption
Plain Text
Encrypted Text
Plain Text

Cryptographic hash function



https://people.scs.carleton.ca/~paulv/toolsjewels/TJrev1/ch2-rev1.pdf

$$f(x) = x \bmod 1000$$

Properties of cryptographic hash functions

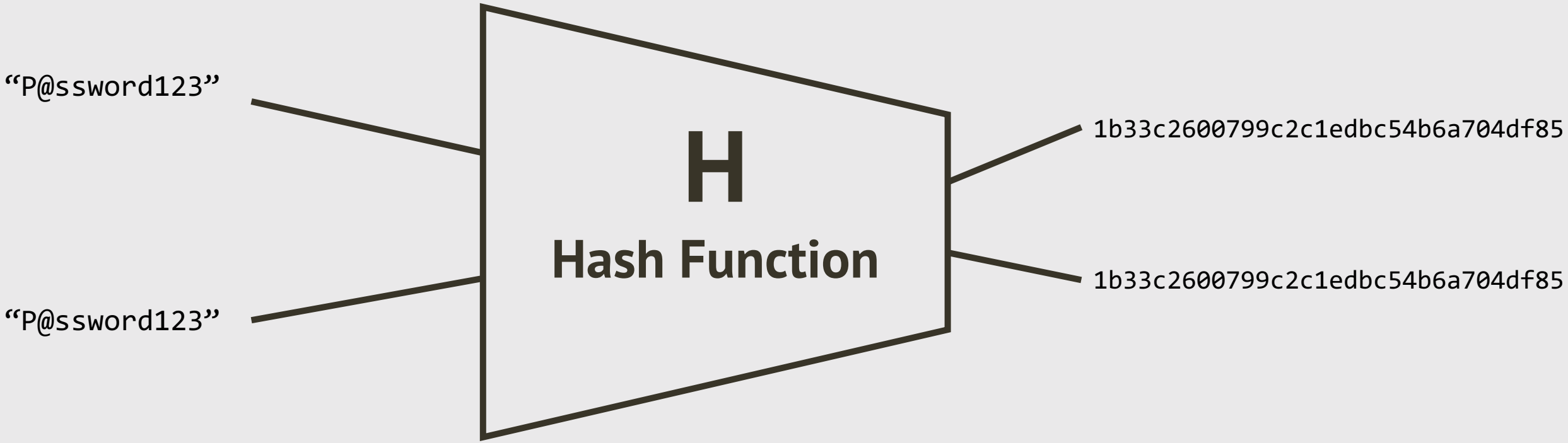
1. **One-Way Property:**
   - For essentially all possible hash values **`h`**
   - given a hash value **`h`**
   - it should be infeasible to find any message **`m`**
   - such that **`H(m) = h`**

Properties of cryptographic hash functions

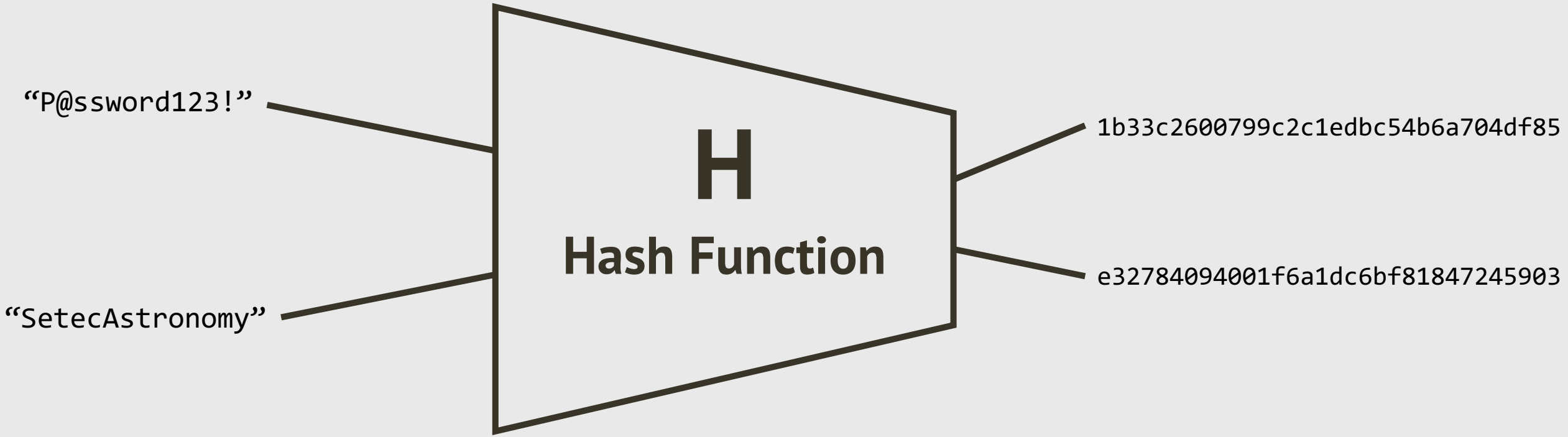
2. **Collision resistance**:
   - It should be infeasible to find any pair of distinct inputs **`m1`**, **`m2`**
   - such that **`H(m1) = H(m2)`**

   - note: here there is free choice of both **`m1`** and **`m2`**
   - When two distinct inputs hash to the same output, we call it a **collision**

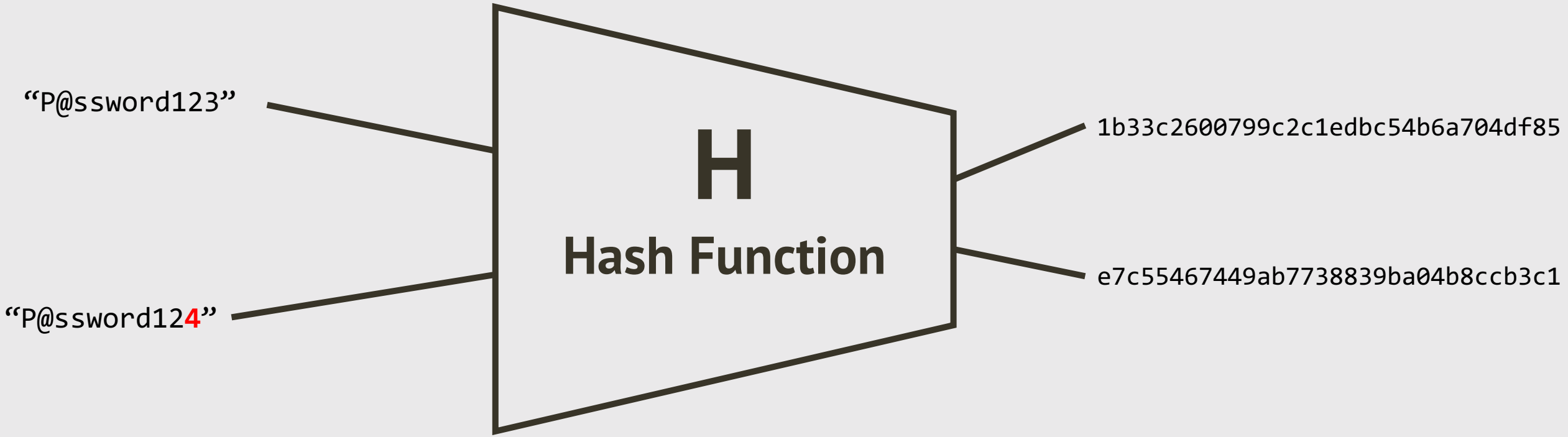
First, if you compute the hash code of the **same string** many times, you **always get the same value**



Second, the hash codes of different inputs are (usually) **very** different from one another

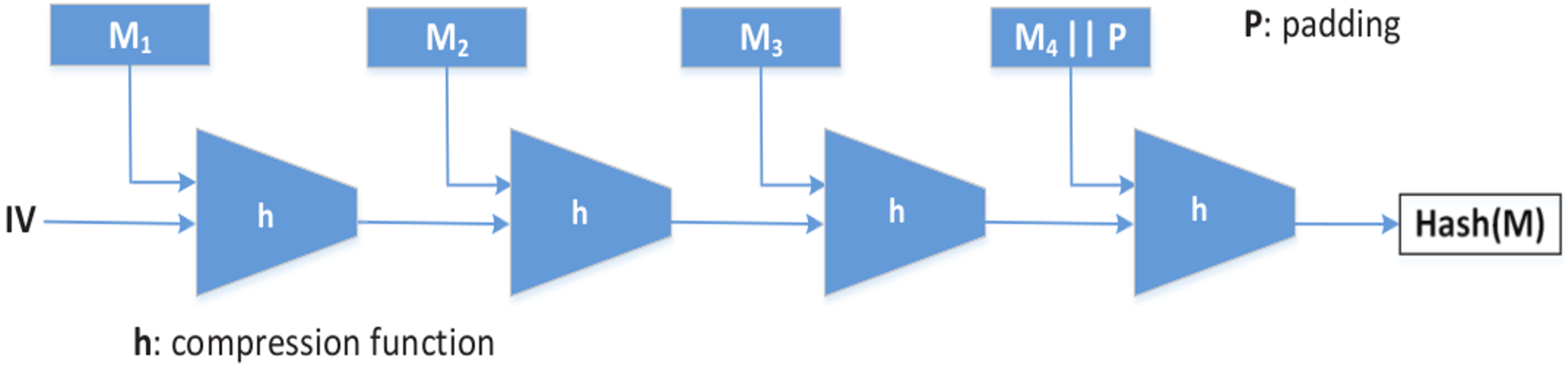


Even very similar inputs give **very different outputs**!



How a one-way hash algorithm works

- Construction method called Merkle–Damgard
- Used by algorithms like MD5, SHA-1, and SHA-2



One-way hash commands

Linux utility programs

- Examples:
  - `md5sum`
  - `sha224sum, sha256sum, sha384sum, sha512sum`

```
$ md5sum file.c
919302e20d3885da126e06ca4cec8e8b   file.c

$ sha256sum file.c
0b2a06a29688...(omitted)...1f04ed41d1   file.c
```

One-way hash commands (continued)

Using the **`openssl`** to calculate a hash

```
$ openssl dgst -sha256 file.c
SHA256(file.c)= 0b2a06a29688...(omitted)...1f04ed41d1

$ openssl sha256 file.c
SHA256(file.c)= 0b2a06a29688...(omitted)...1f04ed41d1

$ openssl md5 file.c
MD5(file.c)= 919302e20d3885da126e06ca4cec8e8b

$ openssl dgst -md5 file.c
MD5(file.c)= 919302e20d3885da126e06ca4cec8e8b
```

Integrity verification

- Changing one bit of the original data changes hash value

```
$ echo -n "Hello World" | sha256sum
a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b57b277d9ad9f146e

$ echo -n "Hallo World" | sha256sum
d87774ec4a1052afb269355d6151cbd39946d3fe16716ff5bec4a7a631c6a7a8
```

- Usage examples:
  - Detect change in system files
  - Detect if file downloaded from website is corrupted
  - https://dev.mysql.com/downloads/mysql/

Committing a secret without telling it

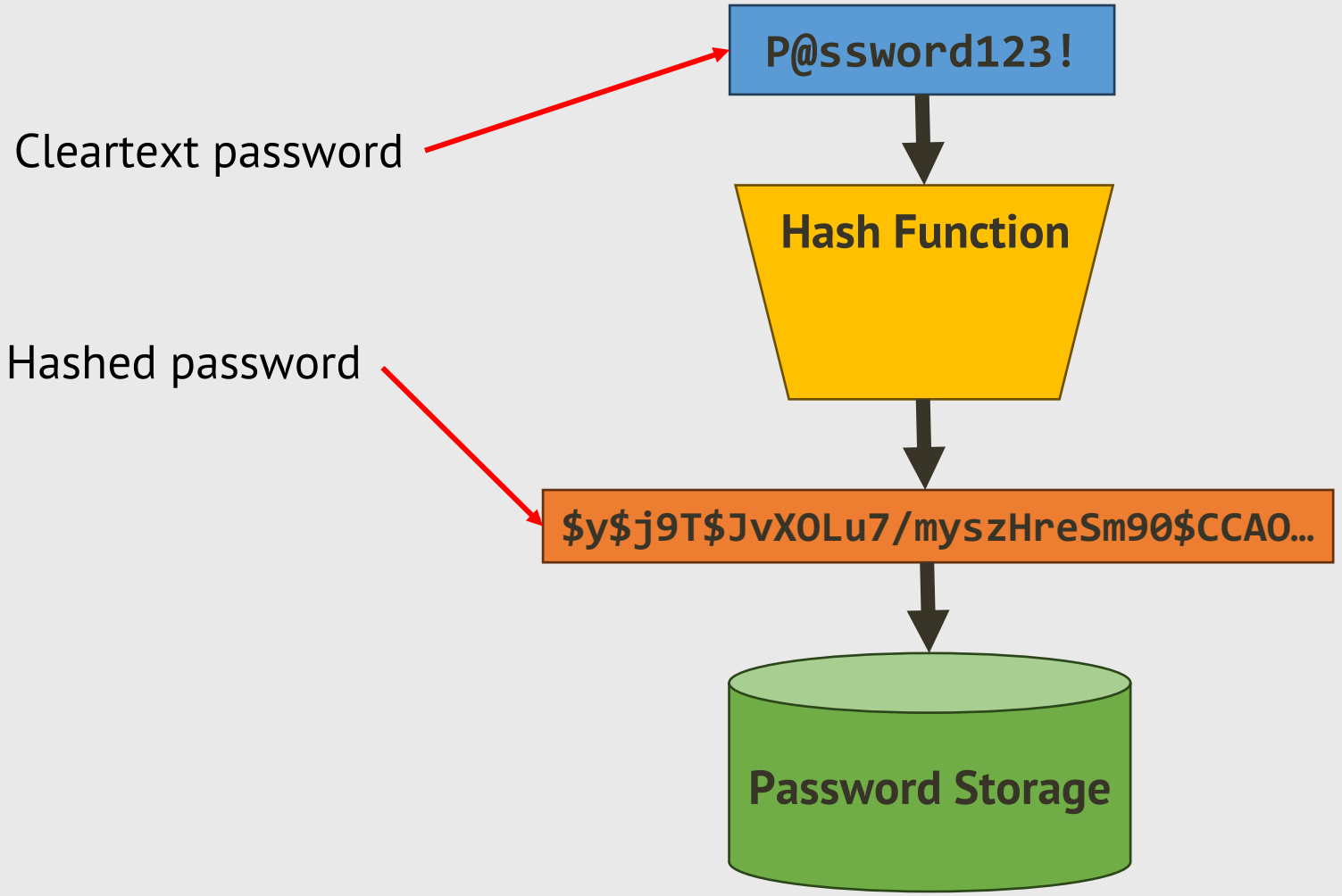
- One-way property
  - Disclosing the hash does not disclose the original message
  - Useful to commit secret without disclosing the secret itself
- Usage Example - Stock Market
  - Need to make prediction about the stock market about a certain day
  - Publish the hash of the secret on your website
  - On the particular day, release the secret
  - Your audience can verify it against the hash

Password Verification

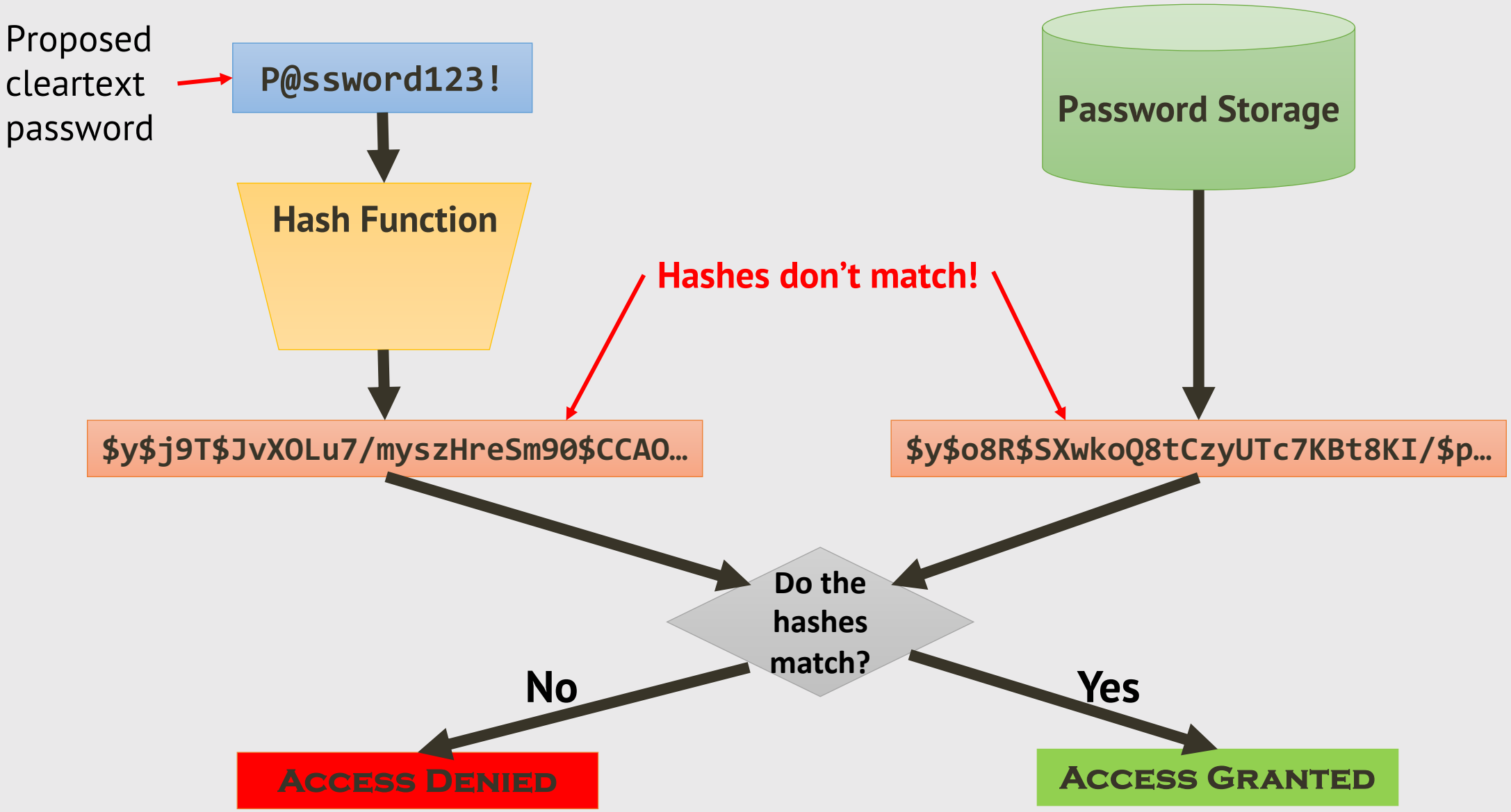
- To login into account, user needs to tell a secret (password)
- Cannot store the secrets in their plaintext
- Need for:
  - Password storage where nobody can know what the password is
  - If provided with a password, it verified against the stored password
- Solution: one-way hash function
- Example: Linux stores passwords in the /etc/shadow file

```
alice:$y$j9T$xwEl…:19741:0:99999:7:::
bob:$y$j9T$q6RlJa…:19741:0:99999:7:::
```

Storing a hash instead of a password



Testing a proposed password against stored



Number Game Replay

Students	vs.	Professor
Pick a number **A**		Pick a number **B**
Hash the number **A**		Hash the number **B**
Share the hash with Professor		Share the hash with Students
Then share the number		Then share the number
Students win if **(A + B)** is odd		Professor wins if **(A + B)** is even

Securing Passwords

1. Hashing
2. Rainbow tables
3. Salts
4. Password cracking tools

Rainbow tables

- Specialized tables used for decrypting hashed passwords

- Compare hashed passwords against precomputed hashed values

- Involves selecting plaintexts, applying hash and generating tables
- Efficiently crack passwords by matching hash values and reversing the process to find plaintexts
- More efficient than brute force
  - but less effective against salted hashes

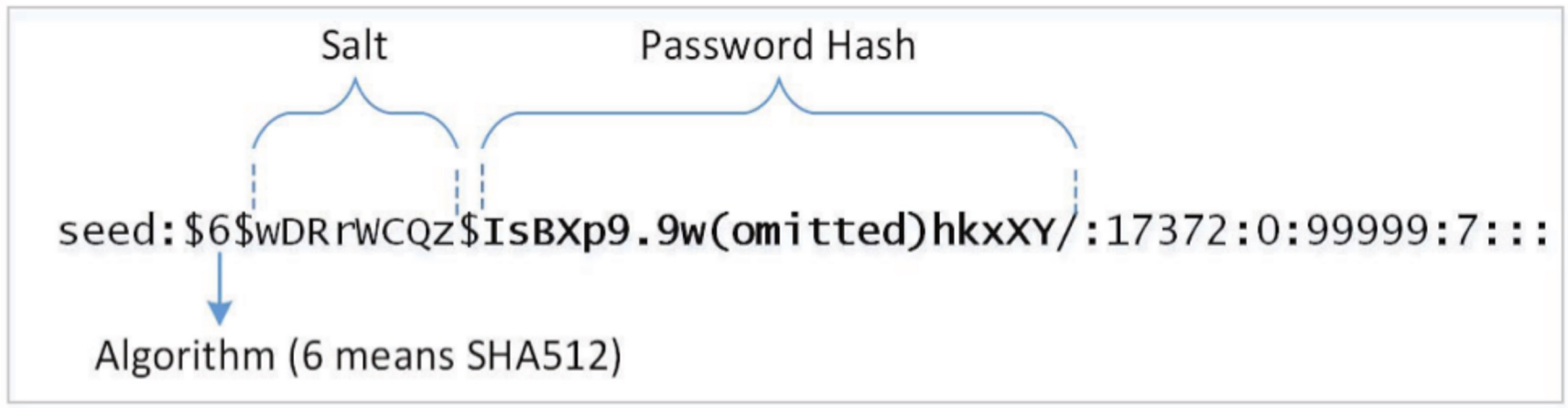
Rainbow table demo

Securing Passwords

1. Hashing
2. Rainbow tables
3. Salts
4. Password cracking tools

/etc/shadow

1. Password field has 3 parts: algorithm used, salt, password hash
2. Salt and password hash are encoded into printable characters
3. Multiple rounds of hash function (slow down brute-force attack)



Purpose of Salt

- Using salt, same input can result in different hashes
- Password hash = one-way hash rounds

  `(password || random string)`
- Random string is the salt

Attacks Prevented by Salt

- Dictionary Attack
  - Put candidate words in a dictionary
  - Try each against the targeted password hash to find a match
- Rainbow Table Attack
  - Precomputed table for reversing cryptographic hash functions
- Why Salt Prevents them?
  - If target password is same as precomputed data, the hash will be the same
  - If this property does not hold, all the precomputed data are useless
  - Salt destroys that property

Securing Passwords

1. Hashing
2. Rainbow tables
3. Salts
4. Password cracking tools

Password cracking strategy

1. Brute force very short passwords
2. Low-hanging fruit
   - Dictionary words that are eight characters long
3. Try common passwords
4. Combine words with numbers
5. Combine words with numbers and special characters