What Is Hashing?

Hashing turns a password into a fingerprint that cannot be turned back — learn why passwords are never saved as plain text.

Introduction

Encryption is a two-way street: lock a message, unlock it later. Hashing is a one-way street: put something in, get a fixed-size fingerprint out — and there is no way back. You can check whether two things match by comparing their fingerprints, but you can never rebuild the original from the fingerprint alone.

That is exactly what a password needs. A website must check your password, but it should never store your password. So it stores the hash — the fingerprint — instead. When you log in, it hashes what you typed and compares fingerprints. Same fingerprint, same password. And if thieves steal the database, they get fingerprints, not passwords.

In this lesson you will hash words yourself in Python, see how one changed letter changes everything, and learn why good password storage adds salt and uses a slow hash.

(Roman Urdu: Hashing aik one-way kaam hai — password ko aik fingerprint mein badal do. Website fingerprint save karti hai, password nahi. Fingerprint se password wapas nahi banta, is liye chor bhi password nahi nikaal sakte.)

The beginner mistake is thinking hashing and encryption are the same thing. They are not: encryption is reversible with a key, hashing is not reversible at all. That one-way property is the whole point.

Key points

- **A hash function takes any input and returns a fixed-size output (the hash or digest).** The same input always gives the same hash. Different inputs give different hashes.
- **Hashing is one-way.** You cannot turn a hash back into the original. That is why passwords are hashed, not encrypted.
- **A tiny change changes everything (the avalanche effect).** Change one letter and the hash looks completely different — which makes tampering easy to spot.
- **A good hash is fast to compute and hard to reverse.** Common hash functions: SHA-256 (general use), bcrypt and Argon2 (passwords — deliberately slow).
- **Salt is random data added before hashing.** Two users with the same password then get different hashes, and attackers cannot pre-compute hashes for common passwords (rainbow tables).
- **A slow hash protects passwords.** bcrypt and Argon2 are slow on purpose: guessing a million passwords takes a million slow steps instead of a million fast ones.
- **Hashes also prove files are unchanged.** Download a file, hash it, compare with the hash the maker published — if they match, the file was not tampered with.

- **MD5 and SHA-1 are broken for security.** They are still fine for finding duplicate files, but never for passwords or signatures — attackers can craft two different inputs with the same hash (a collision).

Examples

Example 1 — SHA-256 in Python (runnable)

```
import hashlib

def sha(text):
    return hashlib.sha256(text.encode()).hexdigest()

print(sha("hello"))
print(sha("hello"))
print(sha("Hello"))
```

What you will get:

```
2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
185f8db32271fe25f561a6fc938b2e264306ec304eda518007d1764826381969
```

"hello" always gives the same hash. "Hello" — one capital letter — gives a totally different hash. That is the avalanche effect.

Example 2 — How a website checks your password

```
Sign up:
  you type "sunflower99"
  site stores: sha256("sunflower99") = "a1b2c3..."  (never the password)

Log in:
  you type "sunflower99"
  site computes sha256("sunflower99") = "a1b2c3..."
  "a1b2c3..." == stored "a1b2c3..."  ->  welcome in!

Wrong password:
  you type "sunflower98"
  site computes a totally different hash  ->  access denied.
```

The site never knew your password — it only compared fingerprints.

Example 3 — Salt: same password, different hash

```
Without salt:
  Ali's "sunflower99"     -> "a1b2c3..."
  Sara's "sunflower99"    -> "a1b2c3..."   (same! attacker sees the match)

With salt (random per user):
  Ali's "sunflower99" + "x7Q2"  -> "9f04e1..."
  Sara's "sunflower99" + "mK91" -> "c77a20..."  (different!)
```

Salt is stored next to the hash — it is not secret. Its job is to make every hash unique, so pre-computed lists of common passwords are useless.

Example 4 — Checking a download was not tampered with

```
import hashlib

data = open("notes.pdf", "rb").read()
print(hashlib.sha256(data).hexdigest())
```

Compare the printed hash with the one published on the official download page. If they match exactly, the file is the real one. If even one byte changed, the hash is completely different.

Quick recap

- Hashing is one-way: input goes in, a fixed-size fingerprint comes out, and there is no way back.
- Same input, same hash; one changed letter, a completely different hash.
- Passwords are stored as salted, slow hashes — never as plain text.
- Salt makes equal passwords produce different hashes.
- Hashes also check that files and messages were not changed.

Common mistakes

1. **Storing passwords as plain text** — one database leak exposes every password. Fix: salted slow hashes (bcrypt/Argon2), never plain text.
2. **Using a fast hash like SHA-256 alone for passwords** — attackers can try billions of guesses per second. Fix: use a slow password hash (bcrypt/Argon2) with salt.
3. **Using MD5 or SHA-1 for security** — both are broken; collisions can be crafted. Fix: SHA-256 or better for fingerprints, bcrypt/Argon2 for passwords.
4. **Thinking a hash is encryption** — there is no key and no "decrypt" step. Fix: if you need to read it back later, you need encryption, not hashing.

5. **Forgetting salt** — without salt, every "123456" in the world has the same hash, and one pre-computed list cracks them all. Fix: a fresh random salt for every password.

Practice

1. Run the Python example: hash "hello", "Hello", and "hello " (with a space). How many characters changed in the hash each time?
2. Hash your own first name, then hash it with one letter changed. Compare the two hashes — that is the avalanche effect.
3. Explain in two lines why a website that stores plain-text passwords is dangerous, even if its login page looks safe.
4. Check: does any app you use show a "checksum" or "SHA-256" next to a download? Find one and write down what it is for.
5. Two users both chose the password "cricket123". Without salt their hashes match; with salt they do not. Write one line explaining why that matters to an attacker.

Practice quiz

1. What is the key difference between hashing and encryption?

- There is no difference
- Hashing is one-way and cannot be reversed; encryption can be reversed with a key
- Encryption is one-way and cannot be reversed
- Hashing needs a key, encryption does not

Answer: Hashing is one-way and cannot be reversed; encryption can be reversed with a key

2. Why do websites store a hash of your password instead of the password itself?

- Hashes are shorter to type
- So a stolen database gives thieves fingerprints, not passwords
- So the website can email you your password
- Hashes are faster to guess

Answer: So a stolen database gives thieves fingerprints, not passwords

3. What does salt do when hashing passwords?

- It makes the password secret
- It adds random data so equal passwords get different hashes
- It encrypts the hash
- It makes hashing faster

Answer: It adds random data so equal passwords get different hashes

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/cs-hashing · More free tools and notes at https://scopetreasure.com