
WEEK 1 – DAY 4

AUTHENTICATION BASICS

(Very Detailed, Easy-to-Explain, Interview-Ready)

---

1. What Is Authentication? (Start From Zero)

Authentication answers one question:

"Who are you?"

It verifies the **identity** of a user before allowing access.

Examples:

- Login with username & password
- OTP
- Biometrics

Authentication is the **first gate** of security.
If authentication is weak → **entire system is compromised**.

---

2. Why Password Storage Is CRITICAL

Many beginners think:

"Just store passwords in the database."

❌ This is **dangerous and wrong**.

Why?

If the database is leaked:

- Attackers get all user passwords
- Users reuse passwords across sites
- Massive damage happens

Correct rule:

"Passwords must NEVER be stored in plain text."

---

3. Plain Text Passwords (WHAT NOT TO DO)

Example (Very Bad Practice):

username | password

kavya | mypassword123

Why This Is Dangerous:

- Anyone with DB access can read passwords
- Insider threats
- Complete account takeover

Interview line:

“Storing passwords in plain text is a critical security vulnerability.”

---

4. Hashing vs Encryption (VERY IMPORTANT)

This is a **frequently asked interview question**.

4.1 Hashing

Simple Meaning:

Hashing converts data into a **fixed-length value** that **cannot be reversed**.

- One-way process
- Original password cannot be retrieved

Example:

password123 → a94a8fe5ccb19ba61c4c0873d391e987

Even if attacker gets the hash:

- They cannot directly get the password

Key Properties of Hashing

- Same input → same hash
- Small change → completely different hash
- Not reversible

4.2 Encryption

Simple Meaning:

Encryption converts data into a secret form that **can be decrypted** using a key.

- Two-way process
- Used for data storage and transmission

Comparison (Interview Table)

Feature	Hashing	Encryption
Reversible	❌ No	✅ Yes
Used for passwords	✅ Yes	❌ No
Needs key	❌ No	✅ Yes

Interview Line (Very Strong)

"Passwords should be hashed, not encrypted, because authentication only needs verification—not recovery."

---

5. Why Normal Hashing (SHA-256) Is NOT Enough

Beginners think:

"I will hash passwords using SHA-256."

❌ This is **still insecure**.

Why?

- SHA-256 is **fast**
- Attackers can brute-force billions of hashes per second using GPUs

---

6. Password Hashing Algorithms (Correct Way)

Password hashing algorithms are **intentionally slow**.

Why slow?

- Makes brute-force attacks expensive
- Protects even if DB is leaked

6.1 bcrypt

- Adaptive hashing algorithm
- Uses **cost factor** (work factor)
- Widely used and trusted

6.2 Argon2 (Modern & Best)

- Winner of Password Hashing Competition
- Memory-hard (resists GPU attacks)
- Very secure

Interview Recommendation

"Argon2 is currently recommended, with bcrypt as a widely supported alternative."

---

7. Salting (EXTREMELY IMPORTANT)

7.1 What Is a Salt?

A **salt** is a **random value added to the password before hashing**.

Example:

password + random_salt → hash

7.2 Why Salting Is Needed

Without salt:

- Same passwords → same hash
- Rainbow table attacks possible

With salt:

- Same passwords → different hashes
- Rainbow tables become useless

Real Example

Without salt:

password123 → hash1

password123 → hash1

With salt:

password123 + salt1 → hashA

password123 + salt2 → hashB

Important Note

Modern algorithms like **bcrypt and Argon2 handle salting automatically**.

---

8. Authentication Flow (Step-by-Step)

Signup:

1. User enters password
2. Server hashes password
3. Hash is stored in DB

Login:

1. User enters password
2. Server hashes input password
3. Compares with stored hash
4. If match → login success

Interview Line

“Passwords are never decrypted or retrieved; they are only verified.”

---

9. Hands-On: Implement Password Hashing in Python

Below is **production-correct, interview-safe code**.

9.1 Using bcrypt (Recommended & Simple)

Install bcrypt

pip install bcrypt

Hashing a Password

```
import bcrypt


password = "MySecurePassword123"


# Convert password to bytes
```

```
password_bytes = password.encode('utf-8')

# Generate salt and hash
hashed_password = bcrypt.hashpw(password_bytes, bcrypt.gensalt())

print("Hashed password:", hashed_password)
```

Verifying a Password

```
import bcrypt

entered_password = "MySecurePassword123"
entered_bytes = entered_password.encode('utf-8')

# Compare with stored hash
if bcrypt.checkpw(entered_bytes, hashed_password):
    print("Password is correct")
else:
    print("Invalid password")
```

Important Observations

- Salt is automatically handled
- Hash looks different every time
- Same password still verifies correctly

9.2 Using Argon2 (Advanced / Best Practice)

Install Argon2

```
pip install argon2-cffi
```

Argon2 Example

```
from argon2 import PasswordHasher
```

```
ph = PasswordHasher()


hash = ph.hash("MySecurePassword123")
print(hash)


# Verify
ph.verify(hash, "MySecurePassword123")
```

---

10. Common Authentication Mistakes (Interview Gold)

❌ Storing plain text passwords
❌ Using fast hashes (MD5, SHA-1)
❌ No rate limiting on login
❌ No password policy
❌ Logging passwords
