

Question 1: Toy Dictionary Attack Open `guessing_demo.py` and read through the code before running it.

The program reads possible passwords from `wordlist.txt`, hashes each guess, and compares the result against an instructor-provided target hash.

1.1. Before running the program, look at the loop that tests each password guess.

What does the program need to compare against `TARGET_HASH`?

1.2. Complete the `hash_guess()` function.

Your function should:

- convert the password guess to bytes,
- hash it using SHA-256, and
- return the hexadecimal representation of the hash.

Then run:

```
python3 guessing_demo.py
```

What password was recovered?

1.3. How many guesses were required?

1.4. Did the program “decrypt” the password hash? Briefly explain what the program actually did.

1.5. Open `wordlist.txt` and move the recovered password earlier or later in the file. Run the program again.

What changes? What does this tell you about password-guessing strategy?

Question 2: Salts Open `salt_demo.py` and read through the code before running it.

Alice and Bob use the **same password**, but the program uses a different salt for each user.

2.1. Before running the program, predict: do you expect Alice and Bob’s stored hashes to be the same or different? Why?

2.2. Complete the `salted_hash()` function.

Your function should:

- convert the password to bytes,
- combine the salt with the password,
- hash the result using SHA-256, and
- return the hexadecimal representation of the hash.

Then run:

```
python3 salt_demo.py
```

Were the two hashes the same or different?

2.3. The password is identical in both cases. Why are the stored hashes different?

2.4. If an attacker is guessing passwords offline, does adding a salt reduce the number of guesses needed to guess a weak password? Explain.

2.5. What work does a unique salt force an offline attacker to repeat?

Question 3: Cost Per Guess Open `cost_demo.py` and read through the code.

The program uses PBKDF2 to repeat cryptographic work many times for each password guess. You will compare several different iteration counts.

3.1. Complete the `hash_password()` function using:

```
hashlib.pbkdf2_hmac(...)
```

Use the provided:

- SHA-256 hash function,
- password,
- salt, and
- number of iterations.

Then run:

```
python3 cost_demo.py
```

3.2. As the number of iterations increases, what happens to the time required for one password guess?

3.3. What happens to the number of guesses that can be attempted per second?

3.4. Look at the estimated time required for 1,000,000 guesses.

What general trend do you observe as the iteration count increases?

3.5. Why might a defender intentionally want password hashing to be slower?

Question 4: Online vs. Offline Defenses Recall:

- **Online guessing:** the attacker submits guesses to the real login service.
- **Offline guessing:** the attacker has obtained stored password hashes and can test guesses locally.

For each defense below, indicate whether it primarily helps against online guessing, offline guessing, or both.

Defense	Online	Offline	Both
Rate limiting / login delays	☐	☐	☐
Unique salts	☐	☐	☐
Slow password hashing	☐	☐	☐
Strong, unique passwords	☐	☐	☐
Multi-factor authentication (MFA)	☐	☐	☐

4.1. Choose one defense above and explain why it helps in that threat model (online/offline).