Question 1: Password entropy and salting

1(a) Password-only entropy

Password scheme

- 3 lowercase letters: 26^3 possibilities
- 2 digits: 10^2 possibilities
- 1 symbol from {+, -, <, >, =}: 5 possibilities
- Fixed order: letters, digits, symbol

Total passwords
N = (26^3) x (10^2) x 5 = 17576 x 100 x 5 = 8788000

Total entropy (bits)
Hpwd=log2N=log2(8,788,000)≈23.07 bits

1(b) Total entropy with salt

Salt
A random 12-bit salt is drawn uniformly and stored with the hash:

hash=SHA3-256(salt ∥ password)

with a 12-bit salt space of (2^12). The salt is independent of the password

Combined search space (salt + password)

Ntotal=(2^12)×8,788,000
Htotal=Hpwd+12≈23.07+12=35.07 bits

Reasoning

The salt does not make any single password harder to guess once the salt is known, but it multiplies the overall space of salt–password pairs that an attacker must cover if they try to reuse precomputed hashes across users.

1(c) The role of the salt

i) "Adding salt does not improve the strength of a given password against targeted brute force attacks."

For a targeted account, the attacker knows the user's salt as it is stored alongside the hash. They must still try each candidate password once with that salt. The number of guesses needed for that user is unchanged: about 8.79 million candidates, so the per-account brute-force cost is the same with or without salt.

ii) "Adding salt is essential to protect against large-scale attacks using rainbow tables and precomputed hashes."

Salts stop reuse of precomputed tables. Without a salt, one rainbow table works for everyone. With a 12-bit salt, you would need separate tables for 212=40962^{12}=4096212=4096 salt values, which is already 4096× more storage and prep work. Salts also prevent identical passwords across users from producing identical hashes, which blocks easy cross-user correlation