

Topic 6: Security, Privacy and Data Integrity

Storing passwords safely, measuring password strength, and the difference between a checksum and a signature.

Optional stretch material. This goes beyond the 9618 syllabus, so you will not be asked about it directly in the exam. Its purpose is to deepen ideas you already know. **Builds on:** 6.1 Data security, 6.2 Data integrity. Try the challenge questions on the last pages before you read the answers.

Storing passwords properly

A system should never store passwords as plain text: one stolen database would expose everyone. Instead it stores a **hash**, a one-way fingerprint, and at login hashes what you type and compares. But plain hashing has two weaknesses: identical passwords give identical hashes, and attackers can pre-compute huge **rainbow tables** of common passwords.

The defence is a **salt**: a random value generated for each user, stored next to the hash and mixed in before hashing.

User	Salt	Stored hash
amy	k3Z9	hash("k3Z9" + amy's password)
ben	Q81x	hash("Q81x" + ben's password)

Even if Amy and Ben pick the same password, their stored hashes differ, and pre-computed tables become useless. Good systems also use deliberately **slow** hash functions (bcrypt, scrypt, Argon2); fast hashes like MD5 let attackers try billions of guesses a second.

How strong is a password? Entropy

If a password is chosen uniformly from an alphabet of N symbols and has length L , there are N^L possibilities, which is **$L \times \log_2(N)$ bits of entropy**. Each extra bit doubles the attacker's work.

Password type	Possibilities	Entropy	Time at 10^{10} guesses/s
8 lowercase letters	$26^8 \approx 2.1 \times 10^{11}$	37.6 bits	About 21 seconds
4 random words (7776-word list)	$7776^4 \approx 3.7 \times 10^{15}$	51.7 bits	About 4 days
12 random printable characters	$94^{12} \approx 4.8 \times 10^{23}$	78.7 bits	About 1.5 million years

Length beats complexity: a long random passphrase can be both stronger and easier to remember than a short jumble. The calculation only holds for *random* choices; "P@ssw0rd!" is predictable and falls to a dictionary attack in seconds.

Integrity is not authenticity

Error-detection methods form a ladder of strength: **parity bit** (detects odd numbers of flipped bits), **checksum** (sums the data), **CRC** (polynomial division; catches burst errors; used in Ethernet and ZIP), and **cryptographic hash** such as SHA-256 (any tiny change gives a completely different output, and engineering a collision is infeasible).

Even so, a hash only proves *the data hasn't changed since the hash was made*. If an attacker can alter the file **and** the hash that sits beside it, the check passes. To prove *who* produced the data you need a **digital signature** (topic 17), which ties the hash to a private key.

Challenge questions

1. A password is 10 characters chosen at random from lowercase letters and digits (36 symbols). Calculate its entropy and the time to try every possibility at 10^{10} guesses per second.
2. A stolen database contains unsalted SHA-256 hashes. Two accounts have the same hash. What can the attacker conclude, and how would salting change this?
3. A website prints the SHA-256 hash of its installer beside the download link. What does a matching hash prove, and what does it not prove?

Worked answers

Only look once you have had a proper go at each question.

1. $10 \times \log_2(36) = 10 \times 5.17 \approx \mathbf{51.7 \text{ bits}}$.
 $36^{10} \approx 3.66 \times 10^{15}$ possibilities; $\div 10^{10}$ per second $\approx 3.66 \times 10^5 \text{ s} \approx \mathbf{4.2 \text{ days}}$ to try them all (about 2 days on average).
2. The two users have the same password, so cracking one cracks both, and a rainbow table lookup works on every account at once. With a unique salt per user the stored hashes differ even for the same password, so the attacker must attack each account separately.
3. It proves your download is identical to the file that was hashed (no corruption in transit). It does not prove the file is genuine: an attacker who controls the page could replace both the installer and the hash. To trust it you need the hash from a separate trusted channel or a digital signature.