

# Identification and Authentication

COMP2700 Cyber Security Foundations

# Outline

- Basis of authentication:
  - what you know, what you possess, who you are.
- Password-related techniques
- Attacks on passwords and defense mechanisms
- Authentication tokens and biometrics

# Authentication

- Entity authentication is a *process* whereby one party (*verifier*) is assured of the identity of a second party (*claimant*) in a protocol.
- Two reasons for authenticating a user:
  - The user identity is a parameter in access control decisions.
  - The user identity is recorded when logging security relevant events in an audit trail.

# Basis of authentication

- *What you know*
  - Authentication is done by exhibiting knowledge of certain secrets.
  - Examples: passwords, Personal Identification Numbers (PINs), private/secret keys.
- *What you have*
  - Magnetic/smart cards, hardware tokens (password generators).
  - Typically combined with passwords, to form a *two-factor authentication*.
- *Who you are*
  - Physical characteristics, e.g., fingerprints, voice, retinal patterns.
  - Behavioral characteristics, e.g., handwritten signatures, keystroke dynamics.

# Authentication protocols

- Weak Authentication (this lecture):
  - Password-based
  - Unilateral: one entity (claimant) proves its identity to the verifier.
- Strong authentication:
  - involves mutual authentication; both parties take both the roles of claimant and verifier:
  - Challenge-response protocols: sequence of steps to prove knowledge of shared secrets.
  - To be covered the second part of this course.

# Password-related techniques

- Password storage:
  - plaintext or “encrypted”.
- Password policies:
  - what rules need to be imposed on the selection of passwords by users, number of failed attempts, etc.
- “Salting” of passwords.
- Alternative forms of passwords: passphrases, one-time passwords, visual passwords.

# One-way functions

- Password storage security relies on a cryptographic construct called *one-way function*.
- A one-way function  $f$  is a function that is relatively **easy to compute** but **hard to reverse**.
  - Given an input  $x$  it is easy to compute  $f(x)$ , but given an output  $y$  it is hard to find  $x$  so that  $y = f(x)$
- Cryptographic hash functions are an example of one-way function:
  - A hash function  $f$  takes an input  $x$  of *arbitrary length*, and produces an output  $f(x)$  of *fixed length*.

# Properties of hash functions

Suppose  $H$  is a hash function. We say  $H$  satisfies:

- *Pre-image resistant* if given a hash value  $y$ , it is computationally infeasible to find  $x$  such that  $H(x) = y$ .
- *Collision resistant* if it is computationally infeasible to find a pair  $(x, y)$  such that  $x \neq y$  and  $H(x) = H(y)$ .



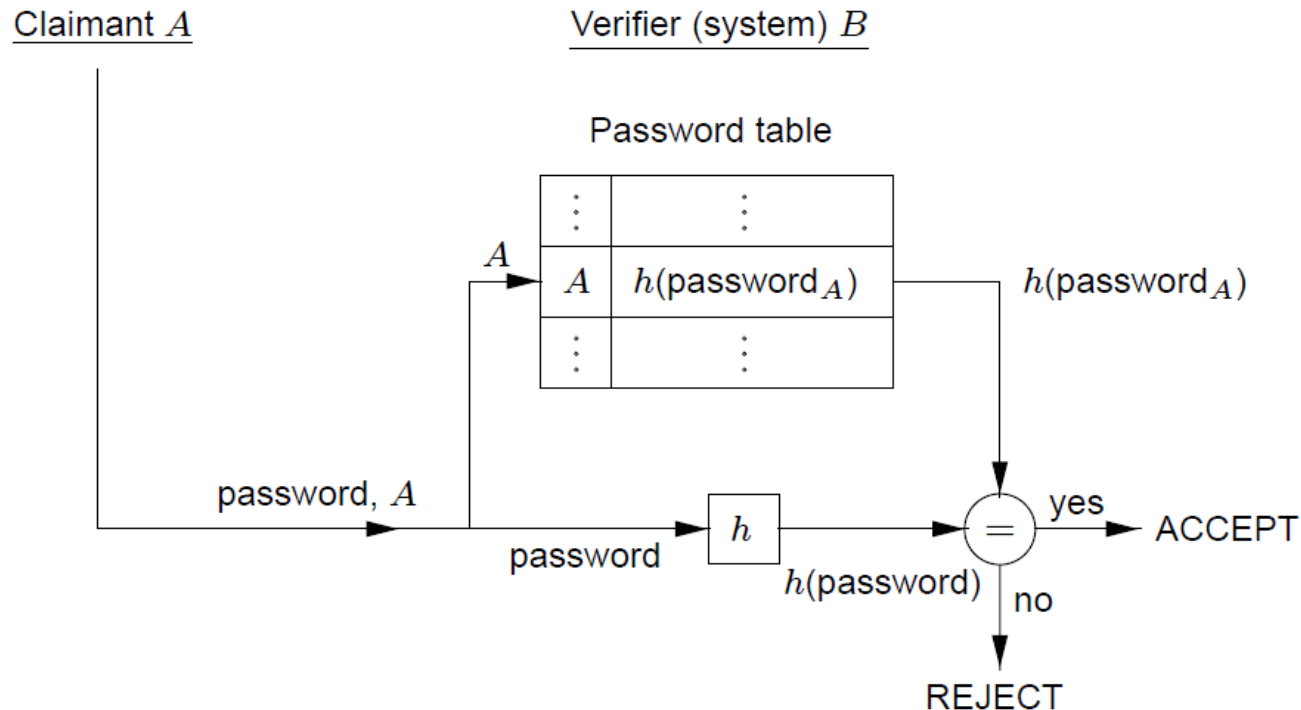
# Password storage

- Plaintext:
  - passwords stored in plaintext.
  - Claimant's password is checked against the database of passwords.
  - No protection against insider (system admin) or an attacker who gains access to the system.
- Hashed/encrypted passwords:
  - Passwords are encrypted, or hashed, and only the encrypted/hashed passwords are stored.
  - Claimant's password is hashed/encrypted, and checked against the database of hashed/encrypted password.
  - Some degree of protection against password recovery by insider/attacker.

# Password Storage

- In operating systems, password hashes are stored in a password file.
  - In Unix, this is `/etc/passwd`, but in modern Unix/Linux systems it is in the *shadow* file in `/etc/shadow`.
  - In Windows system, passwords are stored in Security Accounts Manager (SAM) file (`%windir%\system32\config\SAM`).
- At the application levels, passwords may be held temporarily in intermediate storage locations like buffers, caches, or a web page.
- The management of these storage locations is normally beyond the control of the user.

# Hashed password verification



Source: Menezes et al. *Handbook of Applied Cryptography*.

Notice that the verifier does not store the passwords, only their hashes.

# Attacks on passwords

- Offline guessing attacks
  - Exhaustive attacks
  - Intelligent attacks:
    - Dictionary attacks
- ‘Phishing’ and spoofing
  - Mainly a 'social engineering' attack; tricking user to reveal their passwords voluntarily.
  - Not discussed in this lecture.

# Offline Guessing attack

- Offline guessing attack is an attack where the attacker obtains the hashed passwords and attempts to guess the passwords.
- This is a plausible threat, due to:
  - many incidents of stolen (hashed) passwords as a consequence of hacks on servers.
  - usage of the same passwords across different accounts; so compromise of a password for one account affects other accounts.

# Password-related data breach

Some examples of identity theft:

- [Data Viper breach \(2020\):](#)
  - Security company that collects password databases. Close to 15 billion usernames, passwords and other information exposed.
- [PageUp data breach \(2018\):](#)
  - HR company that manages employee data for many companies; 2 million users.
- [Russian hackers stole 1.2 billion passwords \(2014\)](#)
- [Adobe Inc. data breach \(2013\)](#)
  - 38 million passwords stolen.

# Brute force attack

- Brute force guessing attack against passwords tries to guess password by enumerating all passwords and their hashes in sequence, and check whether they match the target hashes.
- A measure against brute force attack is to increase the space of possible passwords, e.g., longer passwords, allowing more varieties of symbols (alphabets, numerals, signs).

# Password entropy

- A measure of the strength of passwords against brute-force attack.
- Let  $X$  be a random variable which takes on a finite set of values  $x_1, \dots, x_n$ , with probability  $Pr(X = x_i) = p_i$ , where  $0 \leq p_i \leq 1$  for each  $1 \leq i \leq n$ , and  $\sum_{i=1}^n p_i = 1$ .
- The entropy of  $X$  is defined to be:

$$H(X) = \sum_{i=1}^n p_i \log_2\left(\frac{1}{p_i}\right)$$

where, by convention,  $p_i \log_2\left(\frac{1}{p_i}\right) = 0$  if  $p_i = 0$ .



# Password entropy (Example)

- Suppose the set of passwords  $X$  is drawn from any 5 character strings, each character ranges from 'a' to 'z'.
  - So the size of  $X$  is  $26^5$ .
- Assume that each password is equally likely to occur, then every element of has equal probability to occur, i.e.,
$$Pr(X = p) = \frac{1}{26^5} \text{ for every password } p.$$

- Let  $n = 26^5$ . The entropy of  $X$  is

$$H(X) = \sum_{i=1}^n \frac{1}{n} \log_2 n = \log_2 n = \log_2 26^5 \approx 23.5$$

# Password Entropy

$\rightarrow c$ $\downarrow n$	26 (lowercase)	36 (lowercase alphanumeric)	62 (mixed case alphanumeric)	95 (keyboard characters)
5	23.5	25.9	29.8	32.9
6	28.2	31.0	35.7	39.4
7	32.9	36.2	41.7	46.0
8	37.6	41.4	47.6	52.6
9	42.3	46.5	53.6	59.1
10	47.0	51.7	59.5	65.7

**Table 10.1:** Bitsize of password space for various character combinations. The number of  $n$ -character passwords, given  $c$  choices per character, is  $c^n$ . The table gives the base-2 logarithm of this number of possible passwords.

Source: Menezes et al. *Handbook of Applied Cryptography*.

# Dictionary attack

- Choosing passwords with high entropy prevents brute-force attack.
- However, hashed passwords, especially for human-generated passwords, are still vulnerable to *dictionary attack*.
- This exploits weakness in human-chosen passwords, which tend to derive from words in natural languages.

# Some commonly used passwords

Passwords	Frequency
123456	1,911,938
123456789	446,162
password	345,834
adobe123	211,659
12345678	201,580
qwerty	130,832
1234567	124,253

Source: <http://stricture-group.com/files/adobe-top100.txt>

- Data from stolen encrypted passwords of Adobe Inc. (2013)
- Contains more than 38 million passwords (with password hints in plaintext).

# Pre-computed hash table

- A strategy for cracking hashed passwords is to pre-compute a hash table, containing pairs of passwords and their hashes.
- If we have  $k$  password candidates, each password requires  $m$  bits to store, and each hash has  $n$  bits, then we have a table of size  $k \times m \times n$  bits.
- This may not be practical if  $k$  is large.

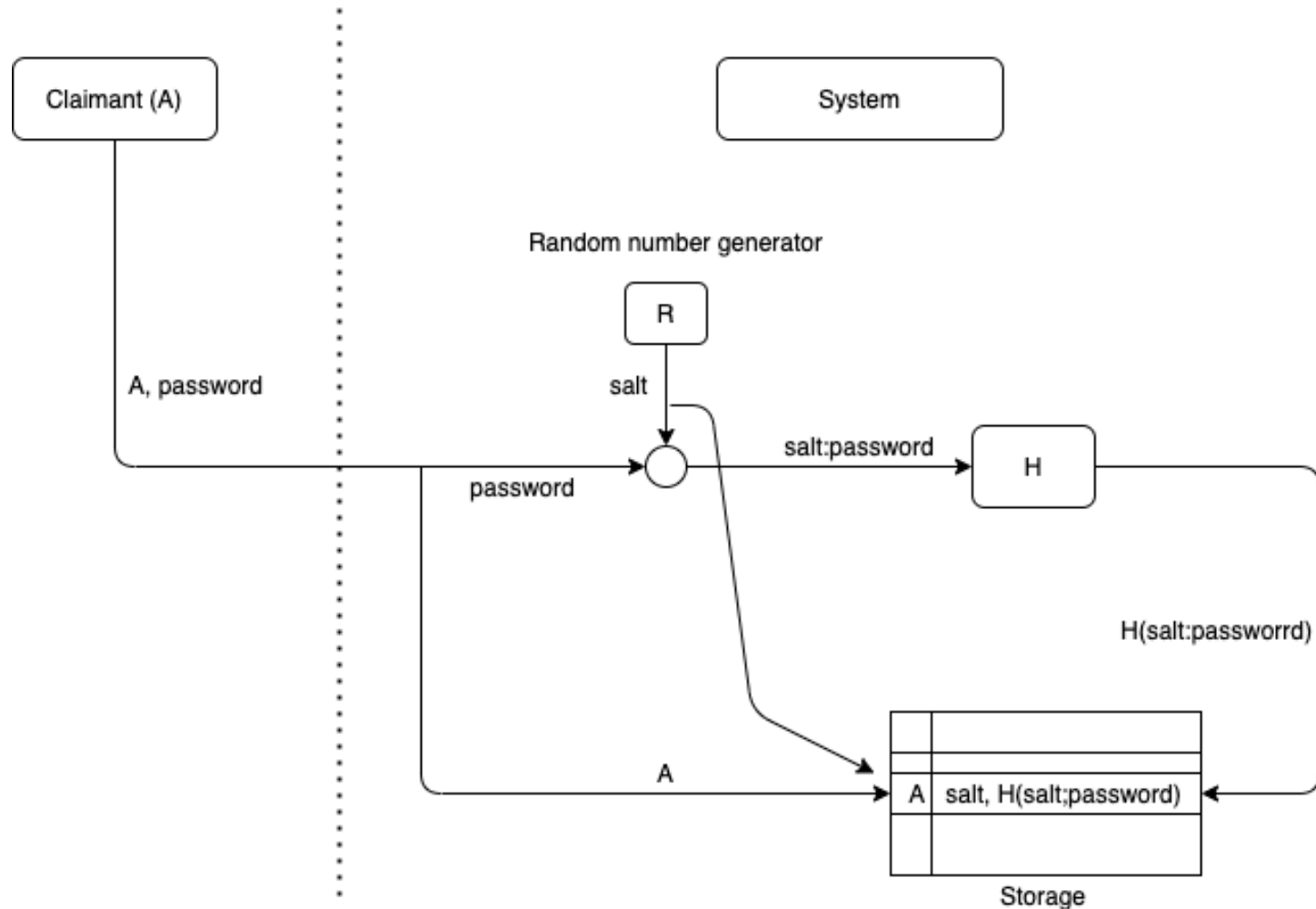
# Pre-computed hash table

- Hash tables are often represented using a data structure called *rainbow table*.
- Not all hashes are stored; some will be computed from the stored hashes.
- Not all hashes are represented.
- Tradeoff between space requirement and query time.

# Password salting

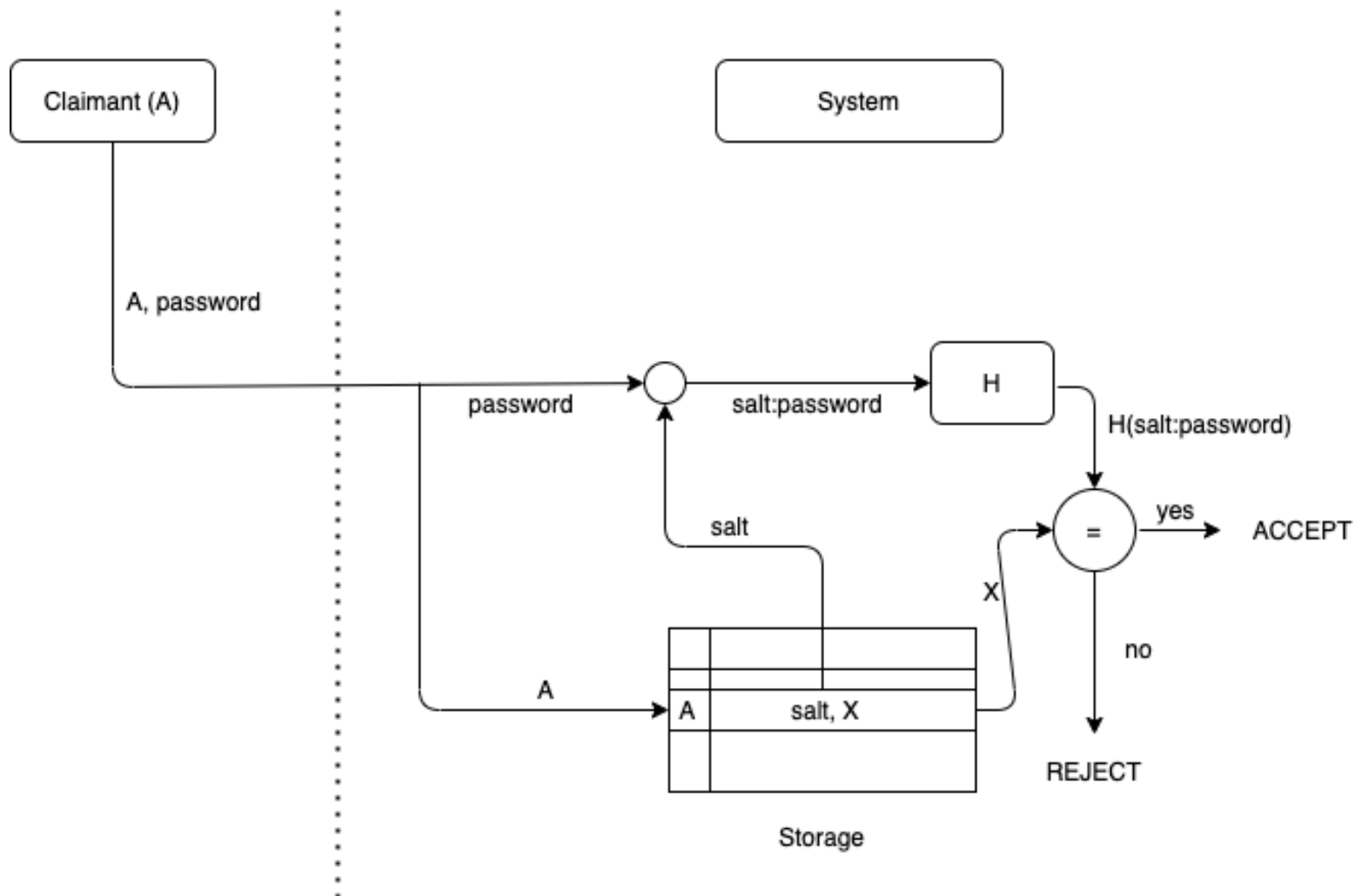
- To reduce the effectiveness of offline attacks using pre-computed hashes, a *salt* is added to a password *before* applying the hash function.
- A salt is just a random string.
- Each password has its own salt.
- The salt value is stored (in ***plaintext***) along with the hash of salt together with the password.
- For a salt of  $n$ -bit, the attacker needs to pre-compute  $2^n$  of hashes for *the same password*.

# Password salting (user registration)





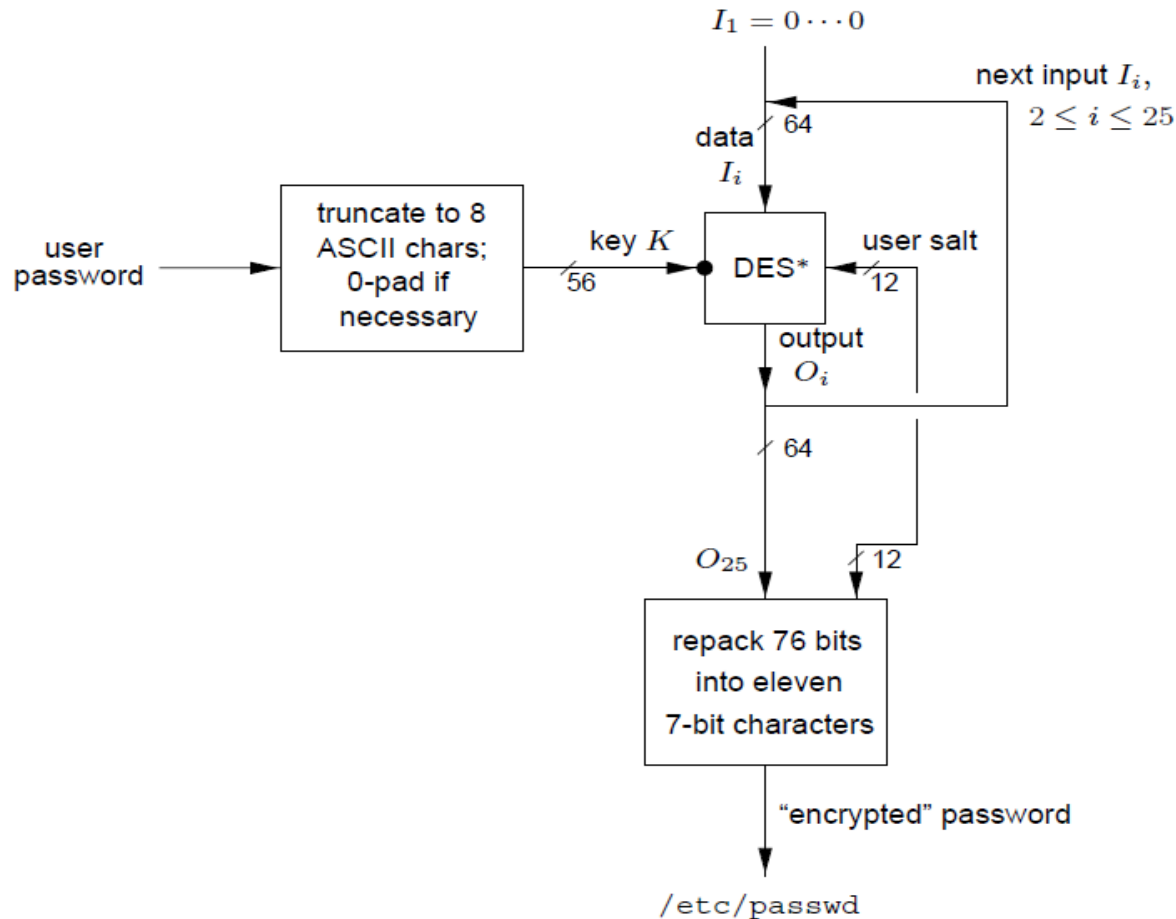
# Password salting (user verification)



## Example: crypt (UNIX)

- Password hashing in Unix/Linux systems is done via the crypt algorithm.
  - The early version is based on a variant of the block cipher DES.
- Features of DES-based crypt:
  - It uses a random 12-bit salt for each password.
  - It accepts passwords of 8 character long (padded with 0 if the length < 8), and produces 64 bit output.
  - User's password is used to derive a 56-bit DES key: take 7 bit from each character in the password.
  - The key is used to encrypt a 64-bit constant 0.
  - DES encryption is iterated 25 times, feeding the output of one iteration to the next (output feedback mode).

# DES-based crypt



Source: Menezes et al. *Handbook of Applied Cryptography*.

## Example: Linux crypt

- DES-based crypt has been phased out in most Unix-based systems (Linux, FreeBSD), due to its cryptographic weakness.
- Modern implementation of crypt supports other, more secure, hash functions: MD5, Blowfish, SHA-256 and SHA-512.
- The overall structure of crypt(3) is similar to DES-crypt: it involves iterations of applications of hash functions (> 1000 iterations).

# Example: Linux crypt

Password hash in Linux has the following format:

`$<id>$<salt>$<hash>`

where

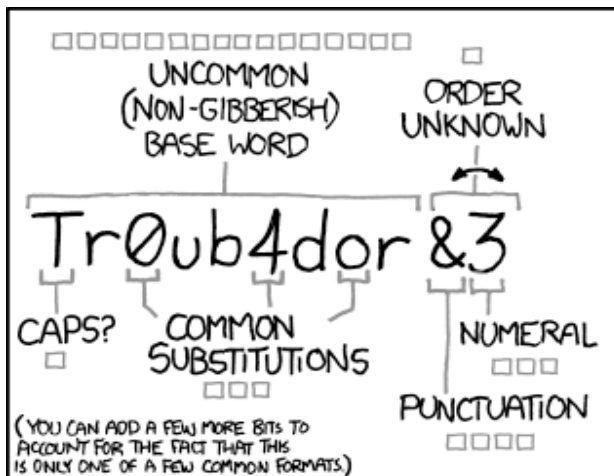
- `<id>` : code for the hash algorithm used, i.e., 1 (MD5), 2a (Blowfish), 5 (SHA-256) and 6 (SHA-512)
- `<salt>`: the salt value, 8 ASCII characters
- `<hash>`: the actual hash value, 22 ASCII characters

## Example: Linux crypt

- Both the salt and the hash are ASCII representations of bit strings in a *base64 encoding*.
- Base64 represents each block of 6 bits using the alphabets: “.”, “/”, “0” to “9”, “A” to “Z” and “a” to “z”.
  - For example, binary string “0000010000011” is represented as “/0”.
- Example: the password ‘123456’ and salt ‘xxxxxxxx’ are hashed (using MD5-based crypt) to  
**\$1\$xxxxxxxx\$8yHoNX3W.aK293K.aT4uJ/**

# Password Security

- Should users be forced to use long passwords, mixing upper and lower case characters and numerical symbols, and changed repeatedly?
- Some considerations:
  - Users may have difficulty memorizing complex passwords.
  - Users may have difficulty dealing with frequent password changes.
  - Users may find ways of re-using their favourite password.



~28 BITS OF ENTROPY

$2^{28} = 3 \text{ DAYS AT } 1000 \text{ GUESSES/SEC}$


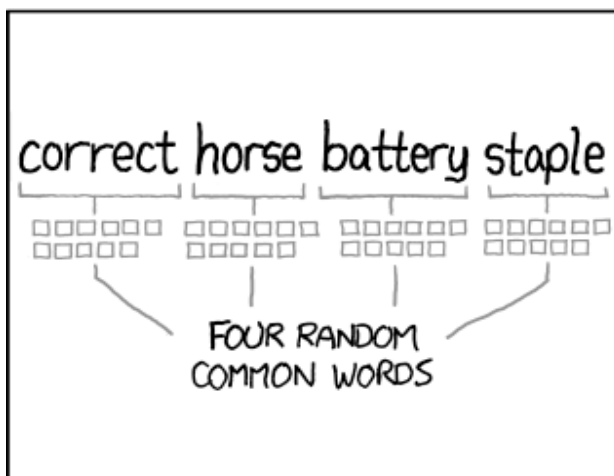
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)

DIFFICULTY TO GUESS:  
**EASY**

WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE 0s WAS A ZERO?

AND THERE WAS SOME SYMBOL...

DIFFICULTY TO REMEMBER:  
**HARD**

~44 BITS OF ENTROPY


$2^{44} = 550 \text{ YEARS AT } 1000 \text{ GUESSES/SEC}$

DIFFICULTY TO GUESS:  
**HARD**

THAT'S A BATTERY STAPLE.

CORRECT!

DIFFICULTY TO REMEMBER:  
YOU'VE ALREADY MEMORIZED IT



THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.



# Password policies

- Set a password: if there is no password for a user account, the attacker does not even have to guess it.
- Change default passwords: often passwords for system accounts have a default value like “manager”.
  - Default passwords help field engineers installing the system; if left unchanged, it is easy for an attacker to break in.
  - Would it then be better to do without default passwords?
- Avoid guessable passwords:
  - Prescribe a minimal password length.
  - Password format: mix upper and lower case, include numerical and other non-alphabetical symbols.
  - Today on-line dictionaries for almost every language exist.

# Password policies

- **Password ageing**
  - set an expiry dates for passwords to force users to change passwords regularly.
  - Prevent users from reverting to old passwords, e.g. keep a list of the last ten passwords used.
- **Limit login attempts**
  - monitor unsuccessful login attempts and react by locking the user account (completely or for a given time interval) to prevent or discourage further attempts.
- **Inform user:**
  - after successful login, display time of last login and the number of failed login attempts since, to warn the user about recently attempted attacks.

# Alternative forms of passwords

- Passphrase: user enters sentences or long phrases that are easy to remember, and the system applies a hash function to compute the (fixed-size) actual passwords.
- Visual drawing patterns (on touch interface). Used in, e.g., used in Android.
- Picture passwords: select objects in pictures and patterns. Used in Windows 8.
- One-time passwords.
- Single sign-on.

# One-time password

- The *one-time password (OTP)* scheme attempts to address a key weakness in the password-based scheme: reuse of stolen passwords.
- The idea is to generate a list of passwords, and each password is *used only once*.
- Different ways OTPs can be generated:
  - Using a *hash chain*, e.g., Lamport's OTP.
  - Using time-synchronised OTP generation (from an initial secret).
  - Using a random challenge and an initial secret.
- We'll cover more details of OTP when we discuss hash functions later in the course.

# Security tokens (what you have)

- Security tokens: offline devices that generates sequences of (seemingly random) numbers.
- The number generation algorithm is deterministic, but is dependent on a 'seed' value and a 'challenge'.
- Effectively the token implements an OTP algorithm.
- Example: RSA SecurID.



Source: wikipedia

# Security tokens (what you have)

- Seed values are secret, and shared between the verifier and the claimant. Challenges provided by verifier each time authentication is requested.
- Seed values are stored in tamper-proof chips inside the tokens, to prevent the tokens being cloned.
- Hardware tokens have been increasingly replaced by *soft tokens*.
  - E.g., Google's Authenticator app for mobile devices.
  - Rely on the secure storage (e.g., [Apple Secure Enclave](#) or [Google StrongBox](#)) of the devices to protect the seed values.

# Biometrics (who you are)

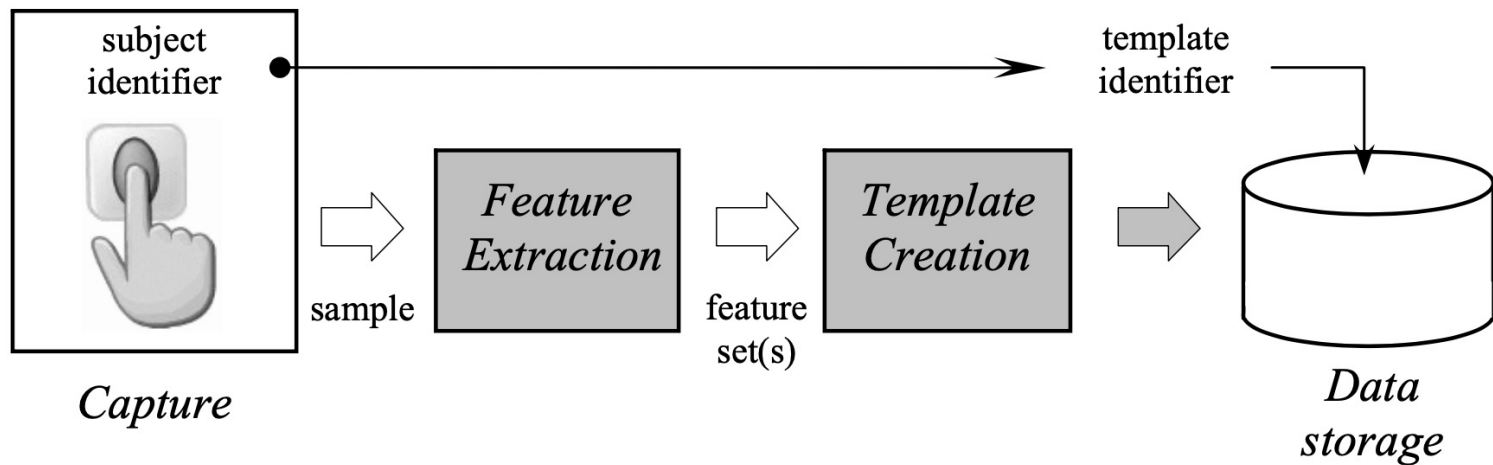
- Biometric schemes use unique physical characteristics (**traits**, **features**) of a person such as face, finger prints, iris patterns.
- Enrolment: reference **sample** of the user's biometric is **acquired** at a biometric reader.
- **Features** are derived from the sample.
  - E.g., Fingerprint minutiae: end points of ridges, bifurcation points, core, delta, loops, whorls, ...
- When the user logs on, a new reading of the biometric is taken; features are compared against the reference features.

# Verification & Identification

- Every biometric system must include an **enrolment process**, where users' biometrics are acquired and stored.
- Biometrics are used for two purposes:
  - **Verification**: **1:1** comparison checks whether there is a match for a given user.
  - **Identification**: **1:n** comparison tries to identify the user from a database of **n** persons.



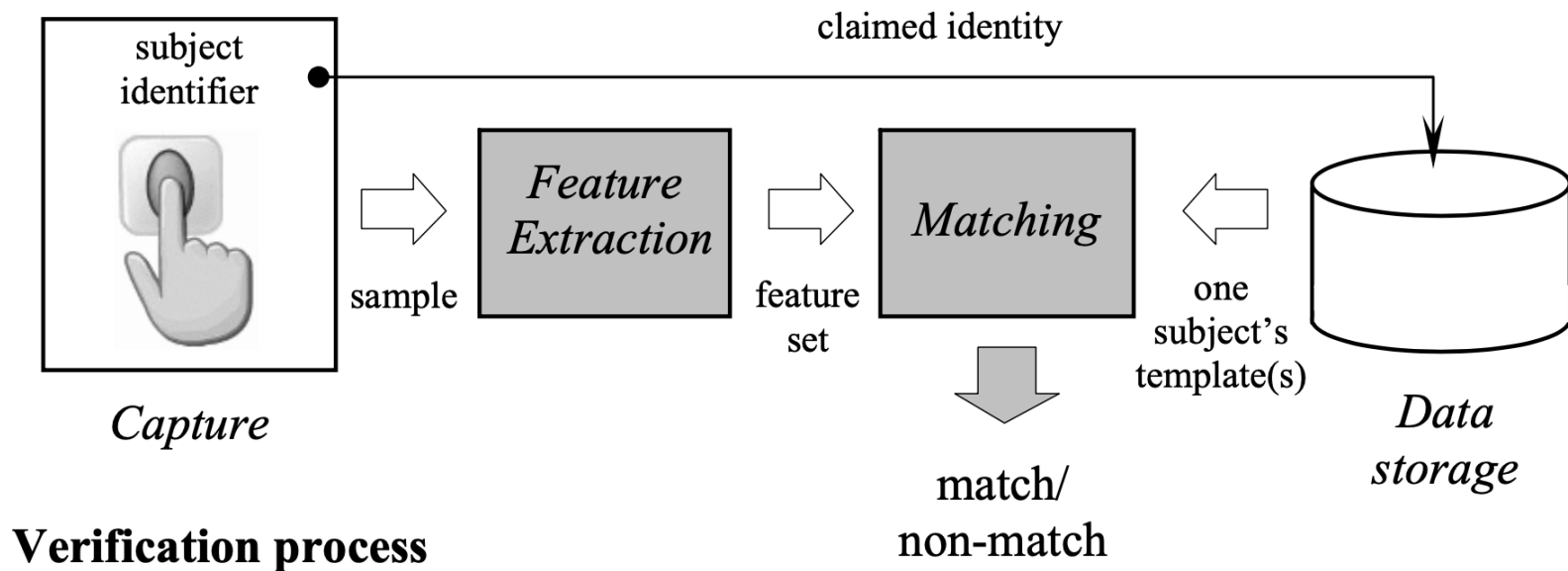
# Biometrics: enrollment process



## Enrollment process

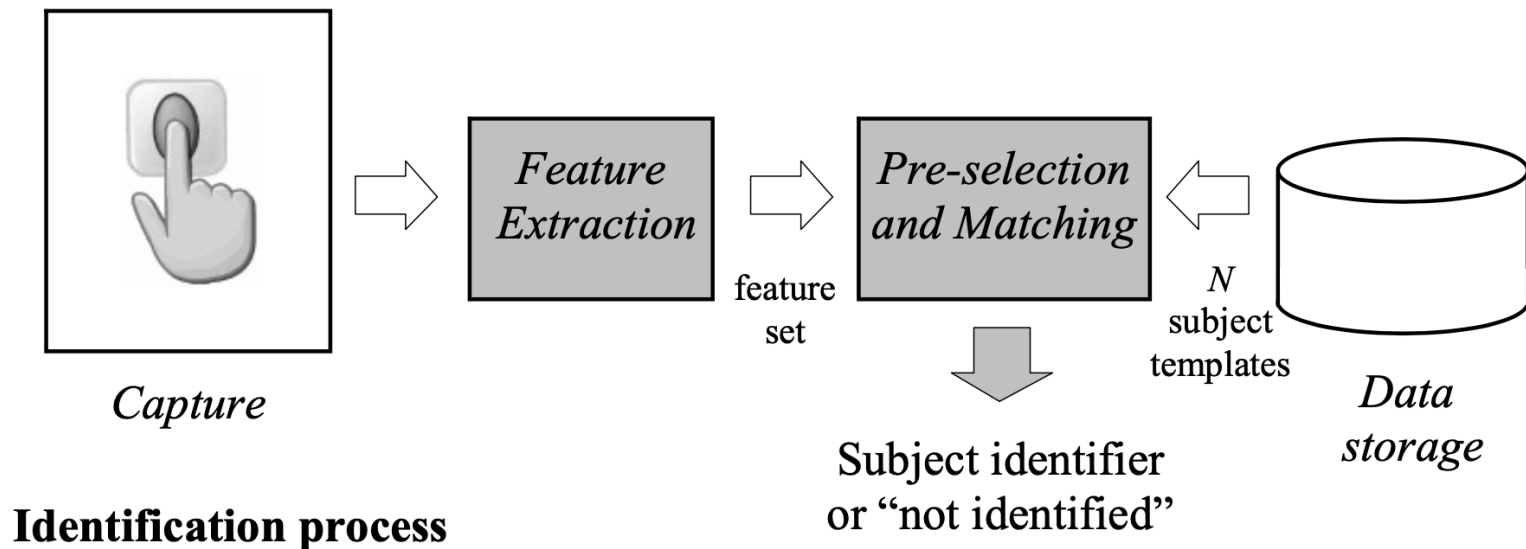
Source: Handbook of Fingerprint Recognition

# Biometrics: verification process



Source: Handbook of Fingerprint Recognition

# Biometrics: identification process



Source: Handbook of Fingerprint Recognition

# Failure Rates

- Measure similarity between reference features and current features.
- User is accepted if match is above a predefined **matching threshold**.
  - A matching threshold is a value between 0 (no matching features) and 1 (all features match).
- **New issue: false positives and false negatives**
- Accept wrong user (**false positive**): security problem.
- Reject legitimate user (**false negative**): usability problem.

# Performance of matching algorithms

- Based on a (given) databases of biometric samples.
- Measures performance of the algorithms extracting and comparing biometric features.
- **False match rate (FMR):**

$$FMR = \frac{\text{number of successful false matches}}{\text{number of attempted false matches}}$$

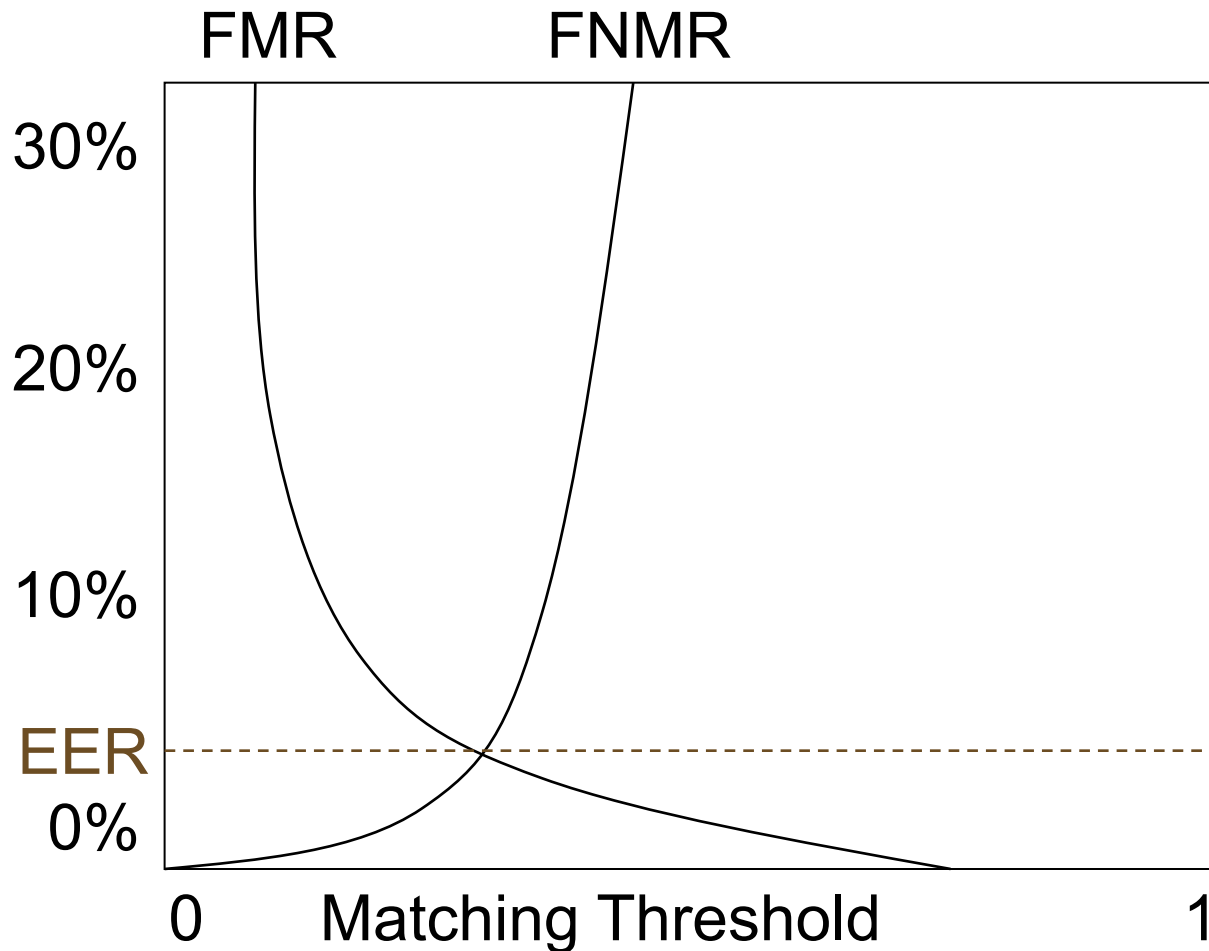
- **False non-match rate (FNMR):**

$$FNMR = \frac{\text{number of rejected genuine matches}}{\text{number of attempted genuine matches}}$$

# Equal-error Rate

- By setting the matching threshold, we can trade off a lower **false match rate** against a higher **false non-match rate**, and vice versa.
- Finding the right balance between those two errors depends on the application.
- **Equal error rate** (EER): given by the threshold value where FMR and FNMR are equal.

# FMR, FNMR, EER



# Scenario Analysis

Records error rates in actual field trials; measures performance of fingerprint reader (hardware and software) capturing templates at log-in time.

- **Failure-to-capture rate (FTC)**: frequency of failing to capture a sample.
- **Failure-to-extract rate (FTX)**: frequency of failing to extract a feature from a sample.



# Scenario Analysis

- **Failure-to-acquire rate:** frequency of failing to acquire a biometric feature:  
$$FTA = FTC + FTX \cdot (1 - FTC)$$
- **False accept rate** for the entire biometric scheme:  
$$FAR = FMR \cdot (1 - FTA).$$
- **False reject rate:**  
$$FRR = FTA + FNMR \cdot (1 - FTA).$$

# Identification with Biometric

- Care must be taken in using biometric for identification.
  - Recall that identification is a 1:n matching problem
- **False positive identification rate** for a database with  $n$  persons:
$$\text{FPIR} = (1 - \text{FTA}) \cdot (1 - (1 - \text{FMR})^n).$$
- Error rate increases as the database size increases.

## Example: Madrid train bombing (2004)

- Coordinated bombings against a commuter train system in Madrid (Spain), on the morning of 11 March 2004 – 192 killed, 2000 injured.
- A fingerprint found in the Madrid train bombing was compared against a database of 530 million entries.
- A match was found and linked by four experts with 100% confidence to a US citizen (Brandon Mayfield).
- Mayfield was innocent; he had not even left the country.

# Issues with biometric

- Fingerprints, and **biometric traits** in general, may be unique but they **are no secrets**.
  - [Hackers show how to lift fingerprints](#) from iPhone 5s and Samsung S5 to defeat fingerprint authentication on the phones (2013).
  - [Iris recognition in Samsung S8 defeated by hackers](#) (2016).
- Biometrics are harder to change (unlike passwords).
- Biometrics should not be used as the sole basis for authentication.

# Summary

- We have seen three forms of authentication: passwords (what you know), authentication tokens (what you have) and biometrics (who you are).
- When designing authentication systems, one needs to consider false positives as well as false negatives.
  - Recall that availability is an important aspect of security.
- Security mechanisms may fail: you need to implement measures to deal with such failures.

# References

Most of this presentation was prepared using the following sources:

- Menezes et al. *Handbook of Applied Cryptography*. Chapter 10.  
<http://cacr.uwaterloo.ca/hac/>
- Gollmann's "Computer Security", Wiley, 2011. Chapter 4.
- R. Anderson. Security Engineering. Chapter 15.  
<http://www.cl.cam.ac.uk/~rja14/book.html>

Further reading:

- ISO/IEC 19795-1:2006 standard – biometric performance testing and reporting.
- D. Maltoni, D. Maio, A. K. Jain, S. Prabhakar. *Handbook of Fingerprint Recognition*. Second edition. Springer 2009.