Module 3: User Authentication

Dr. Natarajan Meghanathan
Associate Professor of Computer Science
Jackson State University, Jackson, MS 39232
E-mail: natarajan.meghanathan@jsums.edu

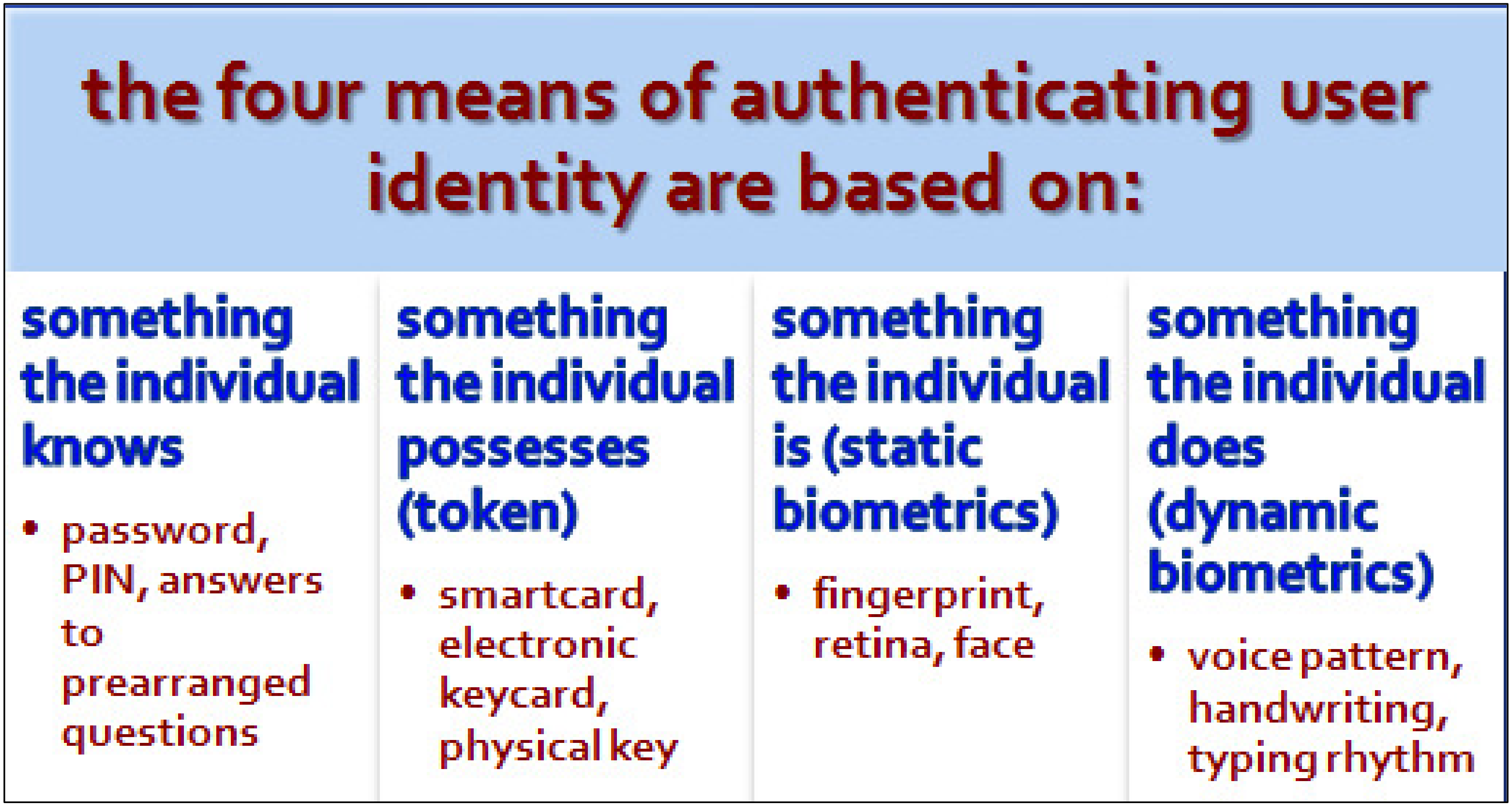
User Authentication

- User authentication: “The process of verifying an identity claimed by or for a system entity.” [RFC 2828]
- Basis for most types of access control and for user accountability.

- An authentication process consists of two steps:
  - Identification Step: Presenting an identifier to the security system [e.g., username]
  - Verification Step: Presenting or generating authentication information that corroborates the binding between the entity and the identifier [e.g., password]

- Message authentication: Making sure that a message was received unaltered and that the source is authentic.

User Authentication



the four means of authenticating user identity are based on:

something the individual knows	something the individual possesses (token)	something the individual is (static biometrics)	something the individual does (dynamic biometrics)
• password, PIN, answers to prearranged questions	• smartcard, electronic keycard, physical key	• fingerprint, retina, face	• voice pattern, handwriting, typing rhythm

Password-based Authentication

- User submits an identifier (ID) and password.
  - The system compares the user entered password to a previously stored password (or a hash) for that user ID, maintained in a system password file.
- The User ID:
  - Determines whether the user is authorized to gain access to the system.
  - Determines the user's privileges (super user, regular user, guest access, etc)
  - Is used in discretionary access control (delegating the access control permissions on user-owned objects to other users; e.g., setting file permissions to other users).
- Common Vulnerabilities of Password-based authentication
  - **Offline dictionary attack**: Comparing the hash values of passwords in the system file with those corresponding to commonly used passwords; and extracting the user ID/password
    - A pre-computed table of hashed passwords of possible passwords is called a **rainbow table**.
  - Guess password for a specific account through repeated login attempts (thru' info about user and system policies)
  - **Social Engineering attacks** – tricking users/admin to reveal password

Countermeasures to Mitigate Attacks on Password-based Authentication

- Controls to prevent unauthorized access to the password file
- Intrusion detection measures to identify a compromise
- Rapid reissuance of passwords should the password file be compromised
- Account lockouts if repeated attempts are noticed to login
- Training in and enforcement of password policies that makes passwords difficult to guess
  - E.g., minimum length of password, use from a character set, length of time to change the password
  - Choose passwords that are easy to remember; but difficult to guess
- Automatically logging out the workstation after a period of inactivity
- Policies against similar passwords on networked devices

Use of Salt to Mitigate Dictionary Attacks

- Motivation: The hash value for a particular password will be the same each time the password is passed as the only input to the cryptographic hashing algorithm.
- With salting, an additional input, a non-secret value (say *s*), could be passed along with the password *p* as the input to the hashing algorithm ***h***(*p*, *s*).
  - Even if two users choose the same password, if their salt values are different, their hashed password values will be different too.
- The tuple <*s*, ***h***(*p*, *s*)> is stored in the password file.
- If the salt value is publicly displayed in the password file, then an attacker has to compute N hash values for every word in the dictionary, where N is the number of users listed in the password file and the N values of the salt for each word in the dictionary will be those of these N users.
- If the salt values could be hidden from the user's view,
  - If the salt is represented using N_s bits, then for every word in the dictionary, the attacker has to now compute 2^{N_s} hash values, one for each possible value of the salt in the range [0,…, 2^{Ns}-1].
- Considering the above, it also becomes very difficult to find out whether a person with passwords on two or more systems has used the same password on all of them [Each system could have its own salt value].

Math Problem on Dictionary Attacks

- (a) If a UNIX system publicly displays the 12-bit salt values of each of its 2^{10} users along with the hash values of the 8-character long passwords, compute the average number of attempts needed for an attacker to launch a dictionary attack. Assume the cardinality of the character set of the passwords is 64 and the size of the dictionary of common passwords is 2^{20}. Also, assume that there is a 25% chance that a user password is chosen from the dictionary.
- (b) If the UNIX system, described in (a), does not publicly display the salt values, compute the compute the average number of attempts needed for an attacker to launch a dictionary attack.

Math Problem on Dictionary Attacks

(a) Salt values displayed publicly in the password file

If all passwords are from the dictionary.

Average # attempts needed = $(2^{10}) * (2^{20}) / 2$

$= 2^{29}$.

If no password is from the dictionary.

Average # attempts needed $= (2^{10}) * (64^{8}) / 2$

$= (2^{10}) * ((2^{6})^{8}) / 2$

$= (2^{10}) * (2^{48}) / 2$

$= 2^{57}$.

There is a 25% chance that a password can be from the dictionary. Hence, the average number of attempts is:

$0.25*2^{29} + 0.75*2^{57} = 2^{29}*(0.25 + 0.75*2^{28}) = 0.75 *2^{29}*2^{28}$.

$= 0.75*2^{57}$.

Math Problem on Dictionary Attacks

(b) Salt values NOT displayed publicly in password file

If all passwords are from the dictionary.

Average # attempts needed $= (2^{12}) * (2^{20}) / 2$

$= 2^{31}$.

If no password is from the dictionary.

Average # attempts needed $= (2^{12}) * (64^{8}) / 2$

$= (2^{12}) * ((2^{6})^{8}) / 2$

$= (2^{12}) * (2^{48}) / 2$

$= 2^{59}$.

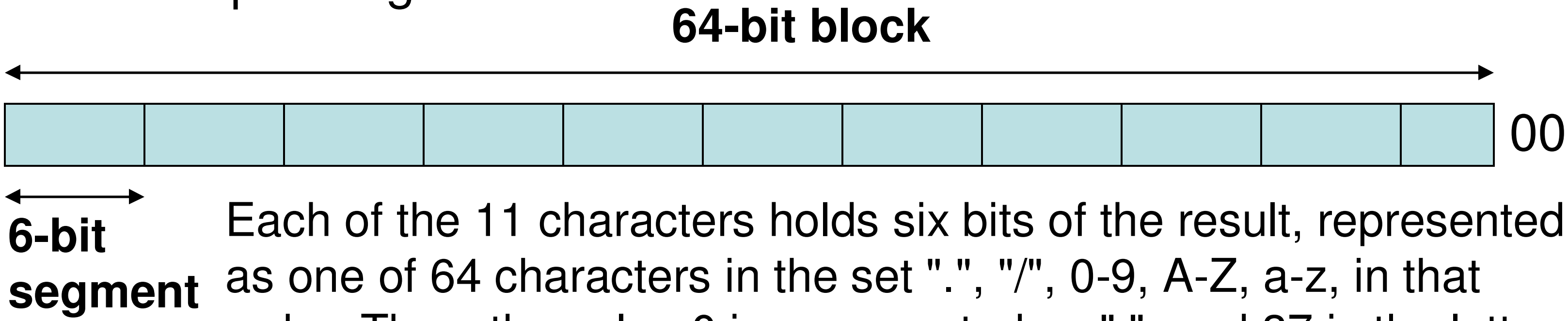
There is a 25% chance that a password can be from the dictionary. Hence, the average number of attempts is:

$0.25*2^{31} + 0.75*2^{59} = 2^{31}*(0.25 + 0.75*2^{28}) = 0.75 *2^{31}*2^{28}$.

$= 0.75*2^{59}$.

Standard UNIX Password Encryption

- The first 8 ASCII characters of a user's password are used. If your password is less than 8 characters in length, then 0 bits are padded to make it 8*7 = 56 bits in length.
- The 56 bits of the user's password is used as the DES key. A constant 64-bit block (consisting of all zero bits) is then encrypted via DES 25 times (the result of each encryption being used to feed the next round), using the 56-bit user password as the key.
- A 12-bit salt value drives the DES P- and S-box tables used.
- The resultant 64-bits is converted into a string of 11 printable ASCII characters, by encoding every six bits into a printable ASCII character and zero padding the 11th character.



Each of the 11 characters holds six bits of the result, represented as one of 64 characters in the set ".", "/", 0-9, A-Z, a-z, in that order. Thus, the value 0 is represented as ".", and 37 is the letter "Z"

.	**/**	**0**	**1**	**.....**	**9**	**A**	**B**	**.....**	**Z**	**a**	**b**	**....**	**Z**
0	**1**	**2**	**3**		**11**	**12**	**13**		**37**	**38**	**39**		**63**

Improved UNIX Implementations

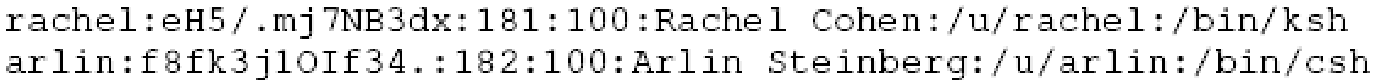
- The DES-based scheme is now regarded as inadequate to withstand dictionary attacks, as it could be broken in months even with a single processor machine available today.
- FreeBSD uses a hash function that is based on MD5
  - salt of up to 48-bits
  - password length is unlimited
  - produces 128-bit hash
  - uses an inner loop with 1000 iterations to achieve slowdown
    - Intentional to slowdown the dictionary attacks
- OpenBSD uses Blowfish block cipher based hash algorithm called Bcrypt
  - most secure version of Unix hash/salt scheme
  - uses 128-bit salt to create 192-bit hash value
- Both the FreeBSD and OpenBSD schemes should be secure from dictionary attack for the foreseeable future.

UNIX System Password File

- On any UNIX-like file system, the user identity information is stored in the "passwd" file and it is located in the "/etc/" directory.
- The "passwd" file has the following format: 7 colon-delimited fields and the fields are in the following order:
  - Username, encrypted password along with the salt, user ID, group ID, full name, Home directory, Shell
  - User ID – is a numeric identifier, which the OS uses to identify which files belong to the user. The system always thinks of the user in terms of a number. It uses the "passwd" file to convert the number into a more human-friendly form, the "username". The "username" is the name assigned by the system administrator and will be used to log in to the system.
  - Group ID – a UNIX group may contain none, one or more users, who will be able to access the files and directories owned by that group, based on that group's permissions. This is useful for sharing files between two people, as a file can have only one owner.
    - User Private Groups – each user is assigned their own group, identified by their username. The user is the only member of the group.
    - User private groups are used in most modern day implementations
  - Home directory – location where all the user files are usually stored
  - Shell – the command line that provides the user interface to the UNIX OS

UNIX System Password File

- The encrypted password + salt field is a 13-character field: the first two characters are the salt and the next 11 characters form the encrypted password.



```
rachel:eH5/.mj7NB3dx:181:100:Rachel Cohen:/u/rachel:/bin/ksh
arlin:f8fk3j1OIf34.:182:100:Arlin Steinberg:/u/arlin:/bin/csh
```

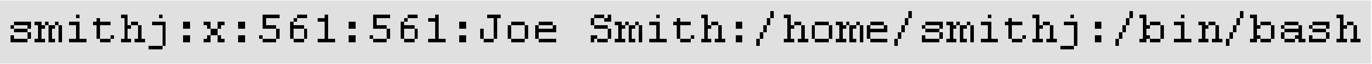
- In the above example, 5/.mj7NB3dx is the encrypted password and eH is the salt for username “rachel”.

- Traditional UNIX systems keep user account information, including the encrypted passwords, in the “/etc/passwd” text file.
- The “/etc/passwd” File is used by many tools (such as “ls”) to display file ownerships, etc., by matching user ID with the username field. As a result, the file needs to be world-readable and consequentially can be somewhat of a security risk.

- Solution: Store the actual encrypted password in another file called the “/etc/shadow” file and it is readable only by the root. This file also contains the password aging information along with the account information.

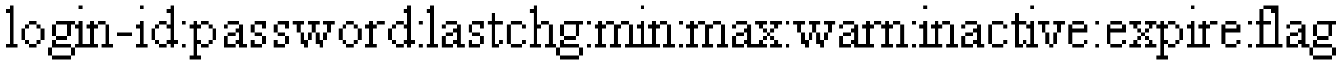
UNIX System Password File

- In most modern day UNIX and LINUX systems, the encrypted password is not stored in the “/etc/passwd” file and is stored in the “/etc/shadow” file.
- If the encrypted password is stored in the “/etc/shadow” file, then the password holder field in each row of the “/etc/passwd” file is filled with just character “x”.



```
smithj:x:561:561:Joe Smith:/home/smithj:/bin/bash
```

- The “/etc/shadow” file contains a row for each user; each row contains 9 fields, each separated by a “:”, in the form:

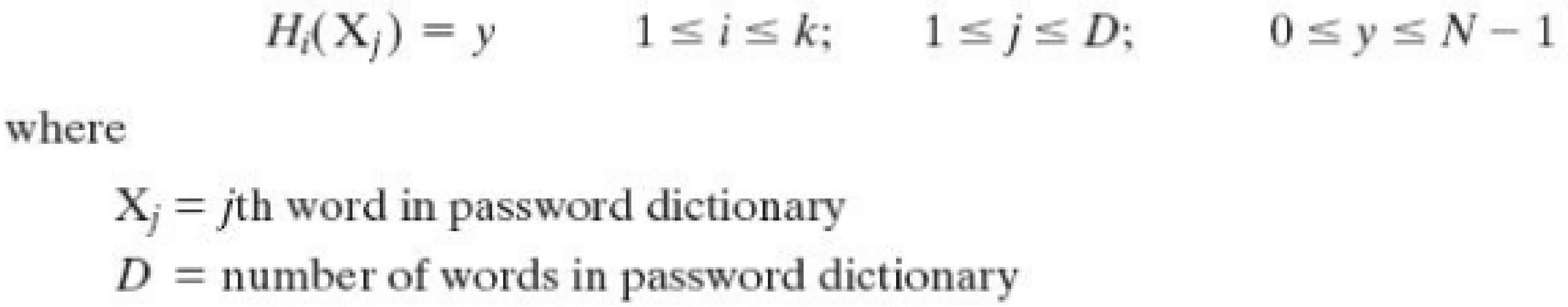


```
login-id:password:lastchg:min:max:warn:inactive:expire:flag
```

  - Login-id: User name
  - Password: 13 character (2 character salt + 11 character encrypted password)
  - lastchg: number of days, since the password was last changed
  - Min: the minimum number of days before password may be changed
  - Max: the maximum number of days, after which the password must be changed
  - Warn: the number of days to warn user of an expiring password
  - Inactive: the number of days the account can be inactive, without being used
  - Expire: the number of days (since Jan 1, 1970) that the account should be disabled
  - Flag – reserved for future use

Proactive Password Cracking using Bloom Filter

- Proactive Password Cracking: Store a list of bad passwords; When a user (re)sets his password, check if it is in the bad list. If so, reject the password; otherwise, accept.
- Bloom Filter: Data structure to capture the list of bad passwords.
  - A Bloom Filter of order *k* consists of a set of *k* independent hash functions $H_1(x)$, $H_2(x)$, …, $H_k(x)$, where each hash function maps a password *x* into a value in the range 0 to N-1.



$$H_i(X_j) = y \qquad 1 \le i \le k; \qquad 1 \le j \le D; \qquad 0 \le y \le N - 1$$

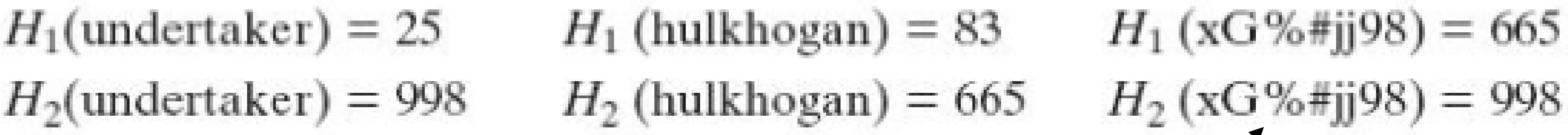
where

X_j = *j*th word in password dictionary

D = number of words in password dictionary

Bloom Filter: Procedure and Analysis

- The Bloom Filter is a hash table.
- The hash table is of size *N* bits, with all the bits initially set to 0.
- For each password, its *k* hash values are calculated, and the corresponding bits in the hash table are set to 1. If the bit already has the value 1, it remains at 1.
- When a new password is presented to the checker, its *k* hash values are calculated. If all the corresponding bits of the hash table are equal to 1, then the password is rejected (considered to be in the list of bad passwords).
- Note that there cannot be false negatives (i.e., a user entered password that is in the bad list has to have all its *k* hash values index in the Bloom Filter to bit positions that are set to 1).
- However, there can be false positives (i.e., a user entered password that is not in the bad list could still have its *k* hash values that index to the Bloom Filter to bit positions that are set to 1).



$H_1(\text{undertaker}) = 25$ $\quad H_1\,(\text{hulkhogan}) = 83$ $\quad H_1\,(\text{xG\%\#jj98}) = 665$

$H_2(\text{undertaker}) = 998$ $\quad H_2\,(\text{hulkhogan}) = 665$ $\quad H_2\,(\text{xG\%\#jj98}) = 998$

Test password

Bloom Filter Analysis

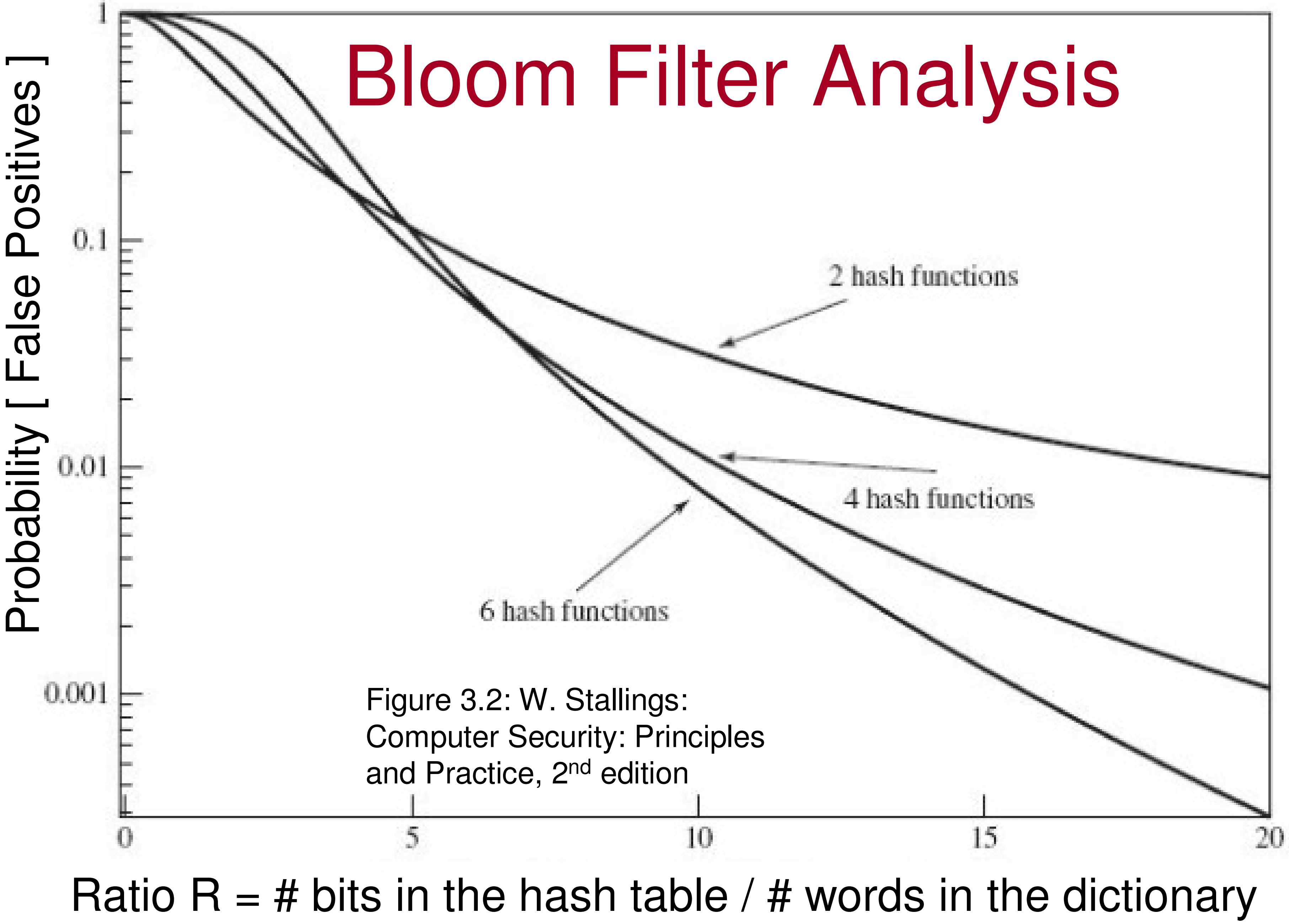


Figure 3.2: W. Stallings: Computer Security: Principles and Practice, 2nd edition

Token-based Authentication

- A token is an object that a user possesses for the purpose of user authentication.
- Traditional token: Memory card (magnetic stripe: containing the user ID) swiped by the user to a card reader. The card reader communicates with the server to authenticate the user.
  - Disadvantage: Anyone who possess the token can get authenticated.
- Memory card along with a PIN
  - Advantage: Multi-factor authentication
  - Disadvantage: The reader needs to act as a middleman between the server and the user; needs to be more specialized and secure not to be tampered.
- Smart card: Add intelligence to the token
  - Include a microprocessor, human-token interface or electronic interface to communicate with a card reader, authentication protocol

Smart Card

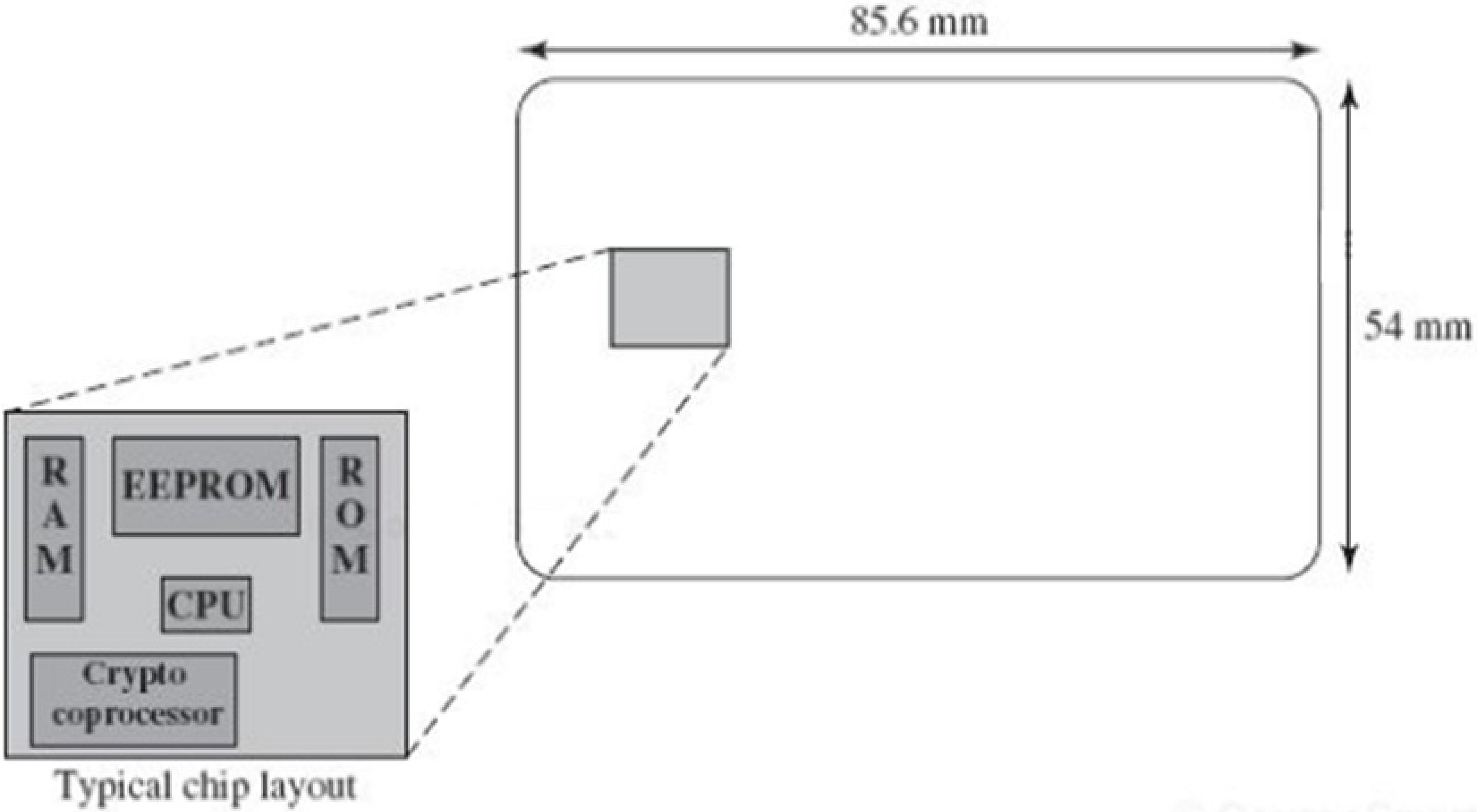


Figure 3.3 **Smart Card Dimensions** The smart card chip is embedded into the plastic card and is not visible. The dimensions conform to ISO standard 7816-2.

Smart Card

- **Memory Components**
  - Read Only Memory (ROM): stores data that does not change during the card's life, such as the card number and the card holder's name.
  - Electrically erasable programmable ROM (EEPROM): holds application data and protocols that the card can execute. Also holds data that may vary with time (e.g., talk time available in a calling card)
  - Random Access Memory (RAM): holds data dynamically generated when applications are executed.
- **Authentication Protocols**
  - Static: The user authenticates to the token and the token in turn authenticates to the server
  - Dynamic Password Generator: The token periodically generates a unique password that is sent to the server when swiped. If the server also generated the same password at that time, the user is authenticated. The token and server need to be synchronized.
  - Challenge-Response: The server generates a challenge for the token when swiped; the token may require the user to enter a PIN for further processing the challenge and sending the appropriate response to the server. The server validates the response (based on the challenge generated and the PIN stored for the user).

Biometric Authentication

- Authenticate a user based on his/her unique physical characteristics
  - Static characteristics: Fingerprints, hand geometry, facial characteristics, and retinal and iris patterns
  - Dynamic characteristics: voiceprint and signature.
- Based on pattern recognition.
- Technically more complex and expensive.

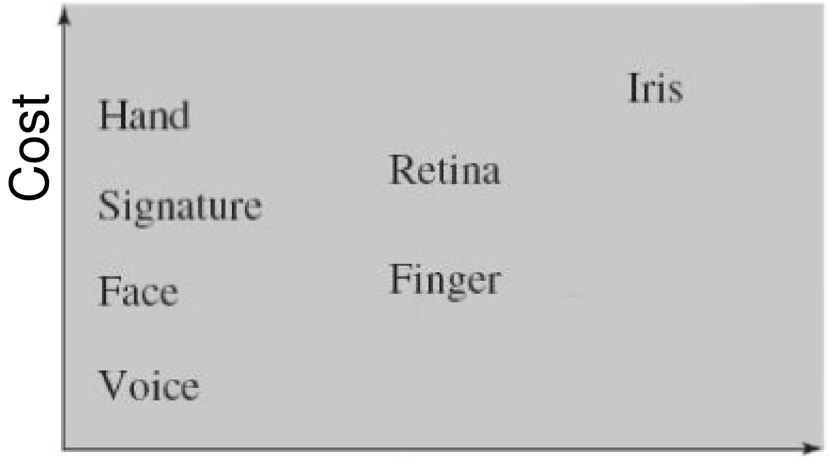


Figure 3.5

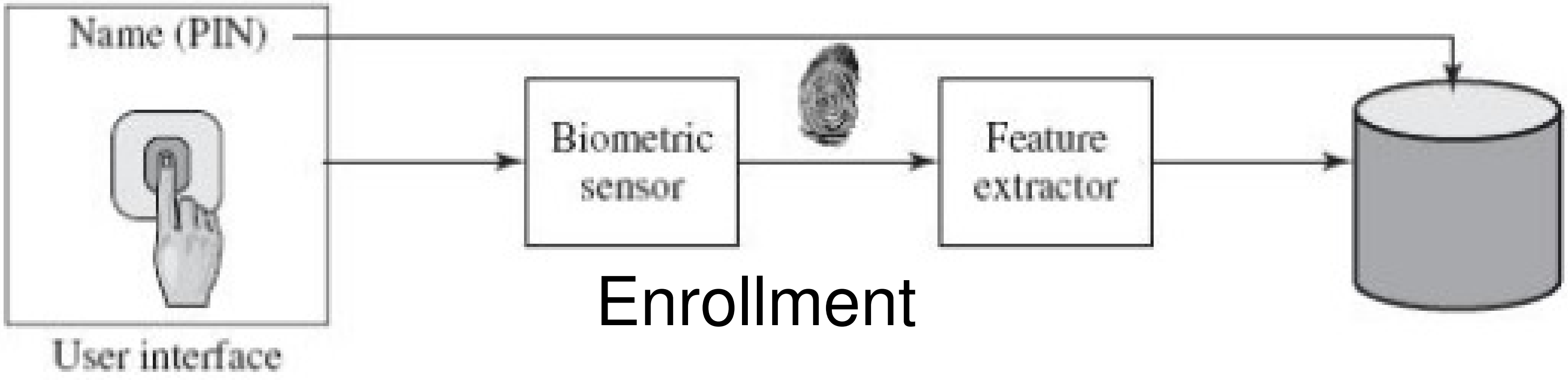
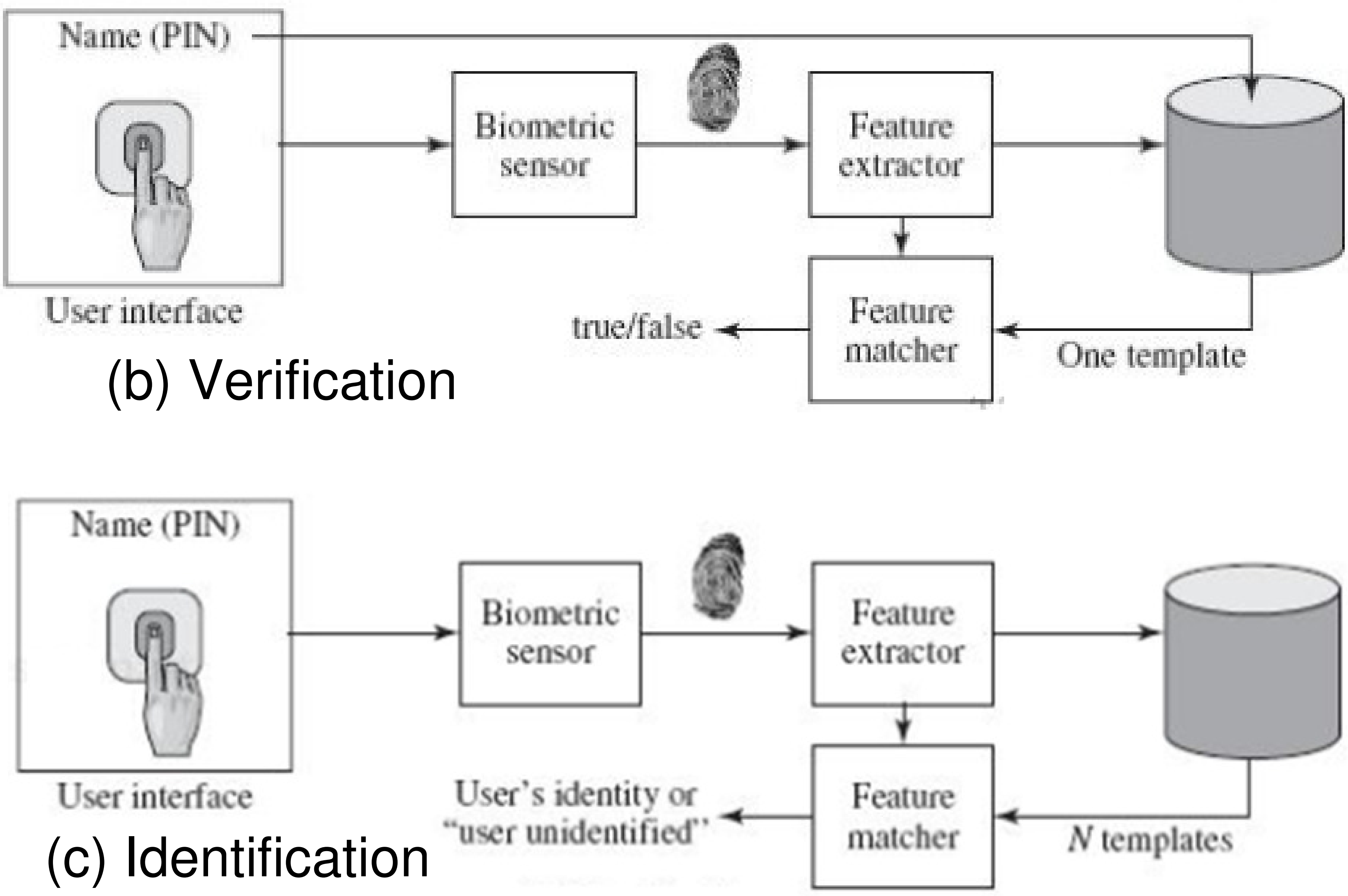


Figure 3.6: W. Stallings: Computer Security: Principles and Practice, 2nd edition



(b) Verification

(c) Identification

Figure 3.6: W. Stallings: Computer Security: Principles and Practice, 2nd edition

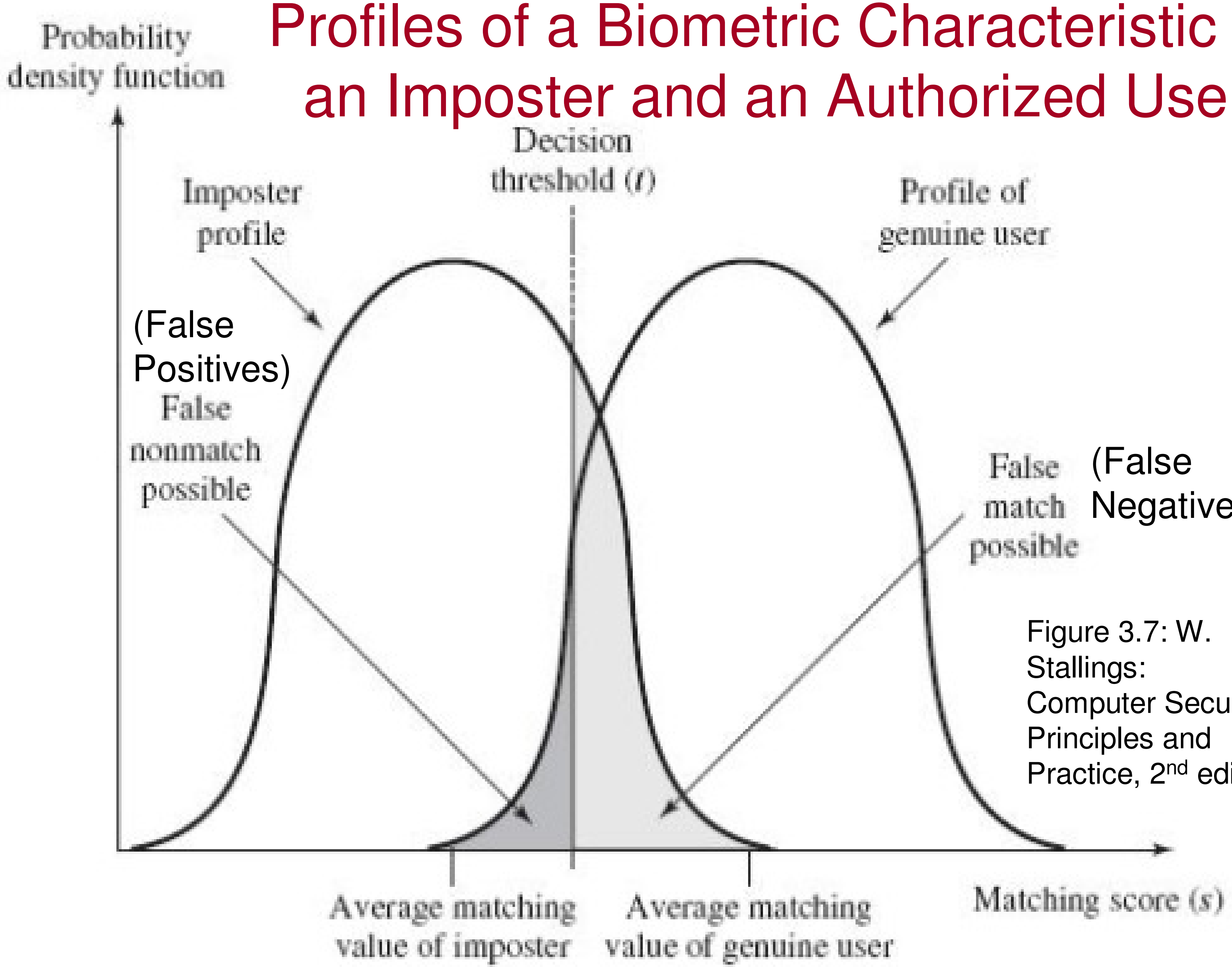
Two Modes of Biometric Systems

- Identification Mode: (primary means/source of authentication)
  - Uses biometric traits that cannot be easily forged and are supposedly unique for each user
  - Could be used for one to many comparison (if username is unknown) and subsequent authentication
  - Low false accept rates
  - More accurate, difficulty associated with data collection and usage
  - Examples: Fingerprint systems, Iris recognition systems, Retinal scans
- Verification Mode: (secondary source of authentication)
  - Uses biometric traits that need not be unique for each user and will incur high false accept and false error rates if used for identification; but, can be used to validate whether a user is whom he/she claims
  - Could be used for one to one comparison
  - Favored for ease associated with data collection and usage
  - Examples: Face recognition systems (only biometric system used for Mass Surveillance), Signature recognition systems, Voice recognition systems

Biometric Accuracy

- In any biometric scheme, some physical characteristic of the individual is mapped into a digital representation (template) that is stored in the authenticating server.
- When the user is to be authenticated, the system compares the stored template to the presented template.
- Given the complexities of the physical characteristics, we cannot expect an expect match between the two templates.
  - The system uses an algorithm (like the Hamming Distance) to generate a matching score (typically a single number) that quantifies the similarity between the input and the stored template.
  - If a single user is tested several times, the matching score may vary (for e.g., in the case of fingerprint, results may vary due to sensor noise, swelling, dryness, etc), leading to a probability density function, typically in the form of a bell-shaped curve.
  - If the matching score is greater than or equal to a threshold, then we accept the user; otherwise, reject the user.

Profiles of a Biometric Characteristic of an Imposter and an Authorized User



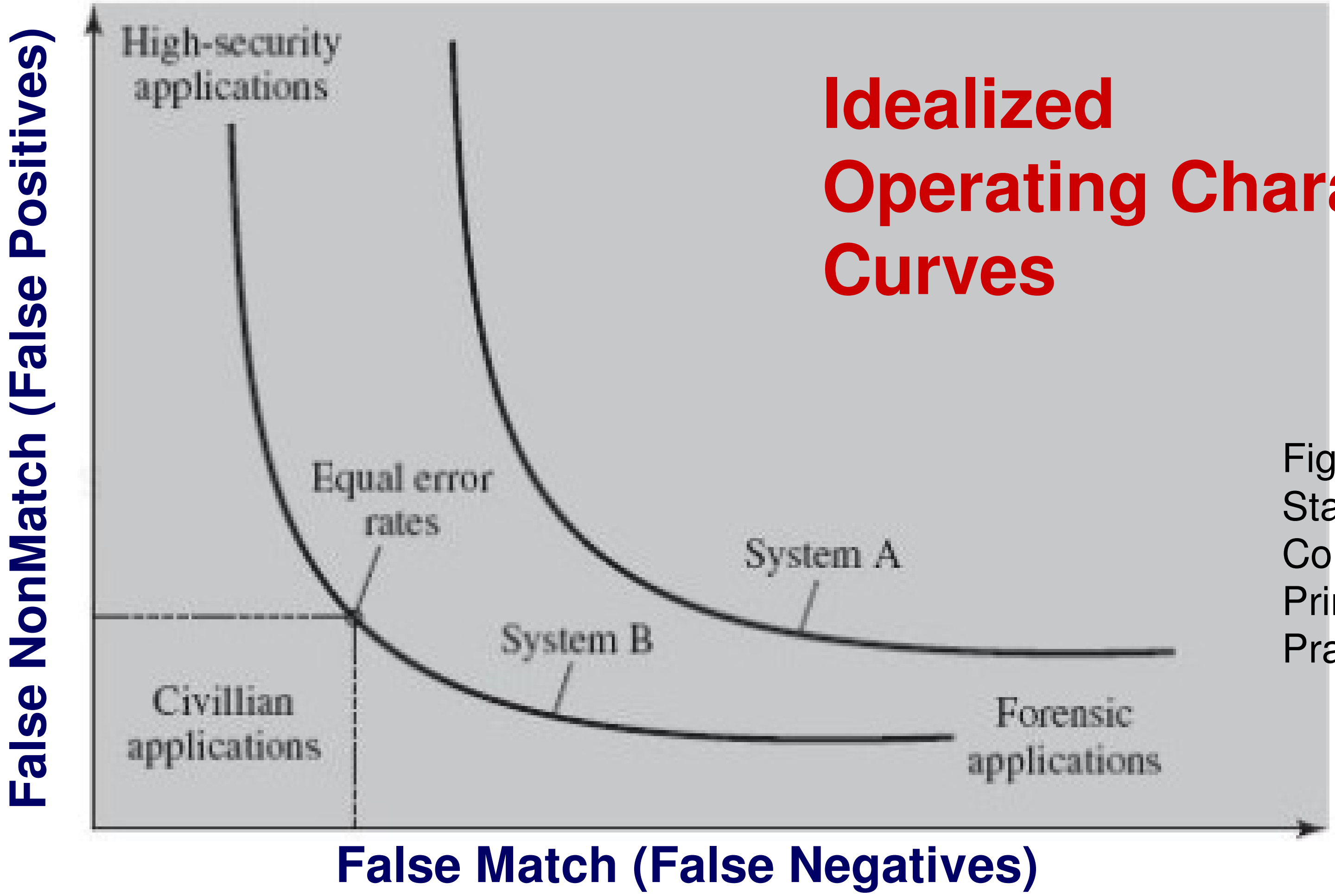
(False Positives)

(False Negatives)

Figure 3.7: W. Stallings: Computer Security: Principles and Practice, 2nd edition

Biometric Accuracy

- By moving the threshold to the right or left, the probability of false match (false negative) and false nonmatch (false positive) can be adjusted (note that, if one increases, the other decreases, and vice-versa).



Idealized Operating Characteristic Curves

Figure 3.8: W. Stallings: Computer Security: Principles and Practice, 2nd edition

Biometric Accuracy

- For high-security applications, unauthorized users should not be let in. Implies, there should be zero or very low false negatives (at the cost of more false positives).
  - A high threshold score

- For forensic applications, we need more users/suspects who can be further probed. Hence, we need more false negatives (users who are not authorized; but appear to have templates that match to the original), also leading to low false positives (i.e., genuine users are not much disturbed).
  - A low threshold score

- For civilian applications (regular use), we need low false positives and low false negatives. Hence, we set the threshold in such a way that the probability of false positives = probability of false negatives (in the operating characteristic curve).

Actual Operating Characteristic Curves for Biometric Systems

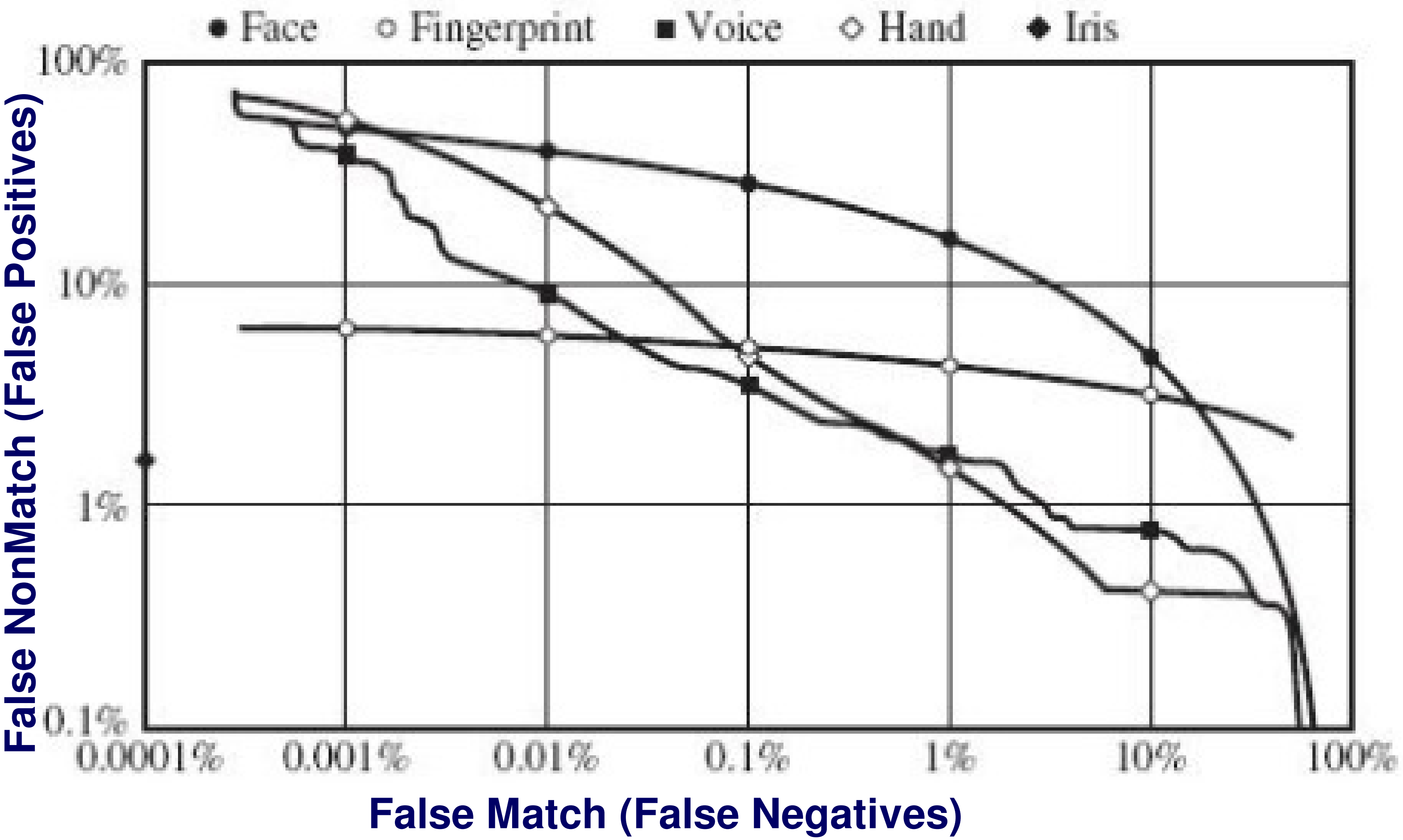


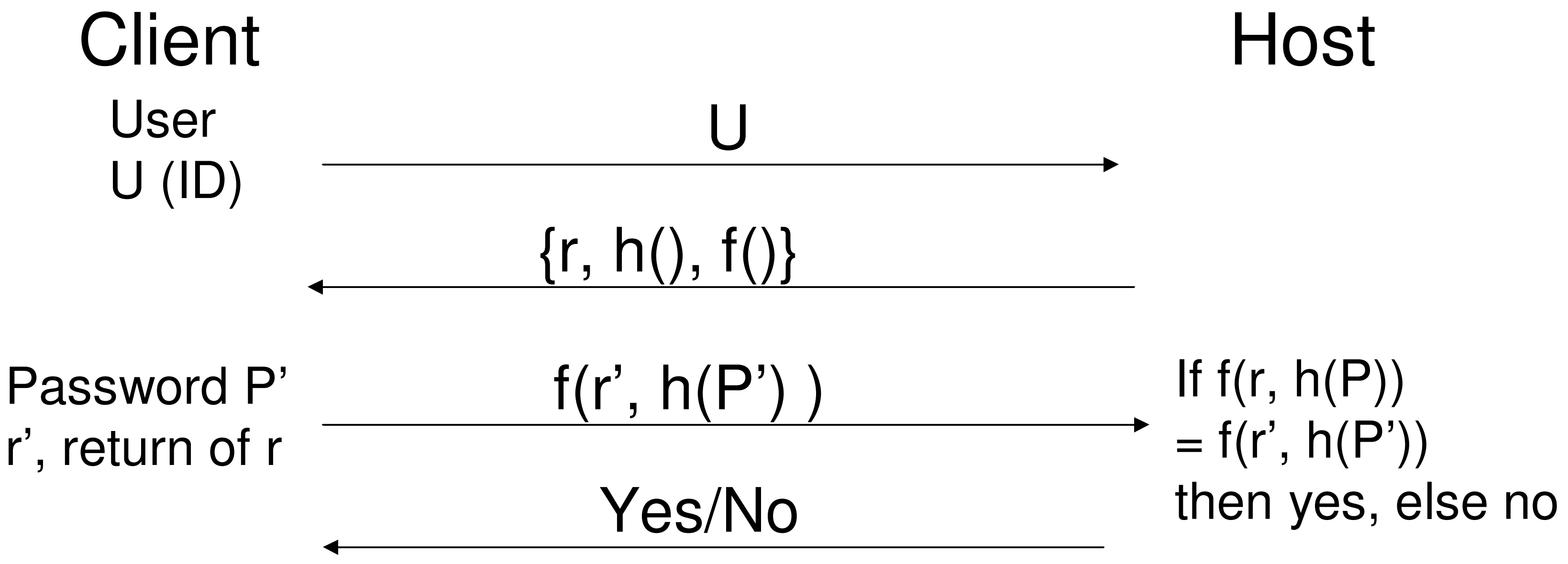
Figure 3.9: W. Stallings: Computer Security: Principles and Practice, 2nd edition

Remote User Authentication

- Remote user authentication occurs over the Internet, a network or a communication link,
  - raising additional security threats like an eavesdropper being able to capture a password, or an adversary replaying an authentication sequence that has been observed.
  - We use Challenge-response protocols

Mostly P = P’ ; the values of r and r’ depends on f

Protocol for Password-based Authentication



Client

Host

User
U (ID)

U

{r, h(), f()}

Password P’
r’, return of r

f(r’, h(P’))

If f(r, h(P))
= f(r’, h(P’))
then yes, else no

Yes/No

Protocol for Token-based Authentication

- User transmits his/her identity to the host.
- The host sends a random number *r* and the identifiers of functions f() and h() to be used in the response.
- At the user end, s/he submits the password P’ to the token (P’ is shared only between the user and the token)
- If the password is valid, the token uses the passcode W’ (either static or generated dynamically) to compute h(W’) and send a response f(r’, h(W’)).
- The host computes f(r, h(W(U))) for the user and if it matches with f(r’, h(W’)) received from the user, then the user is authenticated.

Protocol for Static Biometric-based Authentication

- User transmits its ID to the host that responds with a random number *r* and identifier for an encryption function E( ).
- On the user side is a client system that controls a biometric device (identified by D').
- User submits his/her biometric B' to the device and the system in turn generates a biometric template BT'.
- The client system computes E(r', D', BT') and transmits to the host.
- The host decrypts the incoming message to recover the three transmitted parameters and compares these to locally stored values.
  - r' = r,
  - D' is one of the registered devices and
  - the matching score between BT' and the stored template exceeds a pre-defined threshold, then the host authenticates the user.

Protocol for Dynamic Biometric-based Authentication

- The host provides the user both a random sequence (of numbers, characters or words) and a random number as challenges.
- The human user at the client end must then vocalize (speaker verification), type (keyboard dynamics verification) or write (handwriting verification) the sequence to generate a biometric signal BS'(x').
- The client side encrypts the biometric signal and the random number.
- At the host side, the incoming message is decrypted.
- If incoming random number matches with that sent, the host generates a comparison based on the incoming biometric signal BS'(x'), the stored template BT(U) for this user and the original signal *x*.
  - If the comparison value exceeds a pre-defined threshold, the user is authenticated.