Chapter 4

User Authentication and Key Management

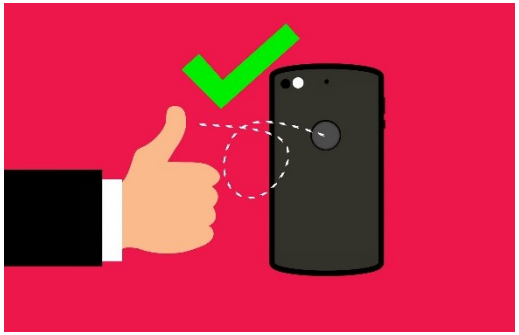
Most important aspect: Usability

When security and/or privacy and usability collide, usability always wins!

- When security methods or implications on users' privacy are not properly understood, systems will be used incorrectly
- Annoying and obtrusive security measures are simply deactivated so that users can get their jobs done
- For example:
  - sharing passwords, never logging out
  - writing PIN on back of card, most often used PINs “1234” and “0000”
  - “ALERT: The URL says www.mybank.com, but the certificate is for cracker.net, really continue?” - “Yeah, whatever, just let me enter my PIN and TAN codes now...”

RFC 2828

RFC 2828 defines user authentication as: “The process of verifying an identity claimed by or for a system entity.”



Authentication process

- Fundamental building block and primary line of defense
- Basis for access control and user accountability
- **Authentication** (proving an identity) **is not the same as authorization** (assigning access control rights / capabilities to an identity)
  - **identification step**
    - presenting an identifier to the security system
      - Note: identifier may be a pseudonym or even “anonymous”
  - **verification step**
    - presenting or generating authentication information that corroborates the binding between the entity and the identifier

Four means of user authentication

Verifying user identity by **something the individual ...**

knows:

- Password / PIN
- Answer to question(s)
- Graphical pattern

possesses (a token):

- Smartcard
- Electronic keycard
- Physical key
- (Embedded software token)

is (static biometrics):

- Fingerprint
- Retina / iris
- Face
- Ear, hand geometry, etc.

does (dynamic biometrics):

- Voice pattern
- Gait
- Handwriting
- Typing rhythm

Password authentication

- Widely used first line of defense against intruders
  - user provides name/login and password
  - system compares password with the one (one-way function derived value) stored for that specified login

- The user ID:
  - determines if the user is *authorized* to access the system
  - determines the user's privileges
  - is used in discretionary/mandatory/role based access control

- Need to protect passwords stored on disk/flash/memory!

Password vulnerabilities

- Offline dictionary attack: see hashed passwords
- Specific account attack: one/few user IDs, many password tries
  - countermeasure is lockout after N failed attempts
- Popular password attack: many user IDs with few popular passwords
  - countermeasure is to force non-dictionary passwords
- Password guessing against single user: try to exploit knowledge about specific user
- Workstation hijacking: use of unlocked workstations / devices
  - countermeasure is automatic screen lock after N seconds/minutes
- Exploiting user mistakes: if password (policy) is too complex, users tend to write them down
- Exploiting multiple password use: using a password from one system on others → "*password stuffing*" attack to try leaked passwords on other sites
- Electronic monitoring: eavesdropping of passwords transmitted over network connections if not properly protected (simply encrypted with shared key is not a proper protection)
  - countermeasure is challenge response protocol

Use of hashed passwords

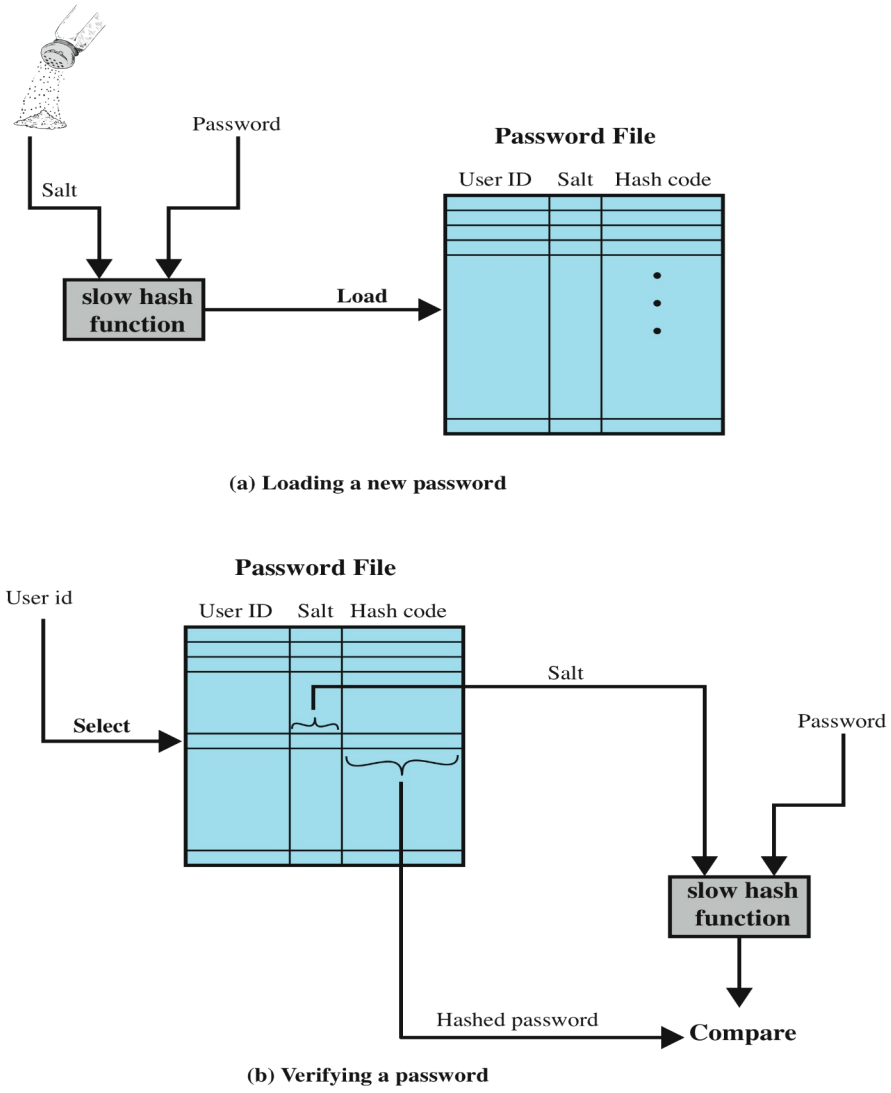


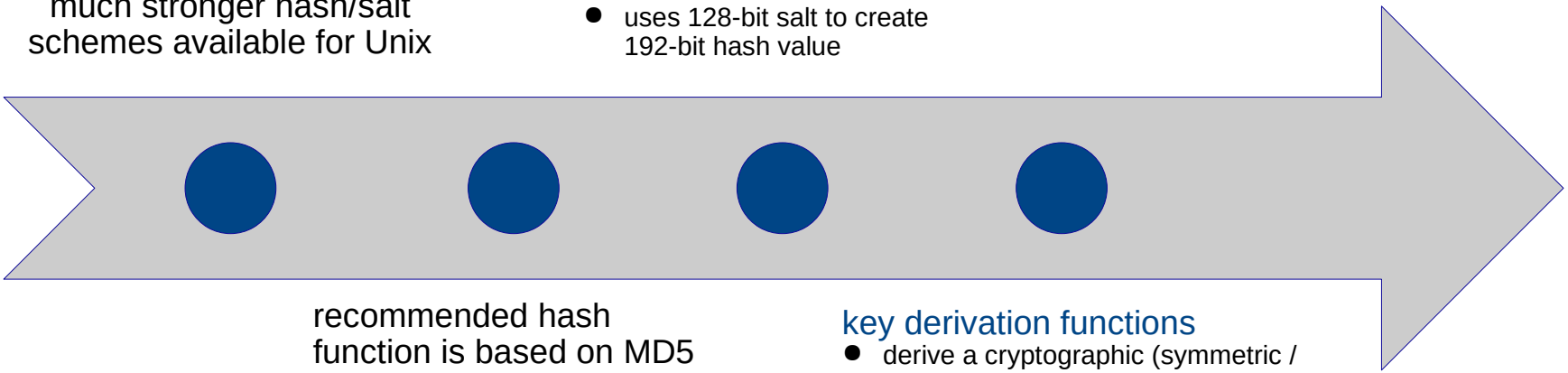
Figure 3.1 UNIX Password Scheme

Improved Implementations over time

much stronger hash/salt schemes available for Unix

OpenBSD uses Blowfish block cipher based hash algorithm called **bcrypt**

- more secure version of Unix hash/salt scheme
- uses 128-bit salt to create 192-bit hash value



recommended hash function 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

key derivation functions

- derive a cryptographic (symmetric / secret) key from user-supplied password
- also use salt as mitigation of low-entropy passwords and rainbow tables
- **scrypt** is currently among strongest key derivation functions because it increases memory requirements along with runtime overhead → hard to brute force on ASICs

Argon2 is a new standard based on BLAKE2
→ recommended to use

Password studies

Many data sources suggest ...

- Purdue 1992 - many short passwords
- Klein 1990 - many guessable passwords
- … and many more results since then, including released password lists (Adobe, Ashley Madison, …)
- (Probably) biggest “study”: https://haveibeenpwned.com/

… that user-chosen passwords are often week

- Conclusion from studies is that users often choose poor passwords
- Need some approach to counter this

Managing passwords – education

- Can use policies and good user education
- Educate on importance of good passwords
- Give guidelines for good passwords
  - minimum length (>6)
  - require a mix of upper and lower case letters, numbers, punctuation
  - not dictionary words
- **But likely to be ignored by many users**

Managing passwords – computer generated

- Let computer create passwords
- If random likely not memorisable, so will be written down (sticky label syndrome)
- Even pronounceable not remembered
- Have history of poor user acceptance
- FIPS PUB 181 one of best generators
  - has both description and sample code
  - generates words from concatenating random pronounceable syllables
  - much longer for given security, but humans can more easily remember

Managing passwords – reactive checking

- Reactively run password guessing tools
  - note that good dictionaries exist for almost any language/interest group
- Cracked passwords are disabled
- But is resource intensive
- Bad passwords are vulnerable till found

- **Check your own passwords**: https://haveibeenpwned.com

Managing passwords – proactive checking

- Most promising approach to improving password security
- Allow users to select own password
- But have system verify it is acceptable
  - simple rule enforcement (see earlier slide)
  - compare against dictionary of bad passwords
  - use algorithmic (Markov model or bloom filter) to detect poor choices

Password cracking

- Dictionary attacks
  - develop a large dictionary of possible passwords and try each against the password file
  - each password must be hashed using each salt value and then compared to stored hash values

- **Rainbow table** attacks
  - pre-compute tables of hash values for all salts
  - a mammoth table of hash values
  - can be countered by using a sufficiently large salt value and a sufficiently large hash length

Token based authentication (possession): Types of cards used as tokens

Card type	Relevant security feature	Example
Embossed / visual	raised characters, maybe visual security markers (holograms, etc.)	old credit card, driving license
Magnetic stripe	magnetic bar on back, characters on front	old bank/credit card, electronic keylock card
Memory	electronic memory cards (no CPU, just storage)	prepaid phone card
Smartcard - contact	electronic memory + CPU, contact pads exposed to card reader on the front or through dedicated port (e.g. USB)	new bank/credit card, citizen identity card, mobile phone SIM card, **FIDO2/U2F USB token**
Smartcard - contactless	electronic memory + CPU, wireless connection through embedded antenna, often powered by reader field (RFID, NFC)	new bank/credit card, new passport (with RFID), new electronic lock cards

Memory cards

- Can store but do not process data
- The most common is the magnetic stripe card
- Can include an internal electronic memory
- Can be used alone for physical access
  - hotel room
  - (old) ATM cards
- Provides significantly greater security when combined with a password or PIN compared at the reader
- Drawbacks of memory cards include:
  - requires a special reader
  - loss of token leaks all contained secrets
  - user dissatisfaction

Smartcard

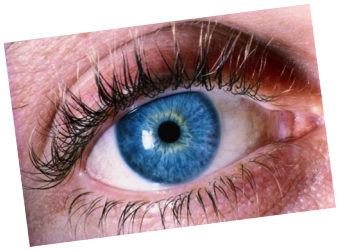
- Physical characteristics:
  - include an embedded (hardened) microprocessor
  - a smart token that looks like a bank card
  - can look like calculators, keys, small portable objects

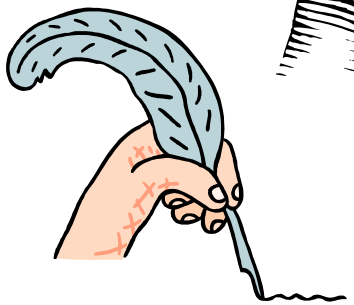
  Built into modern smartphones!

- Interface:
  - manual interfaces include a keypad and display for interaction
  - electronic interfaces communicate with a compatible reader/writer
- Authentication protocol:
  - classified into three categories: **static**, **dynamic password generator**, and **challenge-response**
  - If you can, use FIDO2/U2F!

Biometric authentication

- Attempts to authenticate an individual based on unique physical characteristics
- Based on pattern recognition: no try is exactly the same
- Is technically complex and expensive when compared to passwords and tokens
- Physical characteristics used include:
  - facial characteristics
  - fingerprints
  - hand, ear, ... geometry
  - retinal pattern
  - iris
  - signature
  - voice
  - gait
  - …





Cost versus accuracy

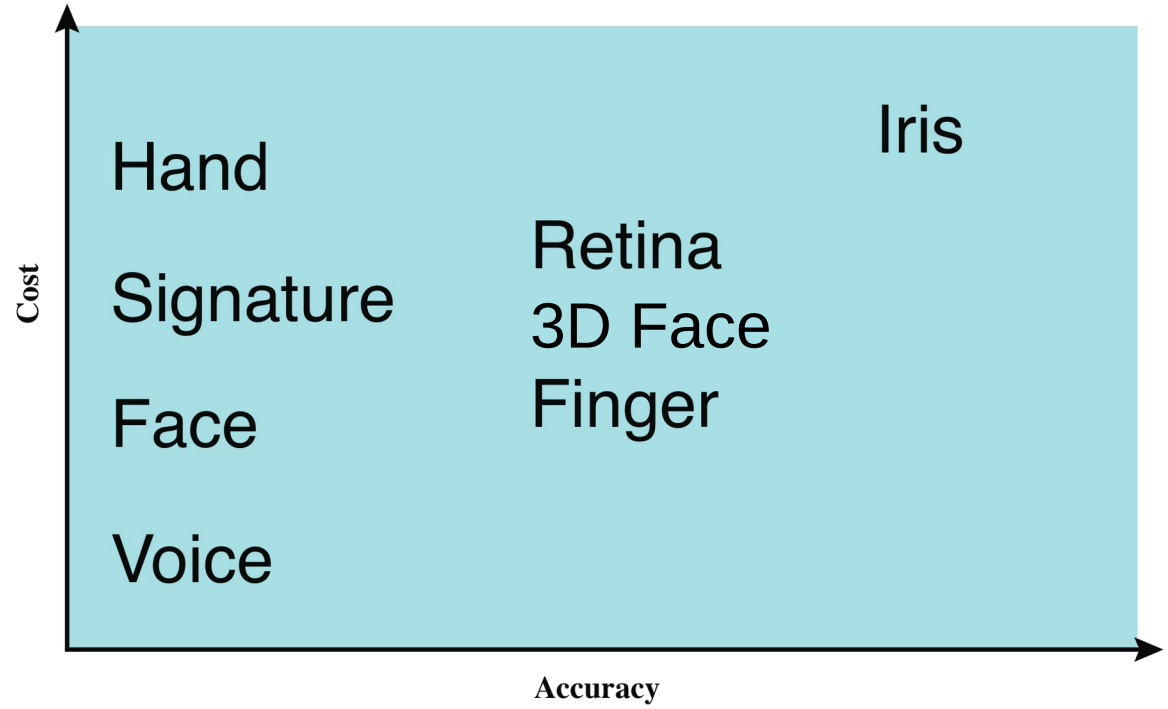
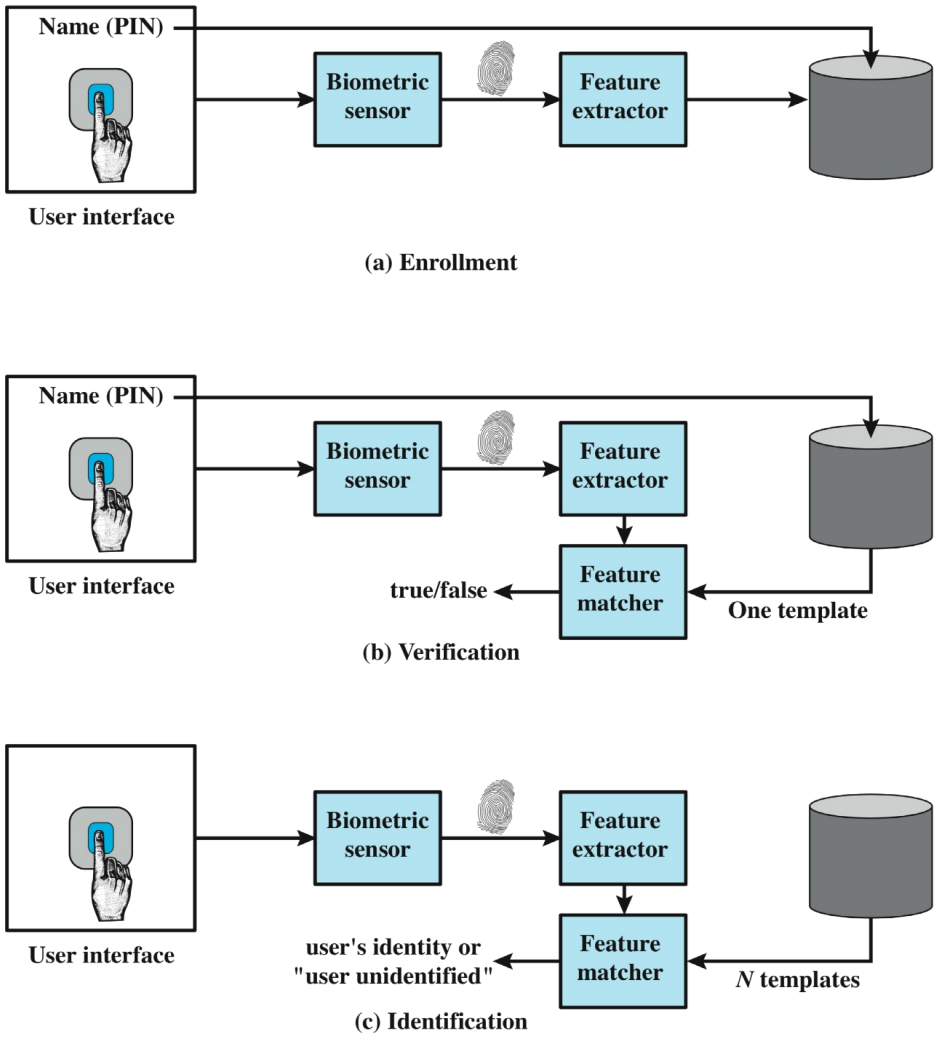


Figure 3.5 Cost Versus Accuracy of Various Biometric Characteristics in User Authentication Schemes.

Operation of a biometric system



A generic biometric system enrollment creates an association between a user and the user's biometric characteristics. Depending on the application, user authentication either involves verifying that a claimed user is the actual user or identifying an unknown user.

Figure 3.6

Biometric accuracy

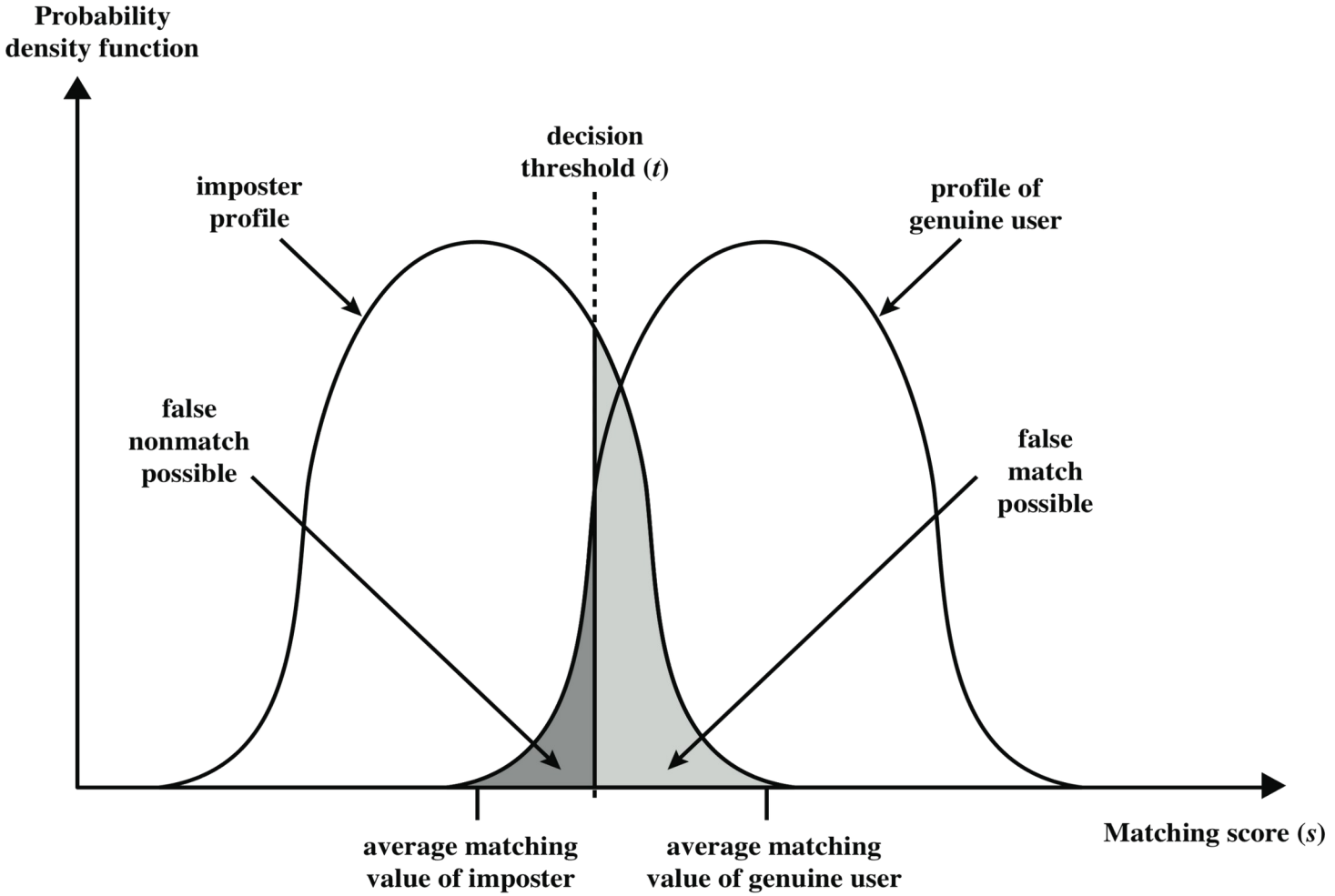


Figure 3.7 Profiles of a Biometric Characteristic of an Imposter and an Authorized Users In this depiction, the comparison between presented feature and a reference feature is reduced to a single numeric value. If the input value (s) is greater than a preassigned threshold (t), a match is declared.

Biometric measurement operating characteristic curves (ROC): theoretical/ideal curves

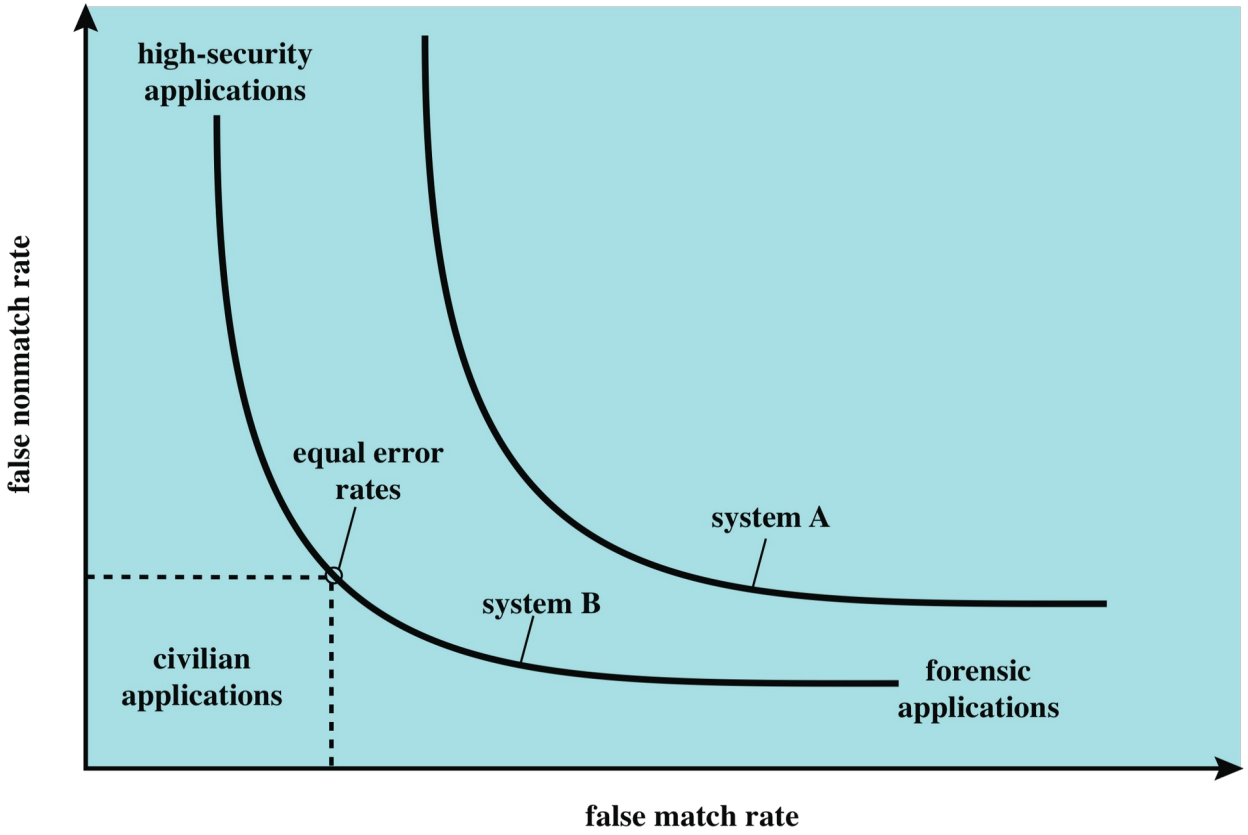


Figure 3.8 Idealized Biometric Measurement Operating Characteristic Curves. Different biometric application types make different trade-offs between the false match rate and the false nonmatch rate. Note that system A is consistently inferior to system B in accuracy performance. [JAIN00]

Actual biometric measurement operating characteristic curves

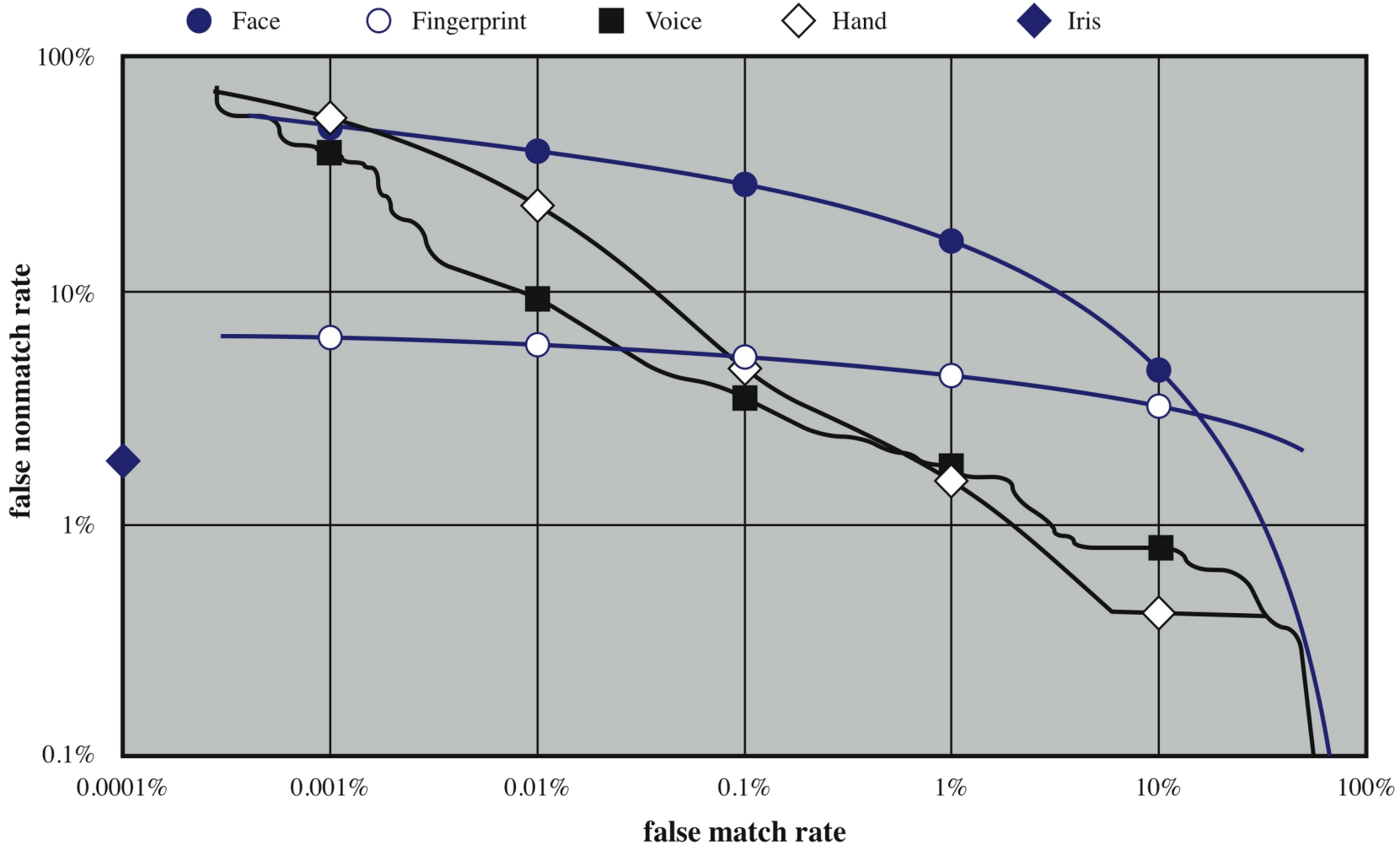


Figure 3.9 Actual Biometric Measurement Operating Characteristic Curves, reported in [MANS01]. To clarify differences among systems, a log-log scale is used.

Remote user authentication

- Authentication over a network, the Internet, or a communications link is more complex
  - additional security threats such as:
    - eavesdropping, capturing a password, replaying an authentication sequence that has been observed

- Generally rely on some form of a challenge-response protocol to counter threats

Potential attacks, susceptible authenticators, and typical defenses

Attacks	Authenticators	Examples	Typical defenses
Client attack	Password	Guessing, exhaustive search	Large entropy; limited attempts
	Token	Exhaustive search	Large entropy; limited attempts, theft of object requires presence
	Biometric	False match	Large entropy; limited attempts
Host attack	Password	Plaintext theft, dictionary/exhaustive search	Hashing; large entropy; protection of password database
	Token	Passcode theft	Same as password; 1-time passcode
	Biometric	Template theft	Capture device authentication; challenge response
Eavesdropping, theft, and copying	Password	"Shoulder surfing"	User diligence to keep secret; administrator diligence to quickly revoke compromised passwords; multifactor authentication
	Token	Theft, counterfeiting hardware	Multifactor authentication; tamper resistant/evident token
	Biometric	Copying (spoofing) biometric	Copy detection at capture device and capture device authentication
Replay	Password	Replay stolen password response	Challenge-response protocol
	Token	Replay stolen passcode response	Challenge-response protocol; 1-time passcode
	Biometric	Replay stolen biometric template response	Copy detection at capture device and capture device authentication via challenge-response protocol
Trojan horse	Password, token, biometric	Installation of rogue client or capture device	Authentication of client or capture device within trusted security perimeter
Denial of service	Password, token, biometric	Lockout by multiple failed authentications	Multifactor with token

Table 3.4

Entropy of passwords

- Can try to estimate Shannon entropy of password string
- But would most probably be overly optimistic, since password characters are not uniformly random and independent, but typically from natural language association
- Better methods account for this practice, e.g. NIST 800-63-1 Appendix A (http://csrc.nist.gov/publications/nistpubs/800-63-1/SP-800-63-1.pdf)

NIST 800-63-1 password entropy estimation

- The entropy of the first character is taken to be 4 bits;
- The entropy of the next 7 characters are 2 bits per character; this is roughly consistent with Shannon's estimate that "when statistical effects extending over not more than 8 letters are considered the entropy is roughly 2.3 bits per character;"
- For the 9th through the 20th character the entropy is taken to be 1.5 bits per character;
- For characters 21 and above the entropy is taken to be 1 bit per character;
- A "bonus" of 6 bits of entropy is assigned for a composition rule that requires both upper case and non-alphabetic characters. This forces the use of these characters, but in many cases these characters will occur only at the beginning or the end of the password, and it reduces the total search space somewhat, so the benefit is probably modest and nearly independent of the length of the password;
- A bonus of up to 6 bits of entropy is added for an extensive dictionary check. If the Attacker knows the dictionary, he can avoid testing those passwords, and will in any event, be able to guess much of the dictionary, which will, however, be the most likely selected passwords in the absence of a dictionary rule. The assumption is that most of the guessing entropy benefits for a dictionary test accrue to relatively short passwords, because any long password that can be remembered must necessarily be a "pass-phrase" composed of dictionary words, so the bonus declines to zero at 20 characters.

Remember!

A note on storing passwords

- Ideally: in new systems, **don't!**
  - use federated authentication instead of storing passwords yourself
  - use FIDO2/WebAuthn instead of passwords for authentication
  - use device-specific tokens instead of global passwords per account
- If password authentication is **really** required
  - **never store plain-text passwords in any form**
  - **don't encrypt passwords** – where do you store the encryption key?
  - only store one-way derived hashes of user passwords
    - best to do this one-way transformation on the client (e.g. in Javascript in the browser or the mobile client) and never even send the password
  - those hashes need to be "salted" with a random number
    - a **new random salt per password** – not a global one!
  - use a **slow derivation function** that ideally requires significant memory to compute, e.g. `Argon2` or `scrypt` (but no longer `PBKDF2`)
    - otherwise attackers can use GPUs/ASICs to compute rainbow tables