HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Identity Authentication

With extra material for further reading, indicated by symbol *

Authentication

- Basics
- Passwords
- Challenge-Response
- Biometrics
- Location
- Multiple Methods

Basics

- Authentication: binding of identity to subject
  - Identity is that of external entity (my identity, Van, *etc*.)
  - Subject is computer entity (process, *etc*.)
- Note:
  - message authentication is a different topic and already mentioned in the applications of hash functions

Establishing Identity

- One or more of the following
  - What entity knows (*eg.* password)
  - What entity has (*eg.* Identity card, smart card)
  - What entity is (*eg.* fingerprints, retinal characteristics)
  - Where entity is (*eg*. In front of a particular terminal)

Authentication System

- We need a formal definition, rather abstract view, of an AS
- A 5-tuple (A, C, F, L, S)
  - *A – a set:* information that proves identity
  - *C – a set:* information stored on computer and used to validate authentication information
  - *F: a set of* complementation functions; $f : A \rightarrow C$
    - *To compute complement information from identity information*
  - *L:* authentication functions that prove identity
  - *S:* functions enabling entity to create, alter information in *A* or *C*

Example

- Password system, with passwords stored on line in clear text
  - *A* set of strings making up passwords
  - *C* = *A*
  - *F* singleton set of identity function { *I* }
  - *L* single equality test function { *eq* }
  - *S* function to set/change password

Passwords

- Sequence of characters
  - Examples: 10 digits, a string of letters, *etc*.
  - Generated randomly, by user, by computer with user input
- Sequence of words
  - Examples: pass-phrases
- Algorithms
  - Examples: challenge-response, one-time passwords

Storage

- Store as cleartext
  - If password file compromised, *all* passwords revealed
- Encipher file
  - Need to have decipherment, encipherment keys in memory
  - Reduces to previous problem need something else
- Solution: Instead store one-way hash of password
  - Got the file, attacker must still guess passwords or invert the hash values

Example: Unix

- By definition, a 5-tuple (*A*, *C*, *F*, *L*, *S*)
  - *A – a set:* information that proves identity
  - *C – a set:* information stored on computer and used to validate authentication information
  - *F: a set of* complementation functions; $f : A \rightarrow C$
  - *L:* authentication functions that prove identity
  - *S:* functions enabling entity to create, alter information in *A* or *C*

Example: Unix

- By definition, a 5-tuple (*A*, *C*, *F*, *L*, *S*)
  - *A – a set:* information that proves identity
    - *A* = { strings of 8 chars or less }
  - *C – a set:* information stored on computer and used to validate authentication information
    - *C* = {hash values of password}
  - *F: a set of* complementation functions; *f* : *A* → *C*
    - *F* = { versions of modified DES }
  - *L:* authentication functions that prove identity
    - *L* = { *login*, *su*, … }
  - *S:* functions enabling entity to create, alter information in *A* or *C*
    - *S* = { *passwd*, *nispasswd*, *passwd+*, … }

Attacking passwords

- Goal: find $a \in A$ such that:
  - For some $f \in F$, $f(a) = c \in C$
  - c is associated with entity
- Two ways to determine whether a meets these requirements:
  - By trying computing f(a) for a set of a values until succeed
  - By trying calling I(a) until succeed (I(a) returns true)

Preventing Attacks

- How to prevent this:
  - Hide one of *a*, *f*, or *c*
    - Prevents obvious attack from above
    - Example: UNIX/Linux shadow password files
      - Hides the *c*'s
  - Block access to all $l \in L$ or result of $l(a)$
    - Prevents attacker from knowing if guess succeeded
    - Example: preventing *any* logins to an account from a network
      - Prevents knowing results of *l* (or accessing *l*)

Dictionary Attacks

- Trial-and-error from a list of potential passwords
  - *Off-line*: know *f* and *c*'s, and repeatedly try different guesses $g \in A$ until the list is done or passwords guessed
    - Examples: *crack*, *john-the-ripper*
  - *On-line*: have access to functions in *L* and try guesses *g* until some $l(g)$ succeeds
    - Examples: trying to log in by guessing a password

Success probability over a time period

Anderson's formula:

- P probability of guessing a password in specified period of time
- G number of guesses tested in 1 time unit
- T number of time units
- N number of possible passwords ($|A|$)
- Then $P \geq TG/N$

Example

- Goal
  - Passwords drawn from a 96-char alphabet
  - Can test 10^4 guesses per second
  - Probability of a success to be 0.5 over a 365 day period
  - What is minimum password length?
- Solution
  - $N \geq TG/P = (365\times24\times60\times60)\times10^4/0.5 = 6.31\times10^{11}$
  - Choose s such that $\Sigma^{s}_{j=0}\ 96^{j} \geq N$
  - So $s \geq 6$, meaning passwords must be at least 6 chars long

Exercise

X = number defined by last 2 digits of your student ID; Y = X mod 4
Assume that H is a cryptographic hash function with output size (Y+2)*16 bits. Assume that Scorpion-***i*** (***i***=1-9) is a specifically designed line of hardware chips for computing H, where Scorpion-***i*** can create 10^{i} * 1000 hash values a second (e.g. Scorpion-2 can do 100,000 hashes/sec). This product line is the best, fastest and affordable, in the market, priced at $\boldsymbol{i^{i/2}}$ *$1000 (e.g $2000 for *i*=2, $16000 for *i*=4).

An authentication system requires its users to pick their passwords of length exactly 6 from an alphabet of size N=(**X** mod 50)+ 40. Using H, this system maintains the hash values of the passwords of all the users. An enemy, who has gained access to this hashed password file, aims to launch an off-line attack to break the password of an important user. Using the Scorpion chips, how much the enemy has to spend in order to finish within a month with success probability (6+Y)*10%?

On password selection

- Random selection
    - Any password from *A* equally likely to be selected
- Pronounceable passwords
- User selection of passwords

Pronounceable Passwords

- Generate phonemes randomly
  - Phoneme is unit of sound, eg. *cv*, *vc*, *cvc*, *vcv*
  - Examples: helgoret, juttelon are; przbqxdfl, zxrptglfn are not
- Problem: too few
- Solution: key crunching
  - Run long key through hash function and convert to printable sequence
  - Use this sequence as password

User Selection

- Problem: people pick easy to guess passwords
  - Based on account names, user names, computer names, place names
  - Dictionary words (also reversed, odd capitalizations, control characters, “elite-speak”, conjugations or declensions, swear words, Torah/Bible/Koran/… words)
  - Too short, digits only, letters only
  - License plates, acronyms, social security numbers
  - Personal characteristics or foibles (pet names, nicknames, job characteristics, *etc*.

Picking Good Passwords

- "LlMm*2^Ap"
  - Names of members of 2 families
- "OoHeO/FSK"
  - Second letter of each word of length 4 or more in third line of third verse of Star-Spangled Banner, followed by "/", followed by author's initials
- What's good here may be bad there
  - "DMC/MHmh" bad at Dartmouth ("<u>D</u>artmouth <u>M</u>edical <u>C</u>enter/<u>M</u>ary <u>H</u>itchcock <u>m</u>emorial <u>h</u>ospital"), ok here
- Why are these now bad passwords? ☹

Proactive Password Checking

- Analyze proposed password for “goodness”
  - Always invoked
  - Can detect, reject bad passwords for an appropriate definition of “bad”
  - Discriminate on per-user, per-site basis
  - Needs to do pattern matching on words
  - Needs to execute subprograms and use results
    - Spell checker, for example
  - Easy to set up and integrate into password selection system

Salting

- Goal: slow dictionary attacks
- Method: perturb hash function so that:
  - Parameter controls *which* hash function is used
  - Parameter differs for each password
  - So given *n* password hashes, and therefore *n* salts, need to hash guess *n*

Examples

- Vanilla UNIX method
  - Use DES to encipher 0 message with password as key; iterate 25 times
  - Perturb E table in DES in one of 4096 ways
    - 12 bit salt flips entries 1–11 with entries 25–36
- Alternate methods
  - Use salt as first part of input to hash function

Unix actually is …

- UNIX system standard hash function
  - Hashes password into 11 char string using one of 4096 hash functions
- As authentication system:
  - *A* = { strings of 8 chars or less }
  - *C* = { 2 char hash id || 11 char hash }
  - *F* = { 4096 versions of modified DES }
  - *L* = { *login*, *su*, … }
  - *S* = { *passwd*, *nispasswd*, *passwd*+, … }

Exercise

Assume that H is a cryptographic hash function with output size (Y+2)*16 bits. Assume that Scorpion-***i*** (***i***=1-9) is a specifically designed line of hardware chips for computing H, where Scorpion-***i*** can create 10^{i} * 1000 hash values a second, priced at $i^{i/2}$ *$1000.

1. An authentication system requires its users to pick their passwords of length exactly 6 from an alphabet of size N=(**X** mod 50)+ 40. Using H, this system maintains the hash values of the passwords of all the users. An enemy, who has gained access to this hashed password file, aims to launch an off-line attack to break the password of an important user. Using the Scorpion chips, how much the enemy has to spend in order to finish within a month with success probability (6+Y)*10%?

2. The owner decides to enhance the above password system by using salt so that the enemy will need to spend at least 10 times the above mentioned amount of money to achieve the same goal. How many salt bits he/she need to use to achieve this purpose ?

Password Cracking: Do the Math*

* Further reading

- Assumptions:
- Pwds are 8 chars, 128 choices per character
  - Then $128^8 = 2^{56}$ possible passwords
- There is a **password file** with 2^{10} pwds
- Attacker has **dictionary** of 2^{20} common pwds
- **Probability** 1/4 that password is in dictionary
- **Work** is measured by number of hashes

Salt with slow hash *

- Hash password with **salt**
- Choose random salt s and compute

  $$y = h(\text{password}, s)$$

  and store (s,y) in the password file
- Note that the salt s is not secret
  - Analogous to IV
- Still easy to verify salted password
- But lots more work for Hacker
  - Why?

Password Cracking: Case I *

- Attack 1 specific password ***without*** using a dictionary
  - E.g., administrator's password
  - Must try $2^{56}/2 = 2^{55}$ on average
  - Like exhaustive key search
- Does **salt** help in this case?

Password Cracking: Case II *

- Attack 1 specific password ***with*** dictionary
- With **salt**
  - Expected work: $1/4\ (2^{19}) + 3/4\ (2^{55}) \approx 2^{54.6}$
  - In practice, try all pwds in dictionary…
  - …then work is at most 2^{20} and probability of success is $1/4$
- What if **no salt** is used?
  - One-time work to compute dictionary: 2^{20}
  - Expected work is of same order as above
  - But with precomputed dictionary hashes, the "in practice" attack is essentially free…

Password Cracking: Case III *

- Any of 1024 pwds in file, ***without*** dictionary
  - Assume all 2^{10} passwords are distinct
  - Need 2^{55} **comparisons** before expect to find pwd
- If **no salt** is used
  - Each computed hash yields 2^{10} comparisons
  - So expected work (hashes) is $2^{55}/2^{10} = 2^{45}$
- If **salt** is used
  - Expected work is 2^{55}
  - Each comparison requires a hash computation

Password Cracking: Case IV *

- Any of 1024 pwds in file, ***with*** dictionary
  - Prob. one or more pwd in dict.: $1 - (3/4)^{1024} \approx 1$
  - So, we ignore case where no pwd is in dictionary
- If **salt** is used, expected work less than 2^{22}
  - See book, or slide notes for details
  - Work $\approx$ size of dictionary / P(pwd in dictionary)
- What if **no salt** is used?
  - If dictionary hashes not precomputed, work is about $2^{19}/2^{10} = 2^9$

Guessing Through *L*

- Cannot prevent these
  - Otherwise, legitimate users cannot log in
- Make them slow
  - Backoff
  - Disconnection
  - Disabling
    - Be very careful with administrative accounts!
  - Jailing
    - Allow in, but restrict activities

Password Aging

- Force users to change passwords after some time has expired
  - How do you force users not to re-use passwords?
    - Record previous passwords
    - Block changes for a period of time
  - Give users time to think of good passwords
    - Don’t force them to change before they can log in
    - Warn them of expiration days in advance

Challenge-Response

- User, system share a secret function *f* (in practice, *f* is a known function with unknown parameters, such as a cryptographic key)

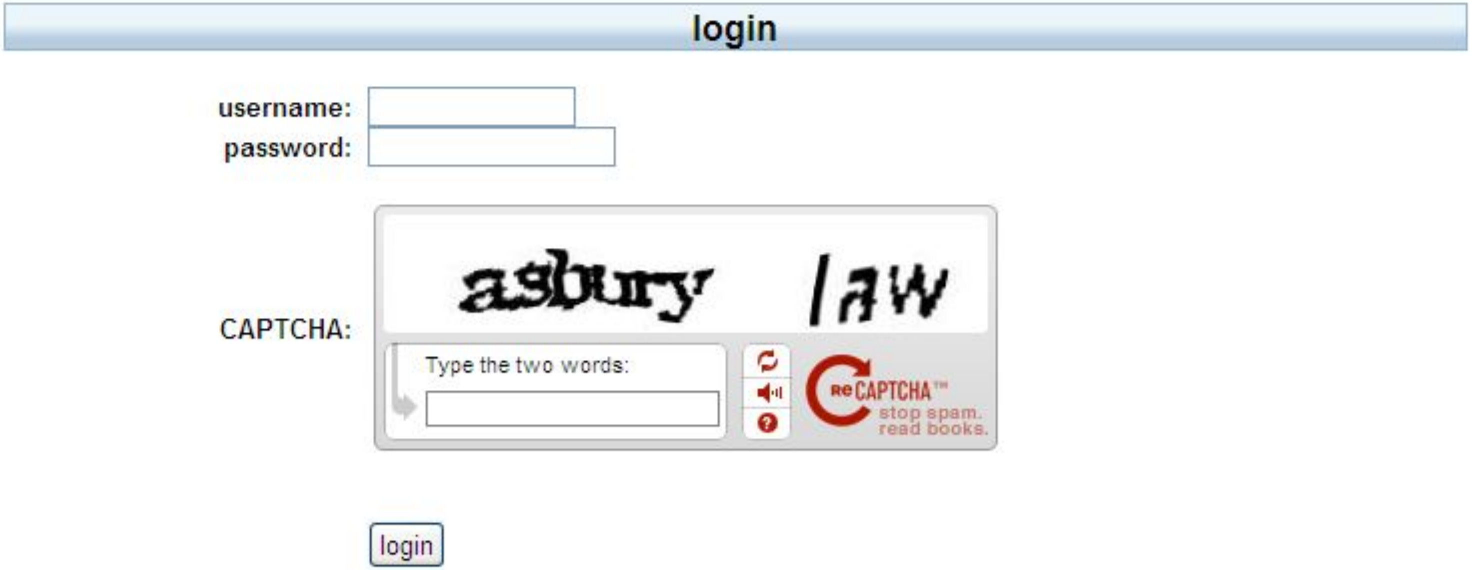
user $\xrightarrow{\textit{request to authenticate}}$ *system*

user $\xleftarrow[\textit{(the challenge)}]{\textit{random message r}}$ *system*

user $\xrightarrow[\textit{(the response)}]{f(r)}$ *system*

Pass Algorithms

- Challenge-response with the function *f* itself a secret
  - Challenge is a random string of characters
  - Response is some function of that string
  - Usually used in conjunction with fixed, reusable password



One-Time Passwords

- Password that can be used exactly *once*
  - After use, it is immediately invalidated
- Challenge-response mechanism
  - Challenge is number of authentications; response is password for that particular number
- Problems
  - Synchronization of user, system
  - Generation of good random passwords
  - Password distribution problem

S/Key

- One-time password scheme based on idea of Lamport
- *h* one-way hash function (MD5 or SHA-1, for example)
- User chooses initial seed *k*
- System calculates:

$$h(k) = k_1,\ h(k_1) = k_2,\ \ldots,\ h(k_{n-1}) = k_n$$

- Passwords are reverse order:

$$p_1 = k_n,\ p_2 = k_{n-1},\ \ldots,\ p_{n-1} = k_2,\ p_n = k_1$$

S/Key Protocol

System stores maximum number of authentications n, number of next authentication i, last correctly supplied password p_{i-1}.

$$user \xrightarrow{\{\ name\ \}} system$$

$$user \xleftarrow{\{\ i\ \}} system$$

$$user \xrightarrow{\{\ p_i\ \}} system$$

System computes $h(p_i) = h(k_{n-i+1}) = k_{n-i+2} = p_{i-1}$. If match with what is stored, system replaces p_{i-1} with p_i and increments i.

C-R and Dictionary Attacks

- Same as for fixed passwords
  - Attacker knows challenge *r* and response *f*(*r*); if *f* encryption function, can try different keys
    - May only need to know *form* of response; attacker can tell if guess correct by looking to see if deciphered object is of right form
    - Example: Kerberos Version 4 used DES, but keys had 20 bits of randomness; Purdue attackers guessed keys quickly because deciphered tickets had a fixed set of bits in some locations

Encrypted Key Exchange *

- Defeats off-line dictionary attacks
- Idea: random challenges enciphered, so attacker cannot verify correct decipherment of challenge
- Assume Alice, Bob share secret password *s*
- In what follows, Alice needs to generate a random public key *p* and a corresponding private key *q*
- Also, *k* is a randomly generated session key, and R_A and R_B are random challenges

EKE Protocol *

$$Alice \xrightarrow{\text{Alice} \,\|\, E_s(p)} Bob$$

$$Alice \xleftarrow{E_s(E_p(k))} Bob$$

Now Alice, Bob share a randomly generated secret session key k

$$Alice \xrightarrow{E_k(R_A)} Bob$$

$$Alice \xleftarrow{E_k(R_A R_B)} Bob$$

$$Alice \xrightarrow{E_k(R_B)} Bob$$

Something You Have

- Something in your possession
- Examples include following…
  - Car key
  - Laptop computer (or MAC address)
  - Password generator (next)
  - ATM card, smartcard, etc.

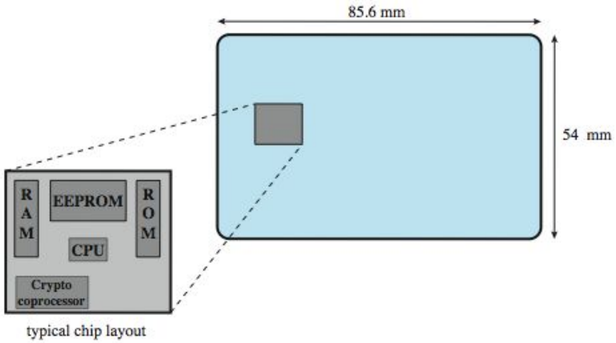
Hardware Support

- Token-based authentication
  - Used to compute response to challenge
    - May encipher or hash challenge
    - May require PIN from user
  - Object user possesses to authenticate, e.g.
    - memory card (magnetic stripe)
    - smartcard
- Temporally-based
  - Every minute (or so) different number shown
    - Computer knows what number to expect when
  - User enters number and fixed password

Memory Card

- store but do not process data
- magnetic stripe card, e.g. bank card
- electronic memory card
- used alone for physical access (e.g., hotel rooms)
- some with password/PIN (e.g., ATMs)
- Drawbacks of memory cards include:
  - need special reader
  - loss of token issues
  - user dissatisfaction (OK for ATM, not OK for computer access)

Smartcard

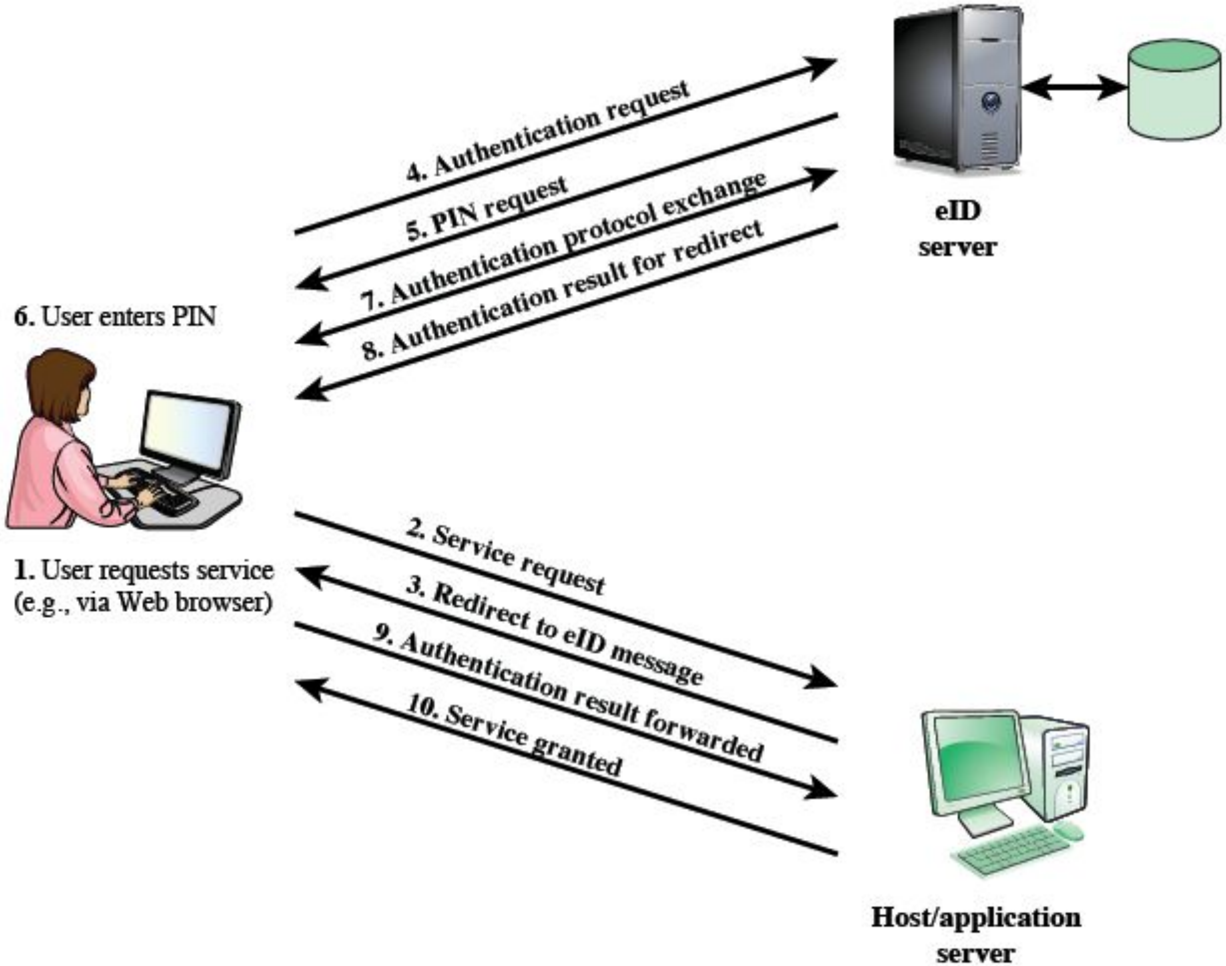


- credit-card like
- has own processor, memory, I/O ports
  - ROM, EEPROM, RAM memory
- executes protocol to authenticate with reader/computer
  - static: similar to memory cards
  - dynamic: passwords created every minute; entered manually by user or electronically
  - challenge-response: computer creates a random number; smart card provides its hash (similar to PK)
- also have USB dongles

Electronic identity cards *

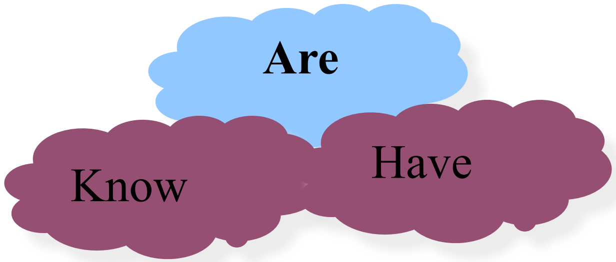
- An important application of smart cards
- A national e-identity (eID)
- Serves the same purpose as other national ID cards (e.g., a driver's licence)
  - Can provide stronger proof of identity
  - A German card
    - Personal data, Document number, Card access number (six digit random number), Machine readable zone (MRZ): the password
    - Uses: ePass (government use), eID (general use), eSign (can have private key and certificate)

User authentication with eID *



Something You Are

- Biometric
  - **"You are your key"** —Schneier
- Examples
  - Fingerprint
  - Handwritten signature
  - Facial recognition
  - Speech recognition
  - Gait (walking) recognition
  - "Digital doggie" (odor recognition)
  - Many more!



Why Biometrics?

- May be better than passwords
- But, cheap and reliable biometrics needed
  - Today, an active area of research
- Biometrics **are** used in security today
  - Thumbprint mouse
  - Palm print for secure entry
  - Fingerprint to unlock car door, etc.
- But biometrics not really that popular
  - Has not lived up to its promise/hype (yet?)

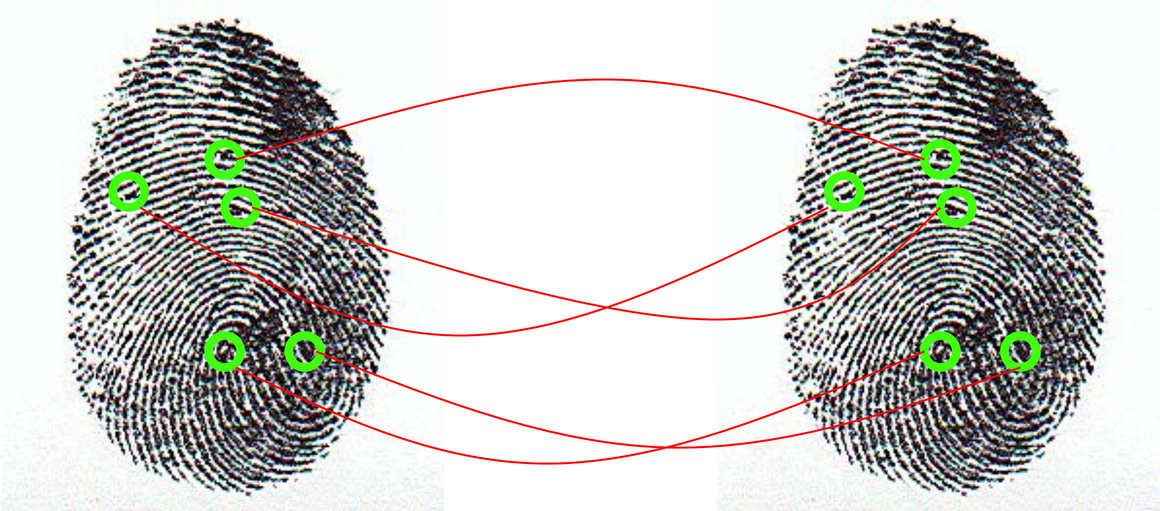
Biometrics: core idea

- Automated measurement of biological, behavioral features that identify a person
  - Fingerprints: optical or electrical techniques
    - Maps fingerprint into a graph, then compares with database
    - Measurements imprecise, so approximate matching algorithms used
  - Voices: speaker verification or recognition
    - Verification: uses statistical techniques to test hypothesis that speaker is who is claimed (speaker dependent)
    - Recognition: checks content of answers (speaker independent)

Fingerprint: Enrollment

- Capture image of fingerprint
- Enhance image
- Identify "points"

Fingerprint: Recognition



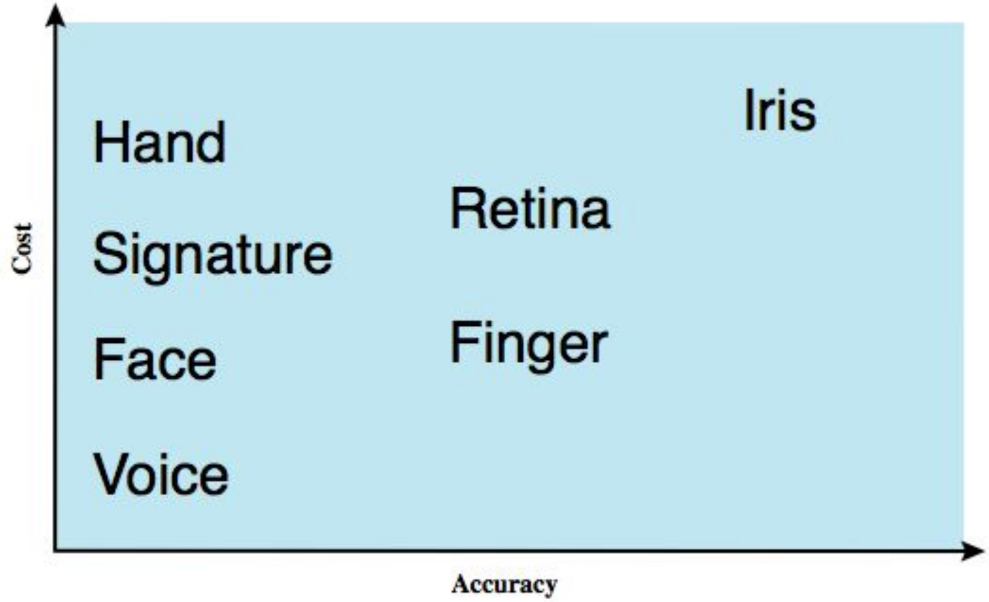
- Extracted points are compared with information stored in a database
- Is it a statistical match?
- Aside: Do identical twins' fingerprints differ?

Other Characteristics

- Can use several other characteristics
  - Eyes: patterns in irises unique
    - Measure patterns, determine if differences are random; or correlate images using statistical tests
  - Faces: image, or specific characteristics like distance from nose to chin
    - Lighting, view of face, other noise can hinder this
  - Keystroke dynamics: believed to be unique
    - Keystroke intervals, pressure, duration of stroke, where key is struck
    - Statistical tests used

Biometric authentication

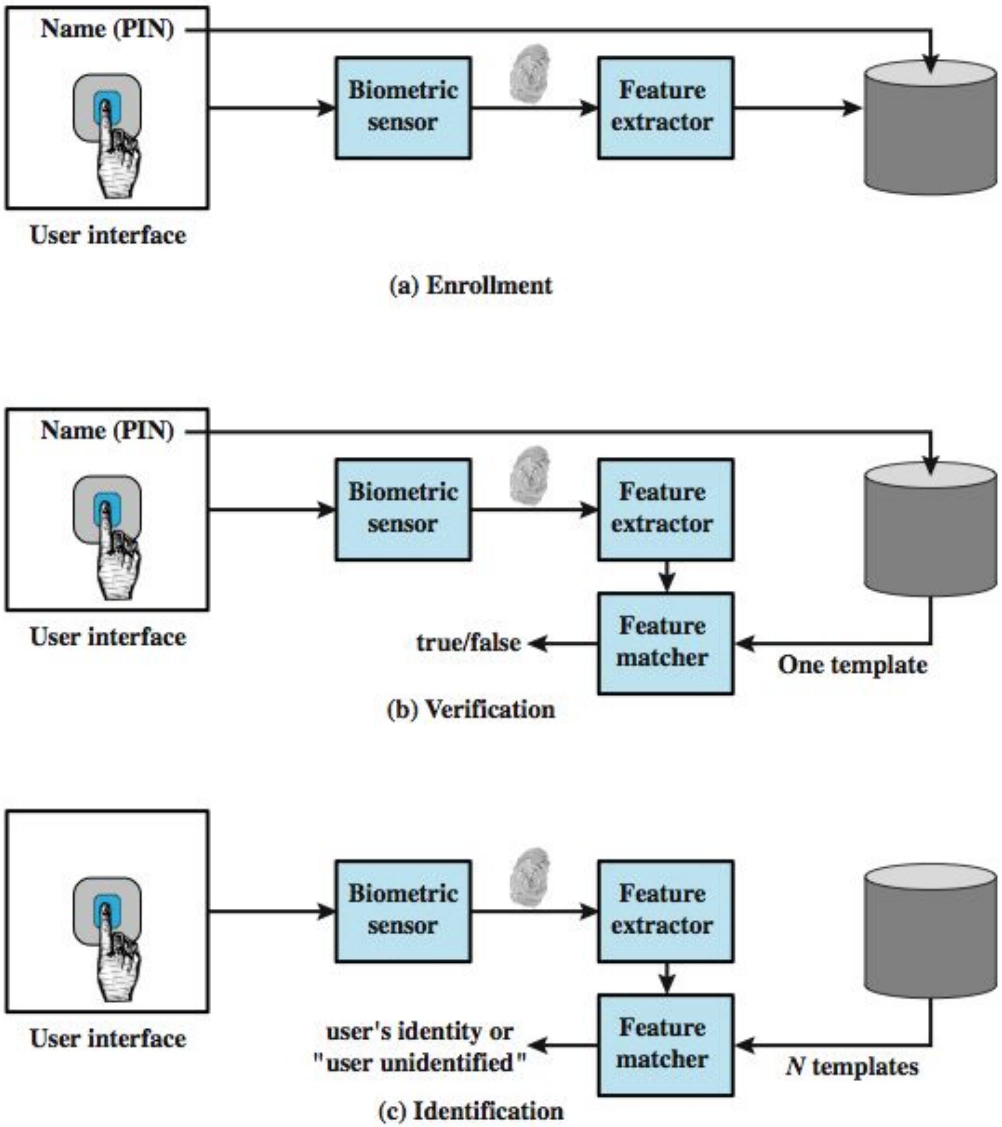
- Authenticate user based on one of their physical characteristics:
  - facial
  - fingerprint
  - hand geometry
  - retina pattern
  - iris
  - signature
  - voice



Operation of a biometric system

Verification *is analogous to user login via a smart card and a PIN*

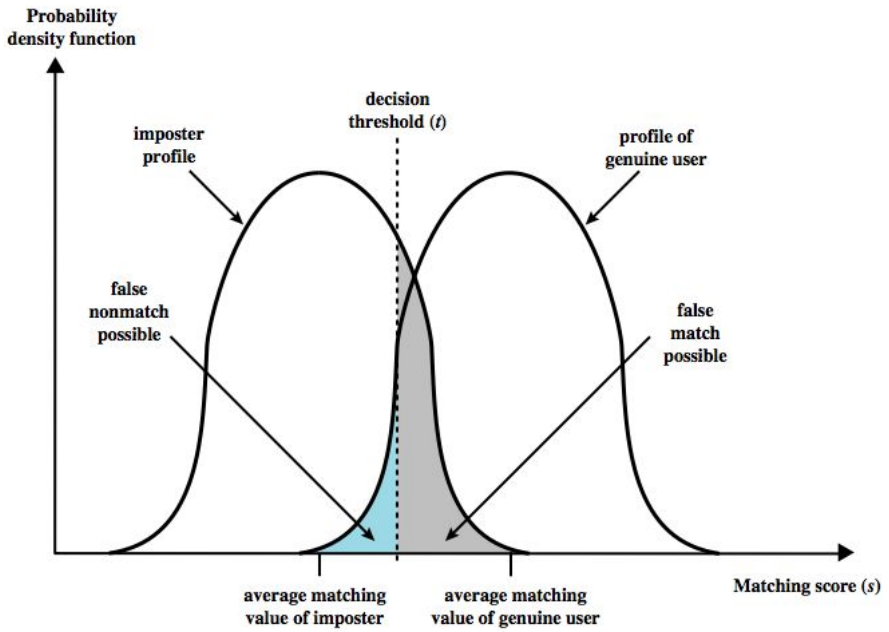
Identification *is biometric info but no IDs; system compares with stored templates*



Biometric Accuracy *

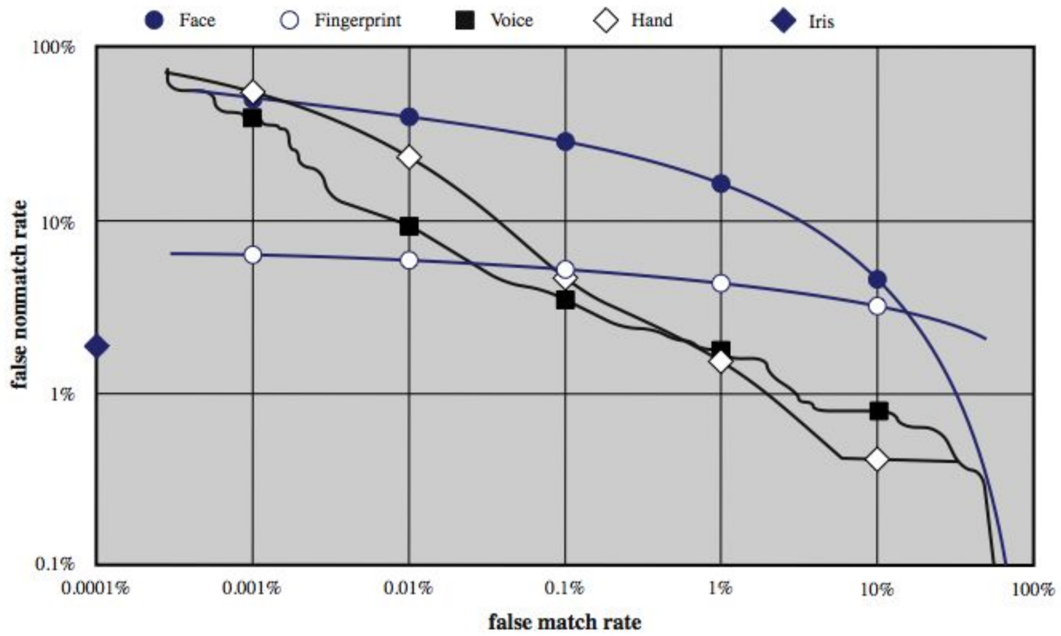
* Further reading
(Stallings textbook)

- Palm print The system generates a matching score (a number) that quantifies similarity between the input and the stored template
- Concerns: sensor noise and detection inaccuracy
- Problems of false match/false non-match



Biometric Accuracy *

- Can plot characteristic curve (2,000,000 comparisons)
- Pick threshold balancing error rates



Cautions

- These can be fooled!
  - Assumes biometric device accurate *in the environment it is being used in!*
  - Transmission of data to validator is tamperproof, correct

Biometrics: The Bottom Line

- Biometrics are hard to forge
- But attacker could
  - Steal Alice’s thumb
  - Photocopy Bob’s fingerprint, eye, etc.
  - Subvert software, database, “trusted path” …
- And how to revoke a “broken” biometric?
- **Biometrics are not foolproof**
- Biometric use is relatively limited today
- That should change in the (near?) future

Location – Just a brief

- If you know where user is, validate identity by seeing if person is where the user is
  - Requires special-purpose hardware to locate user
    - GPS (global positioning system) device gives location signature of entity
    - Host uses LSS (location signature sensor) to get signature for entity

Multiple Methods

- Example: “where you are” also requires entity to have LSS and GPS, so also “what you have”
- Can assign different methods to different tasks
  - As users perform more and more sensitive tasks, must authenticate in more and more ways (presumably, more stringently) File describes authentication required
    - Also includes controls on access (time of day, *etc*.), resources, and requests to change passwords
  - Pluggable Authentication Modules

PAM

- Idea: when program needs to authenticate, it checks central repository for methods to use
- Library call: *pam_authenticate*
  - Accesses file with name of program in */etc/pam_d*
- Modules do authentication checking
  - *sufficient*: succeed if module succeeds
  - *required*: fail if module fails, but all required modules executed before reporting failure
  - *requisite*: like *required*, but don't check all modules
  - *optional*: invoke only if all previous modules fail

Example PAM File

```
auth sufficient /usr/lib/pam_ftp.so
auth required   /usr/lib/pam_unix_auth.so use_first_pass
auth required   /usr/lib/pam_listfile.so onerr=succeed \
            item=user sense=deny file=/etc/ftpusers
```

For ftp:

1. If user “anonymous”, return okay; if not, set PAM_AUTHTOK to password, PAM_RUSER to name, and fail
2. Now check that password in PAM_AUTHTOK belongs to that of user in PAM_RUSER; if not, fail
3. Now see if user in PAM_RUSER named in /etc/ftpusers; if so, fail; if error or not found, succeed

*Extended Material **

Kerberos authentication protocol

Material sources: History & some general info from Wiki;

Details on Kerberos versions 4&5 from Stallings Text and slides

Kerberos

- A computer network authentication protocol
  - which allows nodes communicating over a non-secure network to prove their identity to one another in a secure manner.
  - aimed primarily at a client-server model, and it provides mutual authentication -- both the user and the server verify each other's identity. Messages are protected against eaves dropping & replay attacks.
  - Kerberos builds on SKC and requires a trusted third party, and optionally may use public-key cryptography during certain phases of authentication.

Kerberos

- History [Wiki]
  - named after the character *Kerberos* (or *Cerberus*), the ferocious three-headed guard dog of Hades (from Greek mythology)
  - MIT developed Kerberos in 1988 to protect network services provided by Project Athena.
  - 1st version was primarily designed by Steve Miller and Clifford Neuman based on the earlier Needham–Schroeder symmetric-key protocol. Ver 1 - 3 were experimental, internal.
  - Kerberos version 4, the first public version, was released on January 24, 1989.
  - Neuman and John Kohl published v5 in 1993 with the intention of overcoming existing limitations and security problems. Version 5 appeared as RFC 1510, which was then made obsolete by RFC 4120 in 2005. In 2005, the Internet Engineering Task Force (IETF) Kerberos working group updated specifications.

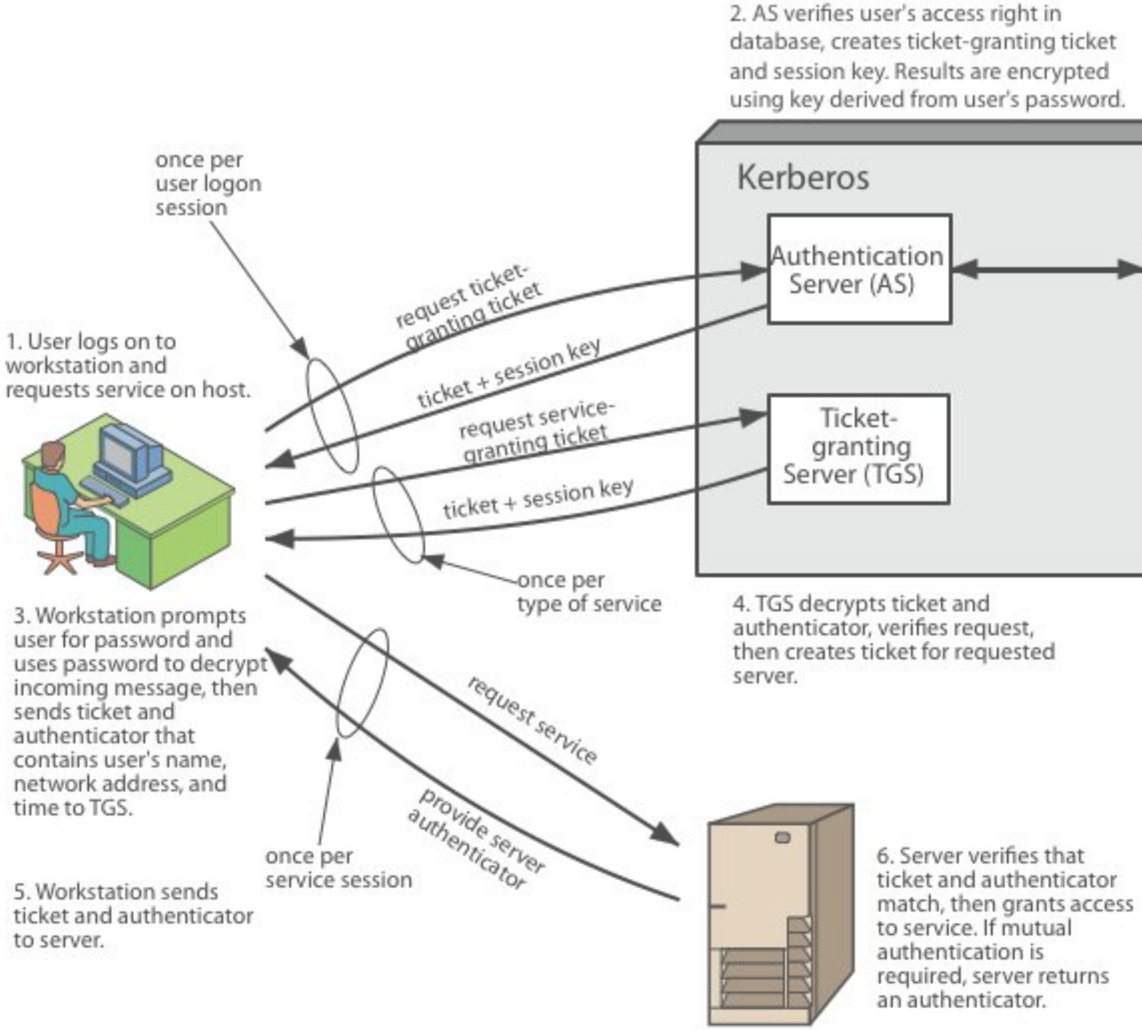
Idea

- Ticket
  - Issuer vouches for identity of requester of service
  - Identifies sender
- Key Distribution Center (KDC) combines two severs:
  - Authentication Server, AS (Also, Kerberos server)
  - Ticket Granting Server, TGS
- User *u* authenticates to AS
  - Obtains ticket $T_{u,TGS}$ for ticket granting service (TGS)
- User *u* wants to use service *s*:
  - User sends authenticator A_u, ticket $T_{u,TGS}$ to TGS asking for ticket for service
  - TGS sends ticket $T_{u,s}$ to user
  - User sends A_u, $T_{u,s}$ to server as request to use *s*

Kerberos v4 Overview

- a basic third-party authentication scheme
- have an Authentication Server (AS)
  - users initially negotiate with AS to identify self
  - AS provides a non-corruptible authentication credential (ticket granting ticket TGT)
- have a Ticket Granting server (TGS)
  - users subsequently request access to other services from TGS on basis of users TGT
- using a complex protocol using DES

Kerberos 4 Overview



Kerberos v4 Dialogue

(1) **C → AS** $ID_c \| ID_{tgs} \| TS_1$

(2) **AS → C** $E(K_c, [K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$

$$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3) **C → TGS** $ID_v \| Ticket_{tgs} \| Authenticator_c$

(4) **TGS → C** $E(K_{c,tgs}, [K_{c,v} \| ID_v \| TS_4 \| Ticket_v])$

$$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$$

$$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4])$$

$$Authenticator_c = E(K_{c,tgs}, [ID_C \| AD_C \| TS_3])$$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

(5) **C → V** $Ticket_v \| Authenticator_c$

(6) **V → C** $E(K_{c,v}, [TS_5 + 1])$ (for mutual authentication)

$$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4])$$

$$Authenticator_c = E(K_{c,v}, [ID_C \| AD_C \| TS_5])$$

(c) Client/Server Authentication Exchange to obtain service

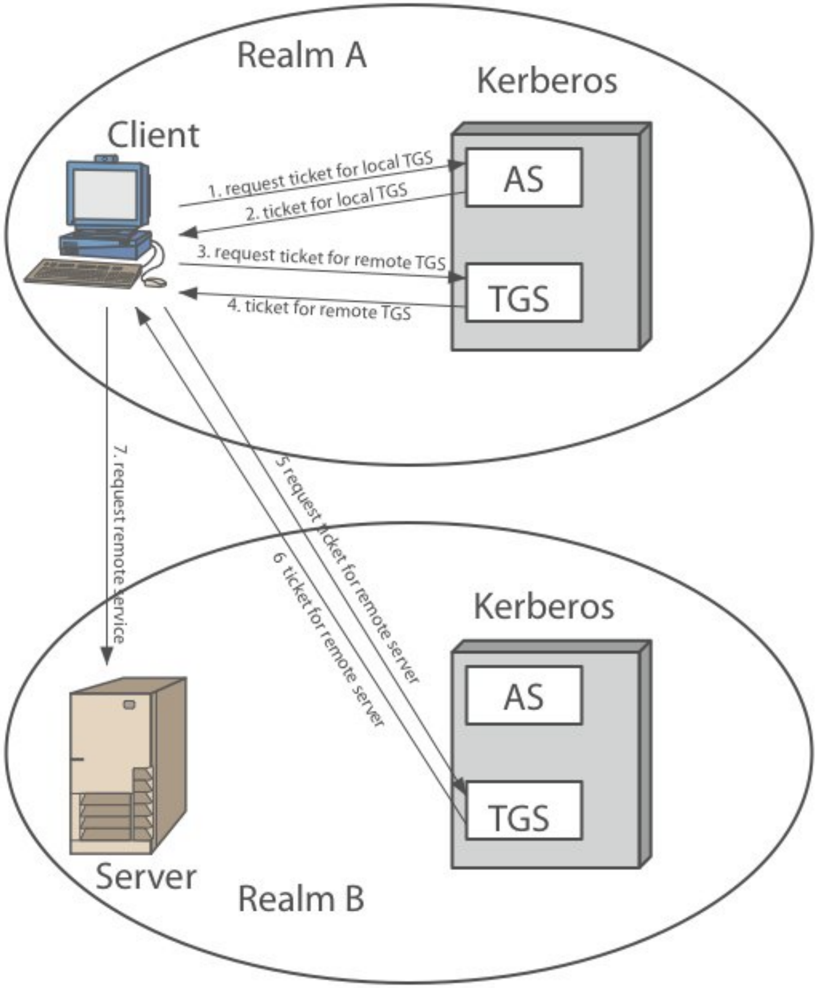
Kerberos Version 5

- developed in mid 1990's
- specified as Internet standard RFC 1510
- provides improvements over v4
  - addresses environmental shortcomings
    - encryption alg, network protocol, byte order, ticket lifetime, authentication forwarding, interrealm auth
  - and technical deficiencies
    - double encryption, non-std mode of use, session keys, password attacks

Kerberos Realms

- a Kerberos environment consists of:
  - a Kerberos server
  - a number of clients, all registered with server
  - application servers, sharing keys with server
- this is termed a realm
  - typically a single administrative domain
- if have multiple realms, their Kerberos servers must share keys and trust

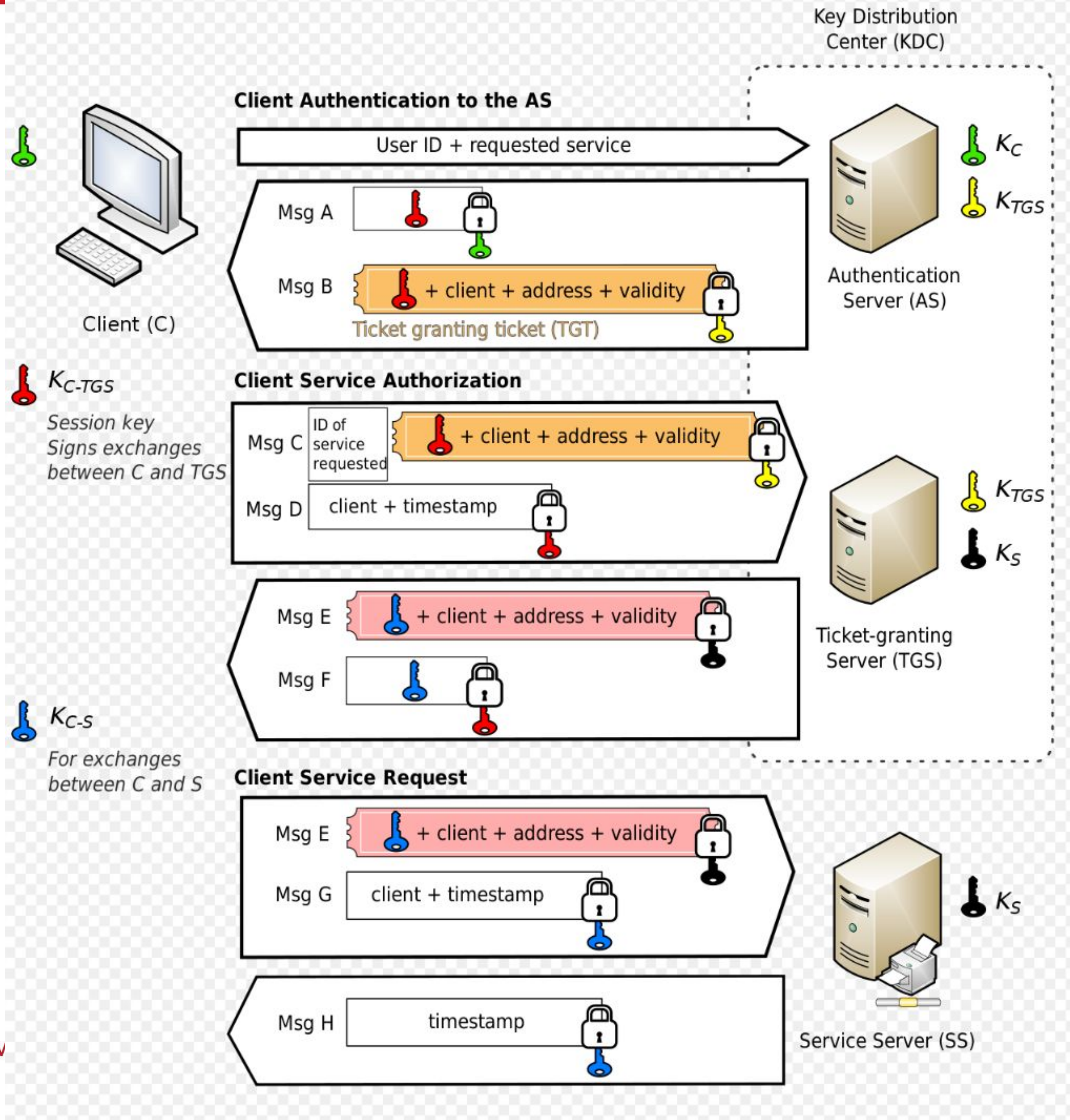
Kerberos Realms



Protocol

[From Wiki]

- Client Authentication to the AS
- Client Service Authorization
- Client Service Request



Kerberos v5 Dialogue

$$\textbf{(1) C} \rightarrow \textbf{AS} \quad \text{Options} \,\|\, ID_c \,\|\, Realm_c \,\|\, ID_{tgs} \,\|\, Times \,\|\, Nonce_1$$

$$\textbf{(2) AS} \rightarrow \textbf{C} \quad Realm_c \,\|\, ID_C \,\|\, Ticket_{tgs} \,\|\, \mathrm{E}(K_c, [K_{c,tgs} \,\|\, Times \,\|\, Nonce_1 \,\|\, Realm_{tgs} \,\|\, ID_{tgs}])$$

$$Ticket_{tgs} = \mathrm{E}(K_{tgs}, [Flags \,\|\, K_{c,tgs} \,\|\, Realm_c \,\|\, ID_C \,\|\, AD_C \,\|\, Times])$$

(a) Authentication Service Exchange to obtain ticket-granting ticket

$$\textbf{(3) C} \rightarrow \textbf{TGS} \quad \text{Options} \,\|\, ID_v \,\|\, Times \,\|\,\|\, Nonce_2 \,\|\, Ticket_{tgs} \,\|\, Authenticator_c$$

$$\textbf{(4) TGS} \rightarrow \textbf{C} \quad Realm_c \,\|\, ID_C \,\|\, Ticket_v \,\|\, \mathrm{E}(K_{c,tgs}, [K_{c,v} \,\|\, Times \,\|\, Nonce_2 \,\|\, Realm_v \,\|\, ID_v])$$

$$Ticket_{tgs} = \mathrm{E}(K_{tgs}, [Flags \,\|\, K_{c,tgs} \,\|\, Realm_c \,\|\, ID_C \,\|\, AD_C \,\|\, Times])$$

$$\mathrm{Ticket}_v = \mathrm{E}(\mathrm{K}_v, [\mathrm{Flags} \,\|\, \mathrm{K}_{c,v} \,\|\, \mathrm{Realm}_c \,\|\, \mathrm{ID}_C \,\|\, \mathrm{AD}_C \,\|\, \mathrm{Times}])$$

$$Authenticator_c = \mathrm{E}(K_{c,tgs}, [ID_C \,\|\, Realm_c \,\|\, TS_1])$$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

$$\textbf{(5) C} \rightarrow \textbf{V} \quad \text{Options} \,\|\, \text{Ticket}_v \,\|\, \text{Authenticator}_c$$

$$\textbf{(6) V} \rightarrow \textbf{C} \quad \mathrm{E}_{K_{c,v}} [\, \mathrm{TS}_2 \,\|\, \text{Subkey} \,\|\, \text{Seq\#} \,]$$

$$\mathrm{Ticket}_v = \mathrm{E}(\mathrm{K}_v, [\mathrm{Flags} \,\|\, \mathrm{K}_{c,v} \,\|\, \mathrm{Realm}_c \,\|\, \mathrm{ID}_C \,\|\, \mathrm{AD}_C \,\|\, \mathrm{Times}])$$

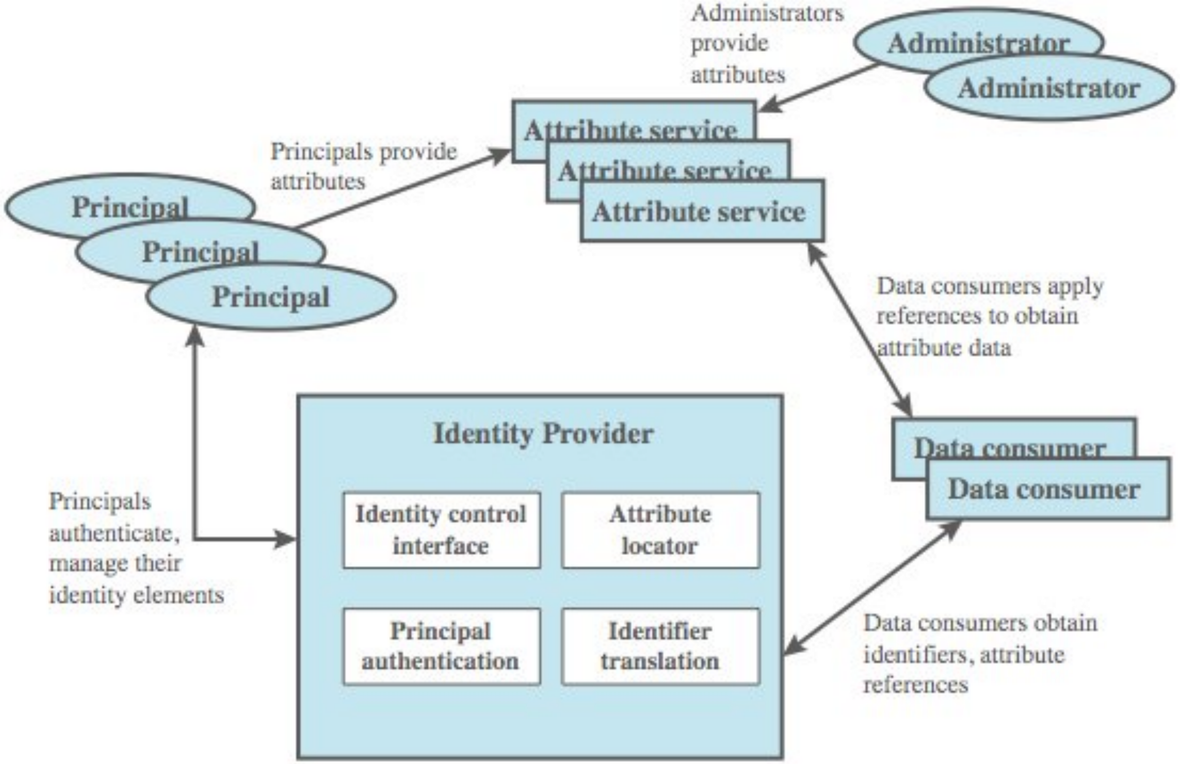
$$Authenticator_c = \mathrm{E}(K_{c,v}, [ID_C \,\|\, Realm_c \,\|\, TS_2 \,\|\, Subkey \,\|\, Seq\#])$$

(c) Client/Server Authentication Exchange to obtain service

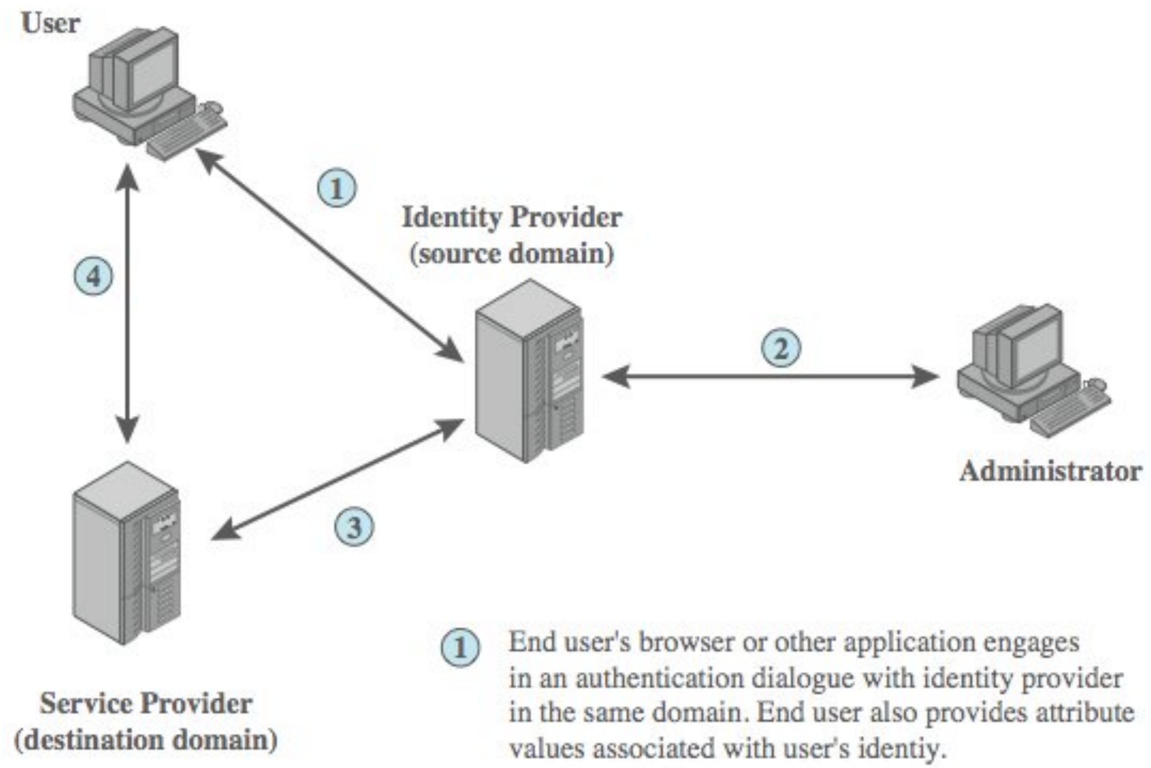
Federated Identity Management

- use of common identity management scheme
  - across multiple enterprises & numerous applications
  - supporting many thousands, even millions of users
- principal elements are:
  - authentication, authorization, accounting, provisioning, workflow automation, delegated administration, password synchronization, self-service password reset, federation
- Kerberos contains many of these elements

Identity Management



Identity Federation

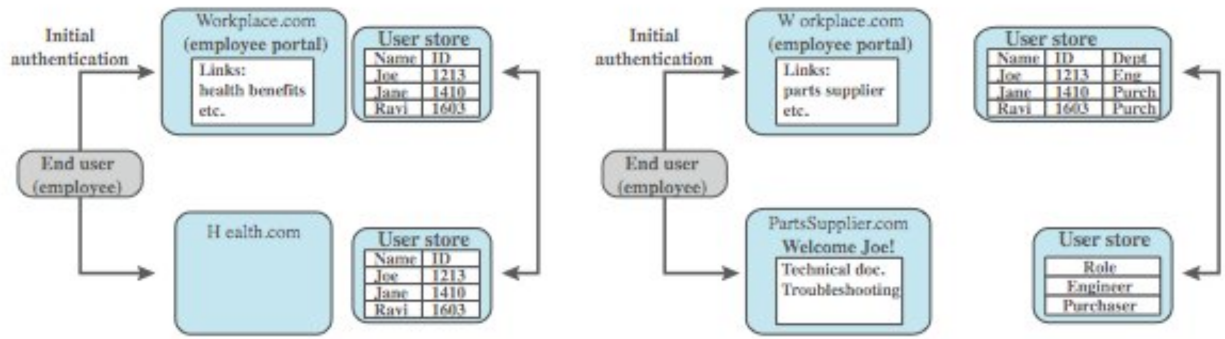


1. End user's browser or other application engages in an authentication dialogue with identity provider in the same domain. End user also provides attribute values associated with user's identiy.
2. Some attributes associated with an identity, such as allowable roles, may be provided by an administrator in the same domain.
3. A service provider in a remote domain, which the user wishes to access, obtains identity information, authentication information, and associated attributes from the identity provider in the source domain.
4. Service provider opens session with remote user and enforces access control restrictions based on user's identity and attributes.

Standards Used

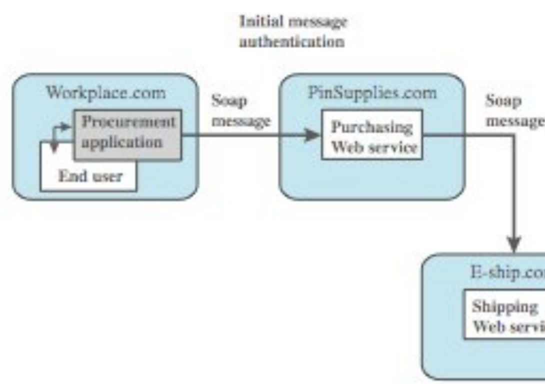
- Security Assertion Markup Language (SAML)
  - XML-based language for exchange of security information between online business partners
- part of OASIS (Organization for the Advancement of Structured Information Standards) standards for federated identity management
  - e.g. WS-Federation for browser-based federation
- need a few mature industry standards

Federated Identity Examples



(a) Federation based on account linking

(b) Federation based on roles



(b) Chained Web Services

FIM vs. SSO

- SSO: Single Sign-On
  - Allows users to access multiple web applications at once, using just one set of credentials.
    - Beyond the workforce, companies can utilize SSO to help customers access various sections of one account.
- FIM
  - As a tool, SSO fits within the broader model of FIM.
  - The key difference between SSO and FIM is while SSO is designed to authenticate a single credential across various systems within one organization, federated identity management systems offer single access to a number of applications across various enterprises.

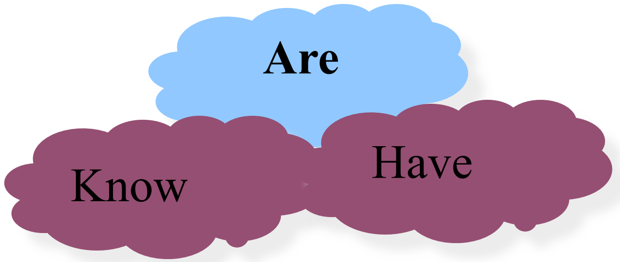
*Extended Material **
Biometrics

Slides borrowed from Mark Stamp's web
https://www.cs.sjsu.edu/~stamp/infosec/powerpoint/

Something You Are

- Biometric
  - **“You are your key”** —Schneier

- Examples
  - Fingerprint
  - Handwritten signature
  - Facial recognition
  - Speech recognition
  - Gait (walking) recognition
  - “Digital doggie” (odor recognition)
  - Many more!



Ideal Biometric

- **Universal** —applies to (almost) everyone
  - In reality, no biometric applies to everyone
- **Distinguishing** —distinguish with certainty
  - In reality, cannot hope for 100% certainty
- **Permanent** —physical characteristic being measured never changes
  - In reality, OK if it to remains valid for long time
- **Collectable** —easy to collect required data
  - Depends on whether subjects are cooperative
- Also, safe, user-friendly, and ???

Identification vs Authentication

- **Identification** —Who goes there?
  - Compare **one-to-many**
  - Example: FBI fingerprint database
- **Authentication** —Are you who you say you are?
  - Compare **one-to-one**
  - Example: Thumbprint mouse
- Identification problem is more difficult
  - More “random” matches since more comparisons
- We are (mostly) interested in authentication

Enrollment vs Recognition

- Enrollment phase
  - Subject's biometric info put into database
  - Must carefully measure the required info
  - OK if slow and repeated measurement needed
  - Must be very precise
  - May be a weak point in real-world use
- Recognition phase
  - Biometric detection, when used in practice
  - Must be quick and simple
  - But must be reasonably accurate

Cooperative Subjects?

- Authentication — cooperative subjects
- Identification — uncooperative subjects
- For example, facial recognition
  - Used in Las Vegas casinos to detect known cheaters (also, terrorists in airports, etc.)
  - Often, less than ideal enrollment conditions
  - Subject will try to confuse recognition phase
- Cooperative subject makes it much easier
  - We are focused on authentication
  - So, we can assume subjects are cooperative

Biometric Errors

- **Fraud rate** versus **insult rate**
  - Fraud — Trudy mis-authenticated as Alice
  - Insult — Alice not authenticated as Alice
- For any biometric, can decrease fraud or insult, but other one will increase
- For example
  - 99% voiceprint match ⇒ low fraud, high insult
  - 30% voiceprint match ⇒ high fraud, low insult
- **Equal error rate:** rate where fraud == insult
  - A way to ***compare*** different biometrics

Fingerprint History

- 1823 —Professor Johannes Evangelist Purkinje discussed 9 fingerprint patterns
- 1856 —Sir William Hershel used fingerprint (in India) on contracts
- 1880 —Dr. Henry Faulds article in *Nature* about fingerprints for ID
- 1883 —Mark Twain's *Life on the Mississippi* (murderer ID'ed by fingerprint)

Fingerprint History

- 1888 — Sir Francis Galton developed classification system
  - His system of "minutia" can be used today
  - Also verified that fingerprints do not change
- Some countries require fixed number of "points" (minutia) to match in criminal cases
  - In Britain, at least 15 points
  - In US, no fixed number of points

Fingerprint Comparison

- Examples of **loops**, **whorls**, and **arches**
- Minutia extracted from these features

Loop (double)

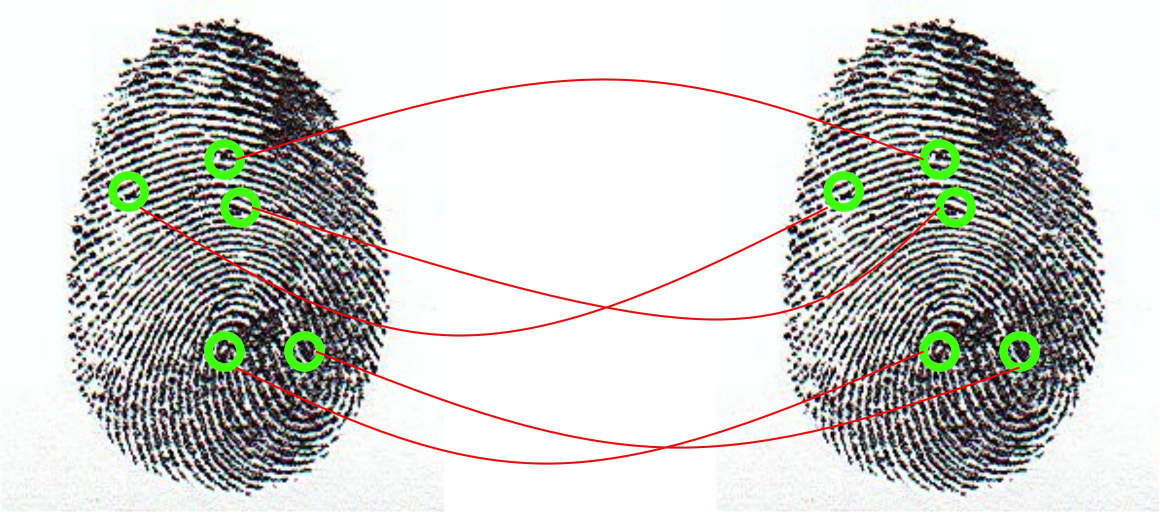
Whorl

Arch

Fingerprint: Enrollment

- Capture image of fingerprint
- Enhance image
- Identify “points”

Fingerprint: Recognition



- Extracted points are compared with information stored in a database
- Is it a statistical match?
- Aside: Do identical twins' fingerprints differ?

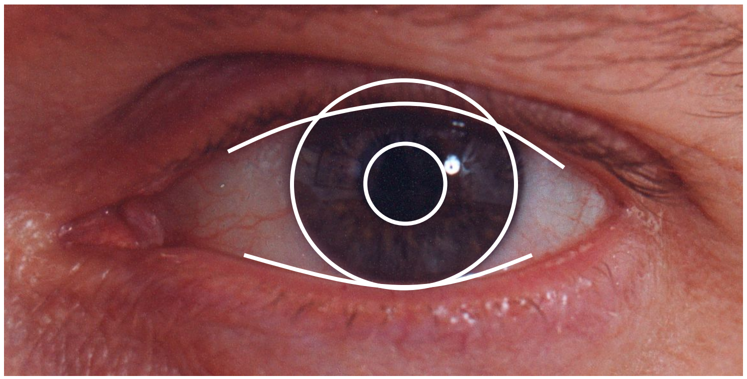
Hand Geometry

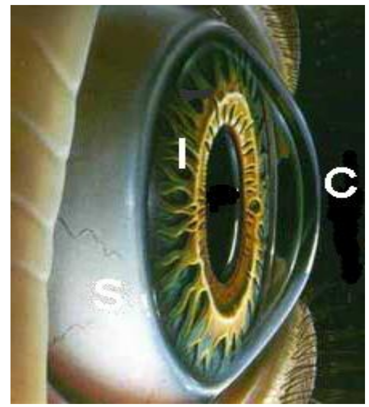
- A popular biometric
- Measures shape of hand
  - Width of hand, fingers
  - Length of fingers, etc.
- Human hands not so unique
- Hand geometry sufficient for many situations
- OK for authentication
- Not useful for ID problem

Hand Geometry

- Advantages
  - Quick — 1 minute for enrollment, 5 seconds for recognition
  - Hands are symmetric — so what?
- Disadvantages
  - Cannot use on very young or very old
  - Relatively high equal error rate

Iris Patterns





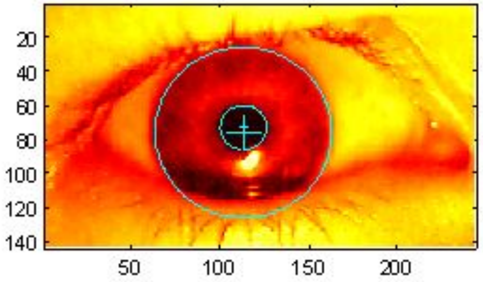
- Iris pattern development is “chaotic”
- Little or no genetic influence
- Even for identical twins, uncorrelated
- Pattern is stable through lifetime

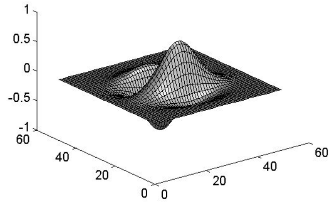
Iris Recognition: History

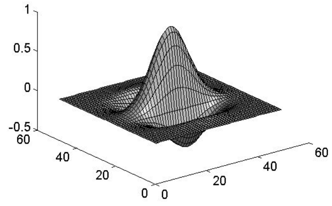
- 1936 — suggested by ophthalmologist
- 1980s — James Bond film(s)
- 1986 — first patent appeared
- 1994 — John Daugman patents new-and-improved technique
  - Patents owned by Iridian Technologies

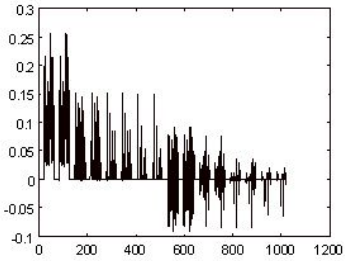
Iris Scan

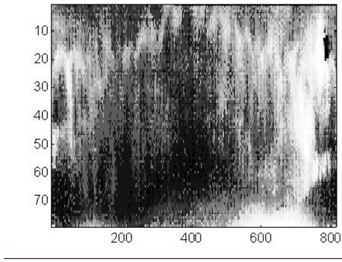
- Scanner locates iris
- Take b/w photo
- Use polar coordinates…
- 2-D wavelet transform
- Get 256 byte iris code











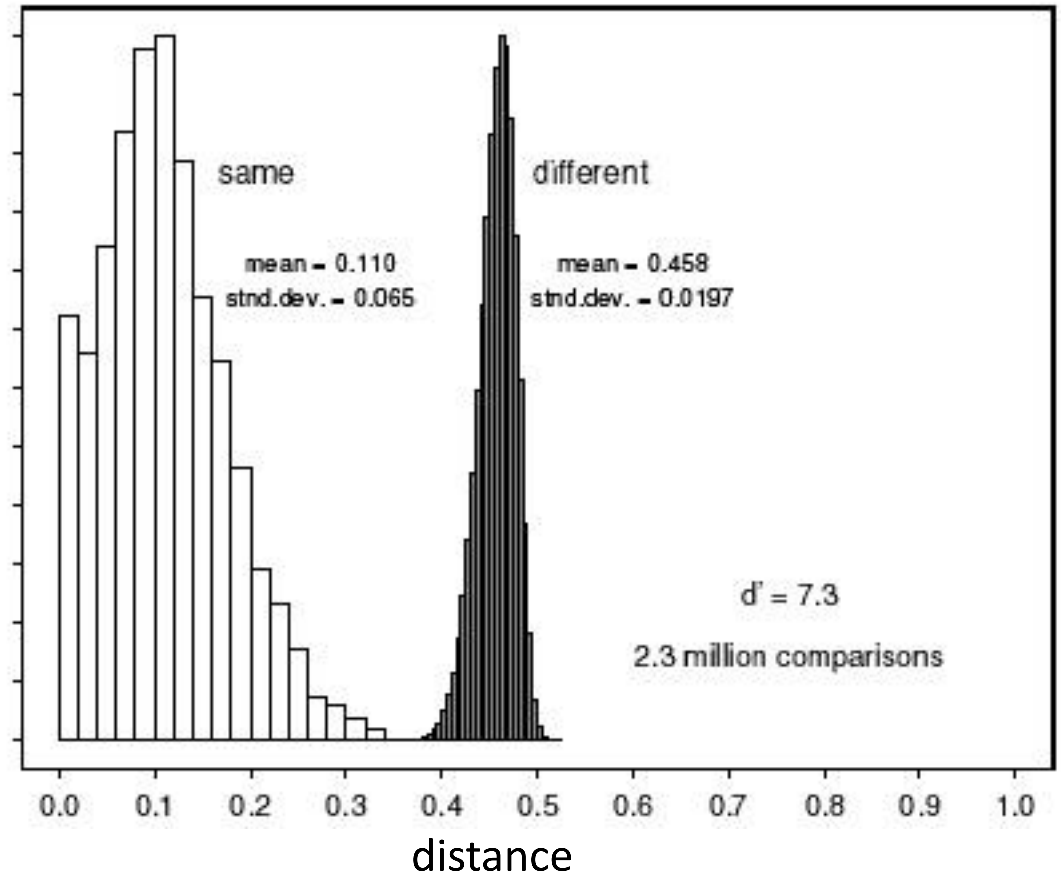
Measuring Iris Similarity

- Based on Hamming distance
- Define $d(x,y)$ to be
  - # of non-match bits / # of bits compared
  - $d(0010,0101) = 3/4$ and $d(101111,101001) = 1/3$
- Compute $d(x,y)$ on 2048-bit iris code
  - Perfect match is $d(x,y) = 0$
  - For same iris, expected distance is 0.08
  - At random, expect distance of 0.50
  - Accept iris scan as match if distance < 0.32

Iris Scan Error Rate

distance	Fraud rate
0.29	1 in $1.3*10^{10}$
0.30	1 in $1.5*10^{9}$
0.31	1 in $1.8*10^{8}$
0.32	1 in $2.6*10^{7}$
0.33	1 in $4.0*10^{6}$
0.34	1 in $6.9*10^{5}$
0.35	1 in $1.3*10^{5}$

== equal error rate



Attack on Iris Scan

- Good **photo** of eye can be scanned
  - Attacker could use photo of eye

- Afghan woman was authenticated by iris scan of old photo
  - Story can be found here
- To prevent attack, scanner could use light to be sure it is a "live" iris

Equal Error Rate Comparison

- Equal error rate (EER): fraud == insult rate
- **Fingerprint** biometrics used in practice have EER ranging from about 10^{-3} to as high as 5%
- **Hand geometry** has EER of about 10^{-3}
- In theory, **iris scan** has EER of about 10^{-6}
  - Enrollment phase may be critical to accuracy
- Most biometrics much worse than fingerprint!
- Biometrics useful for authentication…
  - …but for identification, not so impressive today

Biometrics: The Bottom Line

- Biometrics are hard to forge
- But attacker could
  - Steal Alice's thumb
  - Photocopy Bob's fingerprint, eye, etc.
  - Subvert software, database, "trusted path" …
- And how to revoke a "broken" biometric?
- **Biometrics are not foolproof**
- Biometric use is relatively limited today
- That should change in the (near?) future

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attentions!