Chapter 11: Authentication

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

Basics

- Authentication: binding of identity to subject
  - Identity is that of external entity (Alice, Bob, *etc*.)
  - Subject is computer entity (process, *etc*.)

Establishing Identity

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

Authentication System

- Quintuple (A, C, F, L, S)
  - A information that proves identity
  - C information stored on computer and used to validate authentication information
  - F complementation function; $f : A \rightarrow C$
  - L 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 clear text
  - If password file compromised, *all* passwords revealed
- Encipher file
  - Need to have decipherment, encipherment keys in memory
  - Reduces to previous problem
- Store one-way hash of password
  - If file read, attacker must still guess passwords or invert the hash

Example

- 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+*, ... }

Anatomy of Attacking

- 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:
  - Direct approach: as above
  - Indirect approach: as $l(a)$ succeeds iff $f(a) = c \in C$ for some c associated with an entity, compute $l(a)$

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 *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

Using Time

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
  - $N \geq TG/P = (365 \times 24 \times 60 \times 60) \times 10^4 / 0.5 = 6.31 \times 10^{11}$
  - 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

Approaches: 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

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 ("Dartmouth Medical Center/Mary Hitchcock memorial hospital"), ok here

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

Example: *passwd+*

- Provides little language to describe proactive checking
  - test length("$p") < 6
    - If password under 6 characters, reject it
  - test infile("/usr/dict/words", "$p")
    - If password in file /usr/dict/words, reject it
  - test !inprog("spell", "$p", "$p")
    - If password not in the output from program spell, given the password as input, reject it (because it's a properly spelled word)

Salting

- Goal: slow dictionary attacks that try to find *any* user's password
- Method: perturb hash function so that:
  - Parameter controls *which* hash function is used
  - Parameter (called a salt) differs for each user (password)
  - So given *n* password hashes, and therefore *n* salts, need to hash guess *n* for each password (increase the amount of guess by a factor of *n*)

Examples

- Vanilla UNIX method
  - Use DES to encipher the message of all 0's with password as key; iterate 25 times
  - Perturb E table in DES in one of 4096 ways
    - Each salt results in a different DES variant
    - The salt is chosen randomly

    Salt is *n*, password *p*, then the complementary information for p is: DES_n_p(block of 0's)

User	Password	Salt	Complementary function	Complementary information
Alice	12345	9	$DES\_9_{"12345"}$	$DES\_9_{"12345"}$("00000000")
Bob	asdfg	365	$DES\_365_{"asdfg"}$	$DES\_365_{"asdfg"}$ ("00000000")
Charlie	ikmnj	513	$DES\_513_{"ikmnj"}$	$DES\_513_{"ikmnj"}$ ("00000000")

- Alternate methods
  - Use salt as first part of input to hash function

Guessing Through *L*

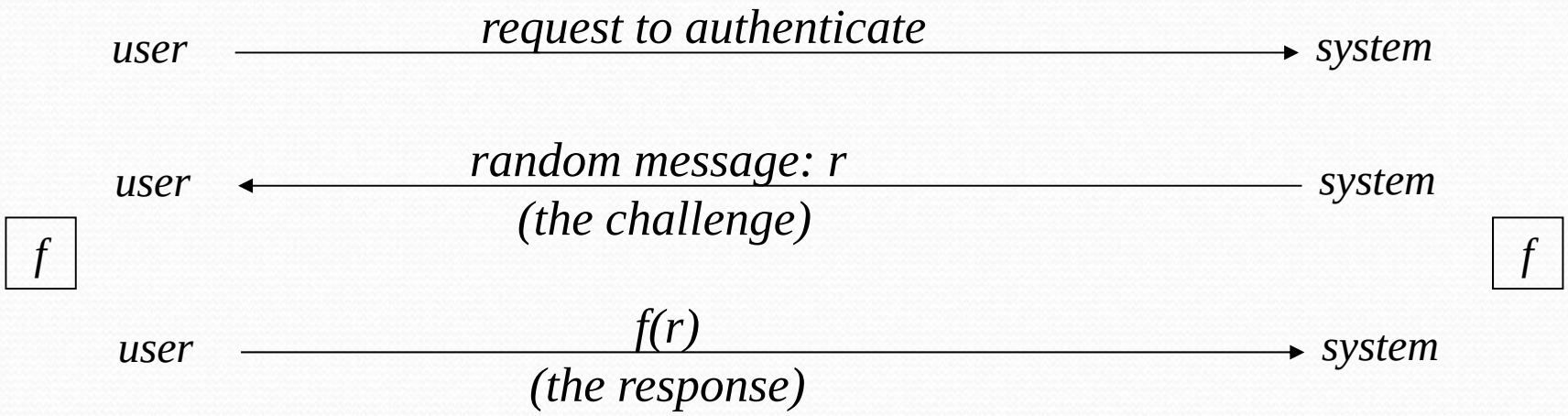
- Cannot prevent these
  - Otherwise, legitimate users cannot log in
- Make them slow
  - Backoff
  - Disconnection
  - Disabling
    - Be very careful with administrative accounts!
  - Jailing
    - Allow in, but restrict activities

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)



Pass Algorithms

- Challenge-response with the function *f* itself a secret
  - Example:
    - Challenge is a random string of characters such as “abcdefg”, “ageksidp”
    - Response is some function of that string such as “bdf”, “gkip”
  - Can alter algorithm based on ancillary information
    - Network connection is as above, dial-up might require “aceg”, “aesd”
  - 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 and 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:

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

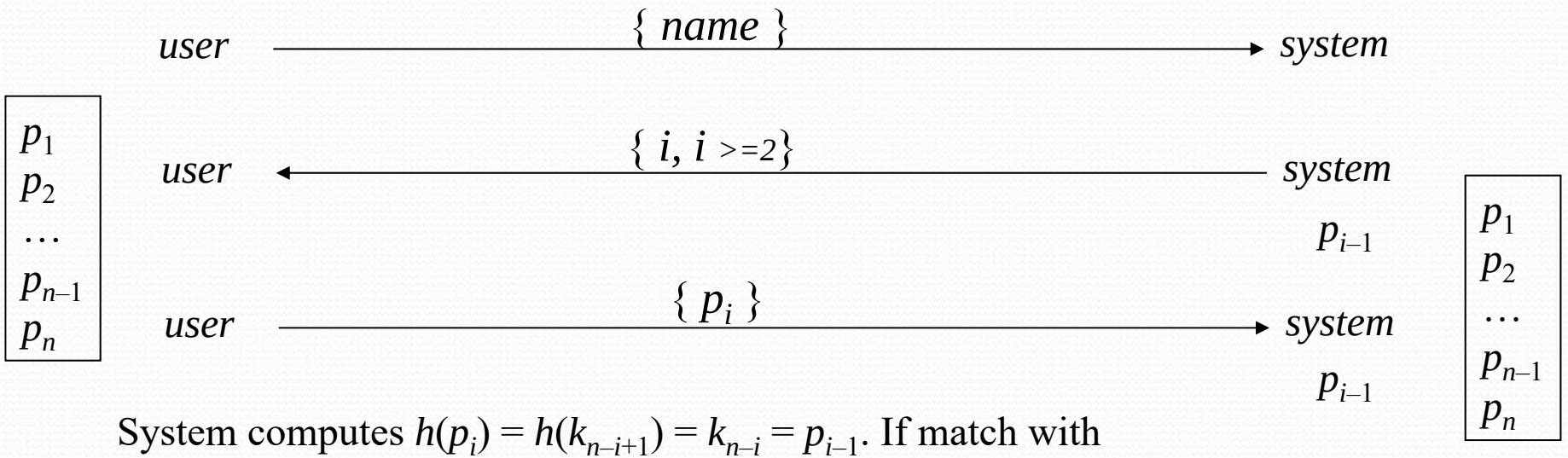
- Passwords are reverse order:

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

$$p_1 = k_n,\ h(p_2) = h(k_{n-1}) = k_n = p_1\ ,\ h(p_3) = h(k_{n-2}) = k_{n-1} = p_2\ ,\ldots$$

S/Key Protocol

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



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

Hardware Support

- Token-based
  - Used to compute response to challenge
    - May encipher or hash challenge
    - May require PIN from user, used as cryptographic key or is combined with the challenge to produce the response
- Temporally-based
  - Every minute (or so) different number shown
    - Computer knows what number to expect when
  - User enters number and fixed password

Biometrics

- 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)

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

Biometrics

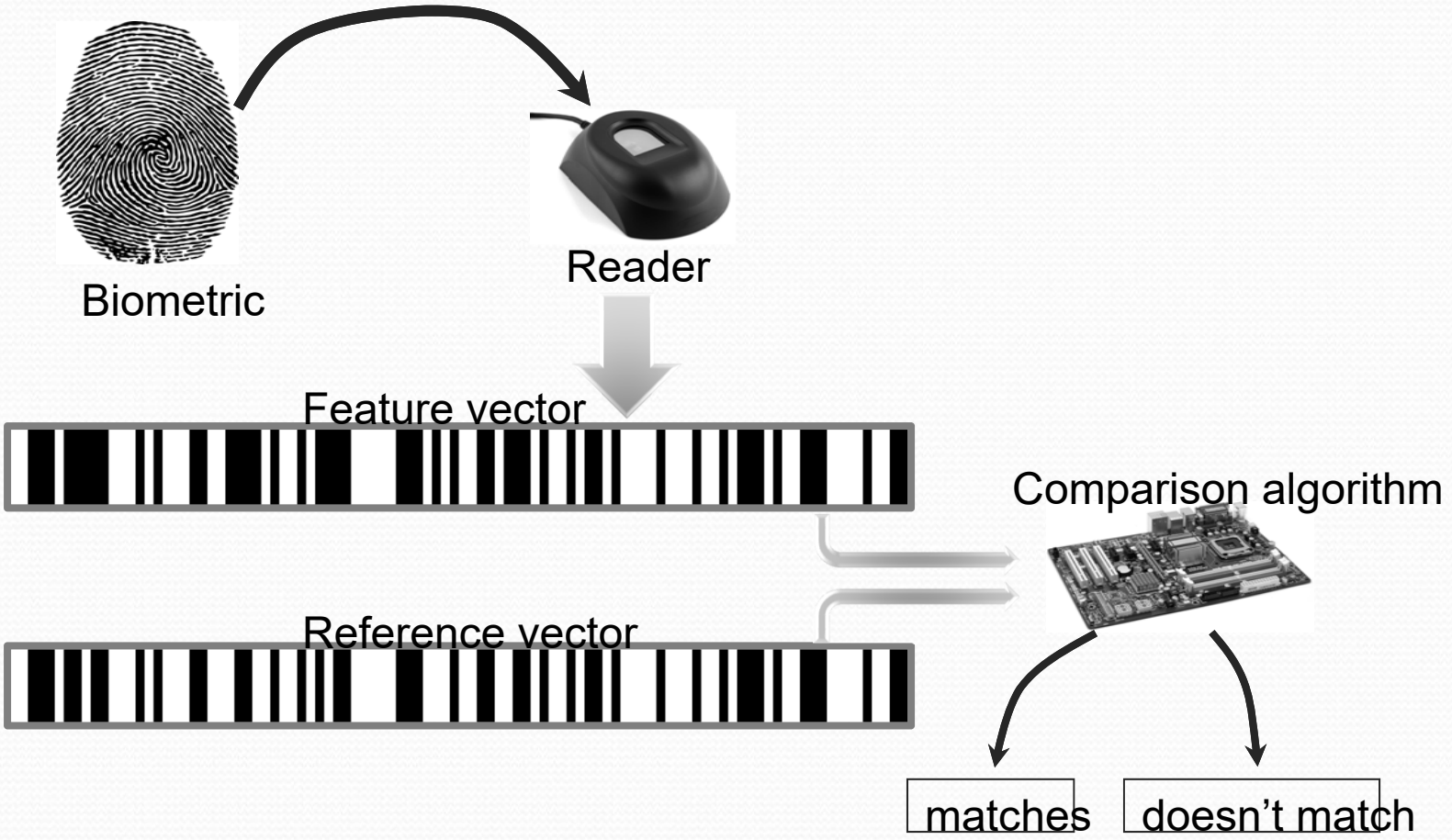
- **Biometric** refers to any measure used to uniquely identify a person based on biological or physiological traits.

- Generally, biometric systems incorporate some sort of sensor or scanner to read in biometric information and then compare this information to stored templates of accepted users before granting access.

Requirements for Biometric Identification

- **Universality.** Almost every person should have this characteristic.
- **Distinctiveness.** Each person should have noticeable differences in the characteristic.
- **Permanence.** The characteristic should not change significantly over time.
- **Collectability.** The characteristic should have the ability to be effectively determined and quantified.

Biometric Identification



Candidates for Biometric IDs

- Fingerprints
- Retinal/iris scans
- DNA
- "Blue-ink" signature
- Voice recognition
- Face recognition
- Gait recognition

Public domain image from
http://commons.wikimedia.org/wiki/File:Fingerprint_Arch.jpg

Public domain image from
http://commons.wikimedia.org/wiki/File:Retinal_scan_securimetrics.jpg

Public domain image from
http://commons.wikimedia.org/wiki/File:CBP_chemist_reads_a_DNA_profile.jpg

- Let us consider how each of these scores in terms of universality, distinctiveness, permanence, and collectability…

Cautions

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

Location

- 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

Multi-Factor Authentication

- Example: "where you are" also requires entity to have LSS and GPS, so also "what you have"
- Example 2: password + hardware-based challenge-response
- 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

Key Points

- Authentication is not cryptography
  - You have to consider system components
- Passwords are here to stay
  - They provide a basis for most forms of authentication
- Protocols are important
  - They can make masquerading harder
- Authentication methods can be combined