NC STATE UNIVERSITY Computer Science

CSC 405
Introduction to Computer Security

Topic 4. Security in Conventional Operating Systems -- Part II

Basic Concepts of UNIX Access Control: Users, Groups, Files, Processes

- Each user has a unique UID
- Users belong to multiple groups
- Each file has 12 permission bits
  - read/write/execute for user, group, and others,
  - suid, sgid, sticky
- Directories are one kind of files
- Files/directories form a hierarchy
- Processes are associated with uid/gid pairs, which are used to determine access

Permissions Bits on Files (Non-directories)

- Having execute but not read, can one run a file?
- Having execute but not read, can one run a script file?
- Having read but not execute, can one run a script file?
- Suid & sgid for executable files (very important)
- Suid & sgid for non-executable files (mostly no effect)
- Sticky bit (no effect now)

Permission Bits on Directories

- The execution bit controls traversing a directory
  - chdir to a directory requires execution
- The read bit allows one to show file names in a directory
- Accessing a file needs execution to all directories along the path
- Deleting/creating a file under a directory requires write & execute for the directory
- suid has no effect; sgid makes new files/directories inherit the same group id
- Sticky bit for directories: when set, only owner of a file (and super user) can delete

Some Examples

- What permissions are needed to access a file/directory?
  - read a file: /dir1/dir2/file3
  - write a file: /dir1/dir2/file3
  - delete a file: /dir1/dir2/file3
  - rename a file: from /dir1/dir2/file3 to /dir1/dir2/file4
  - …
- File/Directory Access Control is by System Calls
  - e.g., open(2), stat(2), read(2), write(2), chmod(2), opendir(2), readdir(2), readlink(2), chdir(2), …

How to Cooperate/Share files under UNIX?

- Goal: I want students in my group to be able to read/write/create files under a certain directory, how to do it?

Process User ID Model in Modern UNIX Systems

- Each process has three user IDs
  - real user ID (ruid) — owner of the process
  - effective user ID (euid) — used in most access control decisions
  - saved user ID (suid) — the euid that the process used to have
- and three group IDs
  - real group ID
  - effective group ID
  - saved group ID

Process User ID Model in Modern UNIX Systems

- When a process is created by *fork*
  - it inherits all three users IDs from its parent process
- When a process executes a file by *exec*
  - it keeps its three user IDs unless the set-user-ID bit of the file is set, in which case the effective uid and saved uid are assigned the user ID of the owner of the file

The Need for suid/sgid Bits

- Some operations are not modeled as files and require user id = 0
  - halting the system
  - bind/listen on “privileged ports” (TCP/UDP ports below 1024)
- File level access control is not fine-grained enough
- System integrity requires more than controlling who can write, but also how it is written

Security Problems of Programs with suid/sgid

- These programs are typically setuid root
- Violates the least privilege principle
  - every program and every user should operate using the least privilege necessary to complete the job
- Why is this bad?
- How to solve this problem?
- How would an attacker exploit this problem?

Changing Effective User IDs

- A process that executes a set-uid program can drop its privilege; it can
  - drop privilege permanently
    - removes the privileged user id from all three user IDs
  - drop privilege temporarily
    - removes the privileged user ID from its effective uid but stores it in its saved uid, later the process may restore privilege by restoring privileged user ID in its effective uid

Access Control in Early UNIX

- A process has two user IDs: real uid and effective uid and one system call setuid
- The system call setuid(id)
  - when euid is 0, setuid set both the ruid and the euid to the parameter
  - otherwise, the setuid could only set effective uid to real uid
    - Permanently drops privileges
- A process cannot temporarily drop privilege

System V

- Added saved uid & a new system call
- The system call seteuid
  - if euid is 0, seteuid could set euid to any user ID
  - otherwise, could set euid to ruid or suid
    - Setting to ruid temp. drops privilege
- The system call setuid is also changed
  - if euid is 0, setuid functions as seteuid
  - otherwise, setuid sets all three user IDs to real uid

BSD

- Uses ruid & euid, change the system call from setuid to setreuid
  - if euid is 0, then the ruid and euid could be set to any user ID
  - otherwise, either the ruid or the euid could be set to value of the other one
    - enables a process to swap ruid & euid

Modern UNIX

- System V & BSD affect each other, both implemented setuid, seteuid, setreuid, with different semantics
  - some modern UNIX introduced setresuid
- Things get messy, complicated, and inconsistent, and buggy
  - POSIX standard, Solaris, FreeBSD, Linux

Suggested Improved API

- Three method calls
  - drop_priv_temp
  - drop_priv_perm
  - restore_priv
- Lessons from this?
- Psychological acceptability principle
  - "human interface should be designed for ease of use"
  - the user's mental image of his protection goals should match the mechanism

Big Picture of UNIX Access Control

- Only by users are not enough
- Using the suid/sgid bits relies on the trustworthiness of programs
- Programs may have bugs
- One attempt to fix it is by temporarily dropping privileges
  - Does this work?

User Authentication

- What the user knows
  - passwords, personal information
- What the user possesses
  - a physical key, a ticket, a passport, a token, a smart card
- What the user is (biometrics)
  - fingerprints, voiceprint, signature dynamics

Passwords

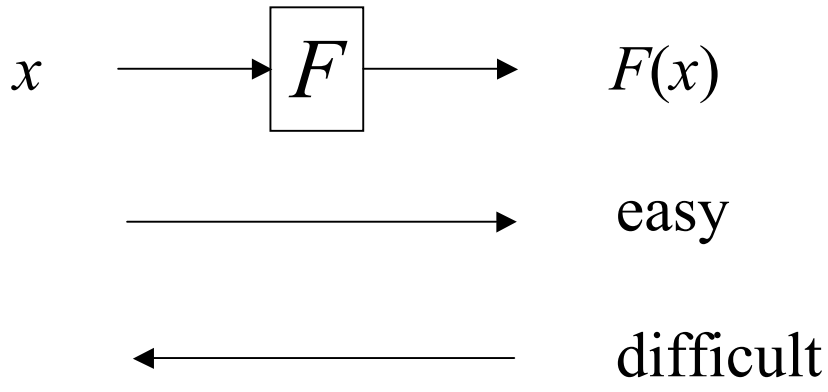
- Most commonly used method.

Storing User Passwords

- Directly Store the Passwords?
  - Not a good idea!
  - High risk
    - Anyone who captures the password database could impersonate all the users.
    - The password database would be very attractive to hackers.

One-Way Hash Function

- One-way hash function F
  - $F(x)$ is easy to compute
  - From $F(x)$, x is difficult to compute
  - Example: $F(x) = g^x \bmod p$, where p is a large prime number and g is a primitive root of p.



Storing Passwords

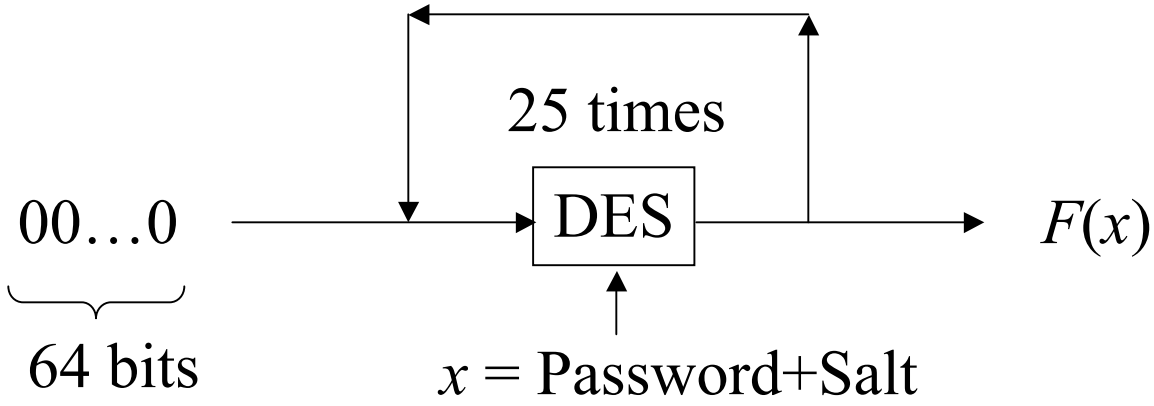
- For each user, system stores

  (*user name*, *F*(*password*))

  in a password file, where *F* is a one-way hash function
- When a user enters the password, system computes *F*(*password*); a match provides proof of identity

What is *F*?

- crypt Algorithm (Unix)
  - Designed by Bob Morris and Ken Thompson
  - Use Data Encryption Standard (DES) encryption algorithm
  - User password and salt is used as the encryption key to encrypt a 64-bit block of zeros
  - This process is repeated 25 times



Choice of Passwords

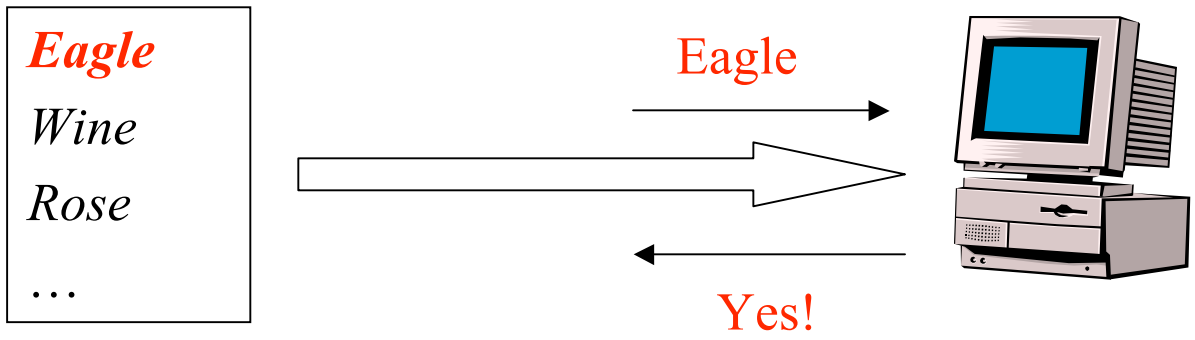
- Suppose passwords can be from 1 to 9 characters in length
- Possible choices for passwords = $26^1 + 26^2 + ... + 26^9 = 5 * 10^{12}$
- At the rate of 1 password per millisecond, it will take on the order of 150 years to test all passwords

Choice of Passwords (Cont'd)

- However, we don't need to try all possible passwords, only the probable passwords
- In a Bell Labs study (Morris & Thompson 1979), 3,289 passwords were examined
  - 15 single ASCII characters, 72 two ASCII characters, 464 three ASCII characters, 477 four alphanumeric character, 706 five letters(all lower or all upper case), 605 six letters all lower case, 492 weak passwords (dictionary words spelled backwards, first names, surnames, etc.)
  - Summary: 2,831 passwords (86% of the sample) were weak, i.e., they were either too easily predictable or too short

Dictionary Attacks

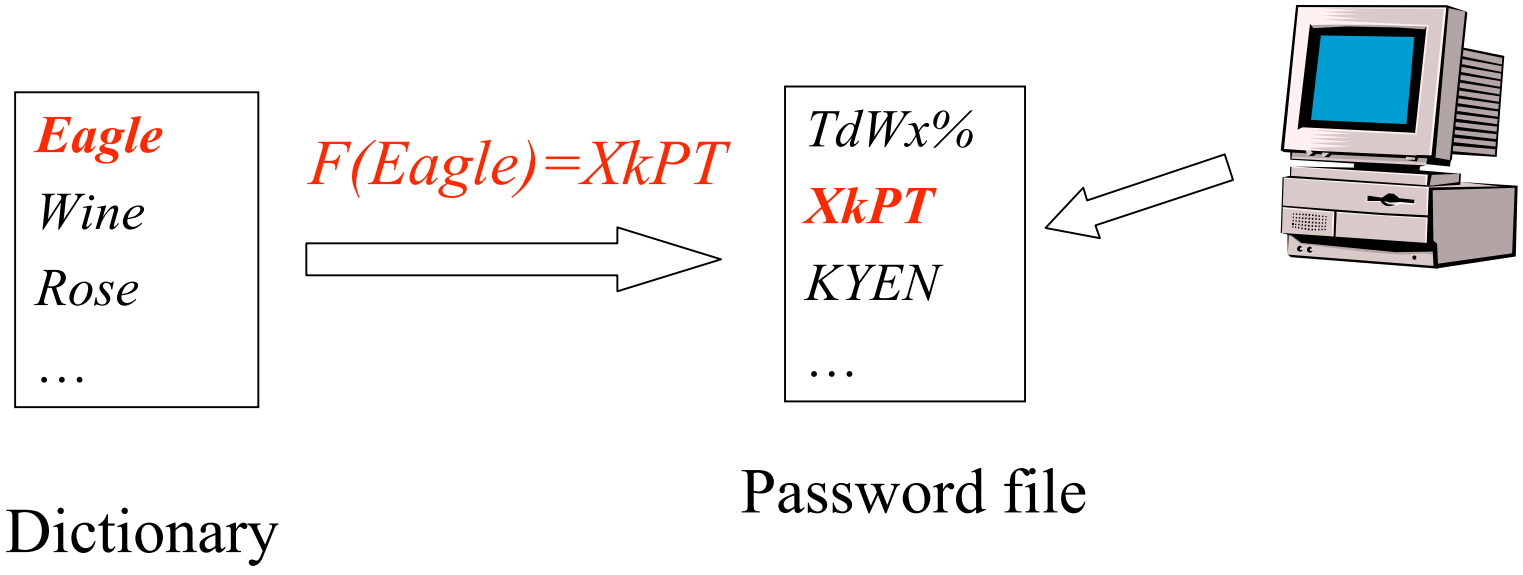
- Attack 1:
  - Create a dictionary of common words and names and their simple transformations
  - Use these to guess the password



Dictionary

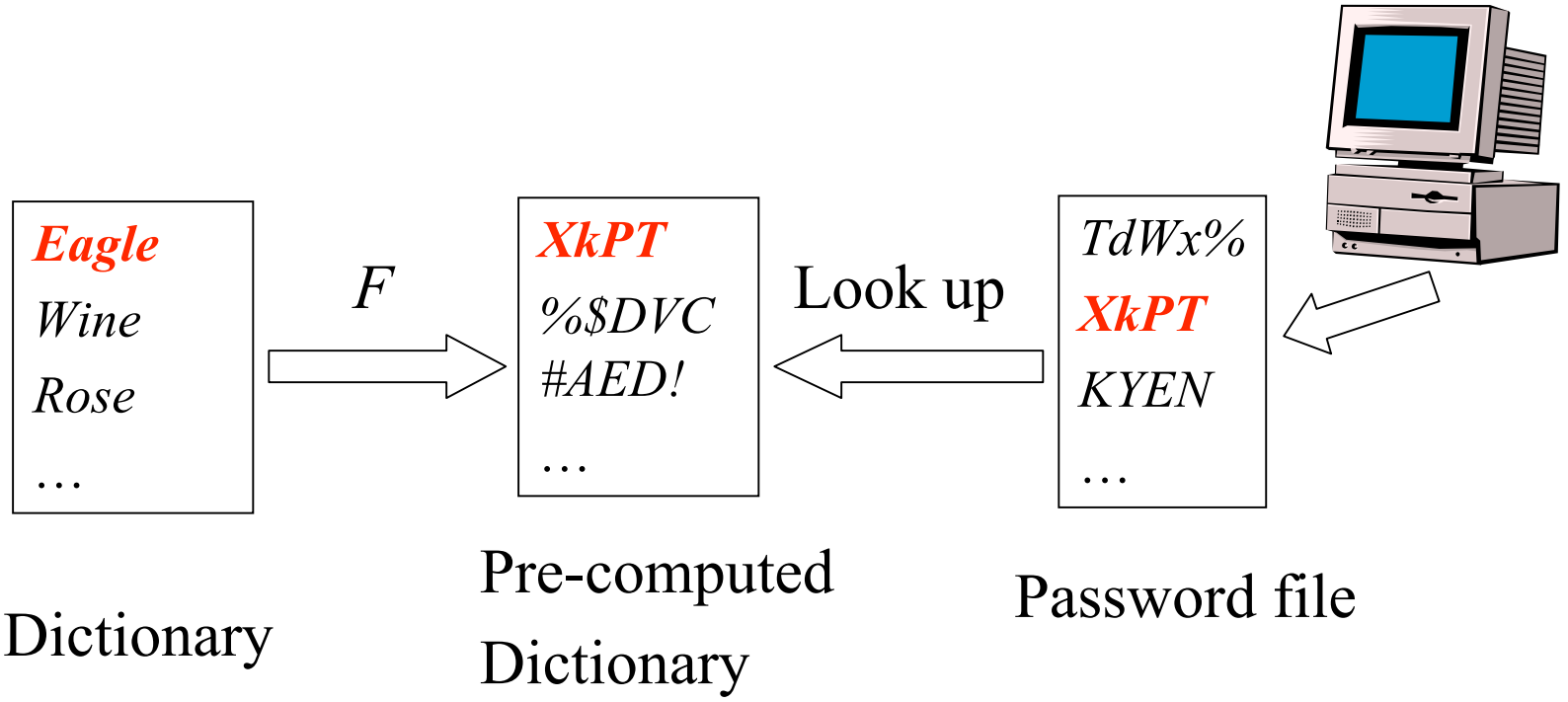
Dictionary Attacks (Cont'd)

- Attack 2:
  - Usually *F* is public and so is the password file
    - In Unix, *F* is crypt, and the password file is /etc/passwd.
  - Compute *F*(*word*) for each word in the dictionary
  - A match gives the password



Dictionary Attacks (Cont'd)

- Attack 3:
  - To speed up search, pre-compute *F*(dictionary)
  - A simple look up gives the password

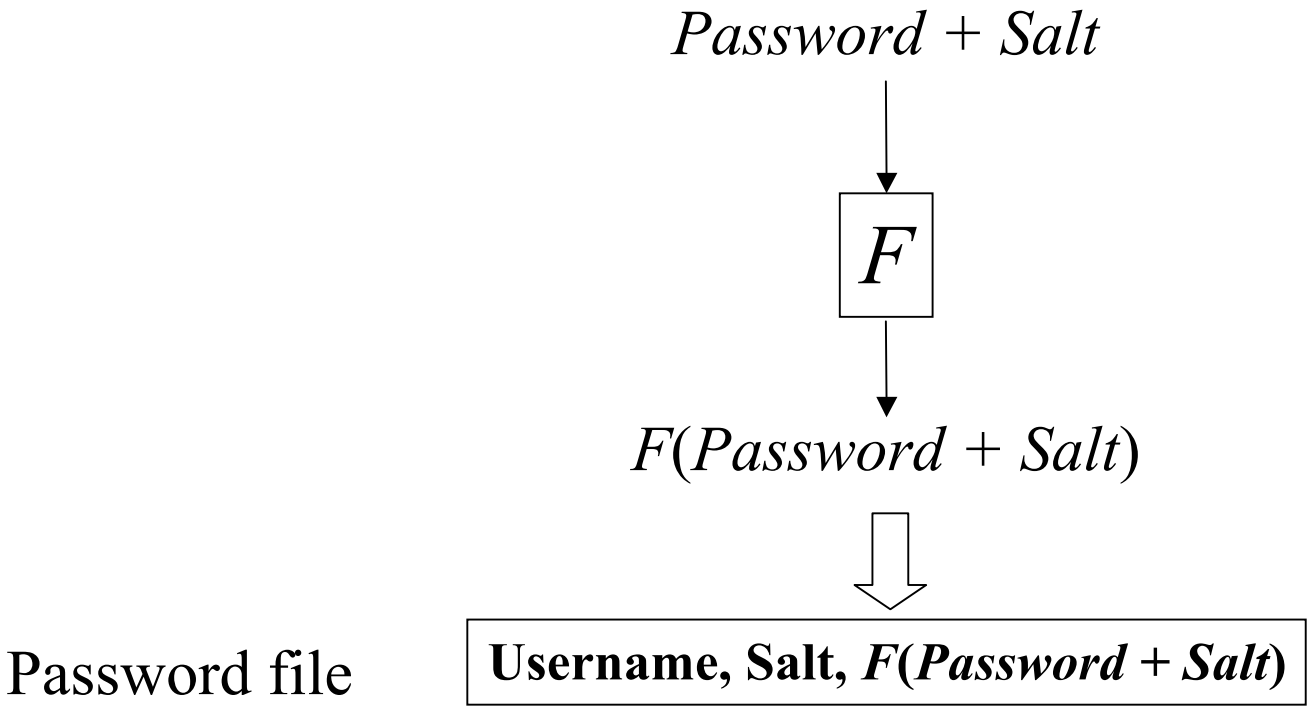


Password Salt

- To make the dictionary attack a bit more difficult
- Salt is a 12-bit number between 0 and 4095
- Derived from the system clock and the process identifier

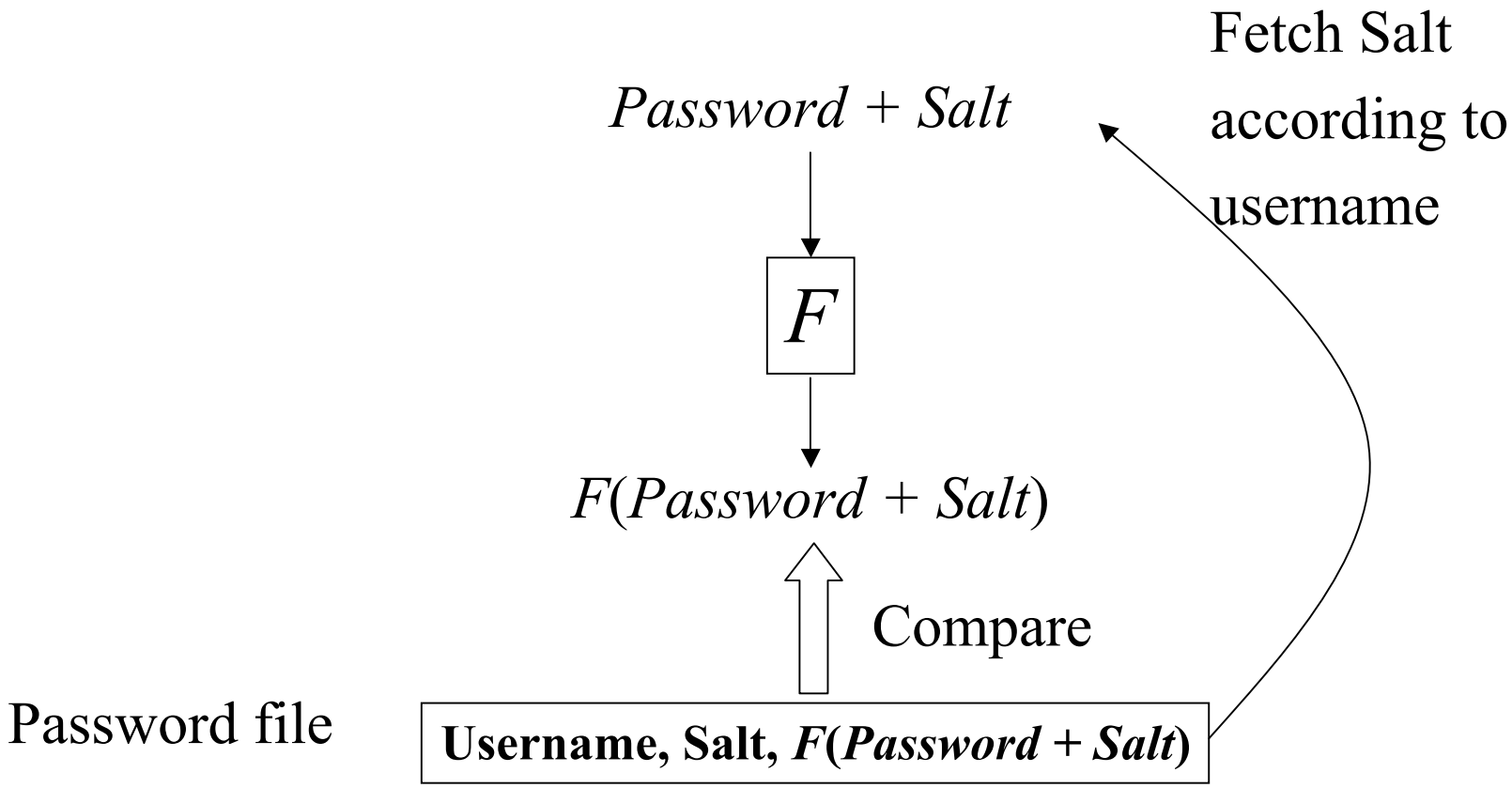
Password Salt (Cont'd)

- Storing the passwords



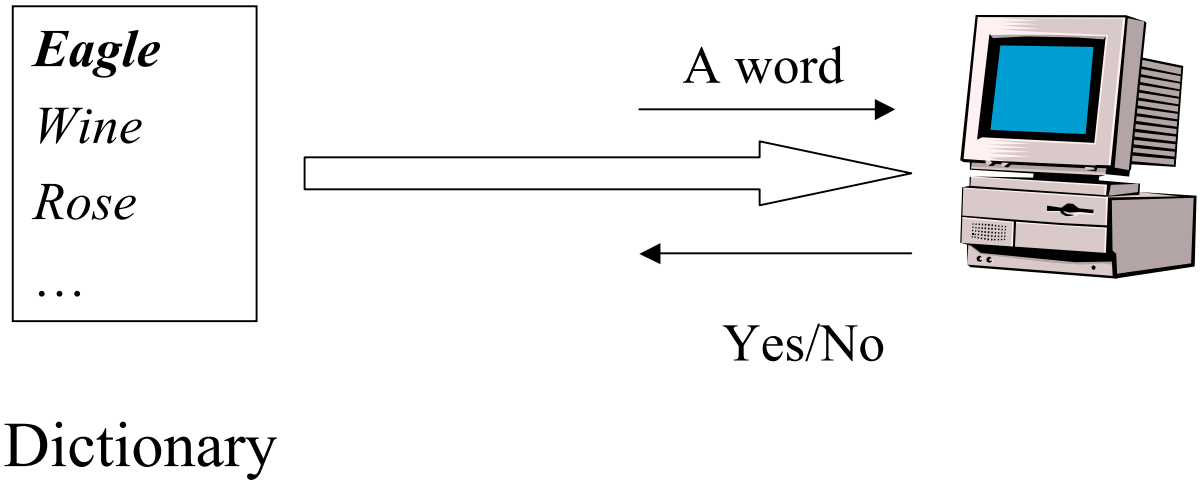
Password Salt (Cont'd)

- Verifying the passwords



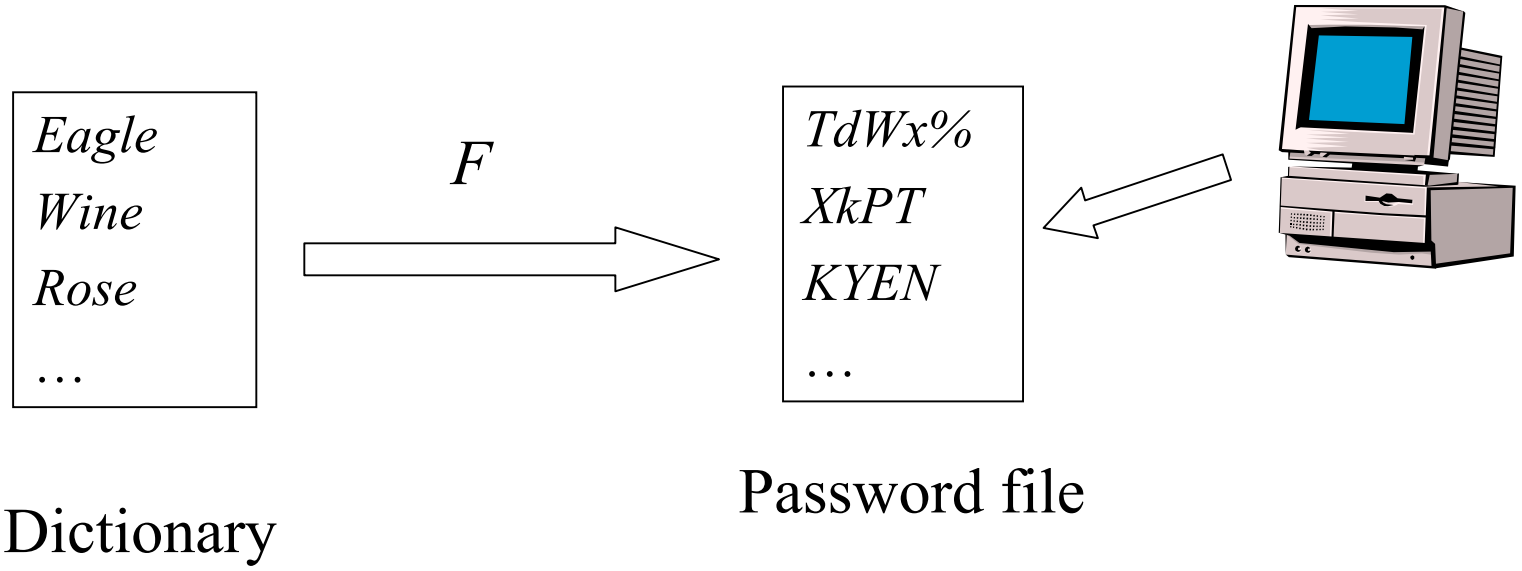
Does Password Salt Help?

- Attack 1?
  - Without Salt
  - With Salt



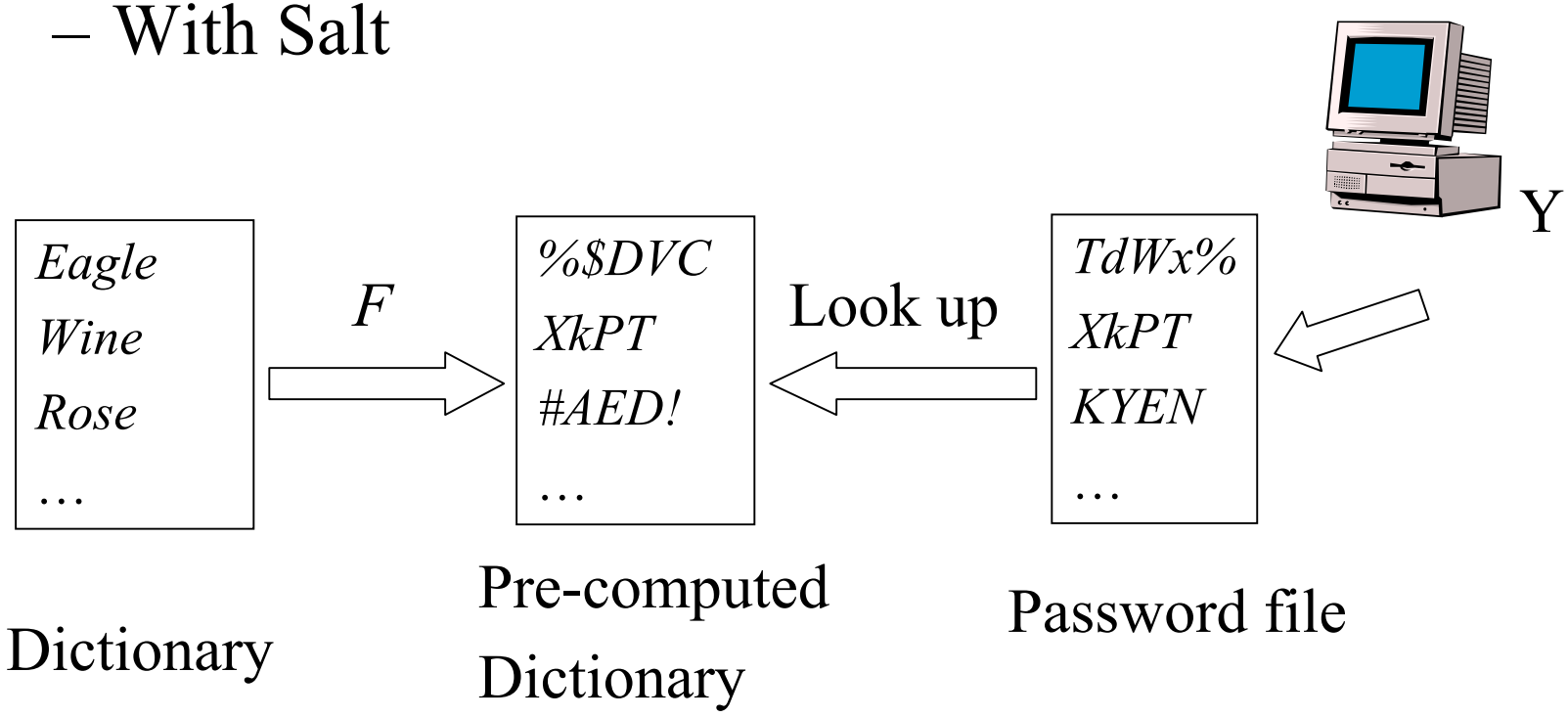
Does Password Salt Help?

- Attack 2?
  - Without Salt
  - With Salt



Does Password Salt Help?

- Attack 3?
  - Without Salt
  - With Salt



Password Management Policy and Procedure

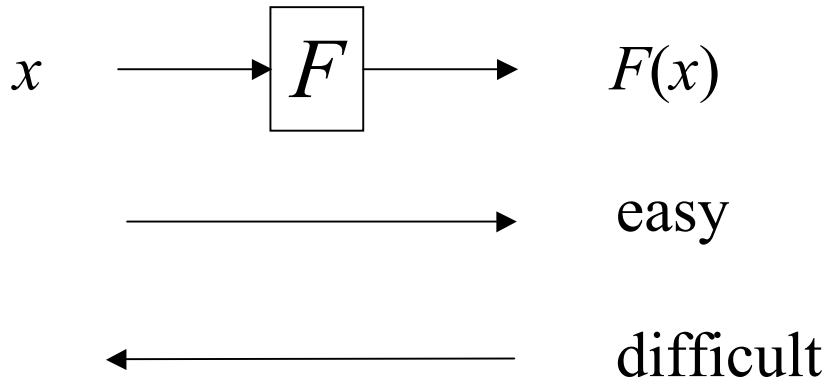
- Educate users to make better choices
  - Does not work if the user population is large or novice
- Define rules for good password selection and ask users to follow them
  - Rules serve as guideline for attackers
- Ask or force users to change their passwords periodically
- Force users to use machine generated passwords
  - Random passwords are difficult to memorize; also password generator may become known to the attacker through analysis
- Actively attempt to break users' passwords; force users to change those that are broken
  - Attacker may have better dictionary
- Screen password choices; if a choice is weak, force users to make a different choice

One-time Passwords

- Use the password exactly once!

Lamport's Scheme (S/Key)

- Take advantage of One-Way function
- One-way hash function F
  - F(x) is easy to compute
  - From F(x), x is difficult to compute

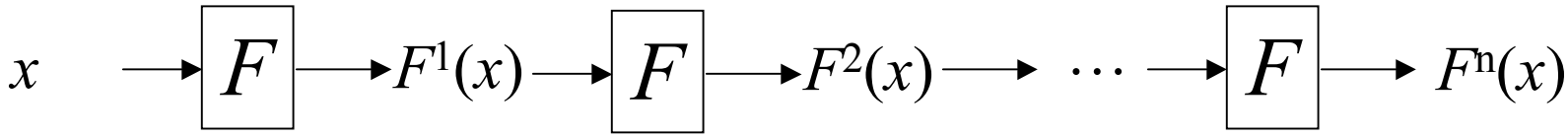


S/Key (Cont'd)

- Pre-computation

The System

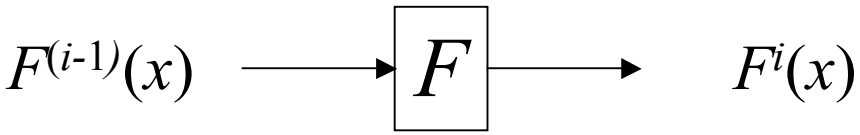
1. Randomly generate x
2. Compute the following



3. Save (*username*, $F^n(x)$), and give x to the user.

S/Key (Cont'd)

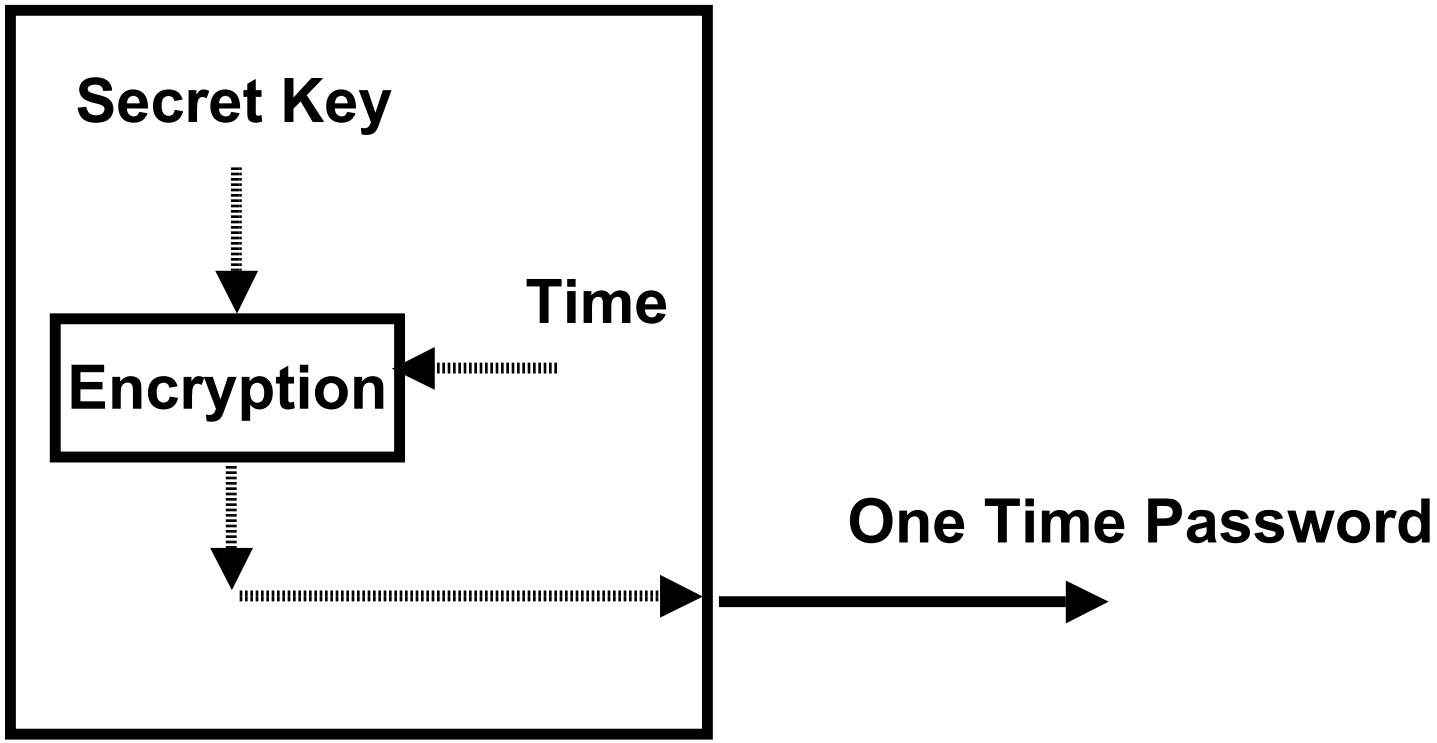
- Authentication
  - The first time, the user supplies $F^{(n-1)}(x)$.
  - The system checks if $F(F^{(n-1)}(x))=F^{n}(x)$. If yes, the user is authenticated and the system replaces $F^{n}(x)$ with $F^{(n-1)}(x)$.
  - The second time, the user supplies $F^{(n-2)}(x)$.
  - The third time, …



Time Synchronized

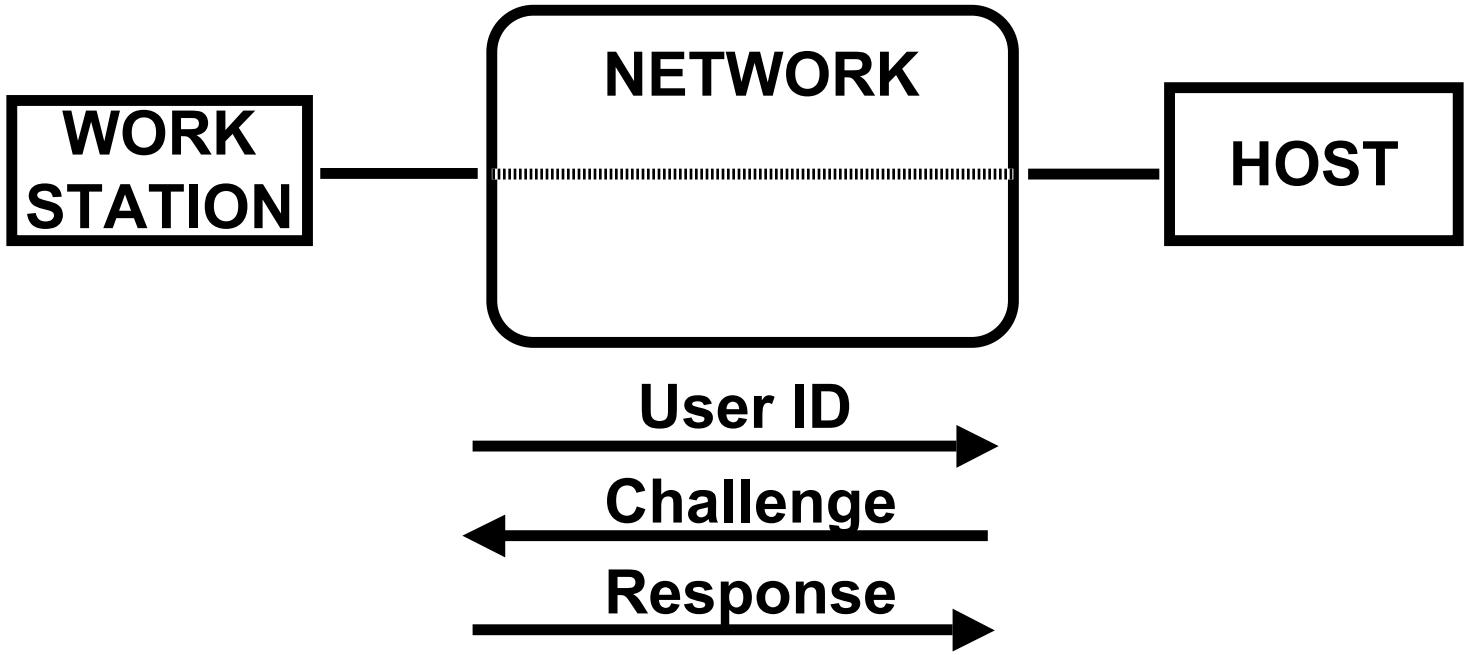
- There is a hand-held authenticator
  - It contains an internal clock, a secret key, and a display
  - Display outputs a function of the current time and the key
  - It changes about once per minute
- User supplies the user id and the display value
- Host uses the secret key, the function, and its clock to calculate the expected output
- The login is valid if the values match
- In practice, the clock skew is a problem

Time Synchronized (Cont'd)

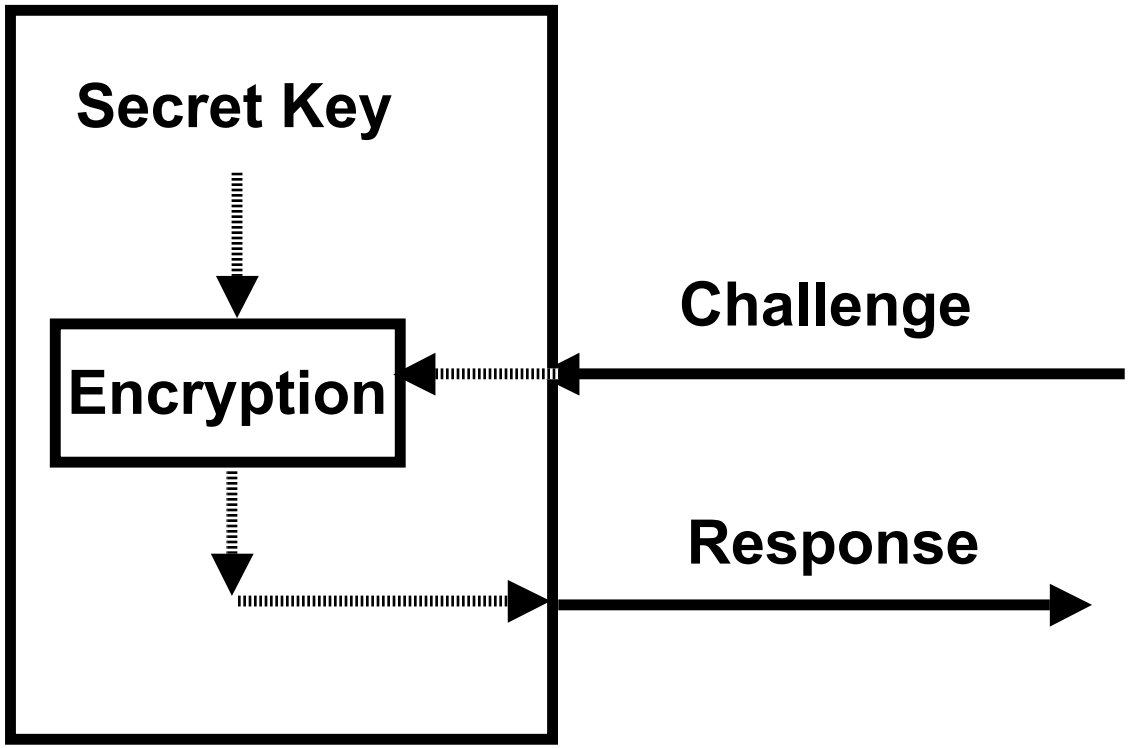


Challenge Response

- A non-repeating challenge from the host instead of a clock is used
- Note that the device requires a keypad.



Challenge Response (Cont'd)



Challenge Response (Cont'd)

- Problems with challenge/response schemes
  - Key database is extremely sensitive
  - This can be avoided if public key algorithms are used; however, the outputs would be too long for users to input conveniently

Biometrics

- Fingerprint
- Retina scan
- Voice pattern
- Signature
- Typing style

Biometrics (Cont'd)

Technique	Description	Min. Cost	False Reading
Retina	Eyes scanned 1 to 2 inches from screening device	$2,400	1/10,000,000+
Iris	Camera image of eye takes from 14 inches	$3,500	1/131,000
Hand	Hand scanned on plate by three video cameras at different angles	$2,150	1/500
Fingerprint	Finger scanned on glass plate	$1,995	1/500
Signature	Written with special pen on digitizer tablet	$1,000	1/50
Voice	Predefined phrase spoken into telephone or microphone	$1,500	1/50

Effectiveness of Biometrics

- Two types of errors for authentication
  - False acceptance (FA)
    - Let imposters in
    - FAR: the probability that an imposter is authenticated.
  - False rejection (FR)
    - Keep authorized users out
    - FRR: the probability that an authorized user is rejected.
- Another type of error for identification
  - False match (FM)
    - One user is mistaken for another (legitimate user)
    - FMR: the probability that a user is incorrectly matched to a different user's profile.
- No technique is perfect!

Multimodal Biometrics

- Use multiple Biometrics together.
  - AND: Accept only when all are passed
    - Why do we need this?
  - OR: Accept as long as at least one is passed
    - Why do we need this?
  - Others