System Security

You Shall Pass with The Right Password

Mohamed Sabt

Univ Rennes, CNRS, IRISA

2023 / 2024

Part I

Introdcution

What do we authenticate?

- Users
  - Requires human interaction
  - Is typically slow
  - Local or remote
  - Goal: Authentication

- Machines
  - Must be transparent to the ”authenticated” user
  - Should be fast and scalable
  - Mostly remote
  - Goal: Key establishment

Authentication Basics

- Authentication binds identity to a subject.

- Two-step process
  - Sign up: establish identity to system.
  - Sign in: verifies and binds (physical) entity to (logical) identity.

Identification vs authentication 1/2

- Identification means one-from-many
  - Find your fingerprints in a police database
- Authentication means one-to-one relations
  - Compare your (based on the username) input to a previously saved one
  - Enrollment (can be slow, must be precise) vs Recognition (must be quick)
- Cooperation
  - In identification the user does not cooperate
  - In authentication the user is cooperative

Identification vs authentication 2/2

- Identify
  - Map a real-person/subject to a virtual account
- Authenticate
  - Request a proof from the account
- Authorize
  - Verify if the account can access a resource
- Accounting
  - Log/monitor what the account is doing

Attacker Model 1/2

Attack	Short Description
dictionary	using a heuristically prioritized list in a guessing attack

Part II

Authentication

The Concept

- The authenticator (e.g. server, website) asks to prove that you are who you pretend to be based on one or more pieces of evidence called **factors**.
  - Can also be mutual. Server also authenticates to the client.

- The evidence can be presented directly (e.g. password authentication protocol) or indirectly.
  - Indirect proof use some form of cryptographic algorithms.

Types of Factors

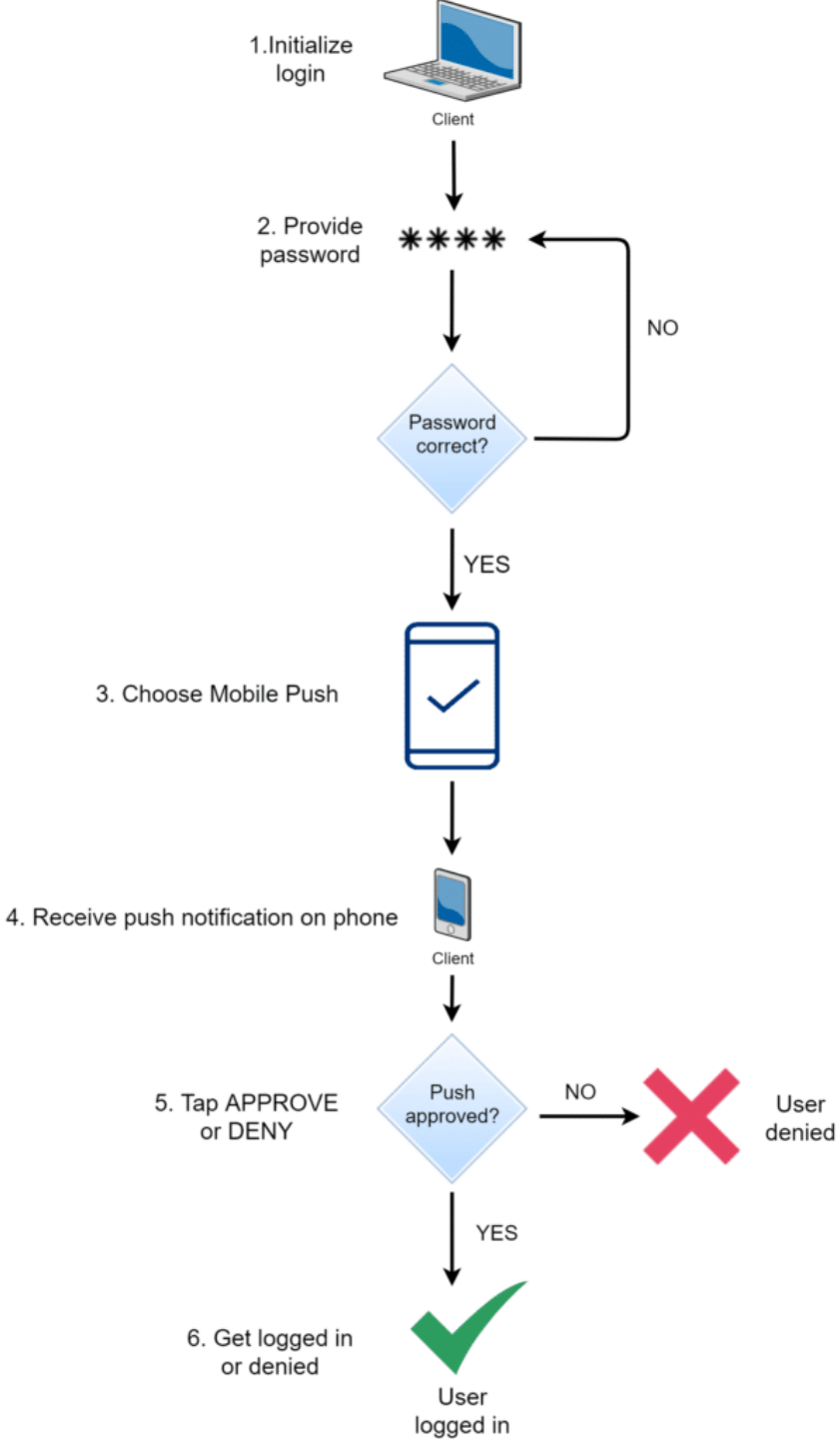
- Something you know (Knowledge Factor)
- Something you have (Possession Factor)
- Something you are (Inherence Factor)

- Other authentication attributes that can be used:
  - Somewhere you are

Chaining Factors

- N-factor authentication
  - Factors should be different

- **First**, you provide your password – something you know. **Then**, you receive a push notification on your mobile phone. Your mobile device is something you have because you need to prove the possession of the mobile device to complete authentication.



Something You Know -- Passwords

- Require people to remember them
  - Used on multiple occasions
- Can be attacked
  - Shoulder surfing easily done
  - Dictionary attack.
- Can be enhanced through policies
  - E.g. Minimum 20 characters

Something You Have

- Phone number.
- Hardware token.
- Cryptographic keys.

Something You Are

- Fingerprint
- Facial recognition
- Speech recognition

Biometric Properties

- Not 100% accurate – measurements change over time
  - Because of sensors
  - Because of changes in biometrics
  - Measurements imprecise, so approximate matching algorithms are used
- Not 100% secure
  - E.g. Fingerprints w/o hands.
  - No need for advanced techniques (i.e., torture) to obtain.
- Typically hard to profile, easy to collect/verify
  - E.g. Scanning of face multiple times to enable FaceID on Apple

Part III

Passwords and Attacks

Authentication System

- (A, C, F, L)
  - A: information that proves identity
  - C: information stored on computer and used to validate authentication info
  - F: transformation function; f: A → C
  - L: function that proves identity

- Example – passwords stored in cleartext
  - A: set of strings making up passwords
  - C: A
  - F: identity function {I}
  - L: single equality test {eq}

Storage

- Store as cleartext
  - If the password file got compromised, all passwords are revealed.

- Encipher file
  - Need to have cryptographic keys.
  - Reduced to previous problem.

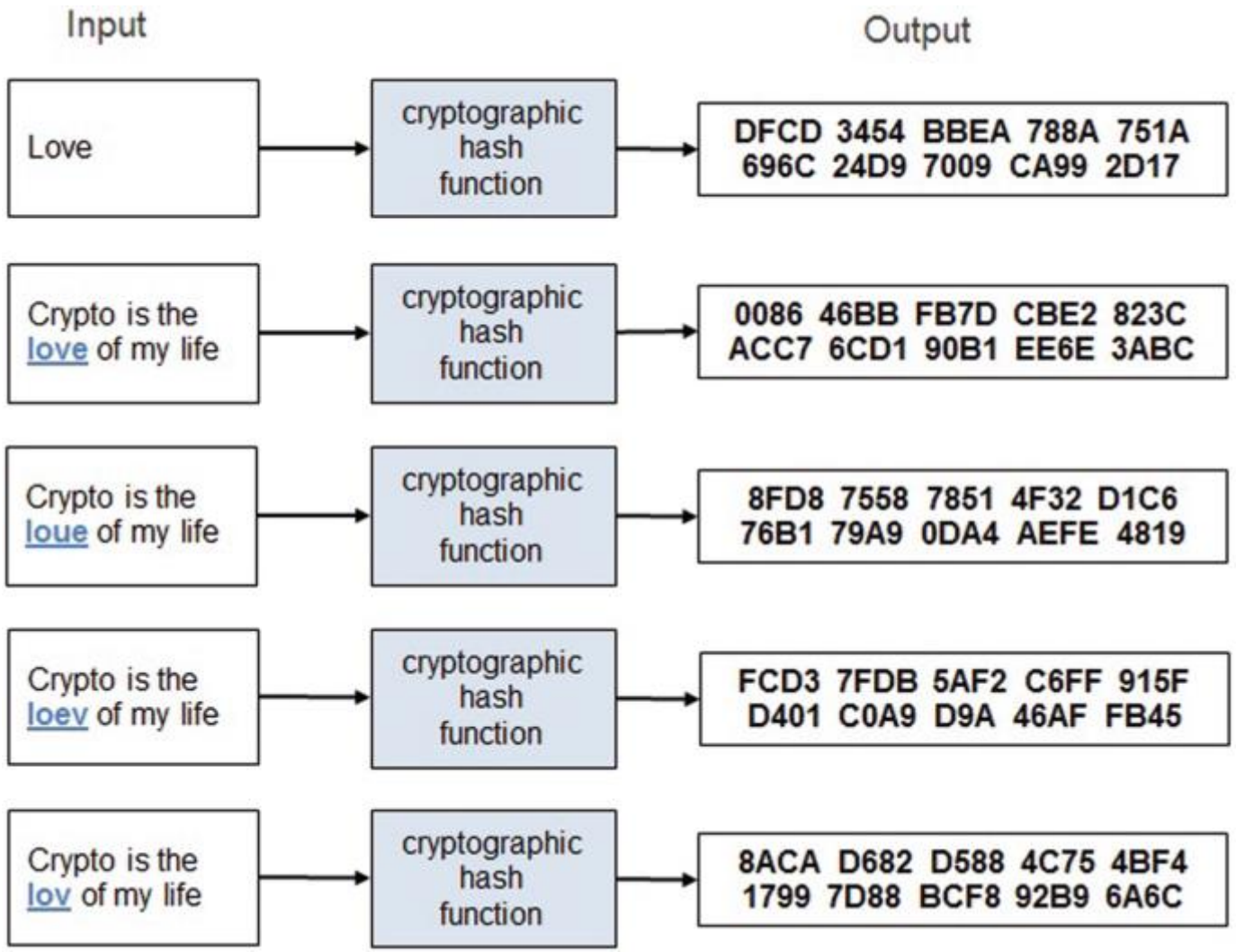
- Store one-way hash of password
  - If file is read, attacker must still guess passwords or invert the hash.

Hash Functions

- Input – data of arbitrary size.
- Output – fixed length.
- Deterministic – same input always produces the same output.
- One way function (pre-image resistance) – hard to deduce input from output.
- Collision resistance – hard to find two inputs producing the same output.

- Different from checksums: e.g., CRC-32

Avalanche Property



Anatomy of Attacking

- Goal: find a ∈ A, such that
  - F(a) = c ∈ C

- Two ways to determine whether a meets these requirements
  - Direct approach: Given c, compute F^{-1}(c).
  - Indirect approach: loop over a ∈ A and compute L(a) until it succeeds.

Dictionary Attacks

- Trial-and-error from a list of potential passwords.
  - Most passwords chosen by humans are from a relatively small set

- offline
  - Know F or C, and repeatedly try different guesses until the list is done or passwords are guessed.
  - John the ripper.

- online
  - Have access to L and try guesses until some password succeeds.
  - Example: try to log in by guessing a password.

Preventing Attacks

- Hide F or C
  - Example: shadow file in Unix.
  - Breaches happen.
  - Alternative: slow down ‘F’.

- Block access to L()
  - Prevents attackers from knowing if guesses succeeded.

Guessing Through L

- Cannot prevent these
  - Otherwise, legitimate users cannot log in.

- Make them slow, while tracking bad guesses
  - Backoff.
  - Disabling

Time is Money

- 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
  - Then, P >= TG/N

Example

- Goal
  - Passwords drawn from a 62-char alpha numeric.
  - 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 ≥ TG/P = (365×24×60×60)×10^4/0.5 = 6.31×10^11
  - Choose s such that $\sum_{j=0}^{s} 62^{j}$ ≥ N
  - So s ≥ 7, meaning passwords must be at least 7 chars long

Slow is Better (Rate Limiting)

- Passwords have weak entropy.
  - Either make passwords longer, or
  - Make guessing slower.

- Slow hash wouldn't bother user: delay in logging hardly noticeable.
  - But would bother attacker constructing dictionary: delay multiplied by number of entries.

- Hashing methods are deliberately engineered to be slow with no parallelization;
  - they use many iterations of an underlying cryptographic primitive to increase the cost of each guess, making brute-force attacks infeasible.

- The number of Iterations is programmable:
  - This makes it possible to keep the hash slow as hardware improves.

Approaches: Password Selection

- Random selection
  - Any password from A is equally likely to be selected.

- User selection of 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/Quran/… words).
  - Too short, digits only, letters only
  - License plates, acronyms, social security numbers
  - Personal characteristics or foibles (pet names, nicknames, job characteristics, etc.

Proactive Password Checking

- Let the user select their own password, but
  - detect, reject bad passwords for an appropriate definition of “bad”.
  - “Bad” is defined according to a “policy”.

- Needs to do pattern matching on words
  - Needs to execute subprograms and use results

- Easy to set up and integrate into password selection System

- Drawback: reduces number of possible passwords

“Secure” Passwords

- Passwords should be uniformly distributed
  - All characters in password chosen with equal probability
  - But not very psychologically acceptable

- Passwords should be long
  - Longer password = larger brute force search space.

- Passwords should never be reused.

- Passwords chosen randomly are difficult to remember
  - How many 15 char passwords can you remember?
  - Tradeoff of security vs. convenience.

Example: Passwd++

- Test length(“$p”) < 7
  - If password is under 7 characters, reject it.

- Test infile(“/usr/dict/words”, “$p”)
  - If password is in file /usr/dict/words, reject it.

- Test not-inprog(“spell”, “$p”)
  - If password is not in the output from a program spell, given the password as input, reject it (because it is a properly spelled word).

Salting

- Goal: slow offline dictionary attacks through Pre-computation or Rainbow Table.

- Method: diversify hash function so that:
  - Parameter differs for each password.
  - So given ‘n’ password hashes, and therefore ‘n’ salts, need to hash guess ‘n’

- Example:
  - Use salt as first part of input to hash function.

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

Other Password Problems

- Keyloggers and server breaches.

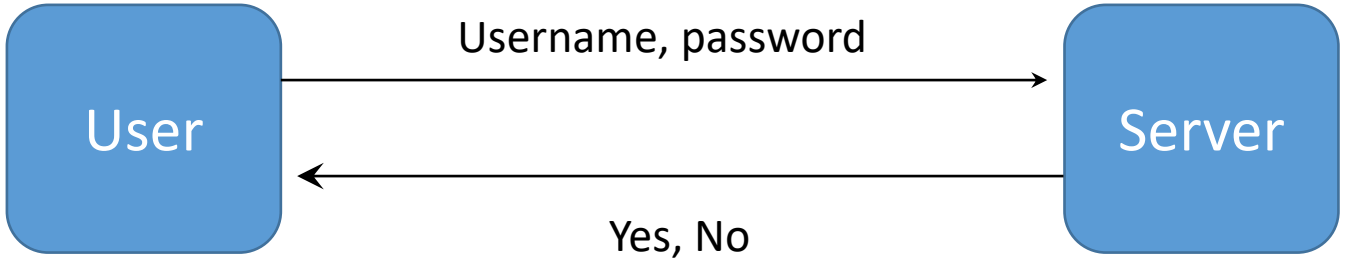
- Social engineering
  - Share passwords.
  - Shoulder surfing.
  - Password reuse.

- Internet
  - How many different passwords can you have?
  - Same one for every server.

- Phishing sites
  - Trick you into revealing your password.
  - Fancy Bear stole Clinton campaign emails.

Remote Authentication 1/2

- “Very bad trip”

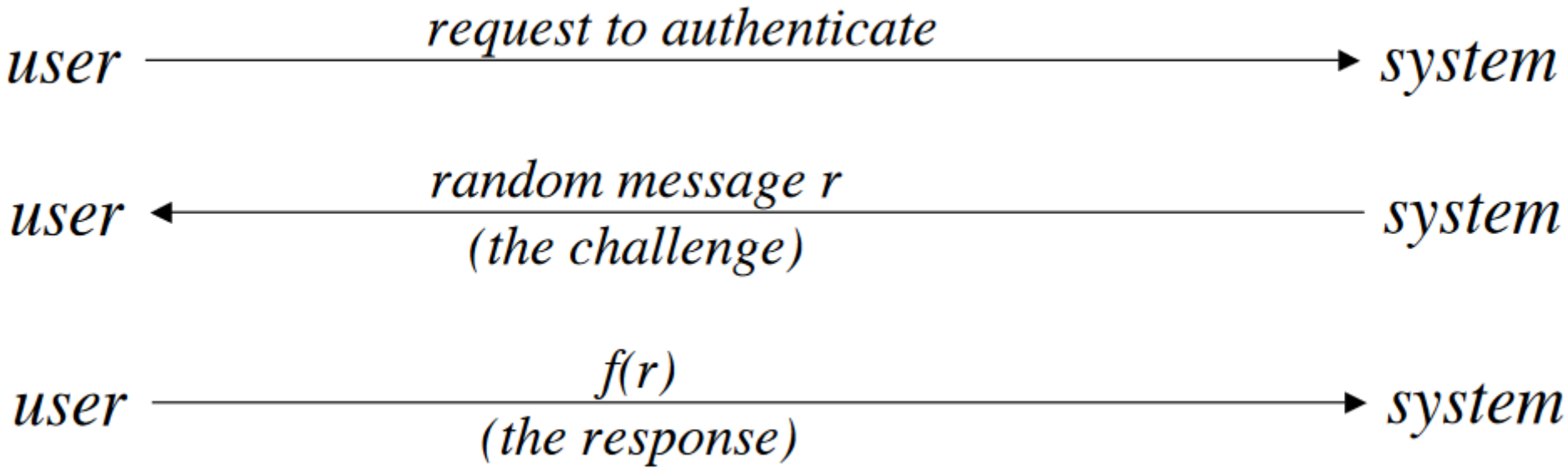


Attacker Model 2/2

Attack	Short Description
replay	reusing a previously captured message in a later protocol run

Remote Authentication 2/2

- ‘f’ is a secret function
  - Or public with some secret parameters, such as the password.



Examples

- FIDO Protocols.
- TOTP.

Part IV

Unix Password Storage

Why /etc/shadow

- /etc/passwd file must be world-readable
  - this file is used to perform the translation from UID to username.

- It is possible to perform attacks against passwords even if the encrypted/hashed password is available.
  - Offline attack.

- Therefore, the /etc/shadow file is readable only by the root user and contains password (and optional information) for each user.
  - As in the /etc/passwd file, each user's information is on a separate line.
  - Each of these lines is a colon delimited list.

/etc/shadow Information 1/3

- **Username**: the name the user types when logging into the system.

- **Encrypted password**: The password is encrypted using the crypt(3) library function. In this field, values other than a validly-formatted encrypted or hashed password are used to control user logins and to show the password status. For example,
  - if the value is ! or *, the account is locked and the user is not allowed to log in.
  - the user, not having set a password, will not be able to log in

- **Date password last changed**: The number of days since January 1, 1970 (also called the epoch) that the password was last changed.

/etc/shadow Information 2/3

- **Number of days before password can be changed**: The minimum number of days that must pass before the password can be changed.

- **Number of days before a password change is required**: The number of days that must pass before the password must be changed.

- **Number of days warning before password change**: The number of days before password expiration during which the user is warned of the impending expiration.

/etc/shadow Information 3/3

- **Number of days before the account is disabled**: The number of days after a password expires before the account will be disabled.

- **Date since the account has been disabled**: The date (stored as the number of days since the epoch) since the user account has been disabled.

- **A reserved field:** ignored in most Linux distributions.

Examples

```
sabtmoha@sabtmohav2:~$ sudo more /etc/shadow | grep -w sabt
sabt:!:19615:0:99999:7:::
```

```
sabtmoha@sabtmohav2:~$ sudo more /etc/shadow | grep -w sabt
sabt:$y$j9T$nUiLPhoYUajxu6QWbUFEz.$3JDQc0B9dhU1NyFo5FyDldYJbBCWIJ2q426JV8rdWX/:19615:0:7:1::19616:
```

Storage Format 1/2

- Hashed passphrase consists of four components delimited by ‘$’: prefix, options, salt, and hash.
- The prefix controls which hashing method is to be used.
- The syntax/contents/length of options, salt, and hash are up to the hashing method.

Storage Format 2/2

- Hashed passphrases are always entirely printable ASCII, and do not contain any whitespace or the characters ‘:’, ‘;’, ‘*’, ‘!’, or ‘\’

- The salt and hash are usually encoded as numerals in base 64.
  - The common “base64” encoding is usually not used.

Linux supported Hash Functions

- yescrypt (based on scrypt)
  - Prefix: y
  - Hash size: 43 chars

- Sha256crypt (based on sha-256 and weak)
  - Prefix: 5
  - Hash size: 43 chars

- md5crypt (too weak)
  - Prefix: 1
  - Hash size: 22

- descrypt (too weak)
  - Prefix: empty
  - Hash size: 13 (and max password length 8 characters)

CPU Time Cost

- Most modern hashing methods allow the number of iterations to be adjusted, using the “CPU time cost” parameter to crypt_gensalt(3).

- The exact meaning of “CPU time cost” depends on the hashing method, but larger numbers correspond to more costly hashes.

- The default “CPU time cost” for Sha256crypt is considered weak.

Part V

Pluggable Authentication Modules (PAM)

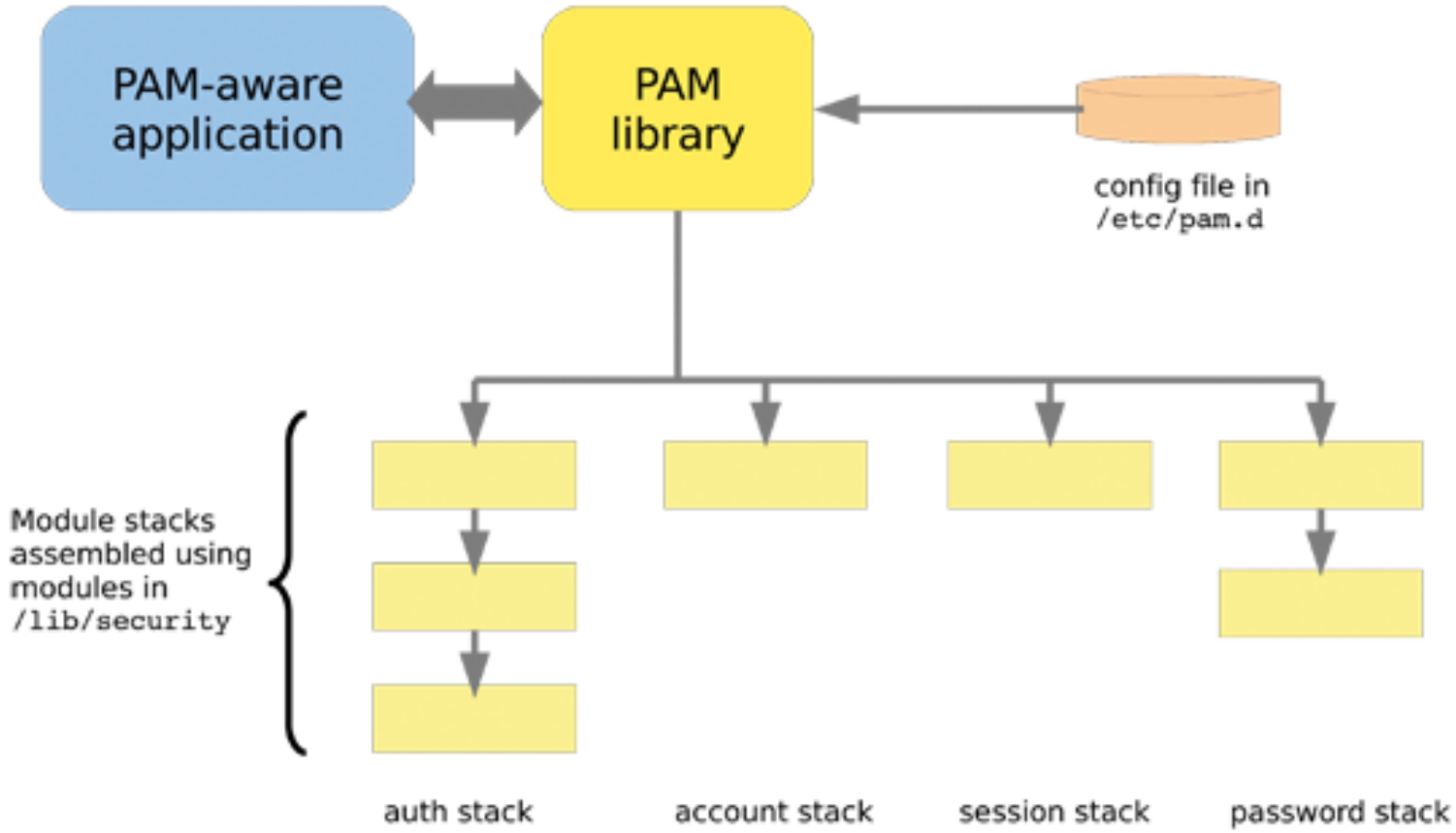
Overview

- Pluggable Authentication Modules (PAM) have been around since 1997.

- PAM separates the standard and specialized tasks of authentication from applications.
  - Centralize authentication, make functionality available through library
  - It would be painful to ask each application developer to provide authentication.
  - Accesses file with name of program in /etc/pam_d

- PAM is modular in that we can add new methods with new libraries.
  - Add a new module (e.g., for fingerprint authentication), directly available to all PAM enabled programs

PAM Design


PAM-aware application
PAM library
config file in /etc/pam.d
Module stacks assembled using modules in /lib/security
auth stack
account stack
session stack
password stack

PAM Activities

- **auth**: The activity of user authentication; typically by password, but can also use tokens, fingerprints etc.
- **account**: After a user is identified, decide whether they are allowed to log in. For example, can restrict login times.
- **session**: Allocates resources, for example mount home directory, set resource usage limits, print greeting message with information.
- **password**: Update the user's credentials (typically the password)

High-Level Steps for local auth 1/2

- The login application prompts for a user name and password,
  - then makes a libpam **authentication** call to ask, "Is this user who they say they are?"
  - The pam_unix module is responsible for checking the local account authentication.
  - Other modules may also be checked,
  - and ultimately the result is passed back to the login process.

- The login process next asks, "Is this user allowed to connect?",
  - then makes an **account** call to libpam.
  - The pam_unix module checks for things like whether the password has expired.
  - Other modules might check host or time-based access control lists.
  - An overall response is handed back to the process.

High-Level Steps for local auth 2/2

- If the login process is continuing at this point, it is ready to create the session.
  - A **session** call to libpam results in the pam_unix module writing a login timestamp to the wtmp file.

- On logout, when the session is closed,
  - another **session** call can be made to libpam.
  - This is when the pam_unix module writes the logout timestamp to wtmp.

PAM Configurations Syntax

- PAM configuration files are located in /etc/pam.d.
  - Each line (called rule) is composed as follow:

```
type control module-path module-arguments
```

- **type:** auth, account, password or session depending on which step you want to configure.

- **control:** indicates if the module should fail or succeed in its authentication task.

- **module-path:** is the PAM module used to perform the action. A lot of PAM modules exist. Note that certain modules cannot be used for every type.

- **module-arguments** are the arguments given to the module.

Control

- The control syntax can be complex.
  - a square-bracketed selection of value=action pairs
- Action can be
  - Ignore, bad, die, ok, done, or N (integer) to skip
- PAM Control flags:
  - **requisite**: if module fails, immediately return failure and stop
  - **required**: if module fails, return failure but continue
  - **sufficient**: if module passes, return pass and stop
  - **optional**: pass/fail result is ignored

Example of PAM Modules

Name	Activities	Description
pam_unix	auth, session, password	Standard UNIX authentication through /etc/shadow passwords
pam_permit	auth, account, session, password	Always returns true
pam_deny	auth, account, session, password	Always returns false
pam_rootok	auth	Returns true if you're root
pam_warn	auth, account, session, password	Write a log message to the system log
pam_cracklib	password	Perform checks of the password strength

Sample PAM Config Examples

- Prevent all users from using su (/etc/pam.d/su)
  - auth requisite pam_deny.so

- Enforce passwords with at least 10 characters and at least 2 special characters, use SHA-512 for password hash (/etc/pam.d/passwd):
  - password required pam_cracklib.so minlen=10 ocredit=-2
  - password required pam_unix.so sha512