Professional Masters in Information and Cyber Security (PMICS)

Computer Science and Engineering,
University of Dhaka

CSE 802

Information Security Fundamentals

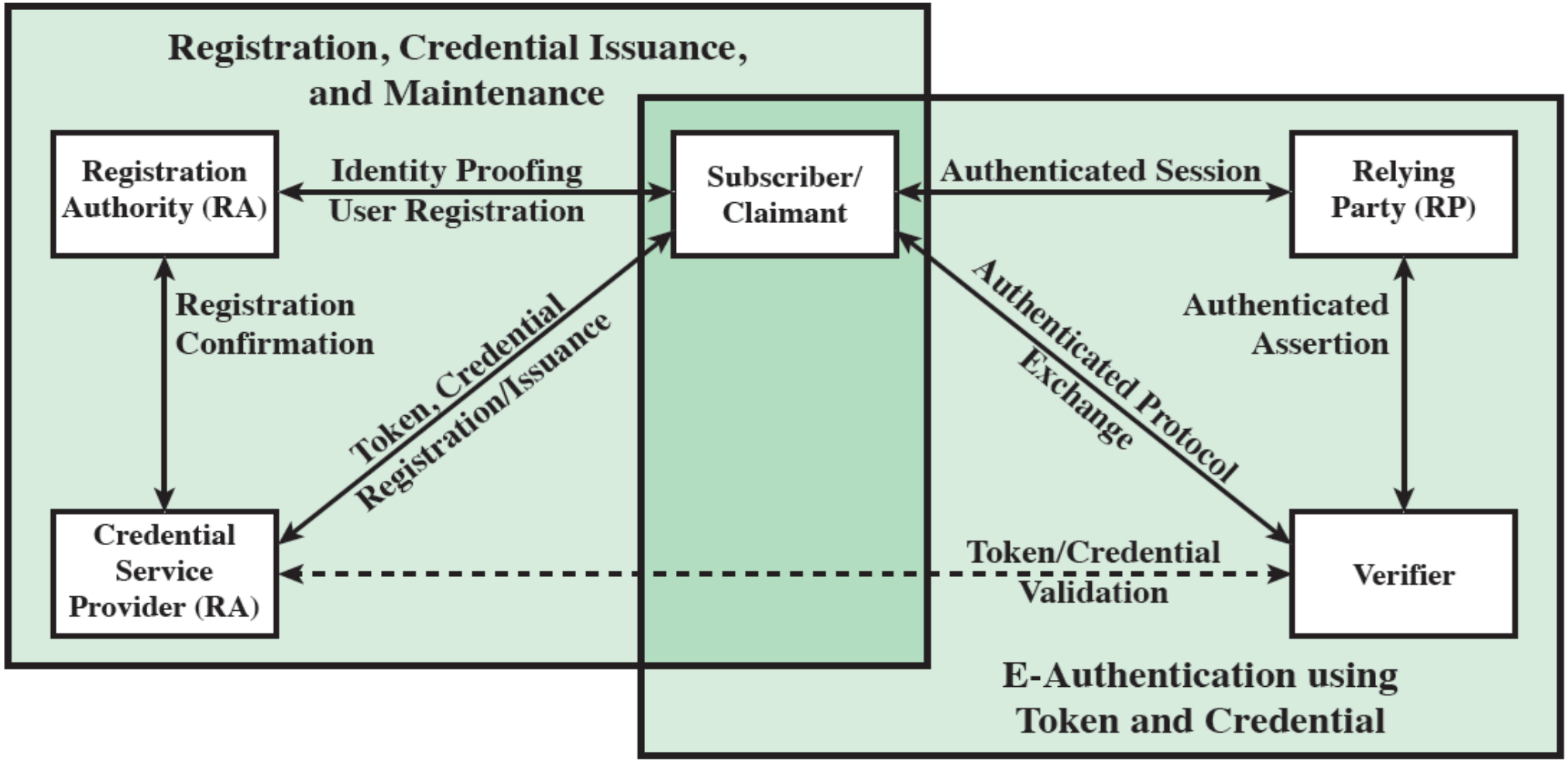
User Authentication

- Fundamental security building block
- basis of **access control** & **user accountability**

- The process of verifying an identity claimed by an entity/system entity
  - Two steps:
    - Identification: specify identifier
    - Verification: bind entity (person) and identifier

- Distinct from message authentication.

A Model for Electronic User Authentication (NIST SP 800-63-2)



Types of User Authentication

- Four means of authenticating user's identity
- Based one something the individual
  - knows, e.g. password, PIN
  - possesses, e.g. key, token, smartcard
  - is (static biometrics), e.g. fingerprint, retina
  - does (dynamic biometrics), e.g. voice, sign
- Can use alone or combined
- All can provide user authentication
- All have issues

Password Based Authentication

- **Widely used user authentication method**
  - user provides name/login and password
  - system compares password with that saved for specified login

- **Authenticates ID of user logging and**
  - that the user is authorized to access system
  - determines the user's privileges
  - is used in discretionary access control

Password Vulnerabilities and Countermeasures

- Offline dictionary attack
  - Prevent unauthorized access to system password file, intrusion detection system to detect compromise and re issuance of compromised password
- Specific Account Attack:
  - Restrict the number of attempts to login.
- Popular Password Attack:
  - Policies to avoid common passwords, scanning IP address of the authentication and client's cookies.
- Password guessing against single user:
  - Policies to form difficult to guess password.
  - addresses the secrecy, minimum length of the password, character set, prohibition against using well-known user identifiers, and length of time before the password must be changed.

Password Vulnerabilities and Countermeasures

- Workstation Hijacking:
  - Automatic logout after pre-defined time, intrusion detection system.
- Exploiting user mistakes:
  - Train the user about the secure usage of password, intrusion detection and password usage associated with another method of authentication.
- Exploiting multiple password use:
  - Imposing the policies to discourage the multiple usage of the same passwords.
- Electronic monitoring

Why Password Based Authentication is Popular?

- Techniques the utilize client-side hardware, such as fingerprint scanners and smart card readers, require the implementation of the appropriate user authentication software to exploit this hardware on both the client and server.

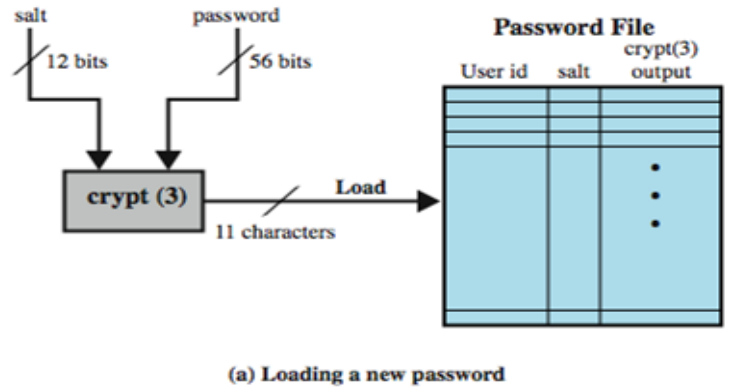
- Physical tokens, such as smart cards, are expensive and/or inconvenient to carry around, especially if multiple tokens are needed.

- Schemes that rely on a single sign-on to multiple services, using one of the non-password techniques create a single point of security risk.

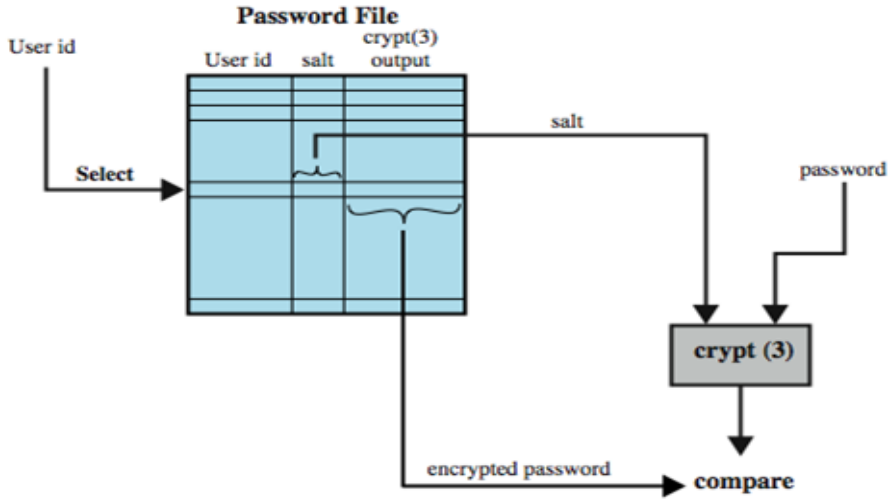
- Automated password managers have poor support for roaming and synchronization across multiple client platforms, and their usability had not be adequately researched.

Hashed Password

- **Benefits of Salt:**
  - Prevents duplicate passwords from being visible in the password file
  - Increases the difficulty of offline dictionary attacks
  - Nearly impossible to tell if a person used the same password on multiple systems



(a) Loading a new password



(b) Verifying a password

Unix Implementation of Password Schemes

- **Original scheme**
  - 8 character password form 56-bit key
  - 12-bit salt used to modify DES encryption into a one-way hash function
  - output translated to 11 character sequence
  - Now regarded as very insecure
- **Many systems now use MD5**
  - with 48-bit salt
  - password length is unlimited
  - is hashed with 1000 times inner loop
  - produces 128-bit hash
- OpenBSD uses Blowfish block cipher based hash algorithm called **Bcrypt**
- uses 128-bit salt to create 192-bit hash value

Password Cracking

Dictionary attacks

Try each word then obvious variants in large dictionary against hash in password file

Rainbow table attacks

- a large dict of possible passwords
- for each password:

precompute tables of hash values for all salts
a mammoth table of hash values: e.g. 1.4GB table cracks 99.9% of alphanumeric Windows passwords in 13.8 secs

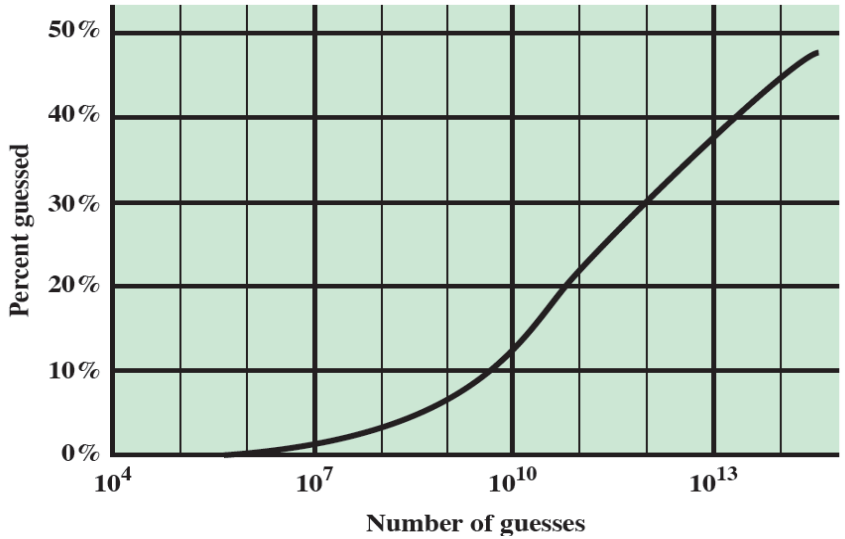
- not feasible if larger salt values and larger hash length is used

Password Cracking

- **users may pick short passwords**
  - e.g. 3% were 3 chars or less, easily guessed
  - system can reject choices that are too short
- **users may pick guessable passwords**
  - so crackers use lists of likely passwords
  - e.g. one study of 14000 encrypted passwords guessed nearly 1/4 of them
- would take about 1 hour on fastest systems to compute all variants, and only need 1 break!

Password Cracking – Modern Approach

- Users use better passwords, organizations impose password policies and sophisticated algorithms to generate passwords.
- Password cracking is still successful:
  - Availability of low cost, computationally powerful resources.
  - Leakage of passwords.



Password File Access Control

- Can block offline guessing attacks by denying access to password file.
- make available only to privileged users
- often using a separate shadow password (for su only)
- Still have vulnerabilities
  - exploit O/S bug
  - accident with permissions making it readable
  - users with same password on other systems
  - access from unprotected backup media
  - sniff passwords in unprotected network traffic

Password File Access Control

- /etc/password is world readable file. It can be modified by root or users with root privilege.
- In older linux, /etc/password contains user information and hashed passwords which is visible to all.
- Subject to password reusability and rainbow table attack.
- To prevent this problem, hashed passwords are separated from /etc/password and kepts in /etc/shadow files.
- It can be accessed and modified by the root user.
- /etc/password still is visible to everyone without the password information.

Password Selection Strategies

- Goal to eliminate guessable passwords
  - Still easy for user to remember
- Techniques:
  - User Education
  - Computer-generated passwords
  - Reactive password checking (periodic checking)
  - proactive password checking (at the time of selection)

Proactive Password Checking

- Rule enforcement plus user advice, e.g.
  - 8+ chars, upper/lower/numeric/punctuation
  - may not suffice
  - Rules can be tuned through proactive password checker.
- Password cracker
  - list of bad passwords
  - time and space issues
- Bloom Filter
  - use to build table based on dictionary using hashes
  - check desired password against this table

Boom Filter

- A Bloom filter of order ***k*** consists of a set of ***k*** independent hash functions $H_1(x), H_2(x), \ldots, H_k(x)$,
- each function maps a password ***x*** into a hash value in the range ***0*** to ***N – 1***.

$$H_i(X_j) = y \qquad 1 \le i \le k; \qquad 1 \le j \le D; \qquad 0 \le y \le N - 1$$

where

X_j = jth word in password dictionary
D = number of words in password dictionary

1. A hash table of N bits is defined, with all bits initially set to 0.
2. For each password, its k hash values are calculated, and the corresponding bits in the hash table are set to 1. Thus, if $H_i(X_j) = 67$ for some (i, j), then the sixty-seventh bit of the hash table is set to 1; if the bit already has the value 1, it remains at 1.

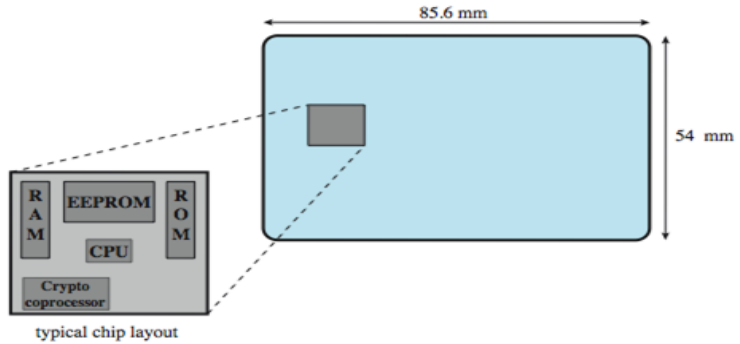
- False positive is possible.

Token-Based Authentication

- Object user possesses to authenticate, such as memory card (magnetic stripe) and smart card.
- **Memory Card:**
  - store but do not process data
  - magnetic stripe card, such as 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)

Token-Based Authentication

Smart Card:



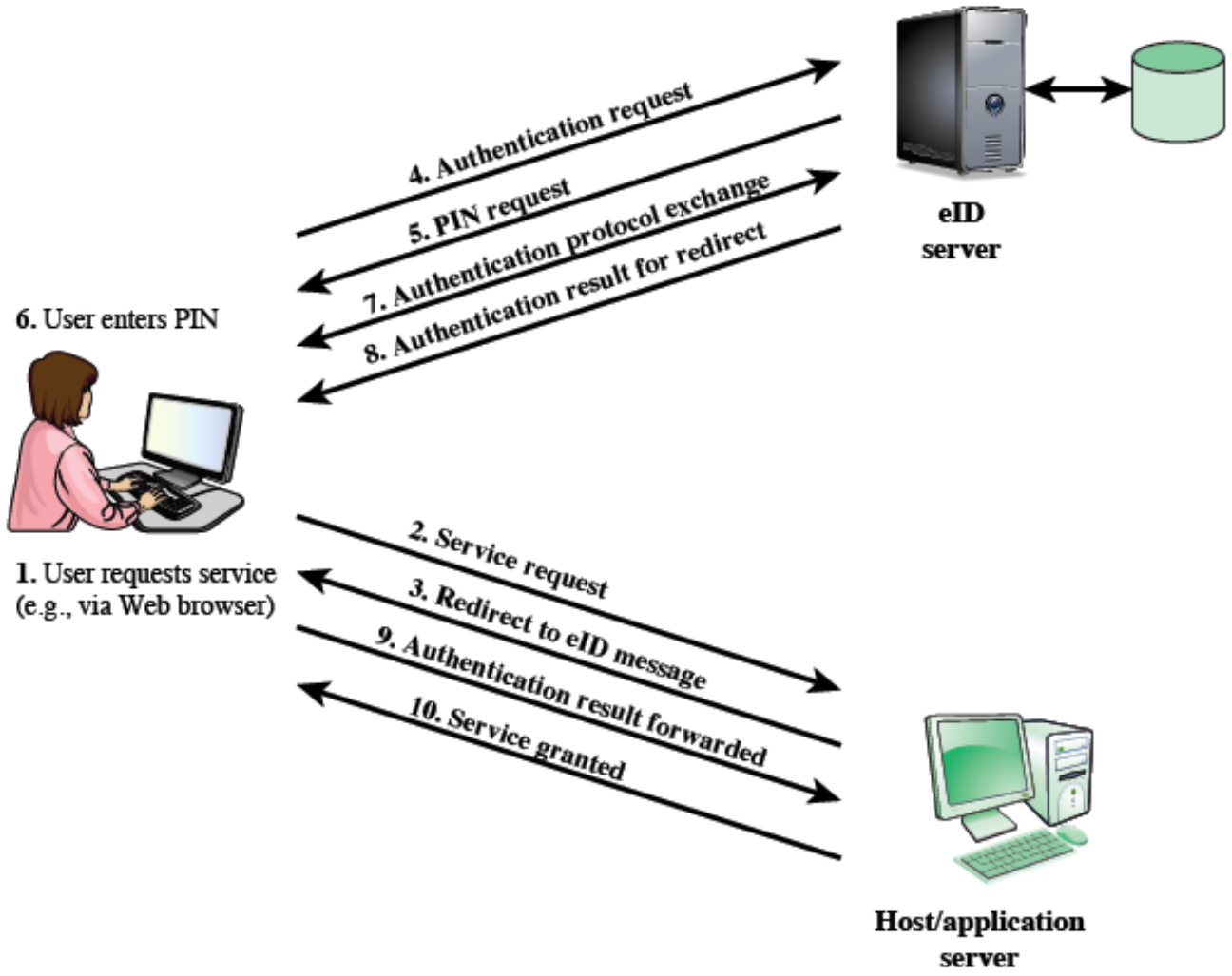
typical chip layout

- has own processor, memory, I/O ports
  - ROM, EEPROM, RAM memory
- Electronic interface: contact/contact less.
- 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)

Token-Based Authentication

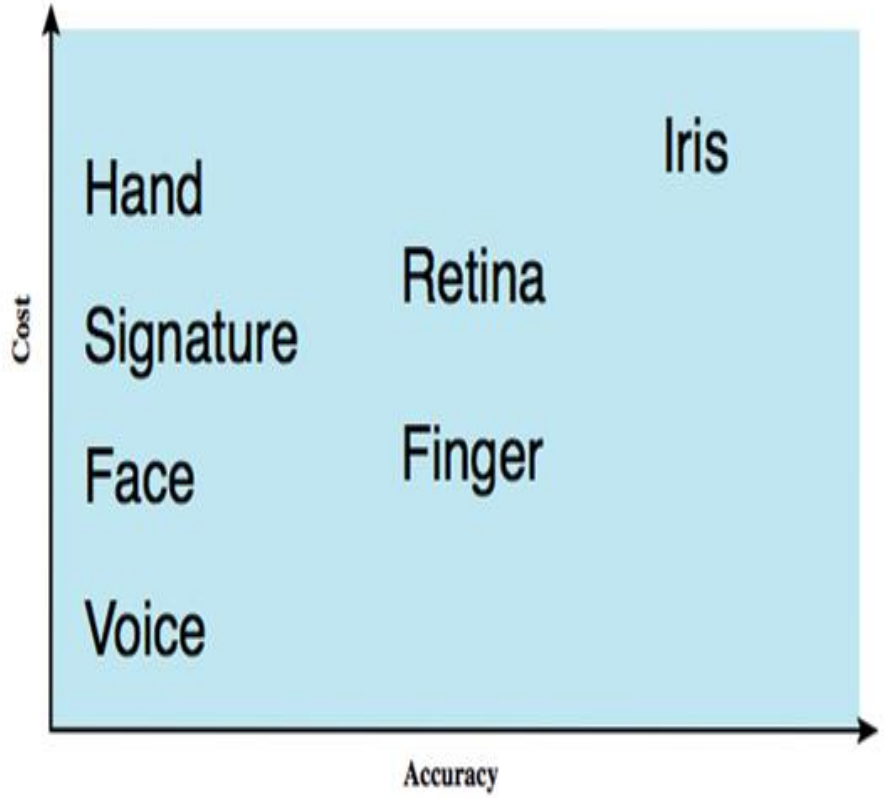
- An important application of smart cards is a national e-identity (eID)
- Serves the same purpose as other national ID cards such as driving license
- Can provide stronger proof of identity
- A German card
  - Personal data, Document number, Card access number (six digit random number), Machine readable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

User Authentication using e-ID

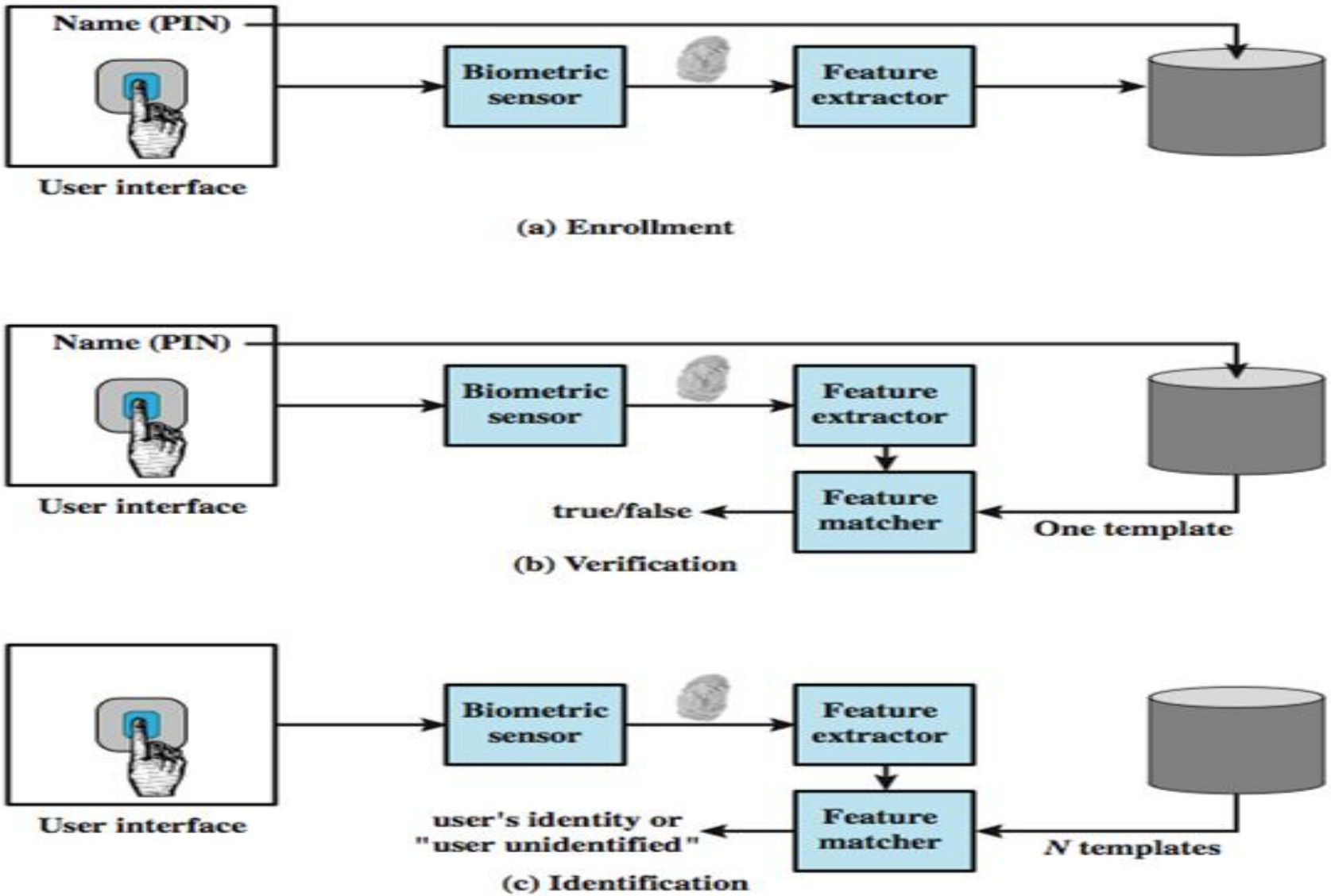


Biometric Authentication

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

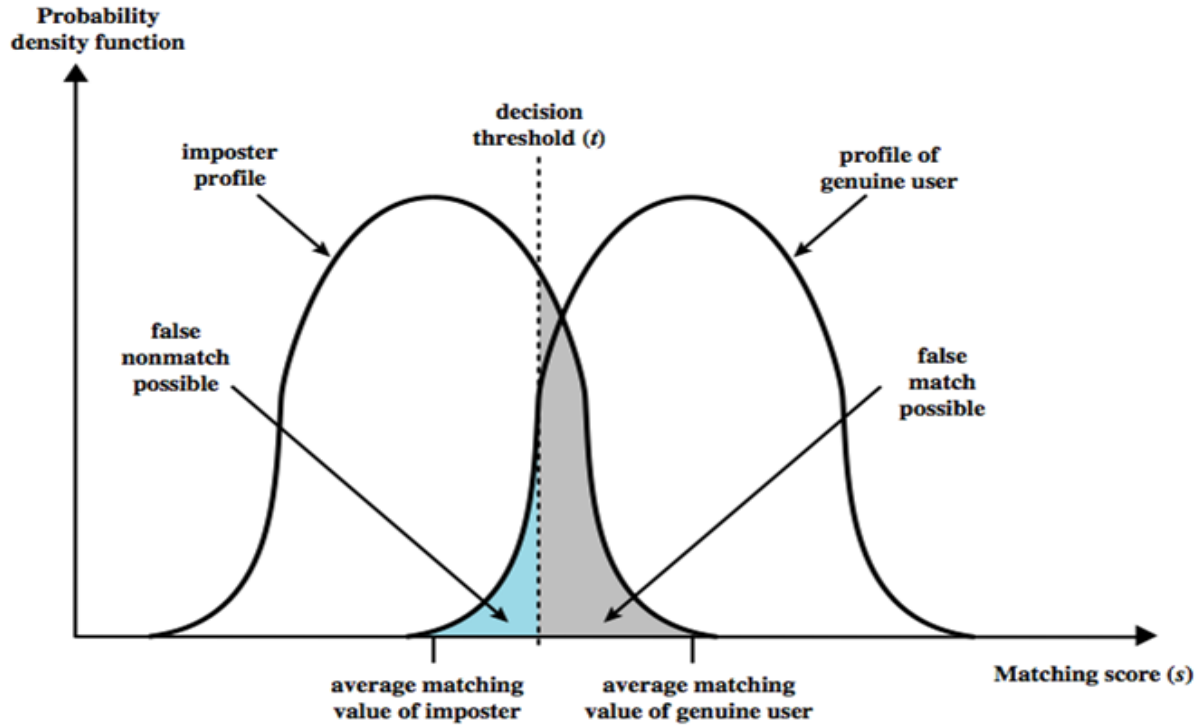


Operation of a Biometric System

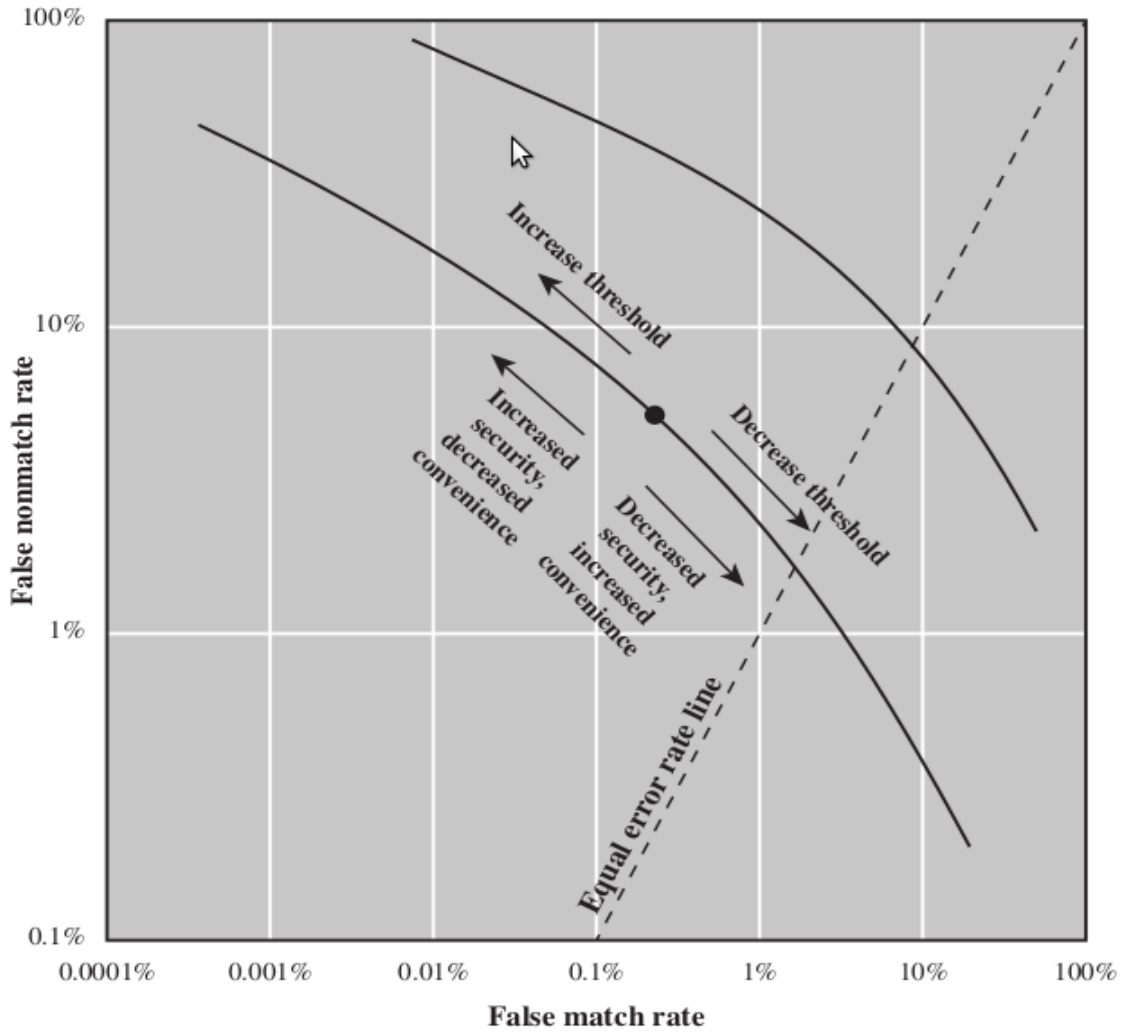


Biometric Accuracy

- 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

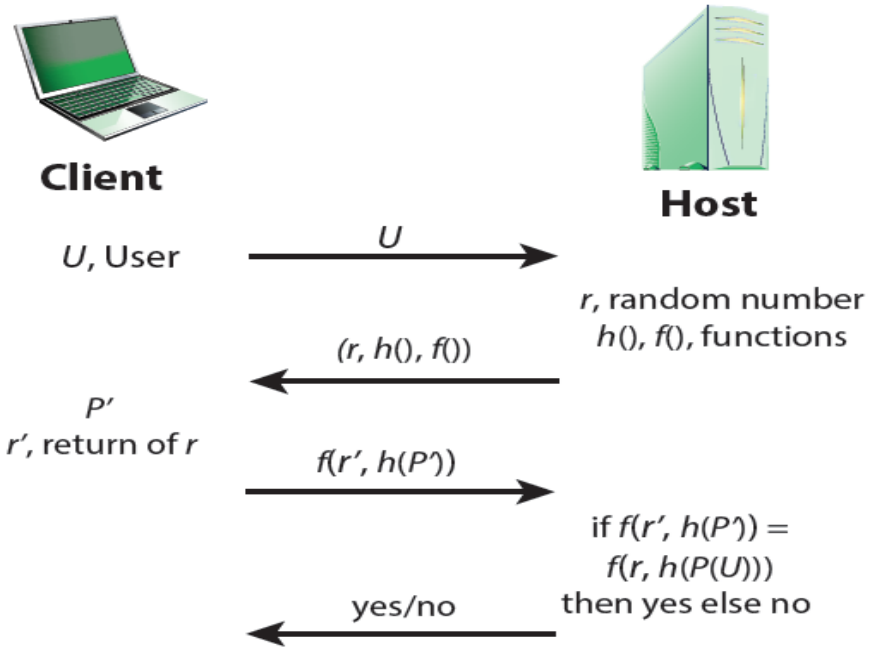


Biometric Accuracy

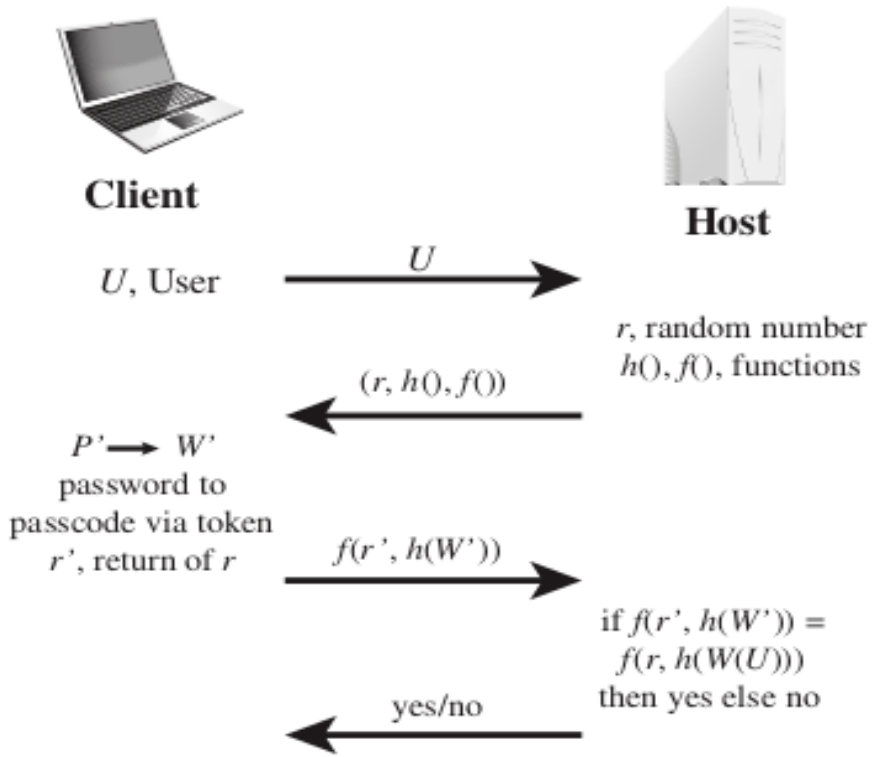


Remote User Authentication

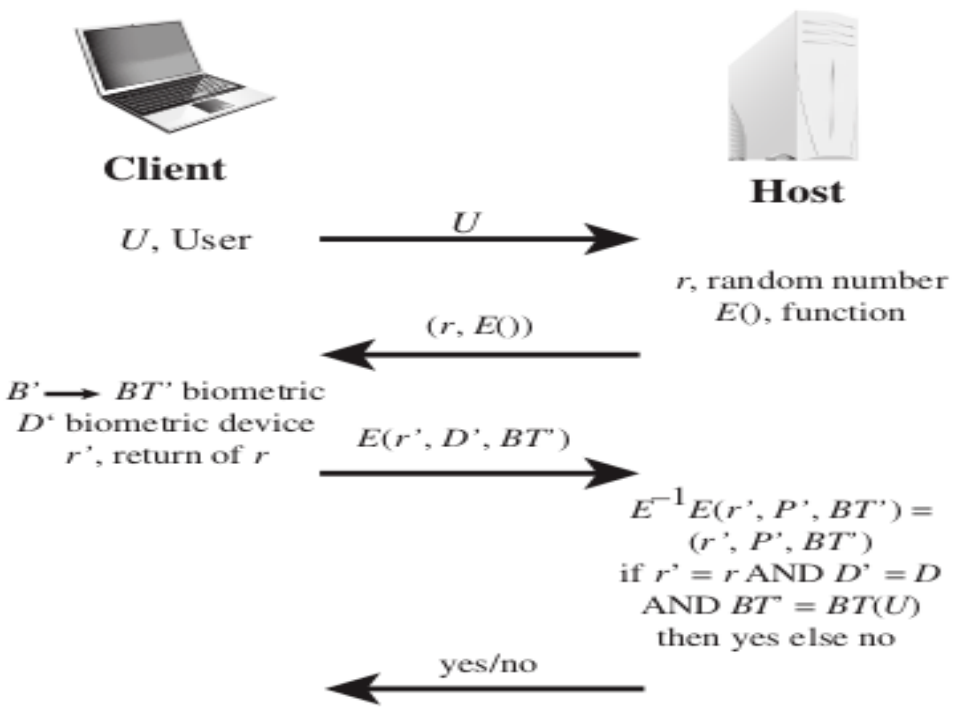
- Authentication over network is more complex
  - Problems of eavesdropping, replay
  - Generally use challenge-response
- **Protocol for a password verification**



Challenge-Response Protocol for Remote User Authentication

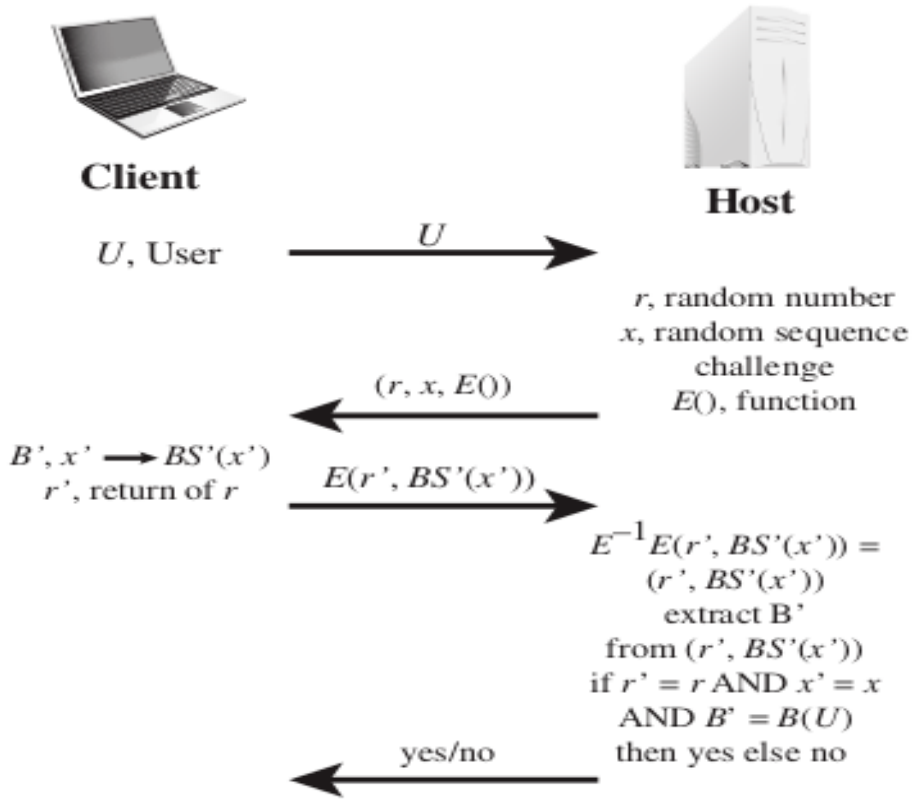


(b) Protocol for a token



(c) Protocol for static biometric

Challenge-Response Protocol for Remote User Authentication



(d) Protocol for dynamic biometric

Case Study of Dictionary Attacks

- We will see the dictionary attack attempts for SSH login on port 22.

```
tail -f /var/log/auth.log  |  sed G
```

- In dictionary attack, an attacker tries with possible account names in a machine and if a match is found, then proceeds with a large set of passwords.
- In brute force attack, an attacker tries with all possible user names and passwords, leading to a large search space. Besides, time and memory requirement increases with the length of user name and password.

Detecting Dictionary Attack with Log Scanning

- Attack with possible account names and then guessed password:

```
username tried: staff

Apr 10 13:59:59 moonshine sshd[32057]: Invalid user staff from 61.163.228.117
Apr 10 13:59:59 moonshine sshd[32057]: pam_unix(sshd:auth): check pass; user unknown
Apr 10 13:59:59 moonshine sshd[32057]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh
Apr 10 14:00:01 moonshine sshd[32057]: Failed password for invalid user staff from 61.163.228.117 port 40805 ssh2


username tried: sales

Apr 10 14:00:08 moonshine sshd[32059]: Invalid user sales from 61.163.228.117
Apr 10 14:00:08 moonshine sshd[32059]: pam_unix(sshd:auth): check pass; user unknown
Apr 10 14:00:08 moonshine sshd[32059]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh
Apr 10 14:00:10 moonshine sshd[32059]: Failed password for invalid user sales from 61.163.228.117 port 41066 ssh2


username tried: recruit

Apr 10 14:00:17 moonshine sshd[32061]: Invalid user recruit from 61.163.228.117
Apr 10 14:00:17 moonshine sshd[32061]: pam_unix(sshd:auth): check pass; user unknown
Apr 10 14:00:17 moonshine sshd[32061]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh
Apr 10 14:00:19 moonshine sshd[32061]: Failed password for invalid user recruit from 61.163.228.117 port 41303 ssh2


username tried: alias

Apr 10 14:00:26 moonshine sshd[32063]: Invalid user alias from 61.163.228.117
Apr 10 14:00:26 moonshine sshd[32063]: pam_unix(sshd:auth): check pass; user unknown
Apr 10 14:00:26 moonshine sshd[32063]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh
Apr 10 14:00:29 moonshine sshd[32063]: Failed password for invalid user alias from 61.163.228.117 port 41539 ssh2
```

Check the IP: 61.163.228.117 using www.ip2location.com

Detecting Dictionary Attack with Log Scanning

- Attack with root account and guessed passwords:

```
Apr 10 16:23:39 moonshine sshd[32305]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh
Apr 10 16:23:41 moonshine sshd[32305]: Failed password for root from 202.99.32.53 port 42732 ssh2

Apr 10 16:23:48 moonshine sshd[32307]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh
Apr 10 16:23:50 moonshine sshd[32307]: Failed password for root from 202.99.32.53 port 42976 ssh2

Apr 10 16:23:58 moonshine sshd[32309]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh
Apr 10 16:23:59 moonshine sshd[32309]: Failed password for root from 202.99.32.53 port 43208 ssh2

Apr 10 16:24:06 moonshine sshd[32311]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh
Apr 10 16:24:08 moonshine sshd[32311]: Failed password for root from 202.99.32.53 port 43439 ssh2

Apr 10 16:24:15 moonshine sshd[32313]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh
Apr 10 16:24:17 moonshine sshd[32313]: Failed password for root from 202.99.32.53 port 43659 ssh2

Apr 10 16:24:24 moonshine sshd[32315]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh
Apr 10 16:24:26 moonshine sshd[32315]: Failed password for root from 202.99.32.53 port 43901 ssh2

Apr 10 16:24:33 moonshine sshd[32317]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh
Apr 10 16:24:35 moonshine sshd[32317]: Failed password for root from 202.99.32.53 port 44128 ssh2

Apr 10 16:24:42 moonshine sshd[32319]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh
Apr 10 16:24:44 moonshine sshd[32319]: Failed password for root from 202.99.32.53 port 44352 ssh2

Apr 10 16:24:51 moonshine sshd[32321]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh
Apr 10 16:24:53 moonshine sshd[32321]: Failed password for root from 202.99.32.53 port 44577 ssh2
```

Detecting Dictionary Attack with Log Scanning

- Attack with root account (exists in every machine) with “failed-POSSIBLE BREAK-IN ATTEMPTS”.

```
Apr 10 21:42:03 moonshine sshd[761]: reverse mapping checking ..... [78.153.210.68] failed - POSSIBLE BREAK-IN ATTEMPT!
Apr 10 21:42:04 moonshine sshd[761]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost
Apr 10 21:42:06 moonshine sshd[761]: Failed password for root from 78.153.210.68 port 44058 ssh2

Apr 10 21:42:08 moonshine sshd[763]: reverse mapping checking ..... [78.153.210.68] failed - POSSIBLE BREAK-IN ATTEMPT!
Apr 10 21:42:08 moonshine sshd[763]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost
Apr 10 21:42:09 moonshine sshd[763]: Failed password for root from 78.153.210.68 port 44210 ssh2

Apr 10 21:42:11 moonshine sshd[765]: reverse mapping checking ..... [78.153.210.68] failed - POSSIBLE BREAK-IN ATTEMPT!
Apr 10 21:42:11 moonshine sshd[765]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost
Apr 10 21:42:12 moonshine sshd[765]: Failed password for root from 78.153.210.68 port 44330 ssh2

Apr 10 21:42:14 moonshine sshd[767]: reverse mapping checking ..... [78.153.210.68] failed - POSSIBLE BREAK-IN ATTEMPT!
Apr 10 21:42:14 moonshine sshd[767]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost
Apr 10 21:42:16 moonshine sshd[767]: Failed password for root from 78.153.210.68 port 44440 ssh2

Apr 10 21:42:17 moonshine sshd[769]: reverse mapping checking ..... [78.153.210.68] failed - POSSIBLE BREAK-IN ATTEMPT!
Apr 10 21:42:17 moonshine sshd[769]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost
Apr 10 21:42:19 moonshine sshd[769]: Failed password for root from 78.153.210.68 port 44568 ssh2

Apr 10 21:42:20 moonshine sshd[771]: reverse mapping checking ..... [78.153.210.68] failed - POSSIBLE BREAK-IN ATTEMPT!
Apr 10 21:42:20 moonshine sshd[771]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost
Apr 10 21:42:22 moonshine sshd[771]: Failed password for root from 78.153.210.68 port 44698 ssh2
```

In SSH connection, after receiving a connection request, domain name is verified for the requesting IP. Any mismatch lead to “failed-POSSIBLE BREAK-IN ATTEMPTS” messages.

```
Apr 10 21:42:45 moonshine sshd[787]: reverse mapping checking \
    getaddrinfo for 210-68.colo.sta.blacknight.ie [78.153.210.68] \
    failed - POSSIBLE BREAK-IN ATTEMPT!
```

Protecting Home Network in SSH Connections

```
/etc/hosts.allow    :    sshd: xxx.xxx.xxx.xxx

/etc/hosts.deny     :    ALL: ALL
```

xxx.xxx.xxx.xxx = IP address from where SSH is allowed.

- /etc/hosts.allow takes precedence over /etc/hosts.deny file.
- It ensures only SSH connection from a specified IP is allowed

Thwarting Dictionary Attack using Log Scanning

- fail2ban is a popular tool to detect intrusion through log scanning.
- It detects intrusion by keeping track of number of login attempts during a prespecified time interval.
- It blacklists IP addresses by adding rules to the iptables firewall.
- It can also identify malicious behavior from an IP address using regular expression-based filter.
- It can block network access for any application that creates a log file for incoming requests.
- See */etc/fail2ban/jail.conf* for list of applications.
- Also you can bring our own server application under the monitoring of fail2ban.

fail2ban

- *sudo apt update* and *sudo apt install fail2ban*
- By default it monitors */var/log/auth.log*
- You can enable fail2ban for any application by setting *enable=true* in the specific section in *jail.conf*.
- To check whether fail2ban is up and running: `sudo fail2ban-client status`

```
Status
|- Number of jail:      1
`- Jail list:   sshd
```

Screen shot of /etc/fail2ban/jail.conf

```
#
# WARNING: heavily refactored in 0.9.0 release.  Please review and
#          customize settings for your setup.
#
# Changes:  in most of the cases you should not modify this
#           file, but provide customizations in jail.local file,
#           or separate .conf files under jail.d/ directory, e.g.:
#
# HOW TO ACTIVATE JAILS:
#
# YOU SHOULD NOT MODIFY THIS FILE.
#
# It will probably be overwritten or improved in a distribution update.
#
# Provide customizations in a jail.local file or a jail.d/customisation.local.
# For example to change the default bantime for all jails and to enable the
# ssh-iptables jail the following (uncommented) would appear in the .local file.
# See man 5 jail.conf for details.
#
# [DEFAULT]
# bantime = 1h
#
# [sshd]
# enabled = true
#
# See jail.conf(5) man page for more information



# Comments: use '#' for comment lines and ';' (following a space) for inline commen


[INCLUDES]

#before = paths-distro.conf
before = paths-debian.conf

# The DEFAULT allows a global definition of the options. They can be overridden
# in each jail afterwards.

[DEFAULT]

#
# MISCELLANEOUS OPTIONS
#

"jail.conf" [readonly] 980 lines, 25607 bytes
```

Iptable after fail2ban installation

```
Chain INPUT (policy ACCEPT)
target     prot opt source       destination
f2b-sshd   tcp  --  anywhere     anywhere         multiport dports ssh

Chain FORWARD (policy ACCEPT)

target     prot opt source       destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source       destination

Chain f2b-sshd (1 references)
target     prot opt source       destination
RETURN     all  --  anywhere     anywhere
```

- One of the table of Iptable is filter consisting of INPUT, FORWARD and OUTPUT section.
- sudo iptables -L

fail2ban monitors only SSH Connections

```
bantime = 3600

findtime  = 3600

maxretry = 5

mta = sendmail

destemail = root@localhost

action = %(action_mwl)s
```

In /etc/fail2ban/jail.local file not in /etc/fail2ban/jail.conf

denyhosts

- Although this package is removed from Ubuntu 20.04 and above, this is a widely deployed tool for intrusion prevention.
- It monitors intrusion in SSH at port 22.
- It adds blacklisted IP in */etc/hosts.deny* and in some other files for synchronization.
- Synchronization allows it to download blacklisted IP list from internet and match with your IP lists in */etc/hosts.deny*.
- Configuration file: */etc/denyhosts.conf*
- Log file: */var/log/denyhosts*

Auth.log after denyhosts is Fired Up

```
tried to connect as root:

Apr 25 16:29:03 moonshine sshd[31037]: reverse mapping .... [190.12.41.50] failed - POSSIBLE BREAK-IN ATTEMPT!
Apr 25 16:29:03 moonshine sshd[31037]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh
Apr 25 16:29:04 moonshine sshd[31037]: Failed password for root from 190.12.41.50 port 54042 ssh2


tried to connect as apple:

Apr 25 16:29:08 moonshine sshd[31039]: reverse mapping .... [190.12.41.50] failed - POSSIBLE BREAK-IN ATTEMPT!
Apr 25 16:29:08 moonshine sshd[31039]: Invalid user apple from 190.12.41.50
Apr 25 16:29:08 moonshine sshd[31039]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh
Apr 25 16:29:10 moonshine sshd[31039]: Failed password for invalid user apple from 190.12.41.50 port 54102 ssh2


tried to connect as magazine:

Apr 25 16:29:13 moonshine sshd[31041]: reverse mapping .... [190.12.41.50] failed - POSSIBLE BREAK-IN ATTEMPT!
Apr 25 16:29:13 moonshine sshd[31041]: Invalid user magazine from 190.12.41.50
Apr 25 16:29:13 moonshine sshd[31041]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh
Apr 25 16:29:15 moonshine sshd[31041]: Failed password for invalid user magazine from 190.12.41.50 port 54163 ssh2


tried to connect as sophia:

Apr 25 16:29:18 moonshine sshd[31043]: reverse mapping .... [190.12.41.50] failed - POSSIBLE BREAK-IN ATTEMPT!
Apr 25 16:29:18 moonshine sshd[31043]: Invalid user sophia from 190.12.41.50
Apr 25 16:29:18 moonshine sshd[31043]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rh
Apr 25 16:29:20 moonshine sshd[31043]: Failed password for invalid user sophia from 190.12.41.50 port 54227 ssh2

                        AND FINALLY CAUGHT BY DENYHOSTS:


 Apr 25 16:29:50 moonshine sshd[31060]: refused connect from ::ffff:190.12.41.50 (::ffff:190.12.41.50)
```

/etc/denyhosts.conf

```
DENY_THRESHOLD_INVALID = 5
DENY_THRESHOLD_VALID   = 10
```

Password Cracking Possible despite One-way Hash Property

- Total number of passwords composed of exactly six characters is only $26^6 = 308915776$.
- Given a hash of such a password, it would be trivial to construct a lookup table for all such hashes and acquire the password in less time than it takes to blink an eye.
- An intruder may not have any desire to crack all the passwords in the /etc/shadow file. All that the intruder wants is to break into just a couple of accounts where he/she can install his own software. For such an intruder, just being able to crack short passwords is good enough.
- **Solution:** salt and round..

Entries in /etc/shadow

- Each entry in /etc/shadow consists of 9 colon-separated fields.
- first field --- username;
- Second field -- the password hash
- Third field -- the date of last password change
- Fourth field -- the number of days the user must wait before he/she is allowed to change the password
- Fifth field -- the number of days after which the user will be forced to change the password.
- Modular Crypt Function (MCF):

```
$6$rounds=40000$ZVzZ72hf$Tf19cHUK0g.nf.I/Bpn5jd3jokKMEAIHssRW2OEUGfneuTUzkhNmGv9iDhjfeDpJtqOyGjtSeXSq8
```

```
  identifier:           6

  number of rounds:     40000

  salt:                 ZVzZ72hf

  actual hash value:    Tf19cHUK0g.nf.I/Bpn5jd3jokKMEAIHssRW2OEUGfneuTUzkhNmGv9iDhjfeDpJtqOyGjtSeXSq8
```

Purpose of Round Value

Password Hashing Scheme	Identifier
md5_crypt	1
bcrypt	2
bcrypt	2a
bcrypt	2x
bcrypt	2y
bsd_nthash	3
sha256_crypt	5
sha512_crypt	6
sun_md5_crypt	md5
sha1_crypt	sha1

- By hashing a multiple number of times, you make it that much harder to crack a password through any sort of a table lookup, rainbow or otherwise.
- The protection provided by salts is considered to be strong enough to thwart any lookup-table based attacks for several more years to come.
- With time computers become even more powerful and should massive disk storage become even more inexpensive, the additional protection made possible a variable number of rounds would certainly be put to greater use.

Salt

- A salt is simply a randomly chosen bit pattern that is combined with the actual password before it is hashed by a hashing algorithm.
- The salt used in the /etc/shadow entry (ZVzZ72hf) are eight Base64 characters, each standing for six bits. Therefore, this salt consists of a 48-bit word.
- In actual practice, a password hashing scheme is likely to create a repetitive concatenation of the salt bits and the password bits to form a bit pattern that is hashed. The precise nature of this concatenation and repetition depends on the password hashing scheme used.

Resources

- **Chapter 3**, Computer Security Principles and Practice --- William Stalling and Lawrie Brown
- Lecture 24 (section 24.1, 24.2, 24.3, 24.5) from https://engineering.purdue.edu/kak/compsec/.

Acknowledgment

Lecture slides 29-42 are adapted from lecture 24 of Avinash Kak from https://engineering.purdue.edu/kak/compsec/.