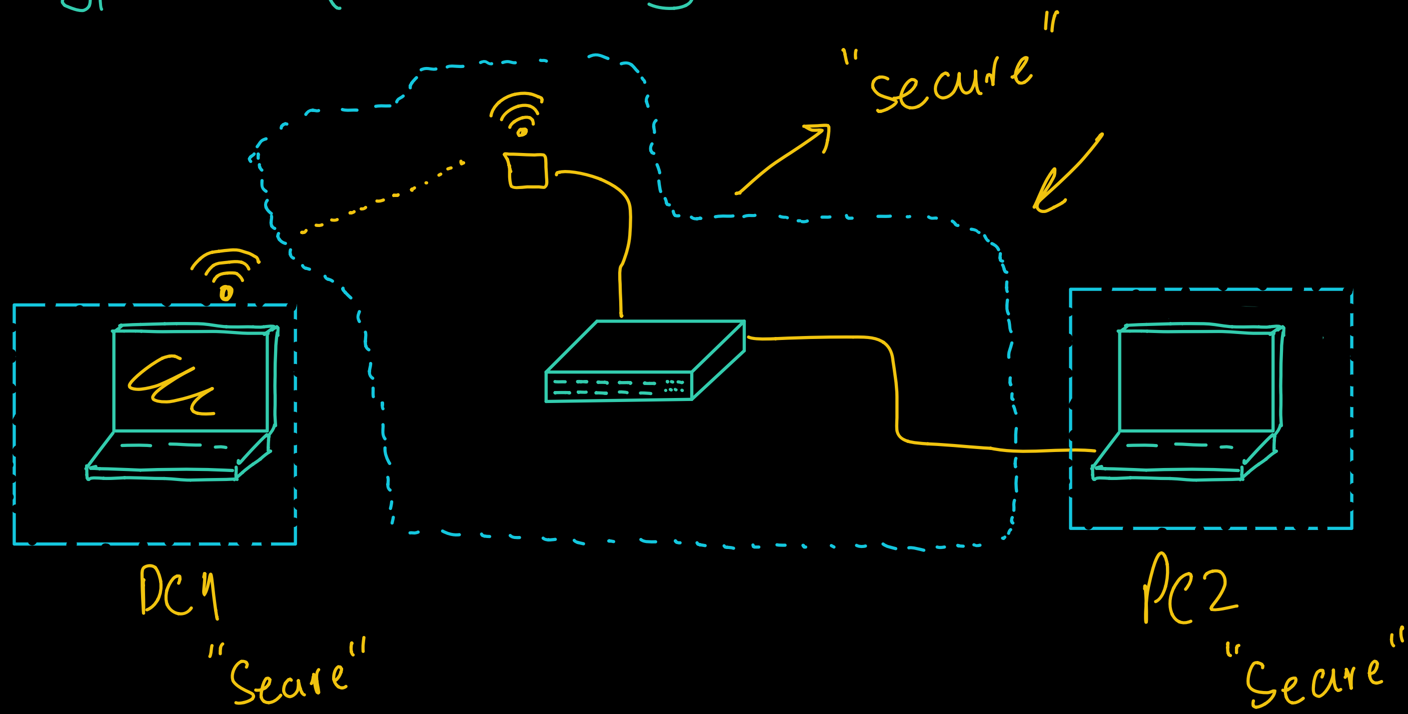
Operating Systems Design
19. Protection

Paul Krzyzanowski
pxk@cs.rutgers.edu

Types of security:



- Network security
- Systems security
  - "Authorization"
  - Malware
  - "Stuff"

Protection & Security

- Security Policy
  - Prevention of unauthorized access to a system
    - Malicious or accidental access
    - "access" may be:
      - user login, a process accessing things it shouldn't, physical access
    - The access operations may be reading, destruction, or alteration

- Protection Mechanism
  - The mechanism that provides and enforces controlled access of resources to processes
  - A protection mechanism *enforces* security policies

Principle of Least Privilege

- At each abstraction layer, every element (user, process, function) should be able to access *only* the resources necessary to perform its task
- Even if an element is compromised, the scope of damage is limited

Security Goals

- Authentication "کون"
  - Ensure that users, machines, programs, and resources are properly identified
- Confidentiality
  - Prevent unauthorized access to data
- Integrity
  - Verify that data has not been compromised: deleted, modified, added
- Availability
  - Ensure that the system is accessible

The Operating System

Subject Object

Action

Right

- The OS provides processes with access to resources

Resource Objects	OS component (Subjects)
Processor(s)	Process scheduler
Memory	Memory Management + MMU
Peripheral devices	Device drivers & buffer cache
Logical persistent data	File systems
Communication networks	Sockets

S
O
R

App
syscall
OS

- Resource access attempts go through the OS
- OS decides whether access should be granted
  - Rules that guide the decision = *policy*

Domains of protection

subjects

- Processes interact with objects
  - Objects:
    hardware (CPU, memory, I/O devices)
    software: files, semaphores, messages, signals

- A process should be allowed to access only objects that it is authorized to access
  - A process operates in a **protection domain**
  - Protection domain defines the objects the process may access and how it may access them

D_0

Modeling Protection: Access Matrix

D0 Bilal, Usman ...

D1 Farooq

- Rows: domains
- Columns: objects
- Each entry represents an access right of a domain on an object

objects

domains of protection

	F_0	F_1	*Printer*
D_0	read	read-write	print
D_1	read-write-execute	read	
D_2	read-execute		
D_3		read	print
D_4			print

F_0

- Desktop ≡ 1
- Documents ≡ 1

F_1

- /var ≡ 1

Access Matrix: Domain Transfers

- ~~Switching from one domain to another is a configurable policy~~

D0 — HoD

D1 — Instructor

objects

domains of protection

	F_0	F_1	*Printer*	D_0	D_1	D_2	D_3	D_4
D_0	read	read-write	print	–	switch	switch		
D_1	read-write-execute	read			–			
D_2	read-execute				switch	–		
D_3		read	print					
D_4			print					

Access Matrix: Additional operations

- Owner: allow new rights to be added or removed
  - An object may be identified as being *owned* by the domain
  - Owner can add and remove any right in any column of the object

objects

domains of protection

	F_0	F_1
D_0	read owner	

Implementing an access matrix

- A single table is usually impractical
  - Big size: # domains (users) × # objects (files)
  - Objects may come and go frequently

Attend Evaluation

Sparse

- Access Control List
  - Associate a column of the table with each object

Implementing an access matrix

- Access Control List (ACL) Default Deny
  - Associate a column of the table with each object

objects

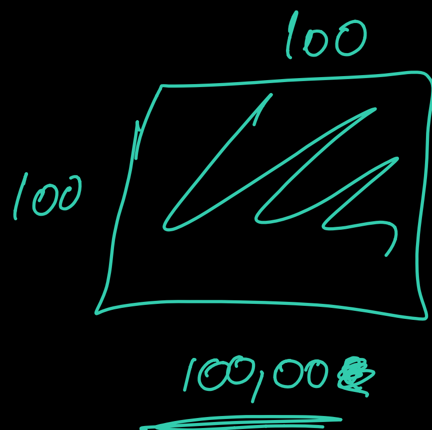
domains of protection

	F_0	F_1	*Printer*	D_0	D_1	D_2	D_3	D_4
D_0	read owner	read-write	print					
D_1	read-write-execute	read*			–			
D_2	read-execute				swtich	–		
D_3		read	print					
D_4			print					

ACL for file F_0

$\boxed{F_0}$ (Printer)

$$= \{ \quad \text{'}D_0\text{'} : [\underbrace{\text{owner}, \quad \text{read}}], \quad \text{'}D_1\text{'} : [\underbrace{\text{read}, \ \text{write}, \ \text{exec}}], \quad \vdots \quad \}$$



$F_0 = \{\}$

Access Control Models: MAC vs. DAC

Owner Policy

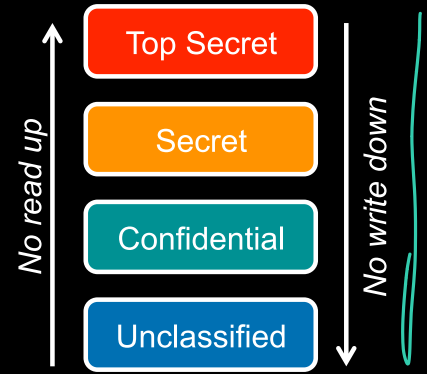
- DAC: Discretionary Access Control
  - A subject (domain) can pass information onto any other subject
  - In some cases, access rights may be transferred
  - *Most systems use this* (er... not really... only OSs)

- MAC: Mandatory Access Control
  - Policy is centrally controlled
  - Users cannot override the policy

Organization

Multi-level Access Control (skip)

- Typical MAC implementations use a Multi-Level Secure (MLS) access model
- Bell-LaPadula model
  - Identifies the ability to access and communicate data
  - Objects are classified into a hierarchy of sensitivity levels
    - Unclassified, Confidential, Secret, Top Secret
  - Users are assigned a clearance
  - "No read up; no write down"
    - Cannot read from a higher clearance level
    - Cannot write to a lower clearance level
- Works well for government information
- Does not translate well to civilian life

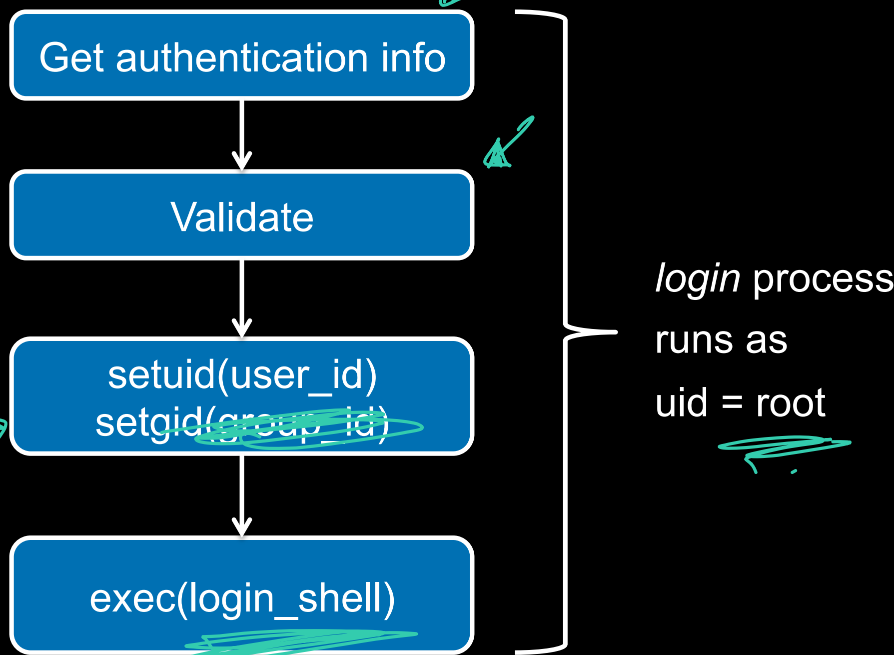


Confidential cannot read Secret
Confidential cannot write Unclassified

Authentication

Authentication

- Establish & verify identity
  - Then decide whether to allow access to resources
- Example: login



Password Authentication Protocol (PAP)

- Reusable passwords
- Server keeps a database of *username:password* mappings

Ghalat

- Prompt client/user for a login name & password
- To authenticate, use the login name as a key to look up the corresponding password in a database (file) to authenticate

match

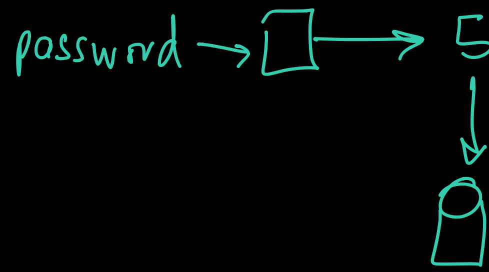
```
if (supplied_password == retrieved_password)
      then user is authenticated
```

hash (hash(sp) + [ ]) =? hash (h (rp) + [ ])

PAP: Reusable passwords

password breach

One problem: what if the password file isn't sufficiently protected and an intruder gets hold of it, he gets all the passwords!

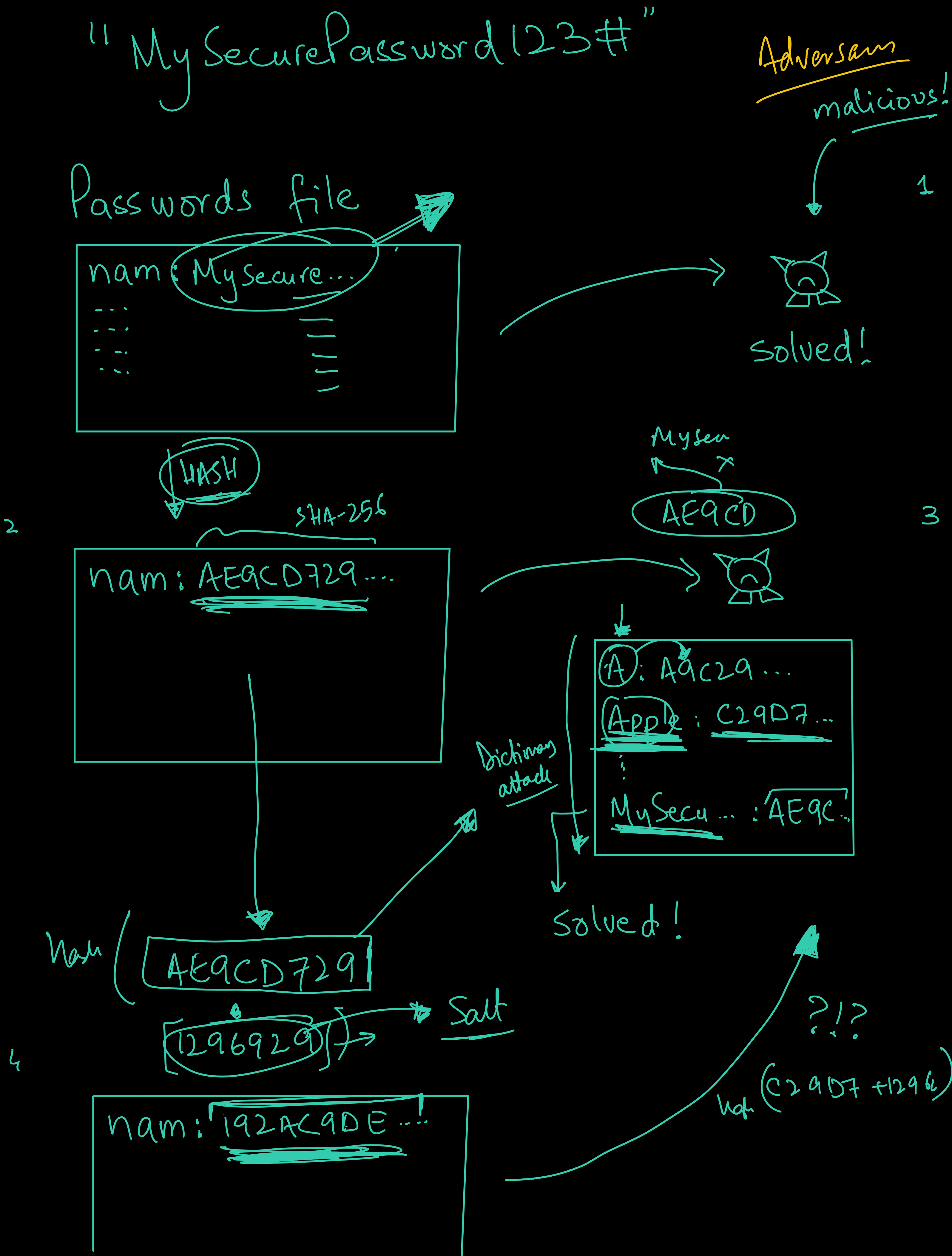


Enhancement:

Store a hash of the password in a file

- given a file, you don't get the passwords
- have to resort to a dictionary or brute-force attack

- Unix approach
  - Password encrypted with 3DES hashes; then MD5 hashes; now SHA512 hashes
  - Salt used to guard against dictionary attacks


"MySecurePassword123#"
Adversary
malicious!
1
Passwords file
nam: MySecure...
solved!
HASH
2
SHA-256
nam: AE9CD729....
Mysecu
AE9CD
3
A : A9C29...
Apple : C29D7...
MySecu... : 4E9C..
Dictionary attack
Solved!
hash (AE9CD729)
1296929
Salt
4
?!?
hash (C29D7+1296)
nam: 192AC9DE....

Hashes

9 mod 9 ≡ 0

5 mod 9 ≡ 5

14 mod 9 ≡ 5

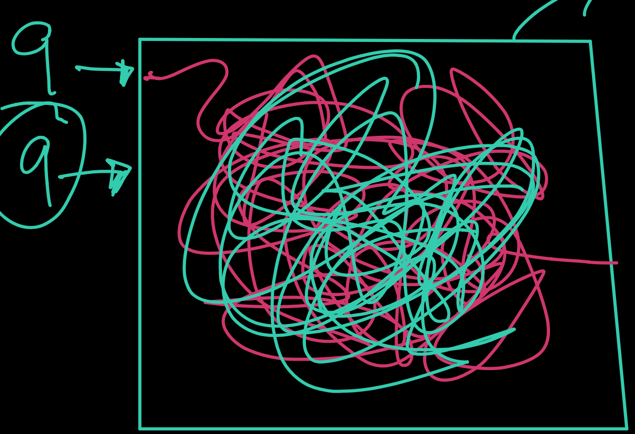
A Hash function

- It's bad because it has "collisions"...
- And it's reversible

Given 5, you can predict what the original number was

Secure hash function

SHA-1
SHA-256
SHA-512



A9C29DE47 .....

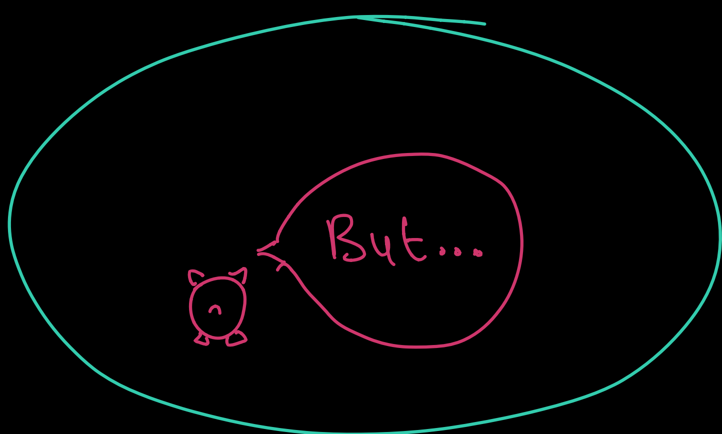
20 bytes

- Same output for same input
- Irreversible !

*

NEVER
EVER
EVER
STORE
PASSWORDS
IN
PLAINTEXT!

— EVER!



INSERT ____

VALUE (Username,
password (pass));

Username	Password
nam	AGCDG

Authentication

Three factors:

- something you have *key, card*, phone !
  - can be stolen
- something you know *passwords*
  - can be guessed, shared, stolen
- something you are *biometrics*
  - costly, can be copied (sometimes)

Authentication

factors may be combined

- ATM machine: 2-factor authentication
  - ATM card *something you have*
  - PIN *something you know*

Versus Authorization

Usman — policy

Authorization defines access control

Once we know a user's identity:

- Allow/disallow request
- Operating system enforces system access based on user's credentials
  - Network services usually run in another context
  - Network server may not know of the user
  - Application takes responsibility
- May contact an authorization server
  - Trusted third party that will grant credentials
  - Kerberos ticket granting service , LDAP, ActiveDirectory
  - RADIUS (centralized authentication/authorization)

Three (Four?) A's of Security

- Authentication
  - Validate an identity or a message
- Authorization (Access Control)
  - Enforce policy
- Accounting logging
- Auditing source

Accounting

If security has been compromised

… *what happened?*

… *who did it?*

… *how did they do it?*

Log transactions

- Logins
- Commands
- Database operations
- *Who looks at audits?*

Log to remote systems

- Minimize chances for intruders to delete logs

Auditing

Go through software source code and search for security holes

- Need access to source
  - Some operating systems > 50 million lines!
- Experienced staff + time
- E.g., OpenBSD

Complex systems will have more bugs

- And will be harder to audit

The End