Security

CS 4410
Operating Systems

[E. Birrell, A. Bracy, E. Sirer, R. Van Renesse]

Cornell CIS
COMPUTING AND INFORMATION SCIENCE

References: Security Introduction and Access Control by Fred Schneider

Historical Context

1961

Compatible Time-Sharing System (CTSS) is Demonstrated

The increasing number of users needing access to computers in the early 1960s leads to experiments in timesharing computer systems. Timesharing systems can support many users – sometimes hundreds – by sharing the computer with each user. CTSS was developed by the MIT Computation Center under the direction of Fernando Corbató and was based on a modified IBM 7094 mainframe computer. Programs created for CTSS included RUNOFF, an early text formatting utility, and an early inter-user messaging system that presaged email. CTSS operated until 1973.

1969

Kenneth Thompson and Dennis Ritchie develop UNIX

AT&T Bell Labs programmers Kenneth Thompson and Dennis Ritchie develop the UNIX operating system on a spare DEC minicomputer. UNIX combined many of the timesharing and file management features offered by Multics, from which it took its name. (Multics, a project of the mid-1960s, represented one of the earliest efforts at creating a multi-user, multi-tasking operating system.) The UNIX operating system quickly secured a wide following, particularly among engineers and scientists, and today is the basis of much of our world's computing infrastructure.

1960's OSes begin to be shared. Enter:

- Communication
- Synchronization
- Security: once a small OS sub-topic. *Not anymore!*

http://www.computerhistory.org, https://en.wikipedia.org

Security Properties: CIA

Confidentiality: *keeping secrets*

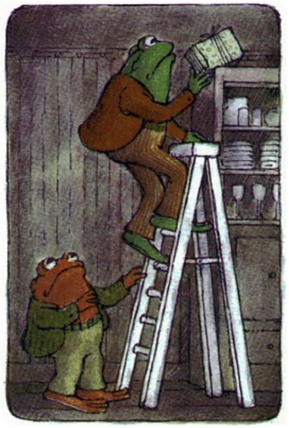
- who is allowed to learn what information

Integrity: *permitting changes*

- what changes to the system and its environment are allowed

Availability: *guarantee of service*

- service should be “timely”



Security in Computer Systems

Gold (Au) Standard for Security [Lampson]

Authorization: mechanisms that govern whether actions are permitted

Authentication: mechanisms that bind *principals* to actions

Audit: mechanisms that record and review actions

Plan of Attack *(no pun intended!)*

- Protection - *This lecture*
  - Authorization: *what are you permitted to do?*
  - Access Control Matrix
- Security – *Next lecture*
  - Authentication: *how do we know who you are?*
  - Threats and Attacks

Access Control Terminology

Operations: how one learns or updates information

Principals: executors (users, processes, threads, procedures)

Objects of operations: memory, files, modules, services

Access Control Policy:

- who may perform which operations on which objects
- enforces confidentiality & integrity

Goal: each object is accessed correctly and only by those principals that are allowed to do so

Access Control Mechanisms

Reference Monitor:

- entity with the power to observe and enforce the policy
- consulted on each operation invocation
- allows operation if invoker has required privileges
- can enforce **confidentiality** and/or **integrity**

Assumptions:

- Predefined operations are the sole means by which principals can learn or update information
- All predefined operations can be monitored (complete mediation)

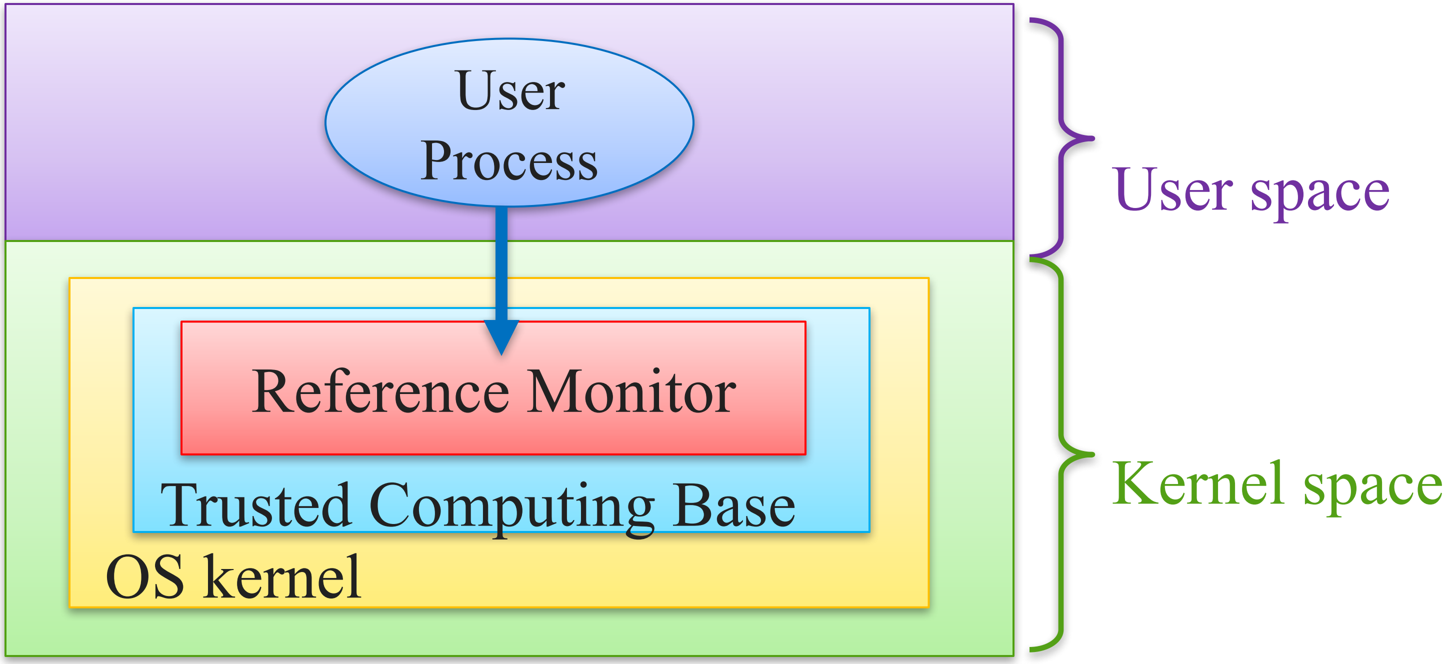
Trusted Computing Base (TCB)

Heart of every trusted system has a small TCB

- HW & SW necessary for enforcing security rules
- Typically has:
  - most hardware, firmware
  - portion of OS kernel
  - most or all programs with superuser power
- Desirable features include:
  - Should be small
  - Should be separable and well-defined
  - Easy to scrutinize independently

TCB and Reference Monitor

- All sensitive operations go through the reference monitor
- Monitor decides if operation should proceed
- Not a separable module in most OSes…



Who defines authorizations?

Discretionary Access Control:

- owner defines authorizations
- Subjects determine who has access to their objects
- Commonly used (Linux/MacOSX/Windows File Systems)
- Flawed for tighter security (program might be buggy)
- **This lecture**

Mandatory Access Control:

- System imposes access control policy that object owner’s cannot change
- centralized authority defines authorizations

Principle of Least Privilege

"Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job."

- Jerome Saltzer
(of the end-to-end argument)

Want to minimize:

- code running inside kernel
- code running as sysadmin

Challenge: It's hard to know:

- what permissions are needed in advance
- what permissions should be granted

Access Control Matrix

- Abstract model of protection
- Rows: principals = users
- Columns: objects = files, I/O, *etc.*

Principals	**OBJECTS**		
	`prelim.pdf`	`jan-hw.tex`	`scores.xls`
rvr (prof)	r, w	r	r, w
jan (student)		r, w	

Unordered set of triples <Principal,Object,Operation>

What does Principal of Least Privilege say about this?

Need Finer-Grained Principals

Protection Domains = new set of principals

- each process belongs to a protection domain
- executing process can transition from domain to domain

Example domain: user ▷ task

- task = program, procedure, block of statements
- task = started by user or in response to user's request
- user ▷ task: holds minimum privilege to get task done for user

→ *task-specific privileges (PoLP is ☺)*

Protection Domain Implementation

Possibilities:

1. Certain system calls cause protection-domain transitions. Obvious candidates:
   - invoking a program
   - changing from user mode to supervisor mode

2. Provide explicit domain-change syscall
   - application programmer or a compiler then required to decide when to invoke this domain-change system call

Access Matrix with Protection Domains

Principals	**OBJECTS**		
	prelim.pdf	jan-hw.tex	scores.xls
rvr▷sh			
rvr▷latex	r, w	r	
rvr▷excel			r, w
jan▷sh			
jan▷latex		r, w	
jan▷excel			

When to transition protection-domains?

- invoking a program
- changing from user to kernel mode
- …

Need to explicitly authorize them in the matrix

Access Matrix with Domain Transitions

Principals	OBJECTS								
	prelim.pdf	jan-hw.tex	scores.xls	rvr▷sh	rvr▷latex	rvr▷excel	jan▷sh	jan▷latex	jan▷excel
rvr▷sh					e	e			
rvr▷latex	r, w	r							
rvr▷excel			r, w						
jan▷sh								e	e
jan▷latex		r, w							
jan▷excel									

e = enter

DAC Implementation Needs

Must support:

- Determining if <*Principal,Object,Operation*> is in matrix
- Changing the matrix
- Assigning each process a protection domain
- Transitioning between domains as needed
- Listing each principal’s privileges (for each object)
- Listing each object’s privileges (held by principals)

2D array?

+ looks good in powerpoint!

– sparse → store only the non-empty cells

How shall we implement this?

Access Control List (ACL): column for each object stored as a list for the object

Principals	OBJECTS		
	prelim.pdf	jan-hw.tex	scores.xls
rvr▷sh			
rvr▷latex	r, w	r	
rvr▷excel			r, w
jan▷sh			
jan▷latex		r, w	
jan▷excel			

How shall we implement this?

Access Control List (ACL): column for each object stored as a list for the object

Capabilities: row for each subject stored as list for the subject

Principals	OBJECTS		
	prelim.pdf	jan-hw.tex	scores.xls
rvr▷sh			
rvr▷latex	r, w	r	
rvr▷excel			r, w
jan▷sh			
jan▷latex		r, w	
jan▷excel			

Same in theory; different in practice!

Access Control Lists

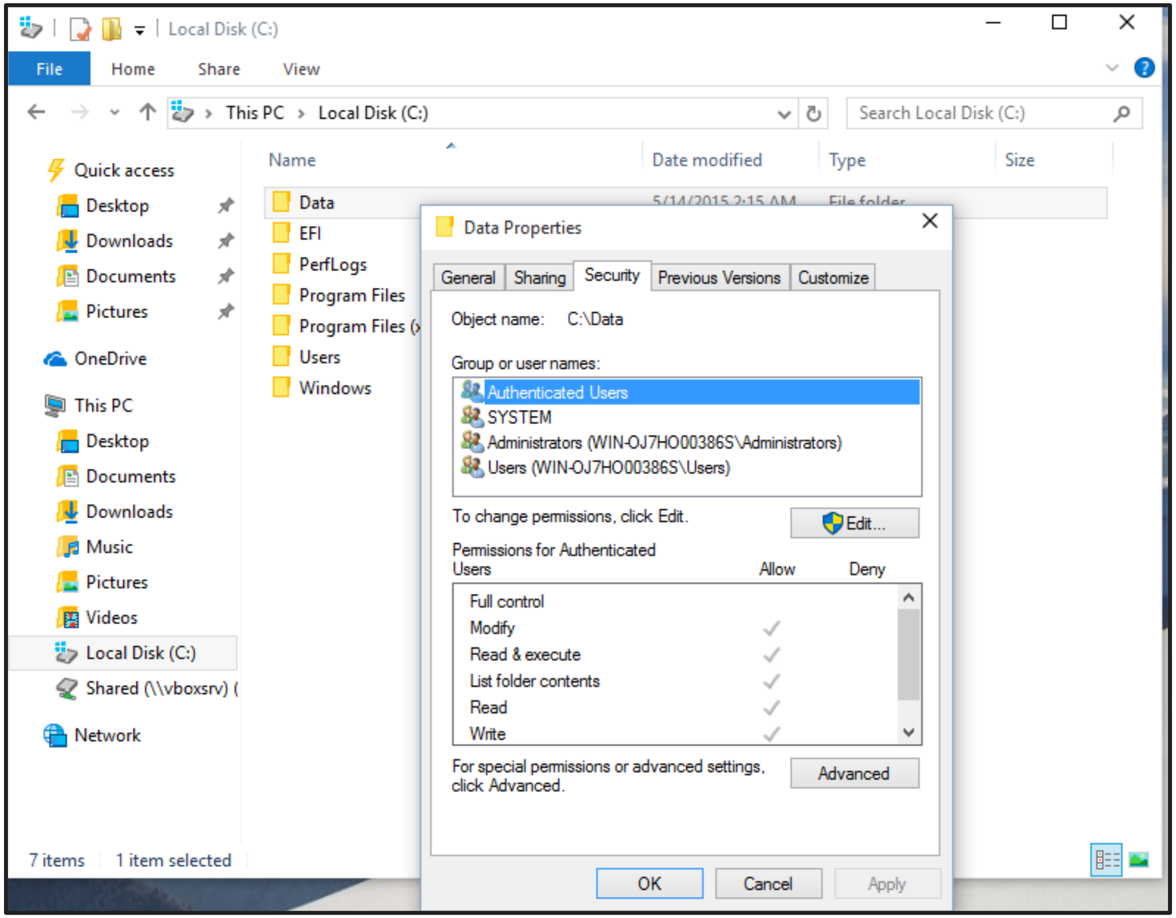
ACL for an object is a list

e.g., ⟨ebirrell, {r,w}⟩ ⟨clarkson, {r}⟩ ⟨student, {r}⟩

To check whether is allowed to perform some operation on some object,

- Look up principal in object's ACL. If not in ACL, reject
- Check whether operation is in the set for that principal. If not, reject

Access Control in Windows



Object name: C:\Data

Group or user names:

- Authenticated Users
- SYSTEM
- Administrators (WIN-OJ7HO00386S\Administrators)
- Users (WIN-OJ7HO00386S\Users)

To change permissions, click Edit.

Permissions for Authenticated Users	Allow	Deny
Full control		
Modify	✓	
Read & execute	✓	
List folder contents	✓	
Read	✓	
Write	✓	

For special permissions or advanced settings, click Advanced.

In NTFS: each file has a set of properties

- Richer set than UNIX: RWX
- P(permission) O(owner) D(delete), read (RX), change (RWXO), full control (RWXOPD)

Access Control Lists Roundup

Advantages:

- Efficient review of permissions for an object
- Centralized enforcement is simple to deploy, verify
- Revocation is straightforward

Disadvantages:

- Inefficient review of permissions for a principal
- Large ACLs take up space in object
- Vulnerable to confused deputy attack

Capability Lists

The capability list for a *principal* is a list

e.g., ⟨dac.tex, {r,w}⟩ ⟨dac.pptx, {r,w}⟩

Capabilities carry privileges:

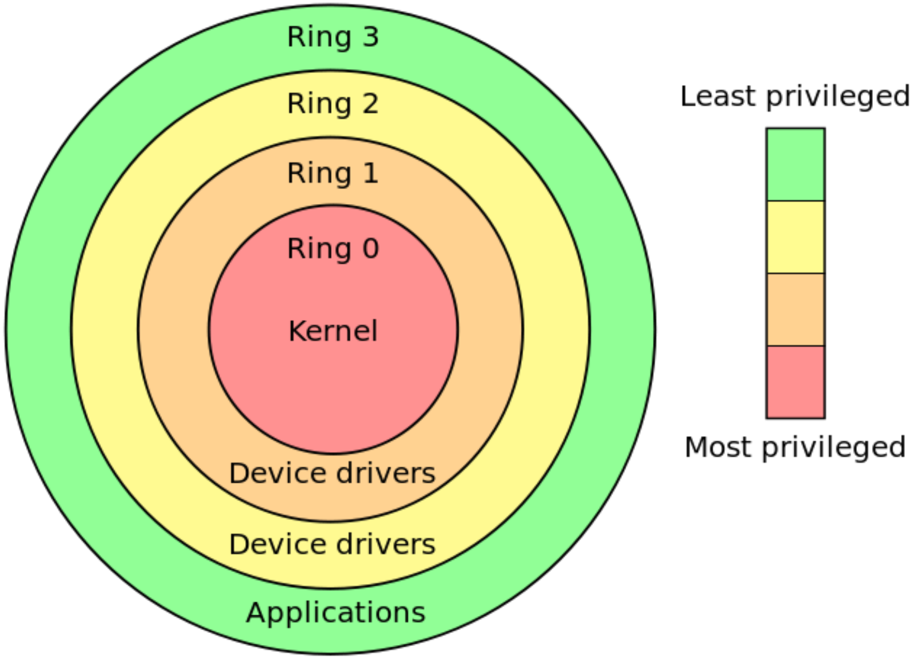
1) **Authorization:** Performing operation on object requires a principal to hold a capability such that
2) **Unforgeability:** Capabilities cannot be counterfeited or corrupted.

Note: Capabilities are (typically) transferable

C-Lists

OS maintains a list of capabilities for each principal (process)

1) **Authorization:** OS mediates access to objects, checks process capabilities
2) **Unforgeability:** capabilities are stored in protected memory region (kernel memory)



Access Control in UNIX

UNIX: has user and group identifiers: `uid` and `gid`

Per process: protection domain = `rvr|faculty▹sh`

Per file: ACL `owner|group|other` → stored in i-node

- Only owner can change these rights (using chmod)
- Each i-node has 3x3 RWX bits for user, group, others
- 2 mode bits allow process to change across domains
  - setuid, setgid bits

(Hybrid!) Approximation of access control scheme:

- Authorization (check ACL) performed at `open`
- Returns a file handle → essentially a capability
- Subsequent `read` or `write` uses the file handle

Capabilities Roundup

Advantages:

- Eliminates confused deputy problems
- Natural approach for user-defined objects

Disadvantages:

- Review of permissions?
- Delegation?
- Revocation?

ACLs vs Capabilities

	ACLs: For each Object: $<P_1,privs_1>$ $<P_2,privs_2>$…	**Capabilities:** For each Principal: $<O_1,privs_1>$ $<O_2,privs_2>$…
Review rights for object O	Easy! Print the list.	Hard. Need to scan all principals' lists.
Review rights for principal P across all objects	Hard. Need to scan all objects' lists.	Easy! Print the c-list.
Revocation	Easy! Delete P from O's list.	If kernel tracks capabilities, invalidates on revocation. Harder if object tracks revocation list.

History of Discretionary Access Control (DAC)

1760+ early philosophical pioneers of private property (Blackston, Bastiat,+)

1965 "access control lists" coined @ MIT describing Multics (CTSS foreshadowed ACLs) (Daley & Neumann)

1966 "capability" coined and OS supervisor outlined @ MIT (Dennis & van Horn)

1974 early computer security: "the user gives access rights at his own discretion" (Walter+)

1983 DoD's Orange book coins the term "discretionary access control"

Plan of Attack

- Protection
  - Authorization: *what are you permitted to do?*
  - Access Control Matrix

- Security
  - Authentication: *how do we know who you are?*
  - Threats and Attacks

Authentication

Establish the identity of user/machine by

- Something you are:
  retinal scan, fingerprint
- Something you have:
  physical key, ticket, credit card, smart card
- Something you know:
  password, secret, answers to security questions, PIN

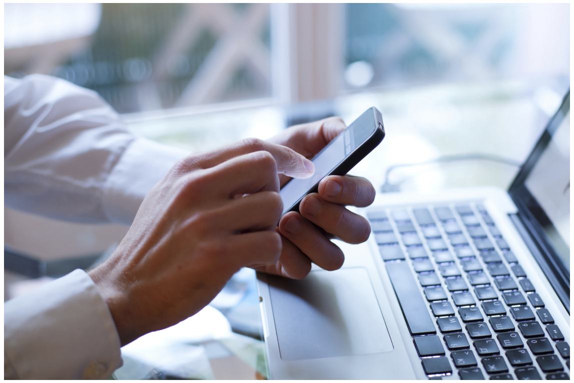
In the case of an OS this is done during login

- OS wants to know who the user is

Multiple Factors

Two-factor Authentication: authenticate based on two independent methods

- ATM card + PIN
- password + secret Q
- password + registered cell phone



Multi-factor Authentication: two or more independent methods

Best to combine separate categories

- 2 passwords from a same person? arguably not independent

Biometrics: something you are

- System has 2 components:
  - **Enrollment:** measure & store characteristics
  - **Identification:** match with user supplied values
- What are good characteristics?

  Finger length, voice, hair color, retinal pattern, blood

Pros: user carries around a good password

Cons: difficult to change password, can be subverted

Authentication with Physical Objects

Door keys have been around long

Plastic cards inserted into reader associated with comp

- Also a password known to user, to protect against lost card

Magnetic stripe cards: ~140 bytes info glued to card

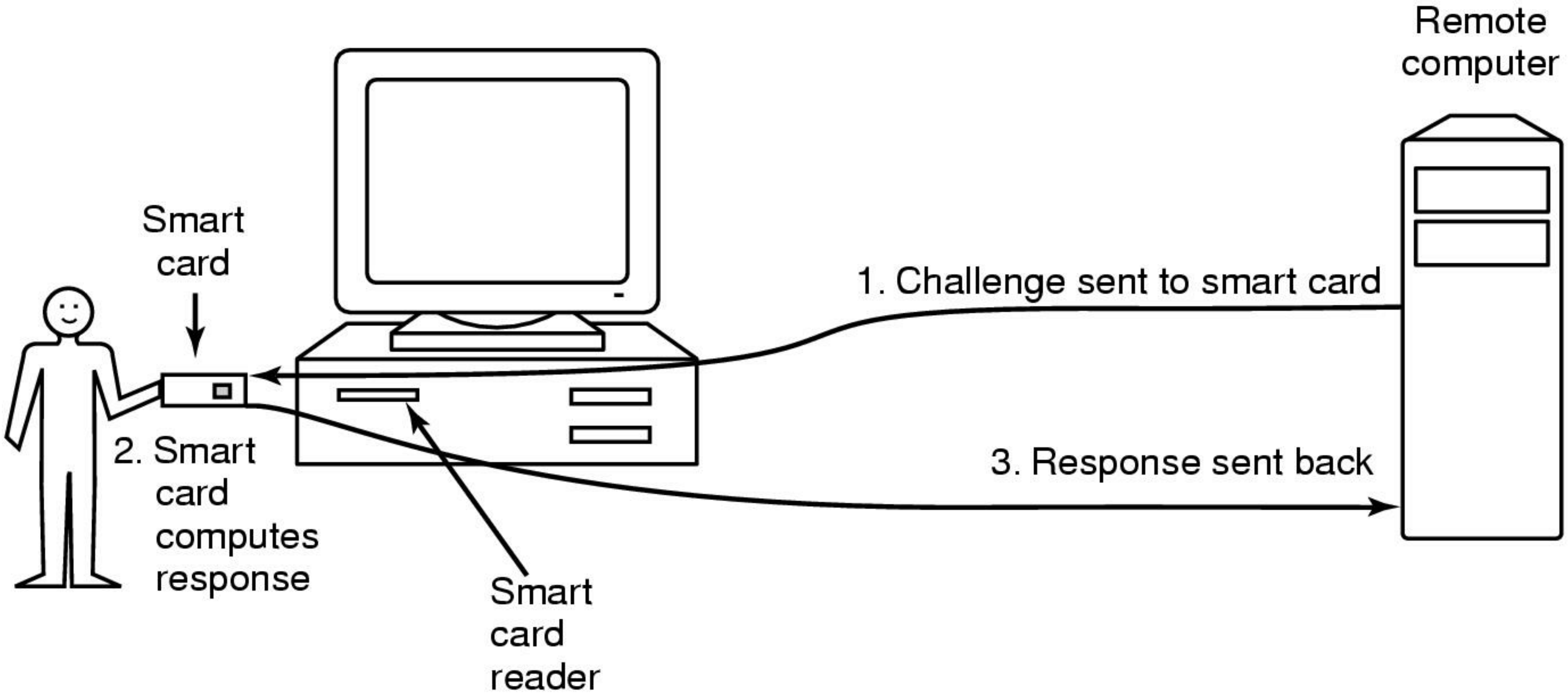
- Is read by terminal and sent to computer
- Info contains encrypted user password (only bank knows key)

Chip cards: have an integrated circuit

- Stored value cards: have EEPROM memory but no CPU
  - Value on card can only be changed by CPU on another comp
- Smart cards: 4 MHz 8-bit CPU, 16 KB ROM, 4 KB EEPROM, 512 bytes RAM, 9600 bps comm. channel

Smart Cards

- Better security than stored value cards
  - Card sends a small encrypted msg. to merchant, who can later use it to get money from the bank
  - **Pros:** no online connection to bank required
- Perform local computations, remember long passwords



Challenge Response Scheme

New user provides server with list of Q/A pairs

- Server asks one of them at random
- Requires a long list of question answer pairs

Prove identity by computing a secret function

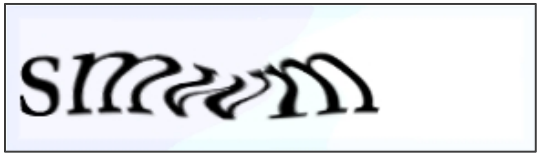
- User picks an algorithm, e.g. x^2
- Server picks a challenge, e.g. $x=7$
- User sends back 49
- Should be difficult to deduce function by looking at results

In practice

- Algorithm is fixed, e.g. one-way hash, but user selects a key
- The server's challenge is combined with user's key to provide input to the function

Authenticate yourself as a human:

CAPTCHA, image tasks, *etc.*



Passwords

Secret known only to the subject

Top 10 passwords in 2017: [SplashData]

1. 123456
2. password
3. 12345678
4. qwerty
5. 12345
6. 123456789
7. letmein
8. 1234567
9. football
10. iloveyou

16: starwars, 18: dragon, 27: jordan23

Top 20 passwords suffice to compromise 10% of accounts

[Skyhigh Networks]

Verifying Passwords

How does OS know that the password is correct?

Simplest implementation:

- OS keeps a file with ⟨login, password⟩ pairs
- User types password
- OS looks for a login→ password match

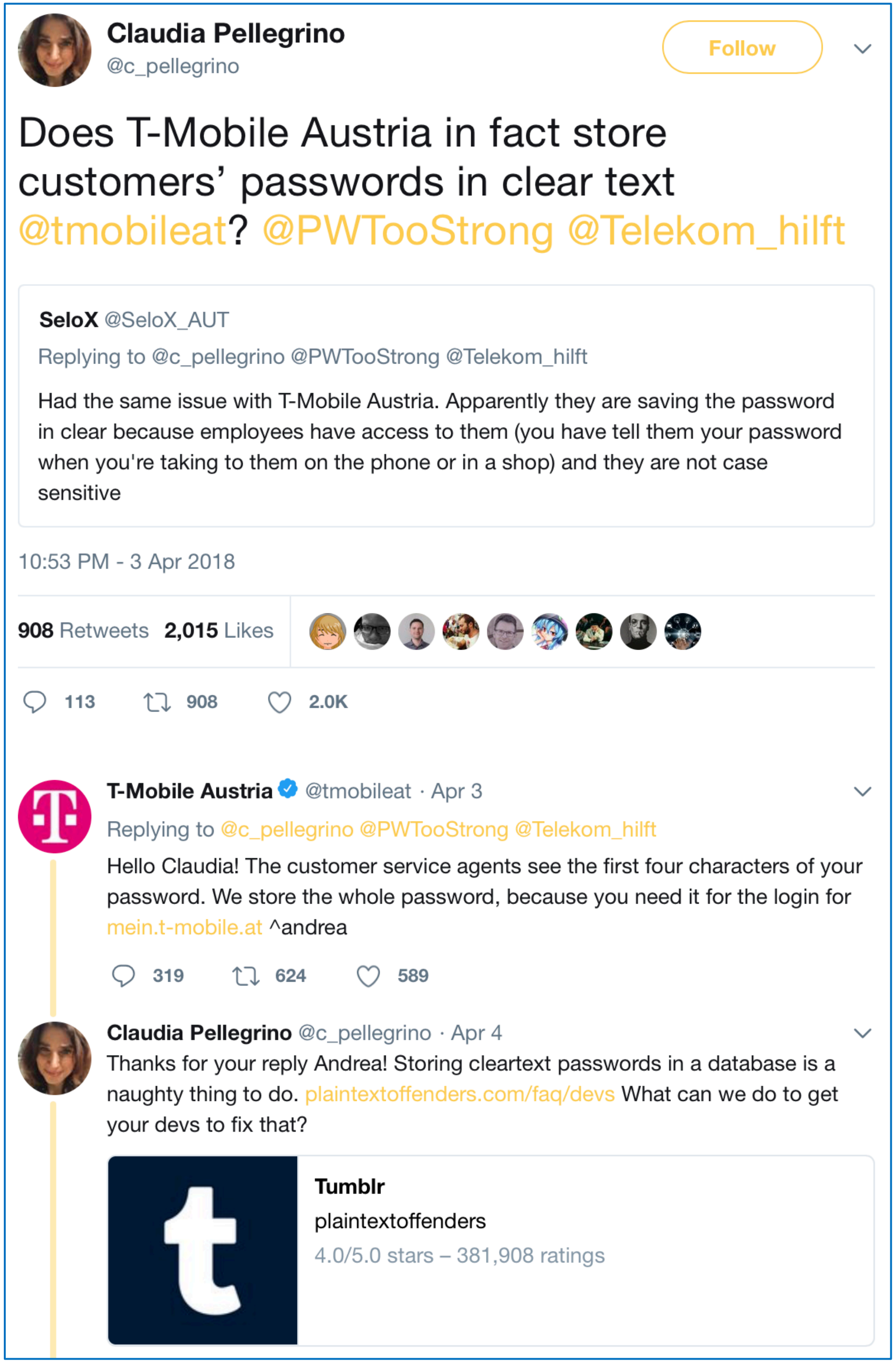
Goal: make this scheme as secure as possible

- display the password when being typed?

Storing Passwords

1. Store username/password in a file
   - Attacker only needs to read the password file
   - Security of system now depends on protection of this file!

     Need: perfect authorization & trusted system administrators



Claudia Pellegrino
@c_pellegrino

Does T-Mobile Austria in fact store customers' passwords in clear text @tmobileat? @PWTooStrong @Telekom_hilft

SeloX @SeloX_AUT
Replying to @c_pellegrino @PWTooStrong @Telekom_hilft

Had the same issue with T-Mobile Austria. Apparently they are saving the password in clear because employees have access to them (you have tell them your password when you're taking to them on the phone or in a shop) and they are not case sensitive

10:53 PM - 3 Apr 2018

908 Retweets 2,015 Likes

113 908 2.0K

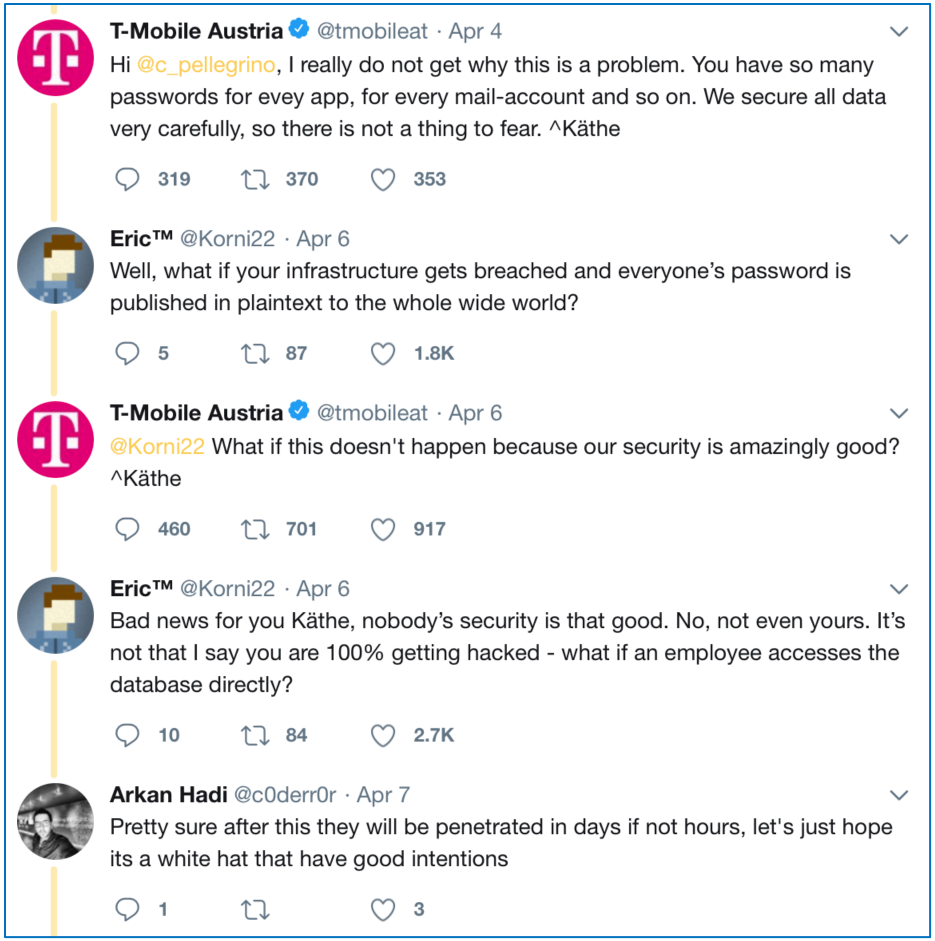
T-Mobile Austria @tmobileat · Apr 3
Replying to @c_pellegrino @PWTooStrong @Telekom_hilft

Hello Claudia! The customer service agents see the first four characters of your password. We store the whole password, because you need it for the login for mein.t-mobile.at ^andrea

319 624 589

Claudia Pellegrino @c_pellegrino · Apr 4
Thanks for your reply Andrea! Storing cleartext passwords in a database is a naughty thing to do. plaintextoffenders.com/faq/devs What can we do to get your devs to fix that?

Tumblr
plaintextoffenders
4.0/5.0 stars – 381,908 ratings



T-Mobile Austria @tmobileat · Apr 4
Hi @c_pellegrino, I really do not get why this is a problem. You have so many passwords for evey app, for every mail-account and so on. We secure all data very carefully, so there is not a thing to fear. ^Käthe

319 370 353

Eric™ @Korni22 · Apr 6
Well, what if your infrastructure gets breached and everyone's password is published in plaintext to the whole wide world?

5 87 1.8K

T-Mobile Austria @tmobileat · Apr 6
@Korni22 What if this doesn't happen because our security is amazingly good? ^Käthe

460 701 917

Eric™ @Korni22 · Apr 6
Bad news for you Käthe, nobody's security is that good. No, not even yours. It's not that I say you are 100% getting hacked - what if an employee accesses the database directly?

10 84 2.7K

Arkan Hadi @c0derr0r · Apr 7
Pretty sure after this they will be penetrated in days if not hours, let's just hope its a white hat that have good intentions

1 3

https://twitter.com/c_pellegrino/status/981409466242486272

Storing Passwords

1. Store username/password in a file
    - Attacker only needs to read the password file
    - Security of system now depends on protection of this file!

      Need: perfect authorization & trusted system administrators

2. Store login/encrypted password in file
    - Access to password file ≠ access to passwords

Hashing

Want a function f such that:

1. Easy to compute and store h(p)
2. Hard to compute p given h(p)
3. Hard to find q such that q ≠ p, h(q)==h(p)

Cryptographic hash functions to the rescue!

h(password) = encrypted-password *e.g.*, SHA

(but don't use SHA…)

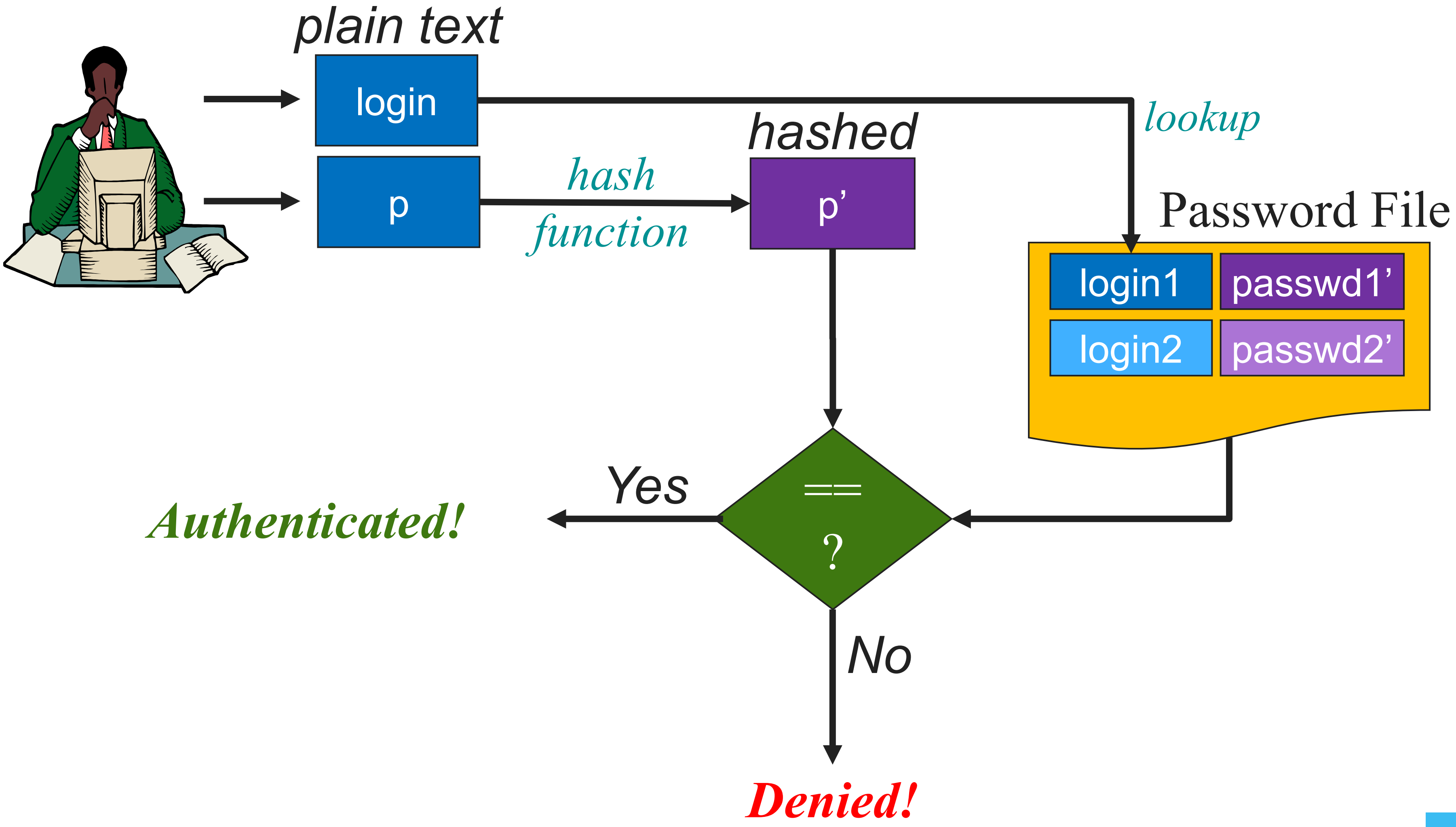
- one-way property gives (1) and (2)
- collision resistance gives (3)

Remember: *h(encrypted-password) ≠ password*

h^{-1}(encrypted-password) = password

h^{-1} hard to compute (hard ≈ impossible)

Storing and Checking Passwords



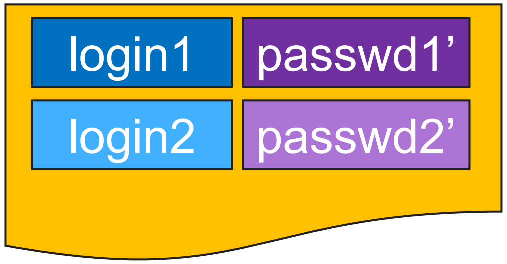
Hashed passwords still vulnerable

Suppose attacker obtains password file:

`/etc/passwd` public, known hash fn known

+ hard to invert → hard to obtain **all** the passwords

Password File



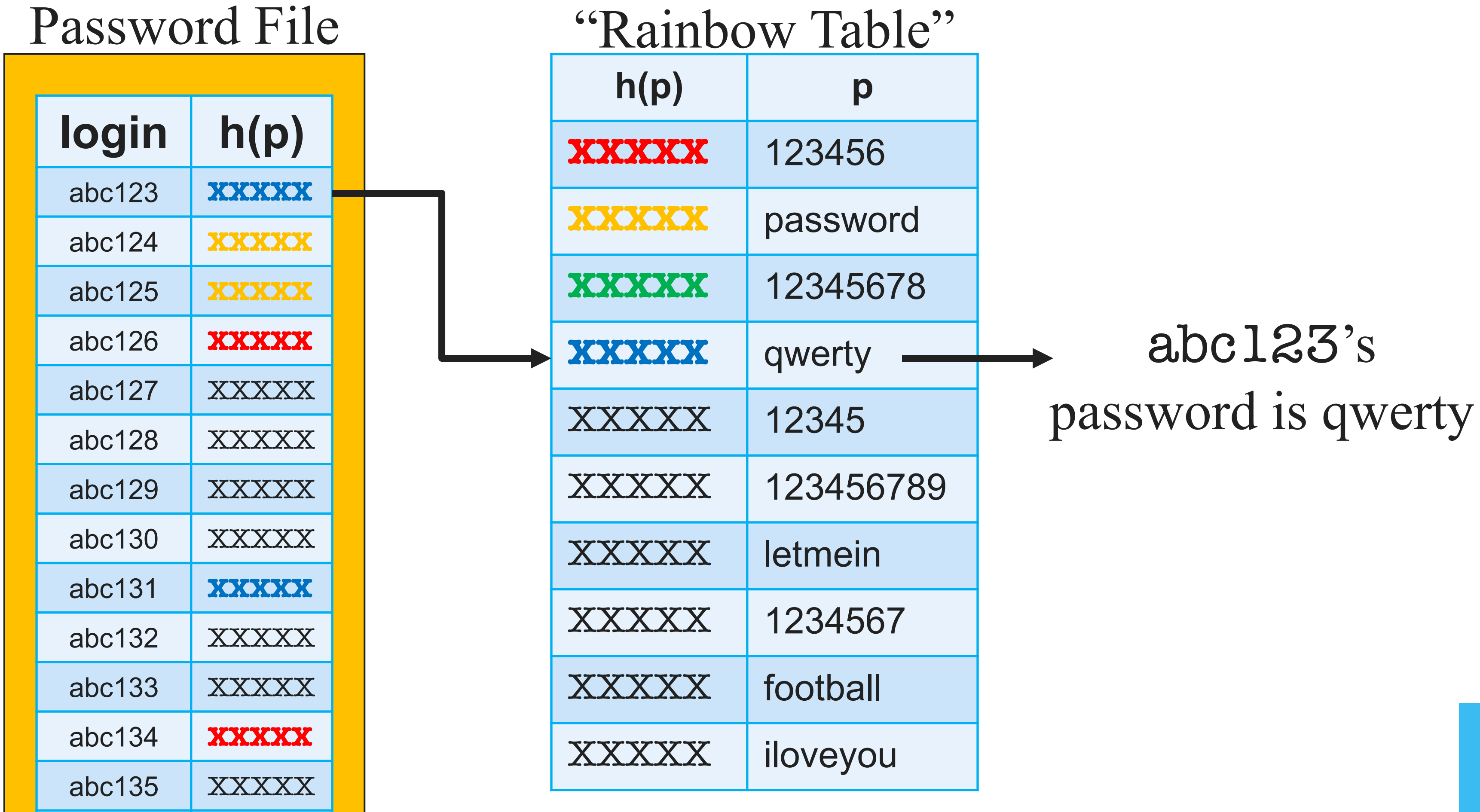
login1	passwd1’
login2	passwd2’

How else can I crack this file?

- Brute Force Attack:
  - Enumerate all possible passwords p, calculate h(p) and see if it matches an entry in the file
- Dictionary Attack
  - List all the likely passwords p, calculate h(p) and check for a match. (recall: top 20 passwords can compromise 10% of accounts)

Rainbow Table Attack

- Pre-compute the dictionary hashes (need space, not time), use hashed passwords as key
- Quick attack: look up each hashed password 1-by-1



Password File

login	h(p)
abc123	XXXXX
abc124	XXXXX
abc125	XXXXX
abc126	XXXXX
abc127	XXXXX
abc128	XXXXX
abc129	XXXXX
abc130	XXXXX
abc131	XXXXX
abc132	XXXXX
abc133	XXXXX
abc134	XXXXX
abc135	XXXXX

"Rainbow Table"

h(p)	p
XXXXX	123456
XXXXX	password
XXXXX	12345678
XXXXX	qwerty
XXXXX	12345
XXXXX	123456789
XXXXX	letmein
XXXXX	1234567
XXXXX	football
XXXXX	iloveyou

abc123's password is qwerty

More offline attacks

- Previous scheme can be attacked: Dictionary Attack
  - Attacker builds dictionary of likely passwords offline
  - At leisure, builds hash of all the entries
  - Checks file to see if hash matches any entry in password file
  - There will be a match unless passwords are truly random
  - 20-30% of passwords in UNIX are variants of common words
    - Morris, Thompson 1979, Klein 1990, Kabay 1997
- Solutions:
  - Shadow files: move password file to /etc/shadow
    - This is accessible only to users with root permissions
  - Salt: store *(user name, salt, E(password+salt))*
    - Simple dictionary attack will not work. Search space is more.

Salting

Vulnerabilities:

- single dictionary compromises all users
- passwords chosen from small space

Countermeasure: include a unique system-chosen nonce as part of each user's password

- make every user's stored hashed password different, even if they chose the same password
- now passwords come from a larger space

Each user has: **login**, unique salt **s**, passwd **p**
System stores: login, s, H(s, p)

Salting Example

login	salt	h(p||s)
abc123	4238	h(423812345)
abc124	2918	h(2918password)
abc125	6902	h(6902LordByron)
abc126	1694	h(1694qwerty)
abc127	1092	h(109212345)
abc128	9763	h(97636%%TaeFF)
abc129	2020	h(2020letmein)

- If the hacker guesses qwerty, has to try:
  h(0001qwerty), h(0002qwerty), h(0003qwerty) …
- UNIX adds 12-bit of salt
- Also, passwords should be secure:
  - Length, case, digits, not from dictionary
  - Can be imposed by the OS! This has its own tradeoffs