CS 165 – Computer Security

Passwords

January 11, 2024

Proving One’s Identity

- There are lots of users
  - Normal users
  - Administrators
- And lots of services to use on computer systems
  - University
  - Banking
  - Conferencing
  - Communication
- Each service may need to know who each user is. Why?

Authentication and Authorization

- To obtain the rights of a principal, one must prove that they can act as that principal
  - Called authentication
- Then, to use those rights a principal can perform authorized operations on the system
  - Called authorization or access control

Authentication in the Real World

- You often have to prove your identity to perform actions in the real world
  - To purchase jewelry at Tiffany’s you have to present a valid credit card
    - In the old days, you had to show additional identification to use the credit card

Authentication in the Real World

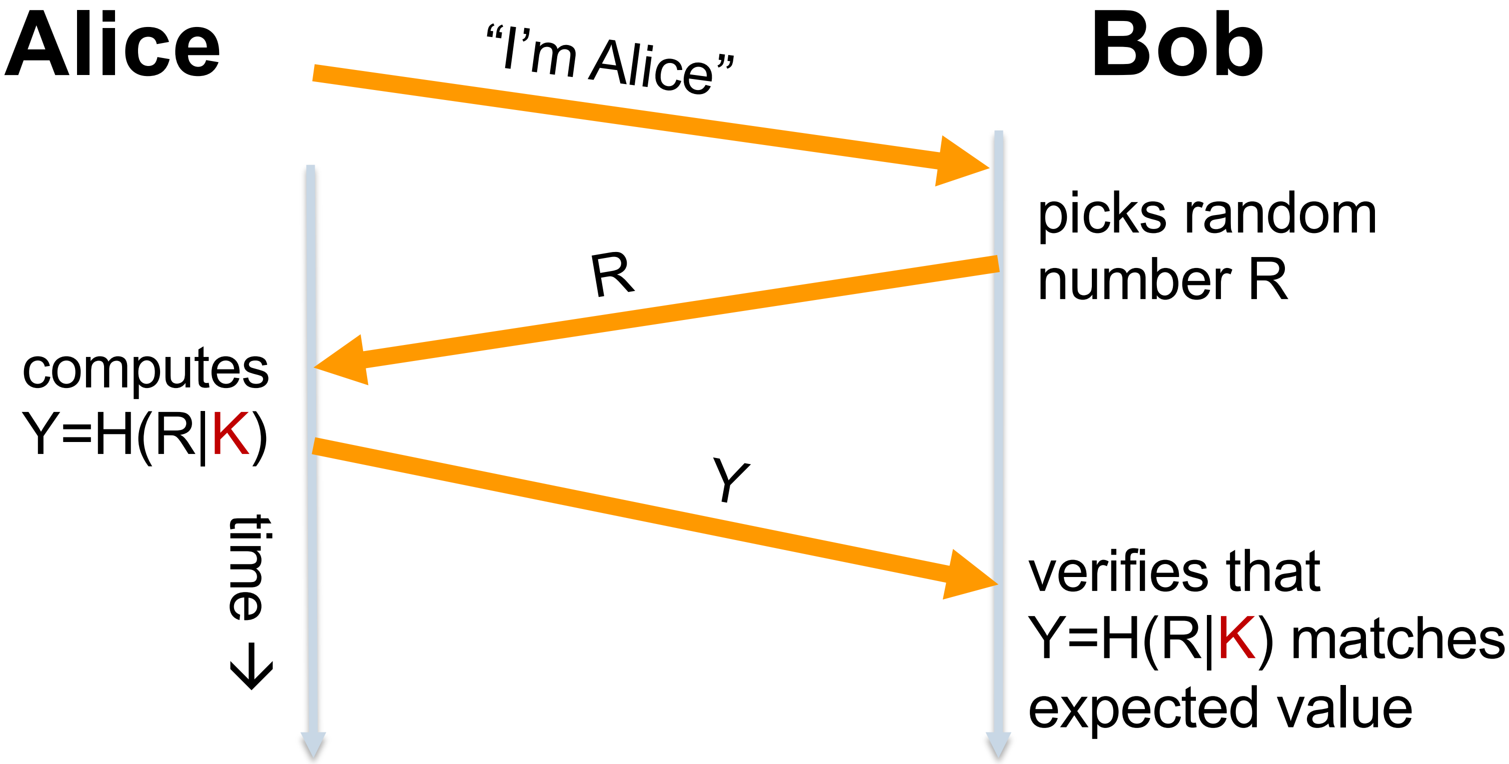
- Lots of identifiers
  - And uses
- Examples of identifiers
  - SSNs prove
  - Driver’s licenses prove
  - Credit cards prove
  - Signatures prove
  - Passwords prove
- Identify a poor mapping between identifier and use

Authentication

- There are four general means of authenticating a user's identity
  - Something the user knows
    - **Password**, personal identification number (PIN)
  - Something the user possesses
    - Smart cards, physical keys, tokens
  - Something the user is (static biometrics)
    - Recognition by fingerprint, face, retina, iris
  - Something the user does (dynamic biometrics)
    - Recognition by voice pattern, handwriting style, typing rhythm
- Can be used in combination
  - Two-factor, multi-factor authentication

Authentication

- To prove who you are



Alice

Bob

“I’m Alice”

picks random number R

R

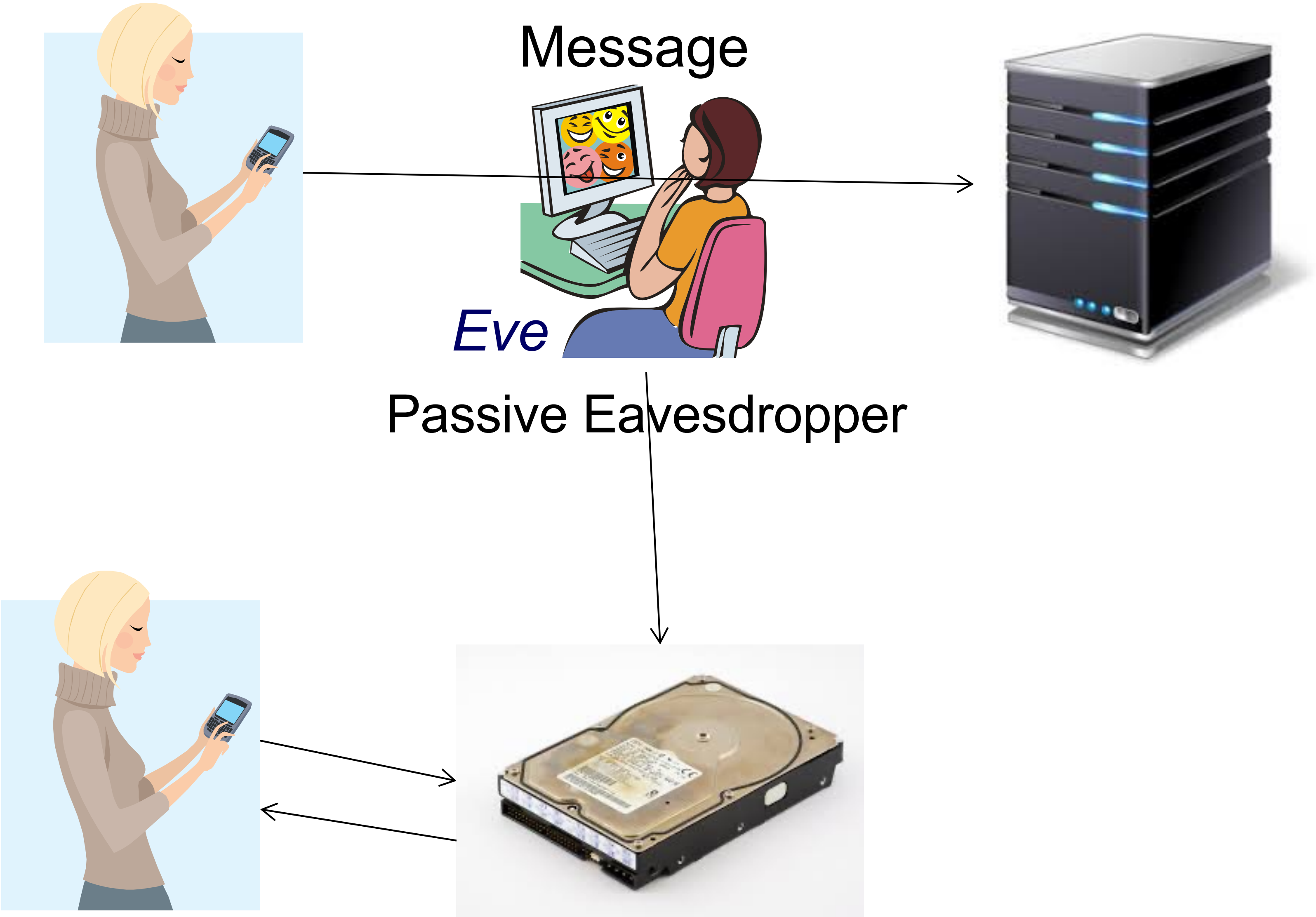
computes Y=H(R|K)

Y

verifies that Y=H(R|K) matches expected value

time ↓

Basic confidentiality requirement



Message

Eve

Passive Eavesdropper

Password Authentication

- □ Most widely used authentication method
- □ Key question: How to store passwords on a server (hard drive)?

Agenda

- How to Store Password NEXT
- UNIX Password System Design

Store in plaintext

Username: password
Alice:123
Bob: 123456
…

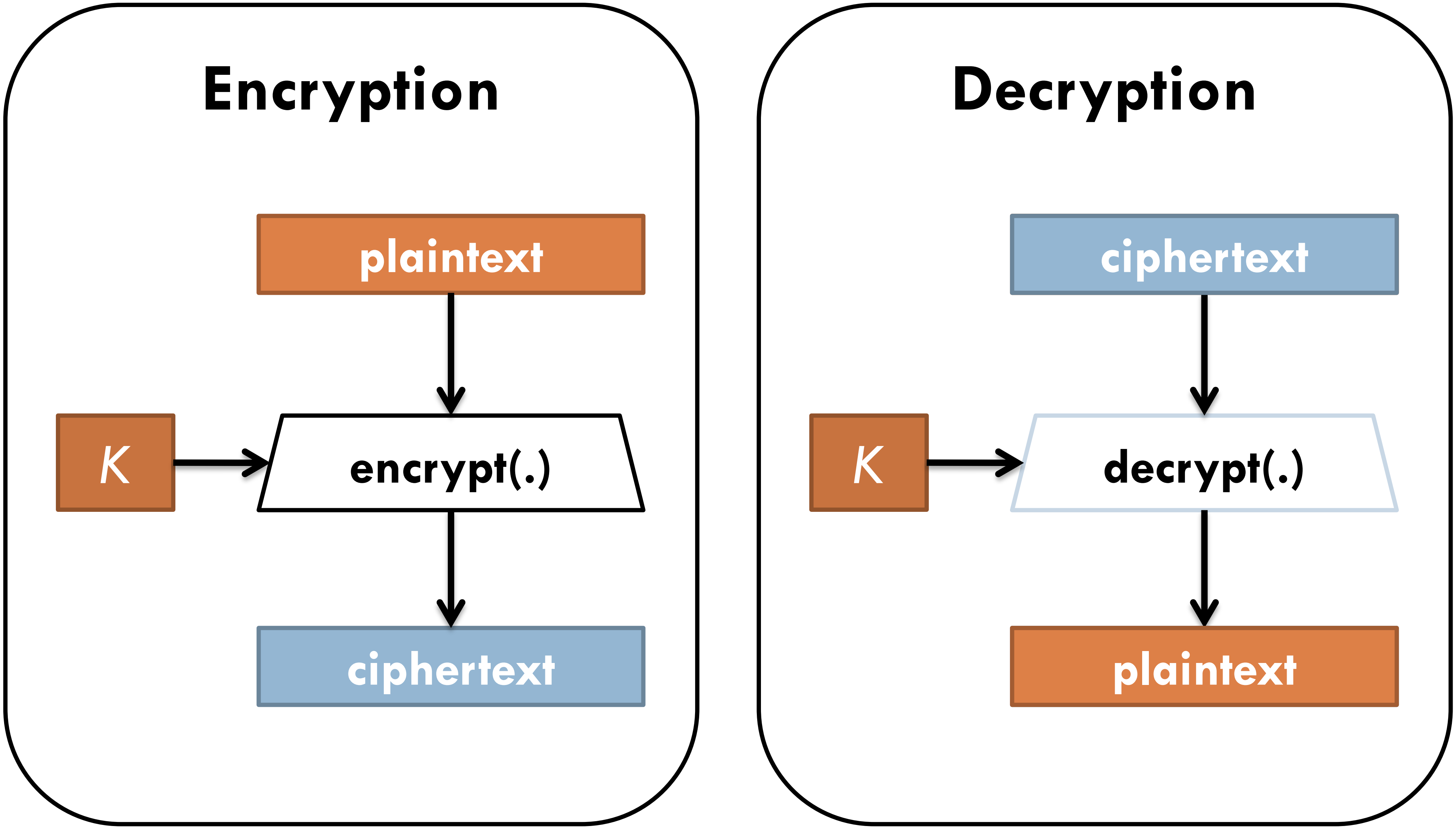
What's the problem of this approach?

RockYou hack compromises 32 million passwords

A hacker was able to break into the database of RockYou and obtain 32 million clear-text passwords through an SQL vulnerability.

http://www.scmagazine.com/rockyou-hack-compromises-32-million-passwords/article/159676/

Confidentiality - Symmetric Key Encryption



Encryption

plaintext

K

encrypt(.)

ciphertext

Decryption

ciphertext

K

decrypt(.)

plaintext

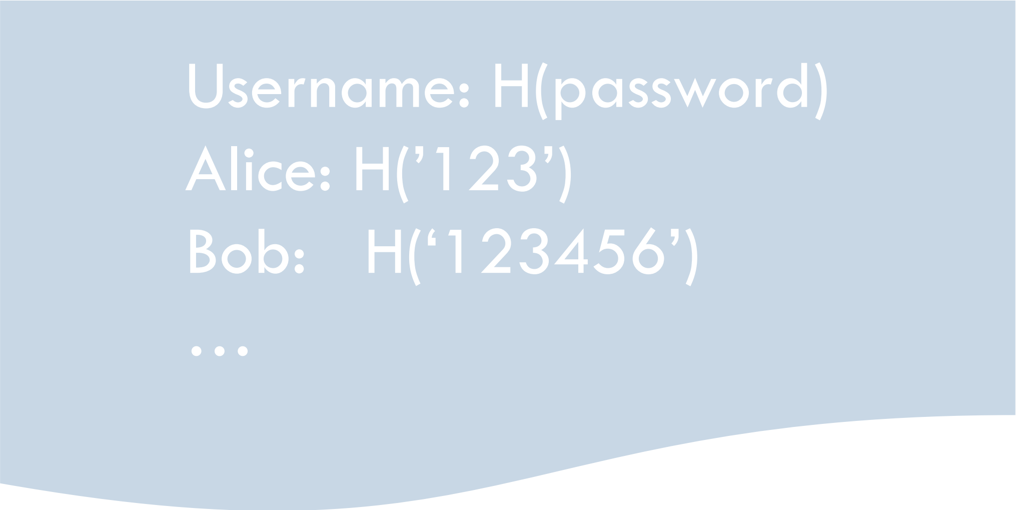
Store E(k, password)

Username: E(k, password)
Alice: E(k, ’123’)
Bob: E(k, ‘123456’)
…

What’s the problem of this approach?

(1) If k gets compromised, all leaked
(2) It reveals two users have the same password if they choose the same one

Store H(password)



Username: H(password)
Alice: H(’123’)
Bob: H(‘123456’)
…

Hash functions are one-way functions

Good idea?

- Do not reveal passwords if file stolen
- Operating systems (e.g., Linux) and server programs (e.g., Apache) store passwords using hashes

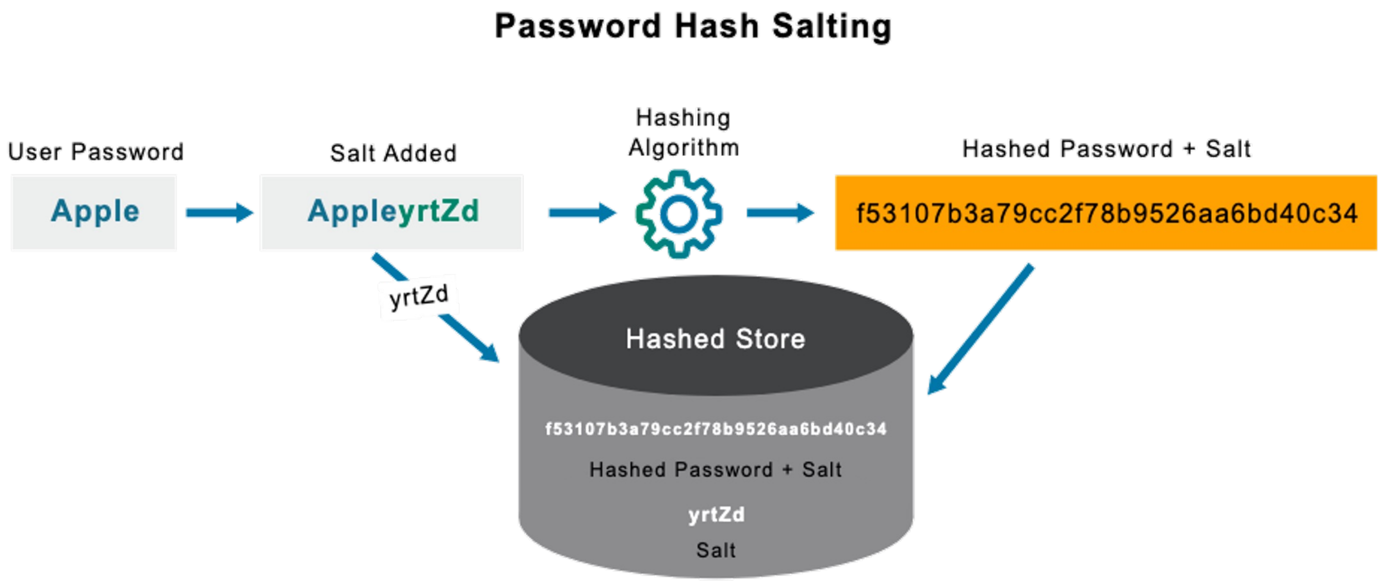
Store H(password)

Username: H(password)
Alice: H(’123’)
Bob: H(‘123456’)
…

Any problem with this approach?

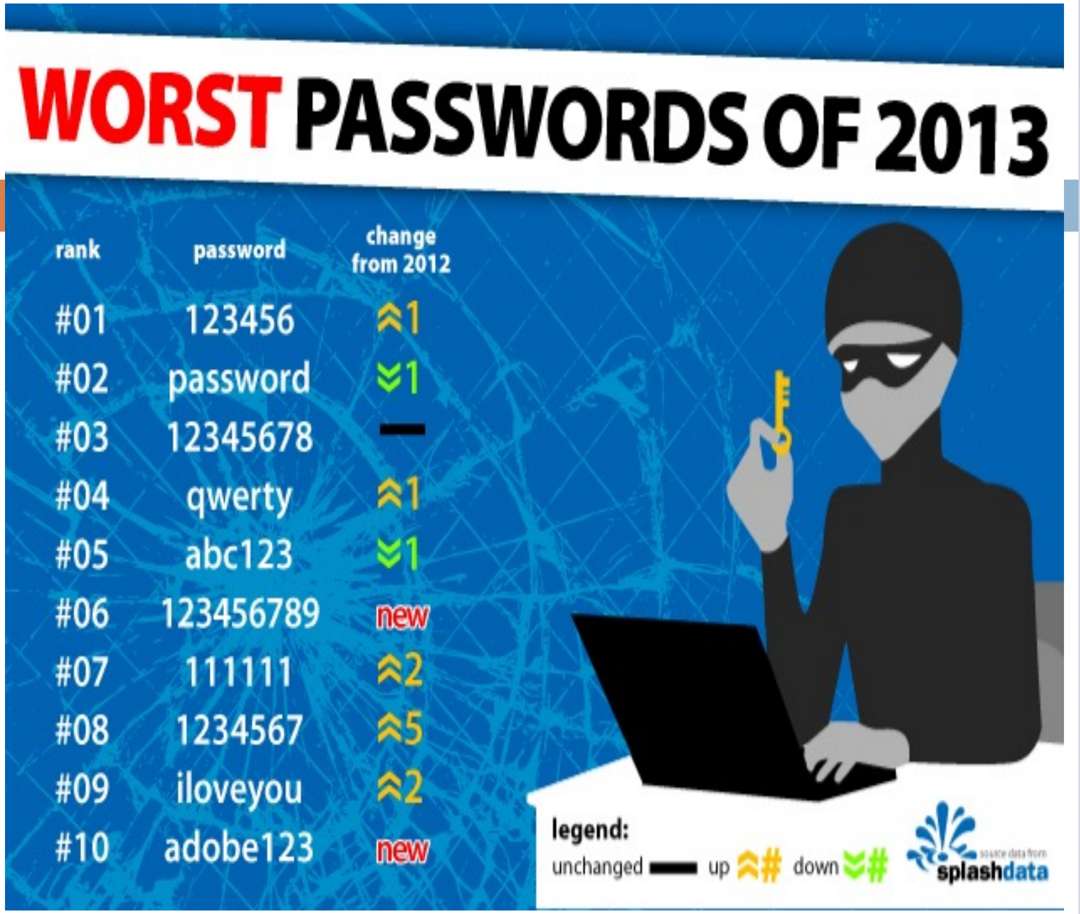
- It reveals two users have the same password if they choose the same one, which still leaks some information

Store H(password|salt)



Username: H(password|salt)
Alice: H('123456'|salt1)
Bob: H('123456'|salt2)
…

Is there **any** way to find out the password given a hash?

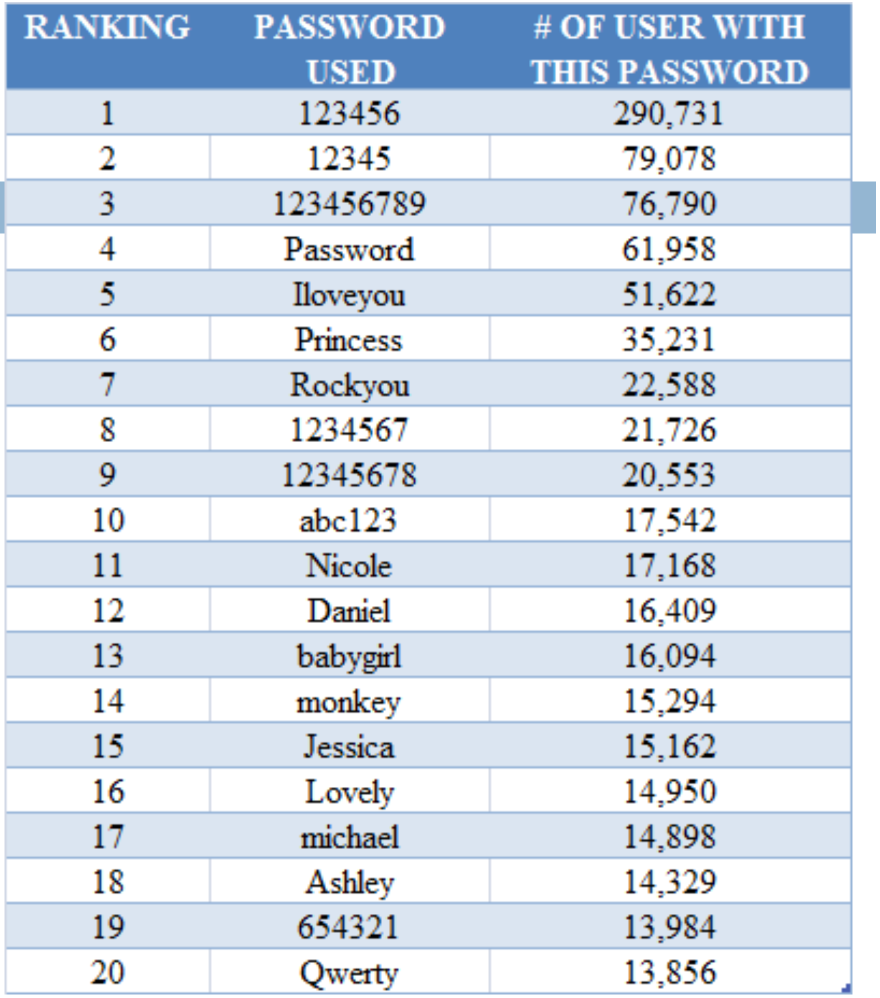


WORST PASSWORDS OF 2013

rank	password	change from 2012
#01	123456	⇑1
#02	password	⇓1
#03	12345678	—
#04	qwerty	⇑1
#05	abc123	⇓1
#06	123456789	new
#07	111111	⇑2
#08	1234567	⇑5
#09	iloveyou	⇑2
#10	adobe123	new

legend: unchanged — up ⇑# down ⇓#

source data from splashdata



RANKING	PASSWORD USED	# OF USER WITH THIS PASSWORD
1	123456	290,731
2	12345	79,078
3	123456789	76,790
4	Password	61,958
5	Iloveyou	51,622
6	Princess	35,231
7	Rockyou	22,588
8	1234567	21,726
9	12345678	20,553
10	abc123	17,542
11	Nicole	17,168
12	Daniel	16,409
13	babygirl	16,094
14	monkey	15,294
15	Jessica	15,162
16	Lovely	14,950
17	michael	14,898
18	Ashley	14,329
19	654321	13,984
20	Qwerty	13,856

http://splashdata.com/press/WorstPasswords-2013.jpg

http://www.cbsnews.com/news/the-25-most-common-passwords-of-2013/

Not much different today

Brute Force – password cracking

Password Guessing (dictionary) Attack:

```
input: passwd_hash to crack
for each i in dictionary file
    if(h(i) == passwd_hash)
        output success;
```

Time Space Tradeoff Attack (rainbow table):

```
precompute: h(i) for each i in dict file in hash_table
 input: passwd_hash
 check if passwd_hash is in hash_table
```

How do these attacks work when a salt is used?

Brute Force – password cracking

How hard is it to crack passwords?

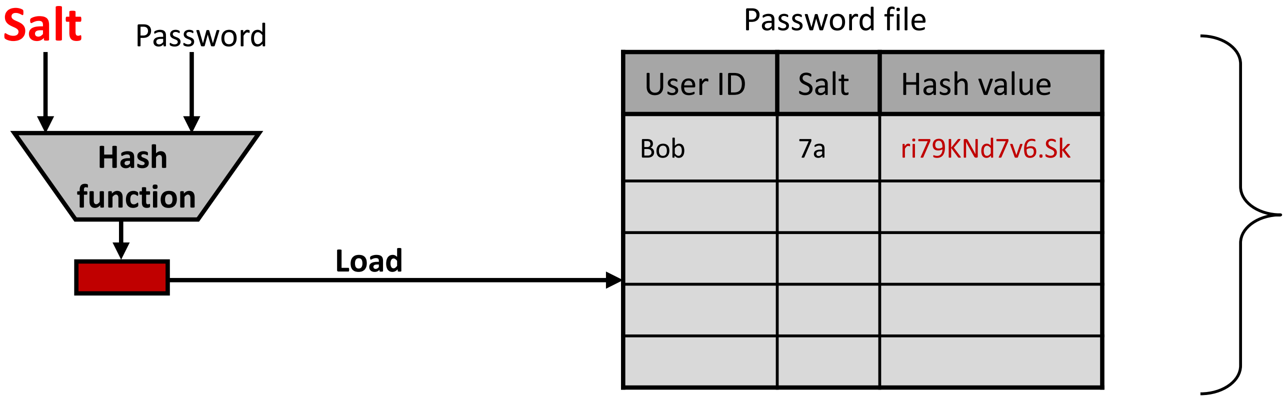
How many 8-character passwords assuming that 52 characters (upper and lower case) can be used?

52^8 = 53 trillion

Agenda

- □ How to Store Password ✓
- □ UNIX Password System Design ← NEXT

Unix Password Scheme

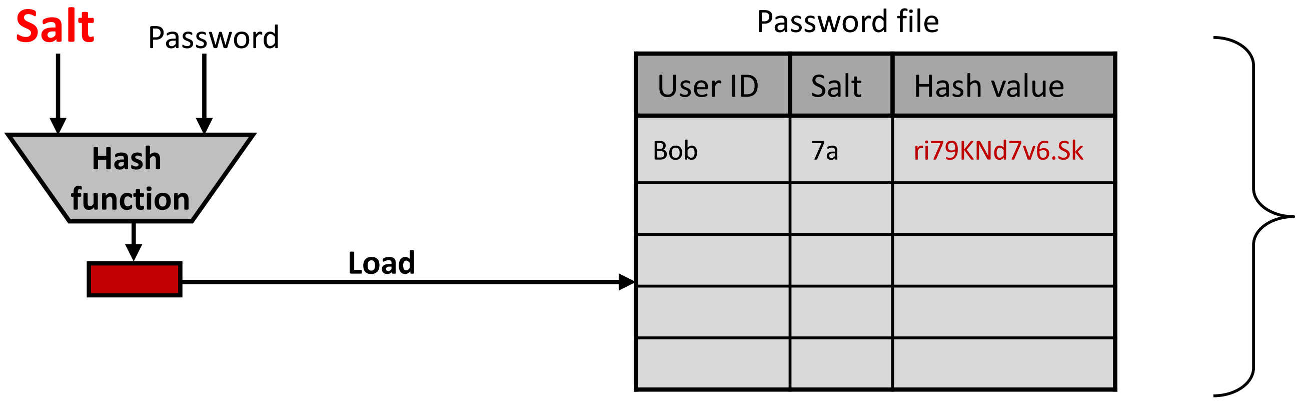


User ID	Salt	Hash value
Bob	7a	ri79KNd7v6.Sk

Loading a new password

How to check the password value?

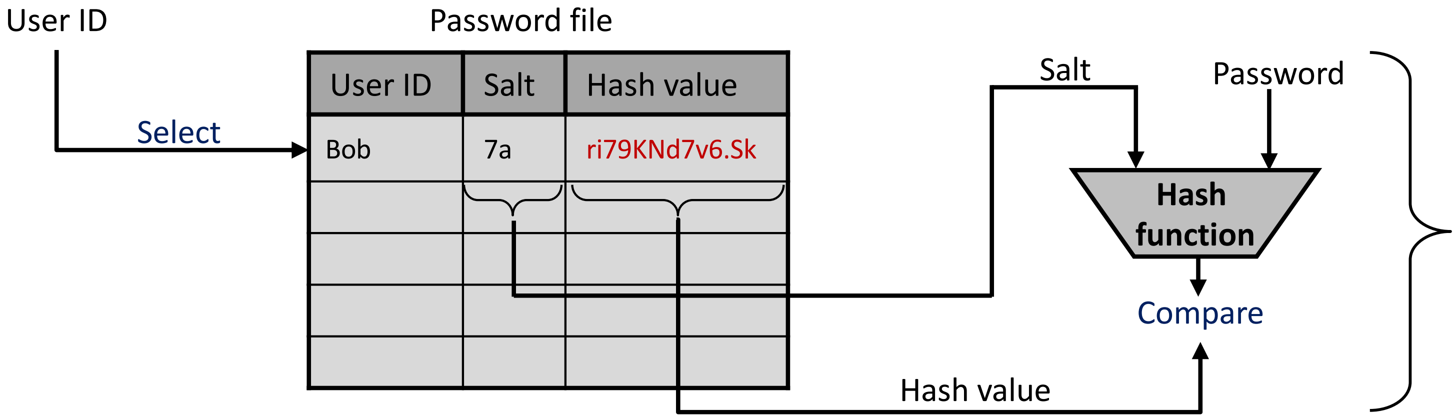
Unix Password Scheme



Password file

User ID	Salt	Hash value
Bob	7a	ri79KNd7v6.Sk

Create a new password



Password file

User ID	Salt	Hash value
Bob	7a	ri79KNd7v6.Sk

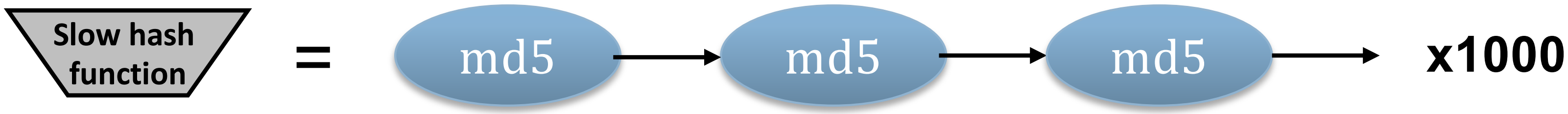
Verifying a password

Cracking Resistance

- How can we make the UNIX password scheme more difficult for cracking?

Slow hash function

- *passwd_hash* = H(passwd)
  - H() is not a single hash function – rather composition of primitive functions
- The composition is called "Slow hash"
  - To slow down password cracking!
  - E.g., 1000 times of simple md5 hashes



How difficult is it to crack passwords?

How many 8-character passwords given that 52 characters (upper and lower case) are available?

52^8 = 53 trillion

CPUs can do millions of primitive hashes per second = thousands (at least) of password hashes

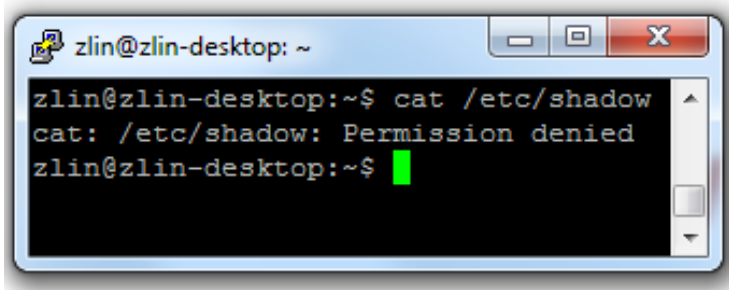
-> ~100,000 days to brute force

UNIX Password Storage

- Old method: names and hashes are stored in /etc/passwd
  - Readable by all processes
  - Programs may want to know the username: UID mapping
- This opens an attack vector
  - What is it?

UNIX Password Storage

- Old method: names and hashes are stored in /etc/passwd
  - Free for anybody to read
  - Opens up ”dictionary attack”
- Safer method: the hashes stored in separate file /etc/shadow
  - Only root can access to this file



```
zlin@zlin-desktop:~$ cat /etc/shadow
cat: /etc/shadow: Permission denied
zlin@zlin-desktop:~$
```

UNIX Password File Access

- Old method: names and hashes are stored in /etc/passwd
  - Free for anybody to read
  - Opens up dictionary attack
- Safer method: the hashes stored in separate file /etc/shadow
  - Only root can access to this file

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
zlin:x:1000:1000:zlin,,,:/home/zlin:/bin/bash
```

```
root:$6$OpBsSYf2$2N7.hAERKFhxFg
HGHLOIz4ngC0wlZATZK.yCZ7capUp
kcHjusp1nmQFATZD
anMt/kTpsHKuZYYTYskillxnE/1:1554
9:0:99999:7:::
```

Password File Access

- Old method: names and hashes are stored in /etc/passwd
  - Free for anybody to read
  - Opens up dictionary attack
- Safer method: the hashes stored in separate file /etc/shadow
  - Only root can access to this file

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
zlin:x:1000:1000:zlin,,,:/home/zlin:/bin/bash
```

```
root:$6$OpBsSYf2$2N7.hAERKFhxFg
HGHLOIz4ngC0wlZATZK.yCZ7capUp
kcHjusp1nmQFATZD
anMt/kTpsHKuZYYTYskillxnE/1:1554
9:0:99999:7:::
```

- Theft of Unix Hashes
  - Goal: gain access to /etc/shadow
  - Take away the hard drive
    - Physical access
  - Obtain root privileges (e.g., by using an exploit)

Questions