Cryptography Engineering

- Lecture 7 (Dec 04, 2024)

- Today's notes:
  - Password and password security
  - Online attacks and Offline attacks
  - Password Authentication, TLS + (hashed) password
  - Salting, TLS + salted and hashed password

- Coding tasks/Homework:
  - Offline dictionary attacks on Passwords
  - Design a password authentication protocol

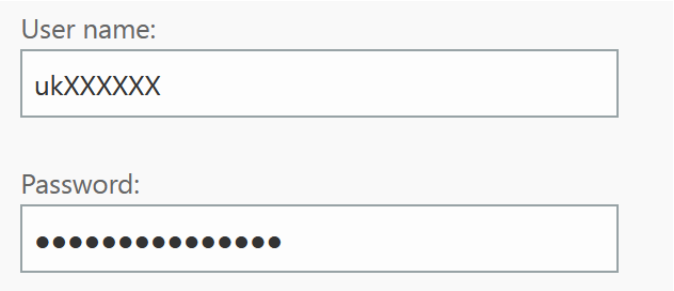
Password and its Security

- **Password:**

  “admin” “123456” “[Your_Name][Your_Birthday]”

  “root” “b8sdhazyn216fsgk.]02=2v4h”

- **Widely used in practice**



Password and its Security

- **Why password security is important:**

Within a couple of weeks, however, Adobe was forced to acknowledge that a more accurate figure for the number of people who were impacted by the hack was some 38 million active users after **a 3.8GB file containing more than 150 million usernames/passwords was dumped on the net.** 5 Nov 2013

Facebook Stored Hundreds of Millions of User Passwords ...

21 Mar 2019 — **Hundreds of millions of Facebook users** had their account passwords stored in plain text and searchable by thousands of Facebook employees ...

LinkedIn
https://www.linkedin.com › news › story › nearly-10-bill...

Nearly 10 billion passwords leaked

In a **leak** that cybersecurity researchers are calling the largest of all time, almost 10 billion unique passwords have been posted to a hacking forum.

The Biggest Password Leak in History

In an unprecedented cyber security event, the largest **password leak** ever recorded has just occurred, exposing over 10 billion passwords.

At the end of 2010, an incident that is known as CSDN Password Leakage Incident happened, and passwords from five websites, including CSDN, Tianya, Duduniu, 7k7k and 178.com, were leaked in several consecutive days. The total number of leaked accounts is over 80 million, and all the leaked passwords are in plaintext. 20 Aug 2014

(source: Google search)

Password and its Security

- We would focus on **How to**
  - Use passwords to authenticate…
  - Securely transmit passwords…
  - Securely store passwords…
  - …

Password and its Security

- **Security Properties:**
  - Mainly used for authentication (e.g., hash and compare), easy to replace,…
  - **Short length,** Human-generated, human-memorizable, **Low Entropy**
  - Highly **Reused**
  - …

Password and its Security

- **Low Entropy**
  - Lack of randomness, predictability, Short length, Limited character set,…
  - Example: (Most people use their personal email as website accounts, e.g., Amazon, …)

    Account: “[YourName]@gmail.com”

    “admin”, “123456”, “hello123”, …

    “[Your/Your Partner’s Name]_iloveu”, “[Your/Your Partner’s Name]_[Birthday]”, …

    “[Your Phone number]”, “[Family’s phone number]”…

    “qwerty” (English keyboard), “qwertz” (German keyboard), …

Password and its Security

- **Low Entropy**
  - Lack of randomness, predictability, Short length, Limited character set,…
  - Example: (Most people use their personal email as website accounts, e.g., Amazon, …)

    Account: “[YourName]@gmail.com”

    “admin”, “123456”, “hello123”, …

    “[Your/Your Partner’s Name]_iloveu”, “[Your/Your Partner’s Name]_[Birthday]”, …

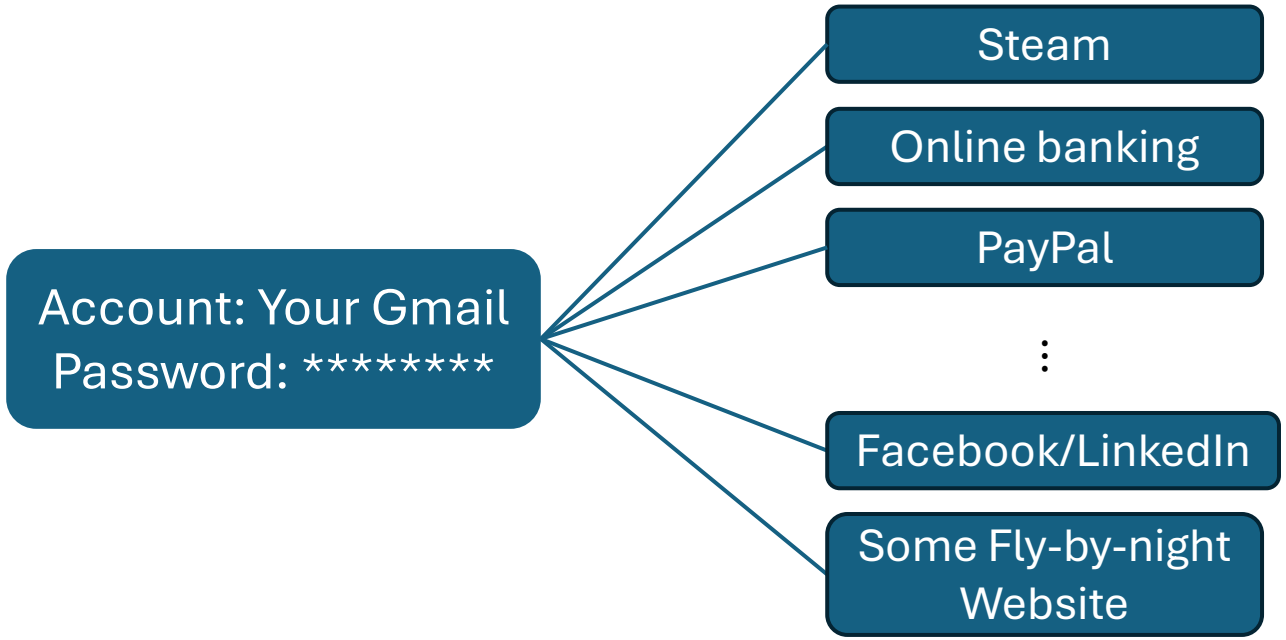
    “[Your Phone number]”, “[Family’s phone number]”…

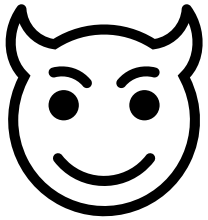
    “qwerty” (English keyboard), “qwertz” (German keyboard), …

  - Short, patterned, no randomness, and highly related to personal information

Password and its Security

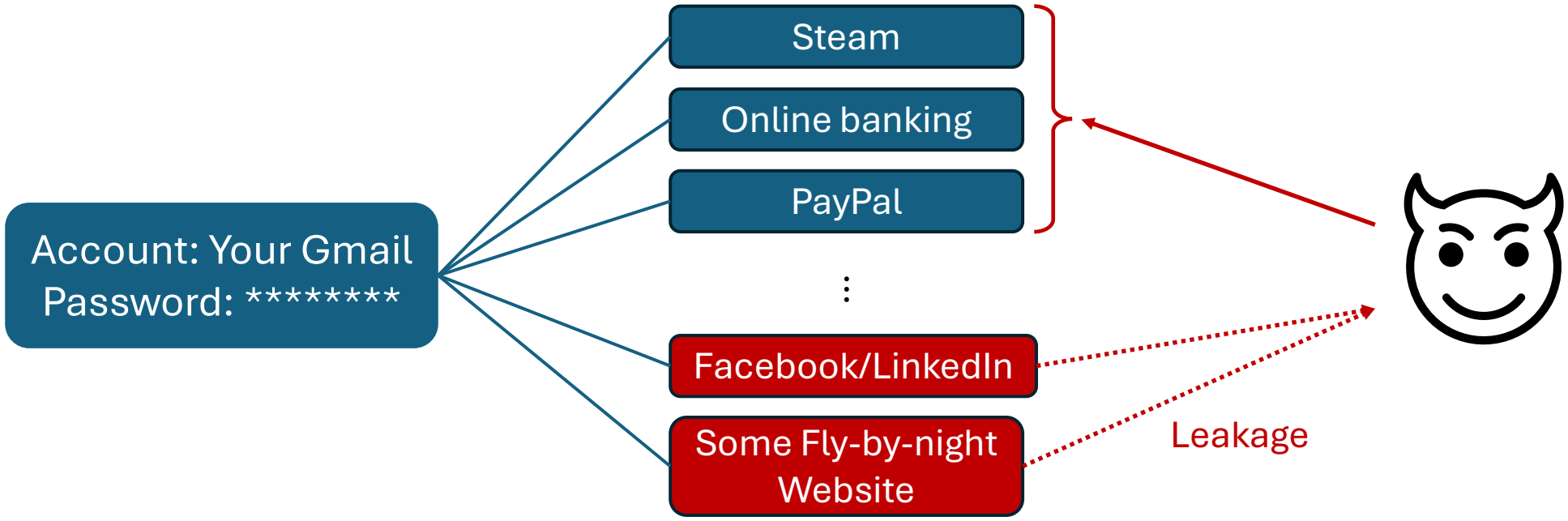
- **Highly reused:** Different portal, but the same password…





Password and its Security

- **Highly reused:** Different portal, but the same password…



Password Dictionary

- **Dictionary Attack:**
  - Attack (Guess) using password dictionaries
  - Focus on known/common password combinations, more efficient than brute force…

123456
password
12345678
qwerty
123456789
12345
1234
111111
1234567
dragon
123123
baseball
abc123
football
monkey
letmein
696969
shadow
master
666666
qwertyuiop
123321
mustang
1234567890
michael
654321
pussy
superman
1qaz2wsx

nascar
monster
tigers
yellow
xxxxxx
123123123
gateway
marina
diablo
bulldog
qwer1234
compaq
purple
hardcore
banana
junior
hannah
123654
porsche
lakers
iceman
money
cowboys
987654
london
tennis
999999
ncc1701
coffee
scooby
0000
miller
boston
q1w2e3r4
fuckoff
brandon
yamaha
chester

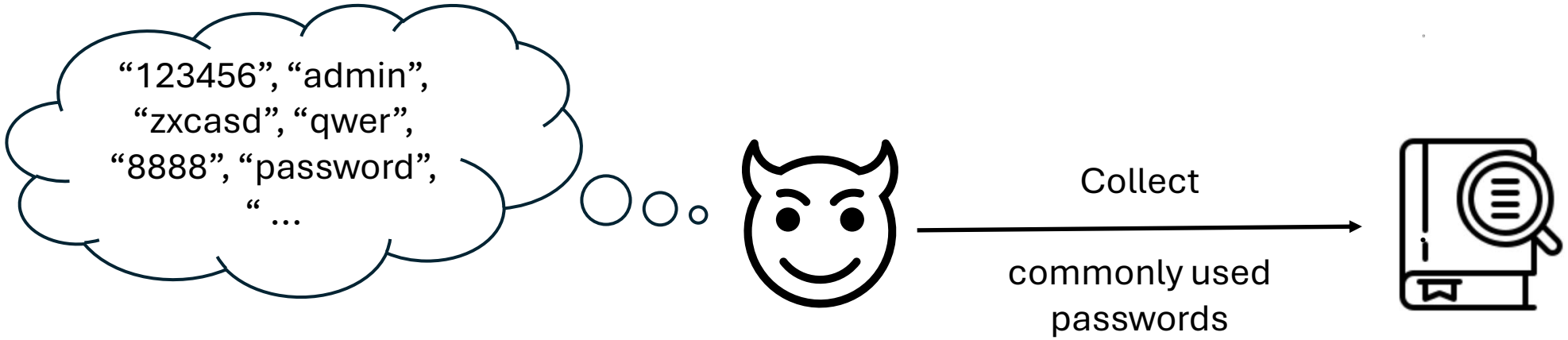
abcd1234
scorpion
qazwsxed
101010
butter
carlos
password
dennis
slipknot
qwerty12
booger
asdf
1991
black
startrek
12341234
cameron
newyork
rainbow
nathan
john
1992
rocket
viking
redskins
butthead
asdfghjk
1212
sierra
peaches
gemini

braves
shelby
godzilla
beaver
fred
tomcat
august
buddy
airborne
1993
1988
lifehack
qqqqqq
brooklyn
animal
platinum
phantom
online
xavier
darkness
blink182
power
fish
green
789456123
voyager
police
travis
12qwaszx
heaven
snowball
lover

bond007
alexis
1111111
samson
5150
willie
scorpio
bonnie
gators
benjamin
voodoo
driver
dexter
2112
jason
calvin
freddy
212121
creative
12345a
sydney
rush2112
1989
asdfghjk
red123
bubba
4815162342
passw0rd
trouble
gunner
happy

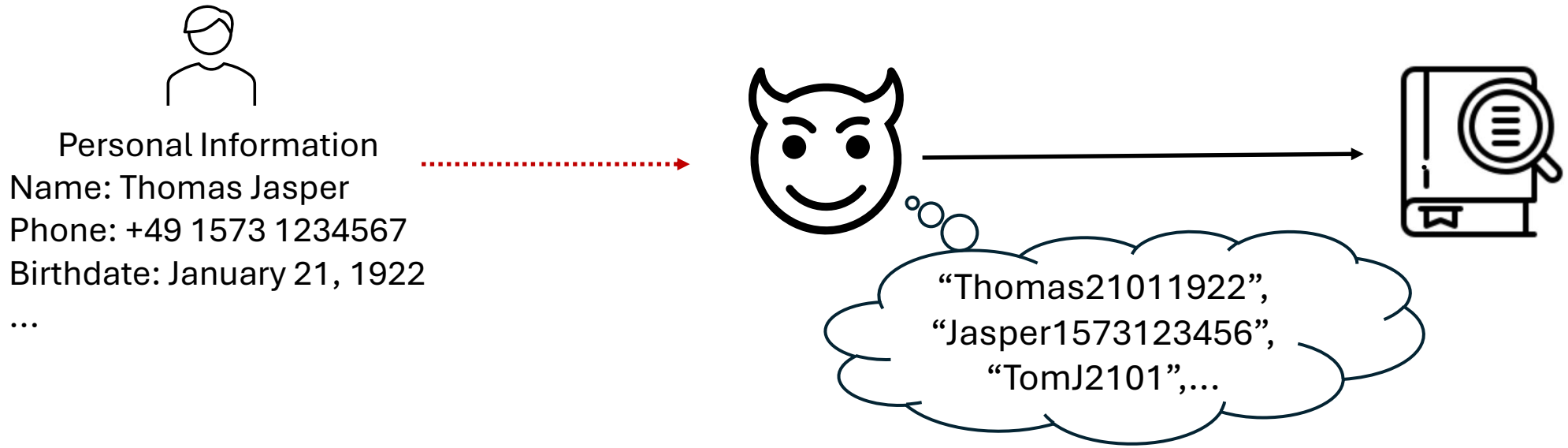
Password Dictionary

- Construct a password dictionary:



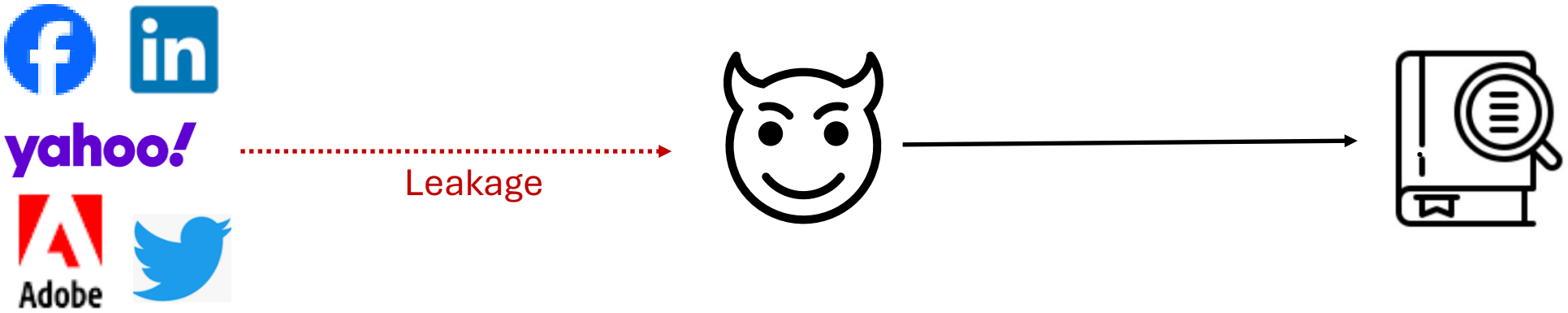
Password Dictionary

- Construct a password dictionary:



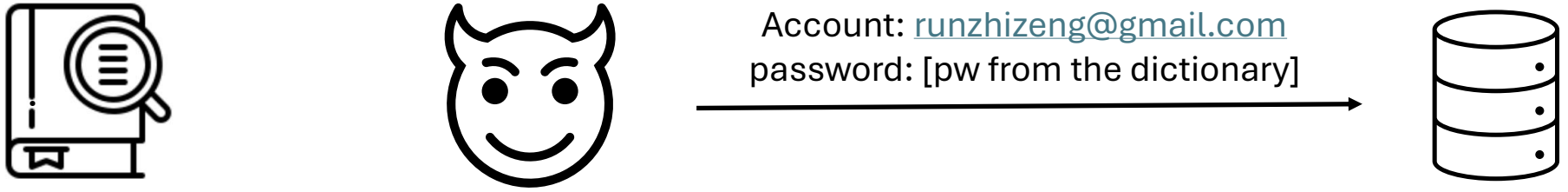
Password Dictionary

- Construct a password dictionary:



(Passwords stored in plaintext)

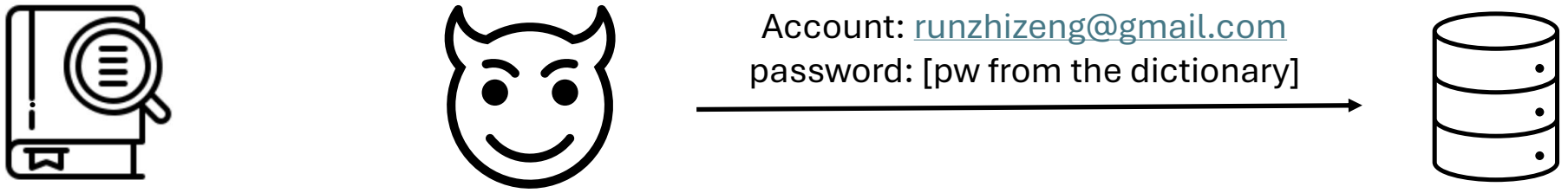
Online Dictionary Attack



Password guessing using the dictionary

- **Online dictionary attack**
  - Attempt passwords from the dictionary until success
  - Require **Online** connections: Verify guess via interacting with the legitimate system

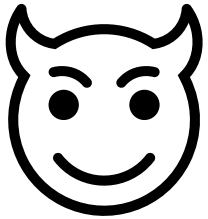
Online Dictionary Attack



Password guessing using the dictionary

- **Online dictionary attack**
  - Attempt passwords from the dictionary until success
  - Require **Online** connections: Verify guess via interacting with the legitimate system
  - **Unavoidable** (in most of cases), but **Detectable** and **Accountable**
  - Non-cryptographic solution: Limit failed trials

Offline Dictionary Attack

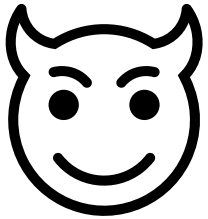


func_pw
= **F**(“RunzhiZeng123456”)

F is some publicly known function with collision resistance

- **Offline dictionary attack**

Offline Dictionary Attack



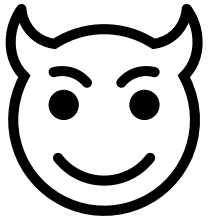
Try all passwords from the dictionary until find a pw such that **F**(pw) = func_pw

func_pw
= **F**(“RunzhiZeng123456”)

F is some publicly known function with collision resistance

- **Offline dictionary attack**
  - Attempt passwords from the dictionary until success

Offline Dictionary Attack



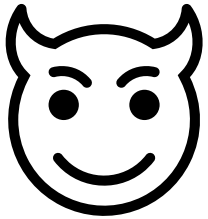
Try all passwords from the dictionary until find a pw such that **F**(pw) = func_pw

func_pw
= **F**(“RunzhiZeng123456”)

F is some publicly known function with collision resistance

- **Offline dictionary attack**
  - Attempt passwords from the dictionary until success
  - **Offline-Performable**: Verify guess without interacting with the legitimate system
  - **Hard to detect and account**

Offline Dictionary Attack



Try all passwords from the dictionary until find a pw such that **F**(pw) = func_pw

func_pw
= **F**(“RunzhiZeng123456”)

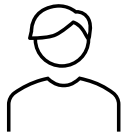
F is some publicly known function with collision resistance

- **Offline dictionary attack**
  - Attempt passwords from the dictionary until success
  - **Offline-Performable**: Verify guess without interacting with the legitimate system
  - **Hard to detect and account**
  - **Primary Goal** of designing secure password-based cryptosystems: resist offline attacks

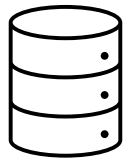
Offline Dictionary Attack

- **Example:** Does this login system resist offline attacks?

Account = "admin"
password = pw
where pw is some string



H is some secure hash function



User	password
admin	pw
Runzhi	pw_1
Tom	pw_2
…	…

1. hash_pw = **H**(pw)

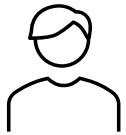
LoginRequest = ("admin", hash_pw)

2. local_hash_pw = **H**(pw),
   // where pw is the password of "admin" from the local database
3. If local_hash_pw == hash_pw:
4. Accept
5. Else: Reject

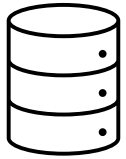
Offline Dictionary Attack

- **Example:** Does this login system resist offline attacks?

Account = "admin"
password = pw
where pw is some string



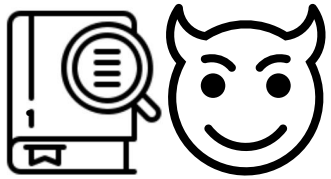
H is some secure hash function



User	password
admin	pw
Runzhi	pw_1
Tom	pw_2
…	…

1. hash_pw = **H**(pw)

LoginRequest = ("admin", hash_pw)



Eavesdropping

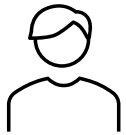
2. local_hash_pw = **H**(pw),
// where pw is the password of "admin" from the local database
3. If local_hash_pw == hash_pw:
4. Accept
5. Else: Reject

Try all pw from the dictionary until find a match: **H**(pw) == hash_pw

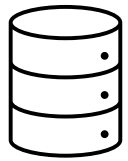
Offline Dictionary Attack

- **Example:** Does this login system resist offline attacks?

Account = "admin"
password = pw
where pw is some string



H is some secure hash function



User	password
admin	$\boldsymbol{H(pw)}$
Runzhi	$H(pw_1)$
Tom	$H(pw_2)$
…	…

1. hash_pw = **H**($\boldsymbol{pw}$)

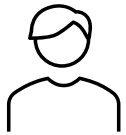
LoginRequest = ("admin", hash_pw)

// $\boldsymbol{H}(pw)$ is the hashed password of "admin" from the local database
2. If $\boldsymbol{H}(pw)$ == hash_pw:
3. Accept
4. Else: Reject

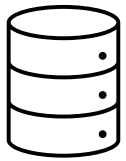
Offline Dictionary Attack

- **Example:** Does this login system resist offline attacks?

Account = "admin"
password = pw
where pw is some string



H is some secure hash function



User	password
admin	$\boldsymbol{H(pw)}$
Runzhi	$H(pw_1)$
Tom	$H(pw_2)$
…	…

1. hash_pw = **H**($\boldsymbol{pw}$)

LoginRequest = ("admin", hash_pw)

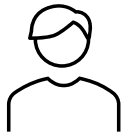
// $\boldsymbol{H}(pw)$ is the hashed password of "admin" from the local database
2. If $\boldsymbol{H}(pw)$ == hash_pw:
3. Accept
4. Else: Reject

A quick question: Can I use hash function without collision-resistance to instantiate this system ?

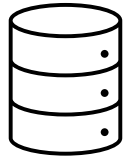
Offline Dictionary Attack

- **Example:** Does this login system resist offline attacks?

Account = "admin"
password = pw
where pw is some string



K is some publicly known symmetric key



User	password
admin	pw
Runzhi	pw_1
Tom	pw_2
…	…

1. enc_pw = AEAD(**K**, pw)

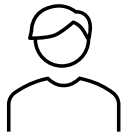
LoginRequest = ("admin", enc_pw)

2. local_enc_pw = AEAD(**K**, pw),
   // where pw is the password of "admin" from the local database
3. If local_enc_pw == enc_pw:
4. Accept
5. Else: Reject

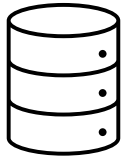
Offline Dictionary Attack

- **Example:** Does this login system resist offline attacks?

Account = "admin"
password = pw
where pw is some string



Run TLS handshake to share a handshake key K



User	password
admin	pw
Runzhi	pw_1
Tom	pw_2
…	…

1. enc_pw = AEAD(**K**, pw)

LoginRequest = ("admin", enc_pw)

2. local_enc_pw = AEAD(**K**, pw),
   // where pw is the password of "admin" from the local database
3. If local_enc_pw == enc_pw:
4. Accept
5. Else: Reject

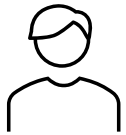
A Summary about Online/Offline Dictionary Attack

	Online Dictionary Attack	Offline Dictionary Attack
	Based on pre-constructed dictionaries	
Type of Interaction	Have to be online, one guess = one interaction with the server	Offline, can be performed locally
Accountability	Easy	Hard
Detectability	Easy	Hard
Security consideration	Unavoidable	**Primary Goal: resist offline attacks**
Solution	Restrict the number of failed attempts, …	**Need cryptographic techniques!**

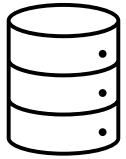
Authentication using Passwords

- Most common practice: TLS + password (e.g., widely used in HTTPs login)

Account = "Runzhi"
password = pw
where pw is some string



Run TLS handshake to share a handshake key K



User	password
Runzhi	pw
Tom	pw_2
…	…

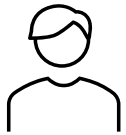
Login Request = ("Runzhi", pw)

(Encrypted by the TLS handshake key K)

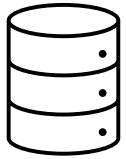
Authentication using Passwords

- Most common practice: TLS + password (e.g., widely used in HTTPs login)

Account = "Runzhi"
password = pw
where pw is some string



Run TLS handshake to share a handshake key K



User	password
Runzhi	pw
Tom	pw_2
…	…

Login Request = ("Runzhi", pw)

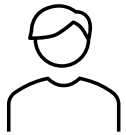
(Encrypted by the TLS handshake key K)

- Advantage: Easy to implement, rely on TLS, …
- Disadvantage: **Passwords are stored in plaintext**

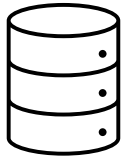
Authentication using Passwords

- Most common practice: TLS + password (e.g., widely used in HTTPs login)

Account = "Runzhi"
password = pw
where pw is some string



Run TLS handshake to share a handshake key K



User	password
Runzhi	$H(pw)$
Tom	$H(pw_2)$
…	…

Login Request = ("Runzhi", **H(pw)**)

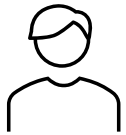
(Encrypted by the TLS handshake key **K**)

- Now the server stores the hashes of passwords…
- **What happens if the database is compromised?**

Authentication using Passwords

- Most common practice: TLS + password (e.g., widely used in HTTPs login)

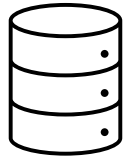
Account = "Runzhi"
password = pw
where pw is some string

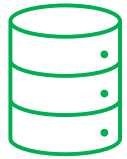


Run TLS handshake to share a handshake key K

Login Request = ("Runzhi", **H(pw)**)

(Encrypted by the TLS handshake key **K**)



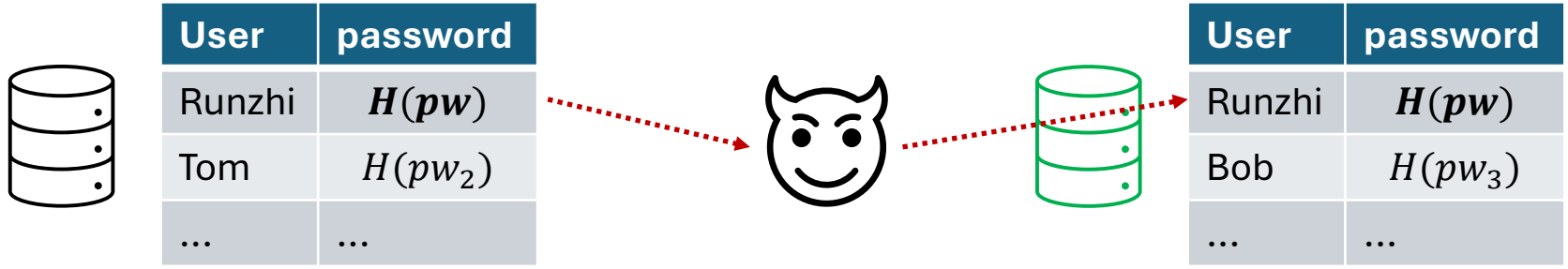


User	password
Runzhi	$H(pw)$
Tom	$H(pw_2)$
…	…

User	password
Runzhi	$H(pw)$
Bob	$H(pw_3)$
…	…

- Now the server stores the hashes of passwords…
- Note: Generally, passwords are **reused across different servers**…

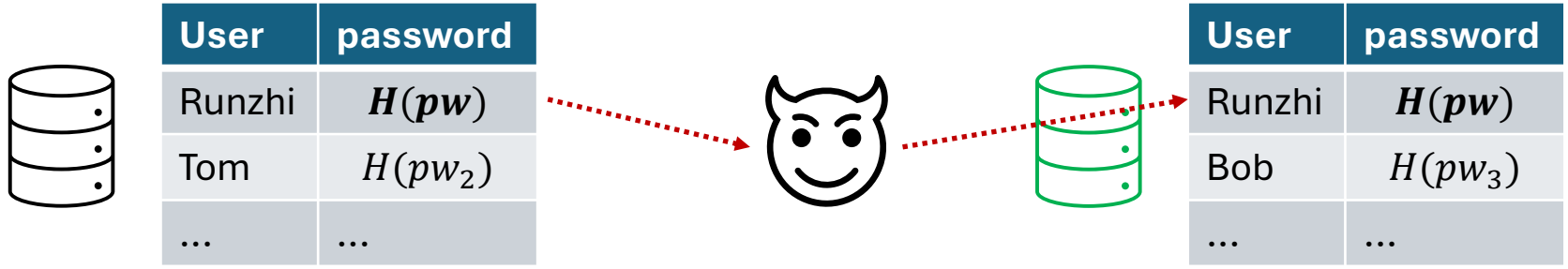
Password Storage and Salting



Store hashes of passwords v.s Store passwords in plaintext

- The former one is almost as insecure as the latter one if different servers store hashes of passwords
- **Why:** Just storing hashes can lead to cross-system compromise, making it nearly as insecure as storing plaintext passwords.

Password Storage and Salting



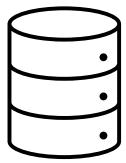
Store hashes of passwords v.s Store passwords in plaintext

- The former one is almost as insecure as the latter one if different servers store hashes of passwords
- **Why:** Just storing hashes can lead to cross-system compromise, making it nearly as insecure as storing plaintext passwords.

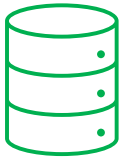
- **Solution: Salting (i.e., store salted hashes of passwords)**

Password Storage and Salting

User	password
Runzhi	$H(pw)$
Tom	$H(pw_2)$
...	...

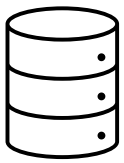


User	password
Runzhi	$H(pw)$
Bob	$H(pw_3)$
...	...

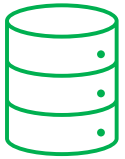


Password Storage and Salting

User	password
Runzhi	$r, H(r, pw)$
Tom	$r_2, H(r_2, pw_2)$
…	…

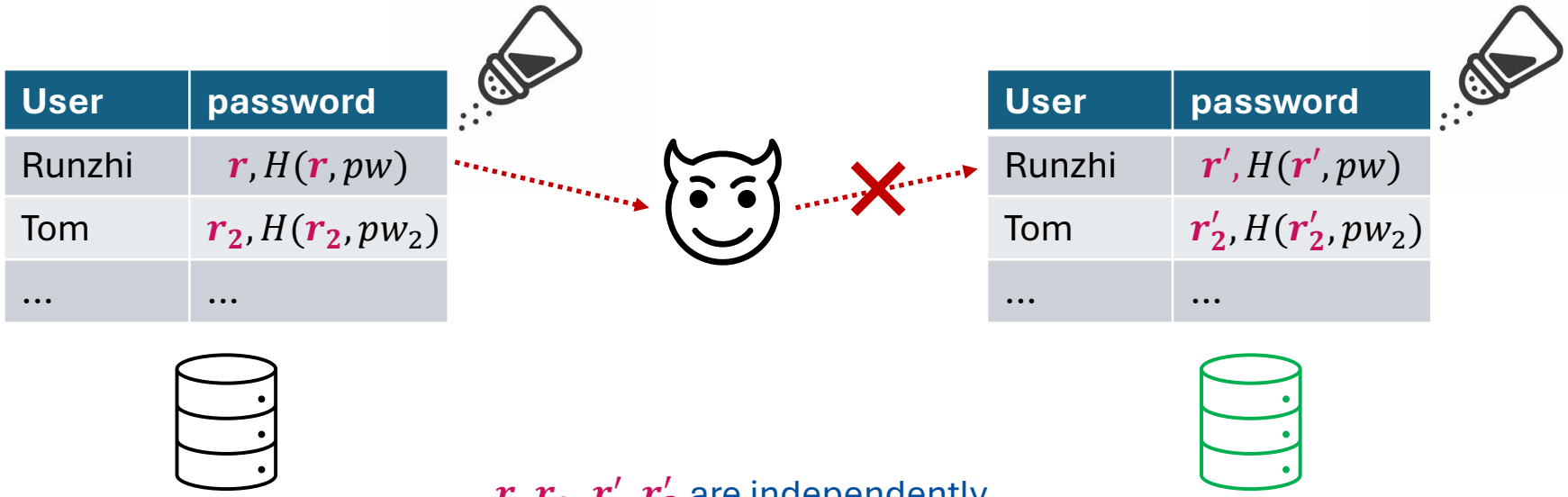


User	password
Runzhi	$r', H(r', pw)$
Tom	$r'_2, H(r'_2, pw_2)$
…	…



r, r_2, r', r'_2 are independently random strings **(salt)**

Password Storage and Salting



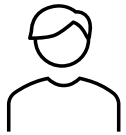
r, r_2, r', r'_2 are independently random strings **(salt)**

- **Resistance to cross-system compromise**

Authentication using Salted Hashes of Passwords

- TLS + salted hashes password

Account = "Runzhi"
password = pw
where pw is some string



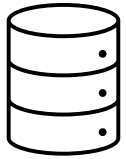
Run TLS handshake to share a handshake key K

LoginRequest = "Runzhi"

r

H(r, pw)

(Encrypted by the TLS handshake key K)

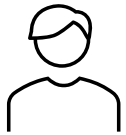


User	password
Runzhi	$r, H(r, pw)$
Tom	$r_2, H(r_2, pw_2)$
…	…

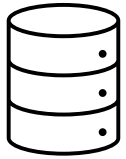
Authentication using Salted Hashes of Passwords

- TLS + salted hashes password

Account = "Runzhi"
password = pw
where pw is some string



Run TLS handshake to share a handshake key K



User	password
Runzhi	$r, H(r, pw)$
Tom	$r_2, H(r_2, pw_2)$
…	…

LoginRequest = "Runzhi"

The server should send the salt of the user in every time it logs in

r

H(r, pw)

(Encrypted by the TLS handshake key K)

Coding tasks

- Perform offline dictionary attacks. Suppose I leaked a SHA3-256 hash of my password (i.e., hash_pw = SHA3-256([my password])) and the password is in a dictionary (in the example code). The hexadecimal value (lower case) of the hash_pw is

  e8acff88511d7f8e48f038001c24d7b1ab76d9233d7894fa936c4c7c93d2c917

- Try to recover my password.

Homework

- Design a password-based login protocol and try analyzing it.
  - ➢ You should specify (1) How the server stores passwords (2) The message flow of the protocol (3) How the server verifies.
  - ➢ Analyze the security of your protocol (e.g., can it resist offline dictionary attacks?)
  - ➢ Hint: You may add some nonces in your protocol

- Implement your login system using sockets.
  - ➢ The “password database” of the server couble be a text file where each row is

([User_name], [Password/Hash_of_password/salted_hash_of_password])