EECS565 Intro to Computer and Information Security

Authentication

Professor Fengjun Li

Fall 2024

Outline

- Password-based Authentication
- Kerberos and Single Sign On (SSO)
- Certificate-based Authentication
  - Certificate, Certificate Authority (CA)
  - Public Key Infrastructure (PKI)
  - Security issues with PKI

Password-based Authentication

Authentication

- Authentication is the process of verifying the identity of a party.
- How do you prove to someone who you are?
  - Show “credentials”
- Credentials can be
  - Something I know password, PIN, passphrase, some secret (security questions), …
  - Something I have IP address, certificates, security token, hardware/mobile devices, …
  - Something I am biometrics

Password-based Authentication

- How do you login to a computer?
  - Username: f123l456 **identification**
  - Password: qweasd **authentication**
- This is called user authentication.
- Password is a secret string that a user types in to prove her identity
  - Created when the account was created for a service for the first time (can be changed)
  - Typed in each time logging in to the service
- Password-based authentication is widely used in OS, web, email, etc.

Password Security

- “8 out of 9 exploits involve password stealing and/or cracking” (Kevin Mitnick)
- 76% of network intrusions exploited weak or stolen credentials (“76% of network intrusions exploited weak or stolen credentials” (2012 Verizon Data Breach Investigations Report)
- “Weak passwords caused 30% of ransomware infections in 2019, and it’s still a major problem for organizations in 2022.” “Stolen credentials led to nearly 50% of attacks.” (2022 Verizon Data Breach Investigations Report)
- How to steal or exploit passwords?
  - After a successful intrusion
  - Steal: Install sniffer or keylogger to steal passwords
  - Exploit: Fetch password files and run cracking tools

Storing Passwords

2013 Adobe data breach

- 153 million accounts with password hints
- Passwords are encrypted using 3DES in ECB

Verifying password + NOT storing info to allow recovery of the password

- How to store password in the system?
  - In password files indexed by user ID
  - In plaintext?
  - Encrypted? No forward security
  - Hashed?
- Security concern is that the attacker may hack into the service and obtain the password file.
- Takeaway: store H(password)

Password Hashing

- For each user, the system stores a hash of her password $\mathrm{H}(\mathrm{password})$
  - Instead of the actual password
- Verification process
  - User enters password
  - System computes $\mathrm{H}(\mathrm{password})$ and checks if it matches any entry in the password file
- Properties of password hash function
  - Deterministic: always generate the same hash for the same password
  - Onewayness: given $\mathrm{H}(\mathrm{password})$, it's hard to deduce the password.
  - "slow to compute": the feature to restrict the speed of brute force attacks
    - If it takes 0.1s to compute hash → attacker can test only 600 passwords in one minute

UNIX Password System

- Password Hash
  - /etc/shadow: only readable by system admin (root)
  - idsalt$hashed

password hash

```
seed:$6$n8DimvsbIgU0OxbD$YZ0h1EAS4bGKeUIMQvRhhYFvkrmMQZdr/hB.0fe3KFZQTgFTcRgoIoK
Zd0OrhDRxxaITL4b/scpdbTfk/nwFd0:18590:0:99999:7:::
```

  - $6: SHA-512 (algorithm used to create hash)
  - $n8DimvsbIgU0OxbD: (salt)
- Why we need salt?

FYI: PBKDF2

- **PBKDF2** (Password-based key derivation function 2)
  - It’s a slow hash, basically compute HMAC 10,000 times
  - Use an underneath hash function, e.g., HMAC-SHA256
  - Input: a password + salt + iteration number
  - Output: a desired length of the output, for example n
- It can derive an arbitrarily long string from the user’s password
- PBKDF2 is slow, but it doesn’t use a lot of memory
  - Output can be used as a symmetric key, a seed for PRNG or for generating public/private key

Password Attacks

- **Brute-force attacks** offline attack
  - If the attacker gets the password file (how?)
  - He can try to hash all possible passwords and compare with the password file
  - Password strength depends on password length and space

Consider an 8-character password:

- Length = 8
- Space = 52 upper- and lower-case letters, 10 digits, and 32 punctuation symbols = 94 candidate characters
- # of possible password = $94^8 \approx$ 6 quadrillion

Password Attacks

- However, passwords are NOT truly random!
  - Humans like to use dictionary words → about 1 million common passwords.

Dictionary attacks are possible!

Most used passwords in 2015:

1	123456	Unchanged
2	password	Unchanged
3	12345678	1 ↗
4	qwerty	1 ↗
5	12345	2 ↘
6	123456789	Unchanged
7	football	3 ↗
8	1234	1 ↘
9	1234567	2 ↗
10	baseball	2 ↘
11	welcome	NEW
12	1234567890	NEW
13	abc123	1 ↗
14	111111	1 ↗
15	1qaz2wsx	NEW
16	dragon	7 ↘
17	master	2 ↗
18	monkey	6 ↘
19	letmein	6 ↘
20	login	NEW
21	princess	NEW
22	qwertyuiop	NEW
23	solo	NEW
24	passw0rd	NEW
25	starwars	NEW

Most used passwords in 2018:

1 123456	Unchanged	13 welcome	Down 1
2 password	Unchanged	14 666666	New
3 123456789	Up 3	15 abc123	Unchanged
4 12345678	Down 1	16 football	Down 7
5 12345	Unchanged	17 123123	Unchanged
6 111111	New	18 monkey	Down 5
7 1234567	Up 1	19 654321	New
8 sunshine	New	20 !@#$%^&*	New
9 qwerty	Down 5	21 charlie	New
10 iloveyou	Unchanged	22 aa123456	New
11 princess	New	23 donald	New
12 admin	Down 1	24 password1	New
		25 qwerty123	New

Password Attacks

- **Dictionary attacks** offline attack
  - Dictionary contains common passwords
  - To speed up cracking, attacker pre-computes $\mathrm{H}(\mathrm{password})$ for every word in the dictionary.
  - Pre-computing needs to be done only once and offline.
  - Once the password file is obtained, cracking can be done immediately (search-and-compare)
  - Password guessing tools also utilize the frequency of letters, password patterns, etc.
- **Rainbow tables**
  - A space-time tradeoff
  - Yes, you can purchase from the Internet

https://freerainbowtables.com/

https://project-rainbowcrack.com

Password Security

userid	salt	hash
jfiore	56129	hash(56129|jfiore's password)
skippy	21592	hash(21592|skippy's password)
lenny	55573	hash(55573|lenny's password)
etc.		

- The countermeasure: Salting
  - Salt is a unique, random value chosen for each user. Not a secret (think of them like IV or nonces)
  - It's chosen randomly when a password is first set and stored in the password file
  - So, password hash = $H(salt \,||\, password)$
- With salting, users with the same password have different entries in the password file
- Salting adds randomness to password hash, making offline dictionary attack harder
  - Why?

Advantages of Salting

- In dictionary attacks, attacker must pre-compute the hashes for all the words in the dictionary using all known hash algorithms
  - But this computation is only once, done offline
  - Sometimes, the hash function is known, e.g., crypt in UNIX
  - For identical passwords, their hashes have the same value

Only need one table for all password files

- With salting, attacker still needs to pre-compute the hashes of all the words
  - For each password entry, he needs to try all possible salts
  - Attacker must try all dictionary words for all salt values

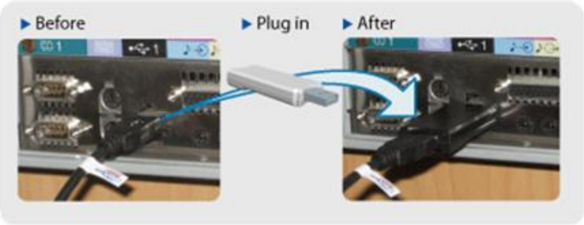
e.g., with 12-bit salt, the same password can have 2^{12} hash values

Online Password Attacks

- The attacker must interact with the service to test the (cracked or guessed) passwords
  - Mallory tries different passwords to log in to a website
  - Each time, the service computes the hash and makes the decision
  - Defenses?
    - Set limits: timeout, preventing attacker from trying too many times
    - Increase time costs: rate limit the number of tries within a period of time
    - Use captcha: detect scripts, preventing automated attacks
- Offline attacks: the attacker performs all the computation without hard limit
  - Assumptions?
  - Defenses?

Other Password Security Risks

- Weak passwords
- Default passwords
  - Mirai botnet
  - US District Courthouse Server “public/public”
- Broken implementations
  - e.g., timing attacks against TENEX operating system
- Keystroke loggers
  - Software: spyware
  - Hardware: KeyGhost, KeyShark
- Social engineering



SYSTEM CONTROL APU “DEFENDED” STANDARD PASSWORDS

magictr | September 27, 2018 | Techno | 0 Comments

The armed forces of Ukraine used to access the servers of the automated command and control system “Dnipro” simple, default password, type admin and 12345.

This became known journalist Alexander Dubinsky, reports UNIAN. According to the documents, which are at the disposal of the journalist, the “gap” in cyber defence automated command and control system found, database specialist, Dmitry Vlasyuk. On many servers, and switches with the IP addresses of the access was carried out according to standard login and password.

The use of standard passwords could cause a big tececo data. The enemy could theoretically scan the entire network ACS Dnipro, gaining access to sensitive data.

Additional Issues

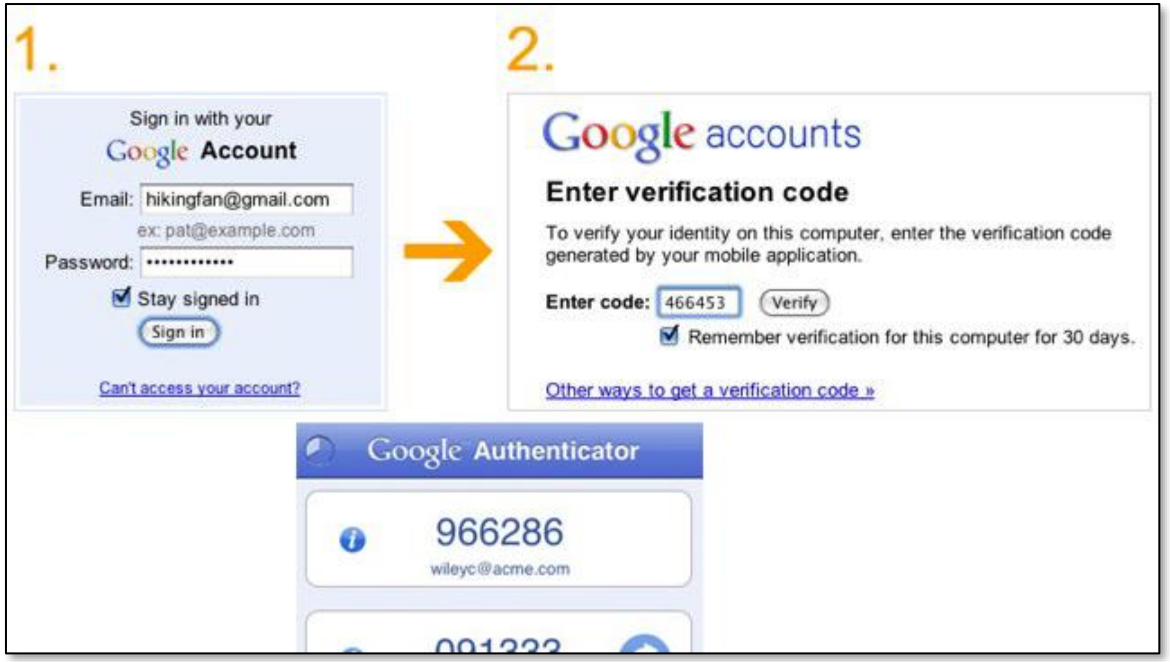
- **Password Strength**
  - Old Password Policies: 7/8 characters, at least 3 out of {digits, upper-case, lower-case, non-alphanumeric}, no dictionary words, change every 4 months, not like previous 12 passwords, etc.
  - NIST recommendation (2017): remove requirement for periodical change, allow but not require arbitrary special characters, allow copy-paste, …
  - Password checkers: https://www.security.org/how-secure-is-my-password/
- **Usability** --- password manager can help
  - Hard-to-remember passwords
  - Password management issues
    - Write them down: NO!
    - Heavy reuse: NO

Improve Password Security

- **Password managers**
  - e.g., LastPass, KeePass, browser built-in solutions
  - What would happen if it’s compromised?
- **Graphical passwords**: easy to remember, no need to write down?
  - e.g., draw on the image (Windows 8), android patterns
  - Side-channel attacks
- Add **biometrics**: unique, hard to fake, no need to remember
  - e.g., fingerprint, iris scan, behavior-based characteristics (handwriting, typing)
  - Additional hardware, private but not secret (can be stolen), hard to revoke, high false positives

Improve Password Security

- **Multi-factor authentication**
  - Use more than one authentication mechanisms for authentication
  - Google: Password + SMS
  - FIDO: Password + hardware

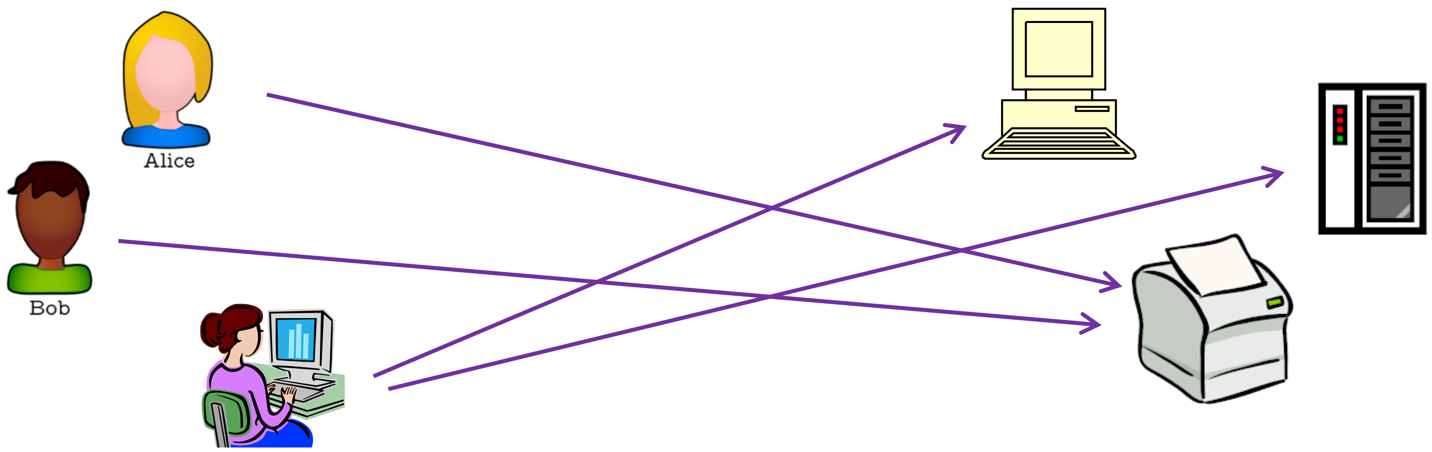


Kerberos

Kerberos History

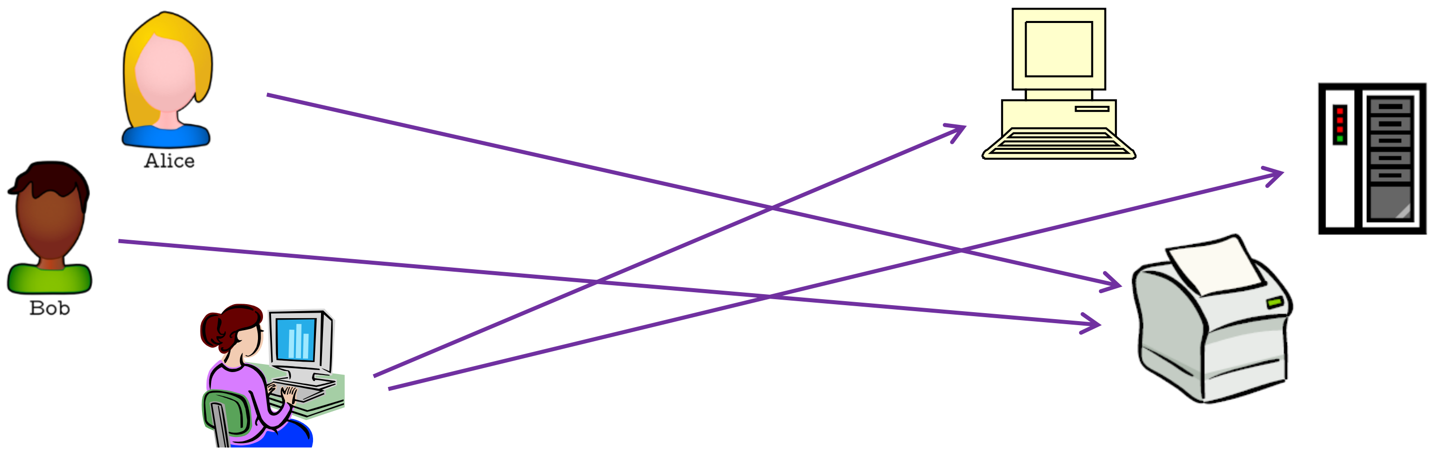
- Developed as a part of Project Athena at MIT to solve password eavesdropping
- Support online authentication – a variant of Needham-Schroeder
- Easy application integration API
- The first single sign-on system – sign-on once, access all services
- The most widely used (non-web) centralized password system in existence
  - Adopted by Windows 2000 and all later versions
  - Also available for Unix/Linux family of OS

Distributed Authentication



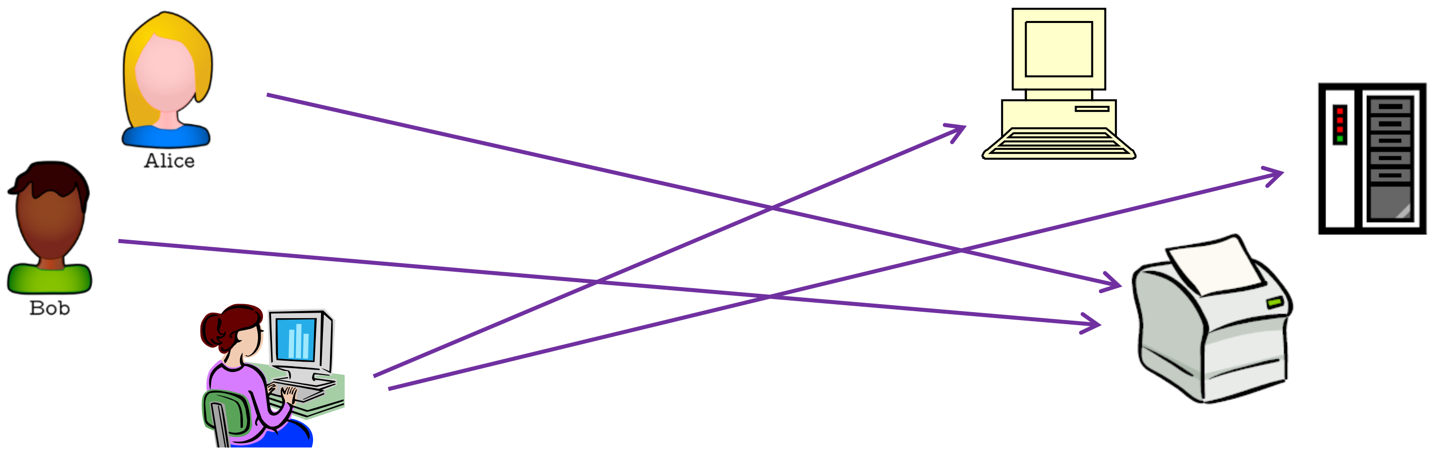
- Threats?
  - **User impersonation:** a malicious user with access to a workstation pretends to be another user using the same station.
  - **Network impersonation:** a malicious user changes the network address of his workstation to impersonate another workstation.
  - **Eavesdropping**, **message modification**, and **replay** attacks

Distributed Authentication



- How to prove user's identity when requesting services from machines on the network?
  - Many-to-many authentication: m clients, n servers
  - Naïve solution: Each server knows every user's password
    - Too many keys: $m \times n$ secret keys shared between each pair of (client, server)
    - Attacker breaking into one server knows all the keys. Besides, key maintenance is complex.
    - Better solution? -- Kerberos

Distributed Authentication

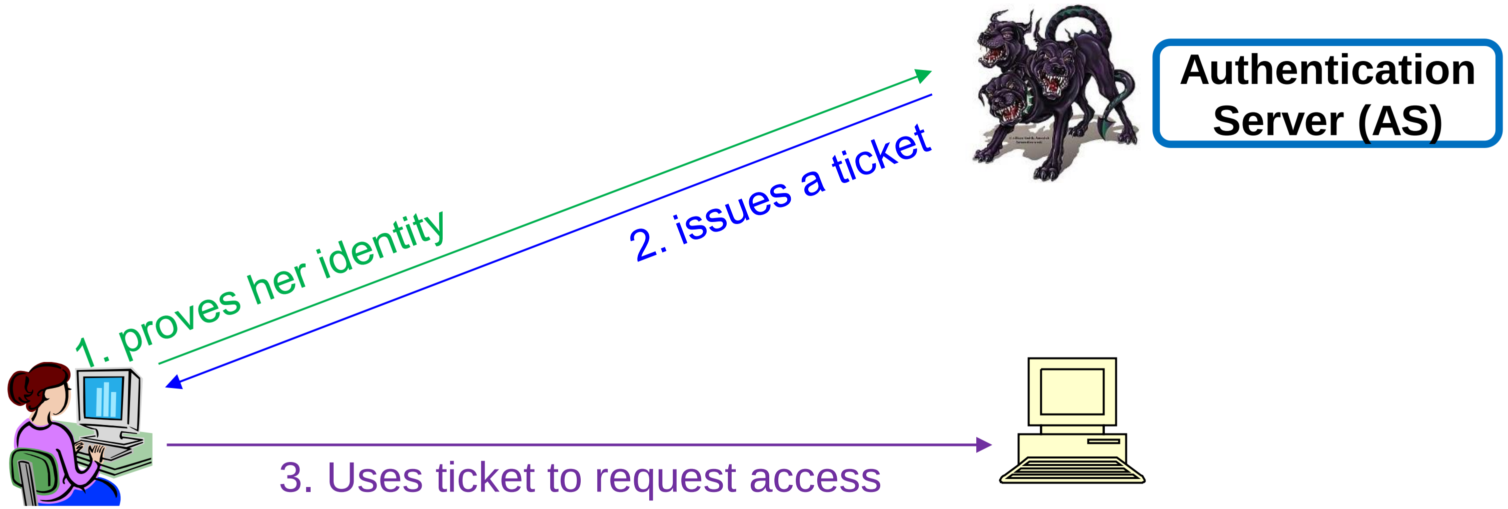


- What do we expect?
  - Secure against attacks by **passive eavesdroppers** and **active malicious attackers**.
  - Transparent, users just need to enter password. They should not notice how authentication is performed.
  - Scalable to serve many users and servers.

Kerberos Overview

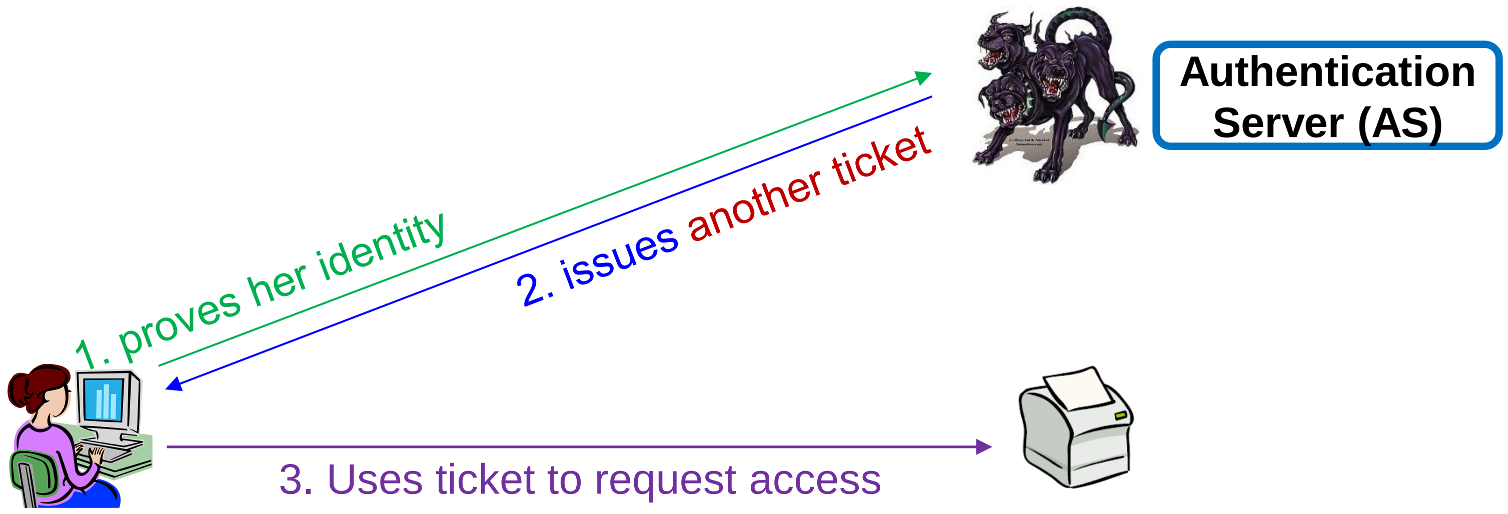
Knows all passwords. Requires high level of security.

- **Key idea:** use a trusted third party
  - So, it's a centralized authentication service.
  - Both clients and servers trust the Kerberos system to mediate authentication.



Kerberos Overview

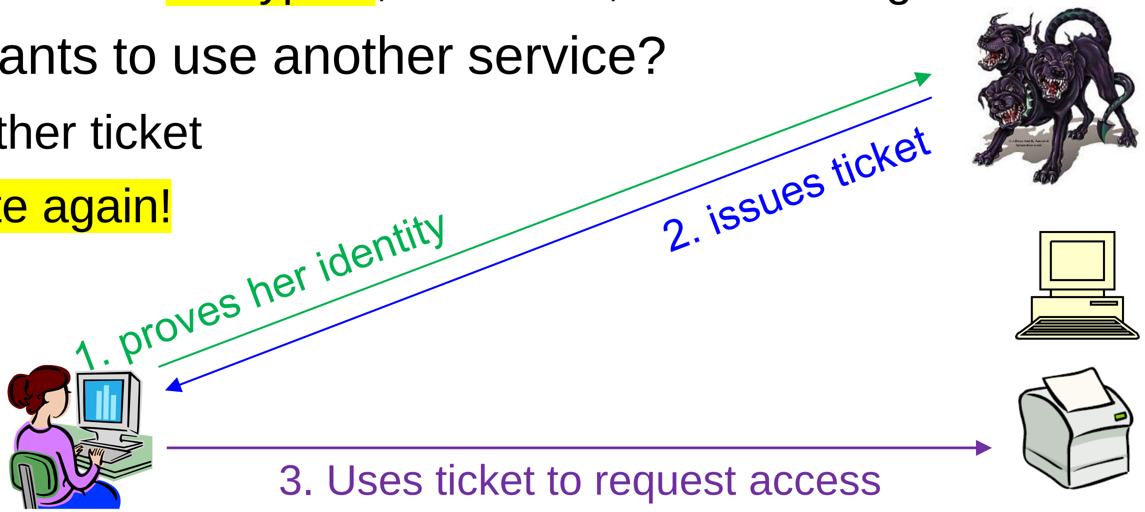
- **Key idea:** use a trusted third party
  - <u>Good</u>: network servers maintain **NO** authentication data or service.
  - <u>Good</u>: client needs to remember only **one** password.
  - <u>Bad</u>: it's a **single point of failure**!



Naïve Approach

- Alice proves her identity
  - Send password to AS – insecure to send password in plaintext
- AS verifies Alice's identity and issues a ticket
  - What should a ticket look like?
  - Ticket needs to be encrypted, otherwise, it can be forged.
- What if Alice wants to use another service?
  - Needs another ticket
  - Authenticate again!

Which key to use?

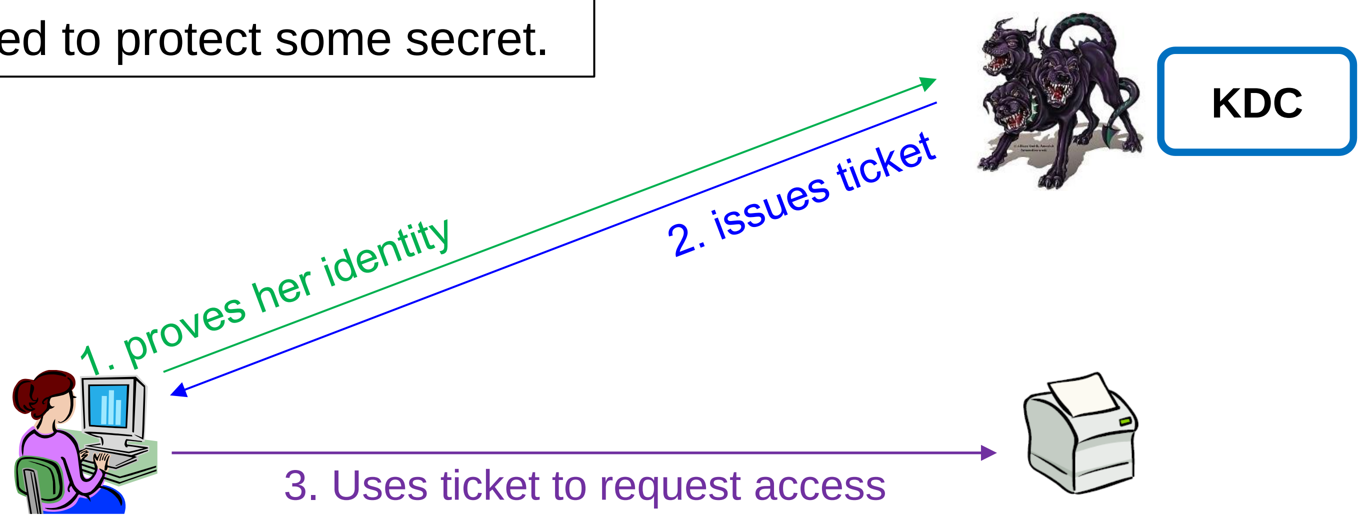


Kerberos Overview

1. Send password to AS – insecure to send plaintext password.

- Convert “password” into client master key: K_A
- K_A is shared with AS or KDC (Key Distribution Center)

K_A is not transmitted over insecure channel but used to protect some secret.

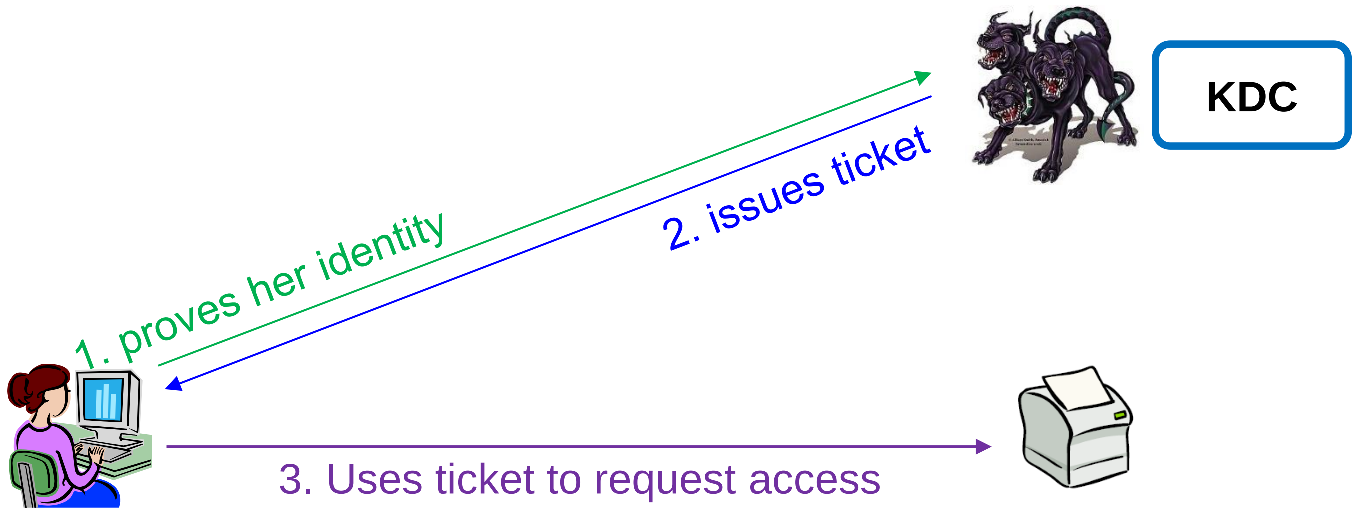


Kerberos Overview

2. Issue ticket – ticket needs to be **encrypted**. Otherwise, it can be forged.

- Client → KDC: "I'm Alice, want to talk to Bob"
  - $\mathrm{ID}_A, \mathrm{ID}_B$
- KDC → Client: encrypted session key & ticket

1. Ticket is protected from Alice. She can only pass it to the server.
2. Ticket proves Alice has authenticated.



Kerberos Overview

2. Issue ticket – ticket needs to be **encrypted**. Otherwise, it can be forged.

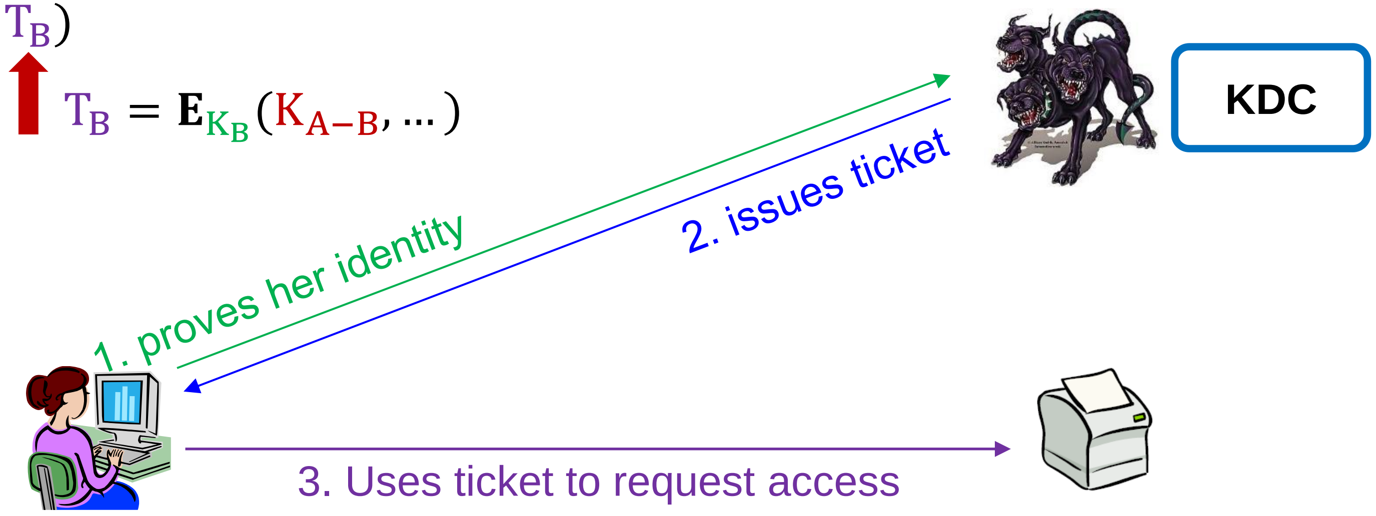
- Client → KDC: "I'm Alice, want to talk to Bob"
  - $\mathrm{ID_A}, \mathrm{ID_B}$
- KDC → Client: encrypted session key & ticket

$$\mathbf{E}_{K_A}(K_{A-B}, \mathrm{ID_B}, T_B)$$

session key

$$T_B = \mathbf{E}_{K_B}(K_{A-B}, \ldots)$$

1. Ticket is protected from Alice. She can only pass it to the server.
2. Ticket proves Alice has authenticated.



Kerberos Overview

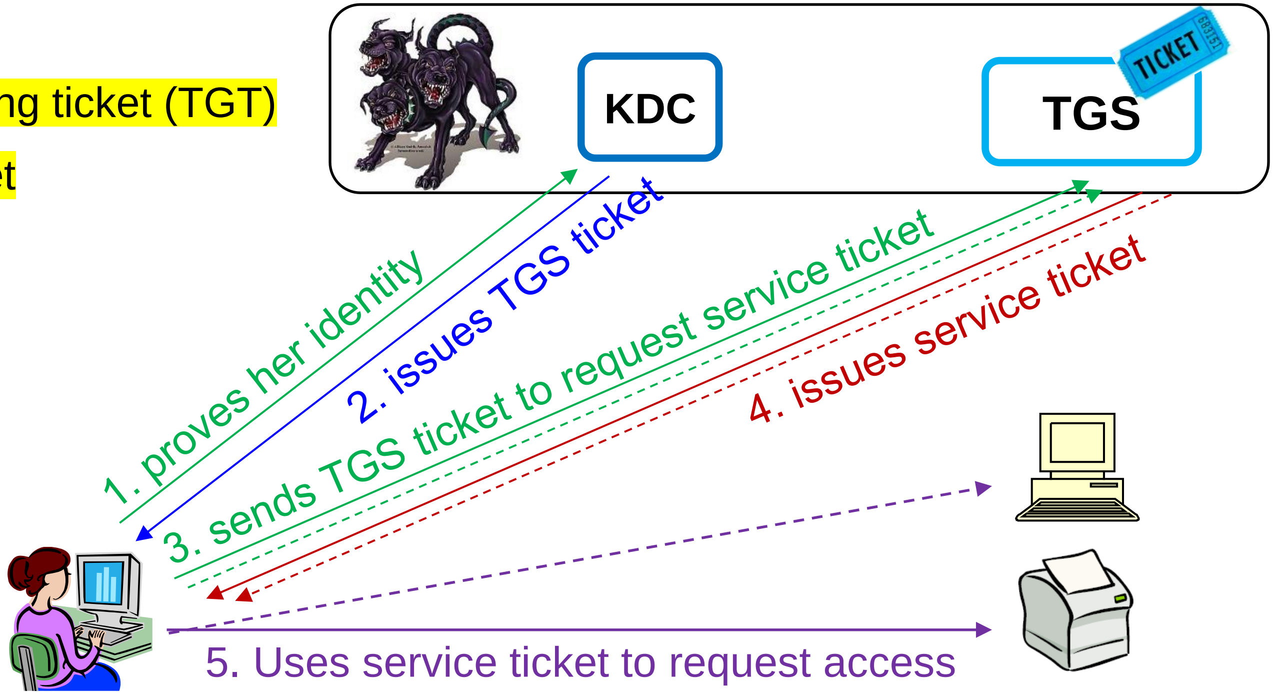
- What should be included in a ticket?
- Ticket proves Alice has authenticated $T_B = \mathbf{E}_{K_B}(K_{A-B}, \ldots)$ $\mathbf{E}_{K_A}(K_{A-B}, ID_B, T_B)$
  - User name: ID_A X **User impersonation**
  - Server name: ID_B
  - Address of user's workstation: IP_A X **Network impersonation**
  - Session key
  - Ticket lifetime
  - A few other things, e.g., timestamp

X **Eavesdropping, message modification, replay attacks**

Kerberos Overview

3. Alice wants to access another service

- **Solution:** Two-step Authentication with KDC and Ticket Granting Server (TGS)
- Two tickets:
  - Ticket granting ticket (TGT)
  - Service ticket



Remaining Threats

- A malicious user Eve could intercept messages between Alice and Kerberos authentication servers (KDC and TGS)
  - Eve gets the service ticket
- Eve could use the same workstation used by Alice
  - Ticket has workstation address, but it doesn’t help.
- Eve could even crack Alice’s password
  - Eve can impersonate Alice!

Remaining Threats

- **Ticket hijacking**
  - Eve steals the service ticket and tries to use it on the same workstation
  - So, servers must verify the user presenting the ticket is the same user issued with the ticket.

  Alice must show a proof (called authenticator) along with the service ticket.

  A ticket must be fresh and expire after a period of time.

- **No server authentication**
  - A malicious server could hijack Alice's requests and pretends to be Bob
  - So, servers must prove their identity to users

  Server must show a proof to the user.

KDC

TGS

$\mathrm{ID_A},\ time_A$

$\mathbf{E}_{\mathrm{K_A}}(\mathrm{K_{A-TGS}}, \mathrm{ID_{TGS}}, time_{\mathrm{KDC}}, \text{lifetime}, \mathrm{TGT})$

$\mathrm{TGT} = \mathbf{E}_{\mathrm{K_{TGS}}}(\mathrm{K_{A-TGS}}, \mathrm{ID_A}, \mathrm{ID_{TGS}}, time_{\mathrm{KDC}}, \text{lifetime})$

$\text{authenticator}_1 = \mathbf{E}_{\mathrm{K_{A-TGS}}}(\mathrm{ID_A}, \mathrm{ID_B}, time_{\mathrm{A}})$

$\mathrm{ID_A},\ \mathrm{ID_B},\ \mathrm{TGT},\ \text{authenticator}_1$

$\mathbf{E}_{\mathrm{K_{A-TGS}}}(\mathrm{K_{A-B}}, \mathrm{ID}_B, time_{\mathrm{TGS}}, \text{lifetime}, \mathrm{T_B})$

$\mathrm{T_B} = \mathbf{E}_{\mathrm{K_B}}(\mathrm{K_{A-B}}, \mathrm{ID_A}, \mathrm{ID_B}, time_{\mathrm{TGS}}, \text{lifetime})$

$\mathrm{ID_A},\ \mathrm{T_B},\ \text{authenticator}_2$

$\text{authenticator}_2 = \mathbf{E}_{\mathrm{K_{A-B}}}(\mathrm{ID_A}, time_{\mathrm{A}})$

$\mathbf{E}_{\mathrm{K}_{A-B}}(time_A + 1)$

Client authentication

KDC

TGS

Sign on once

transparent

scalable

$\mathrm{ID_A}, \textit{time}_{\mathrm{A}}$

$\mathbf{E}_{\mathrm{K_A}}(\mathrm{K_{A-TGS}}, \mathrm{ID_{TGS}}, \textit{time}_{\mathrm{KDC}}, \text{lifetime}, \mathrm{TGT})$

$\mathrm{TGT} = \mathbf{E}_{\mathrm{K_{TGS}}}(\mathrm{K_{A-TGS}}, \mathrm{ID_A}, \mathrm{ID_{TGS}}, \textit{time}_{\mathrm{KDC}}, \text{lifetime})$

$\mathrm{ID_A}, \mathrm{ID_B}, \mathrm{TGT}, \text{authenticator}_1$

$\text{authenticator}_1 = \mathbf{E}_{\mathrm{K_{A-TGS}}}(\mathrm{ID_A}, \mathrm{ID_B}, \textit{time}_{\mathrm{A}})$

No ticket replay

$\mathbf{E}_{\mathrm{K_{A-TGS}}}(\mathrm{K_{A-B}}, \mathrm{ID}_B, \textit{time}_{\mathrm{TGS}}, \text{lifetime}, \mathrm{T_B})$

$\mathrm{T_B} = \mathbf{E}_{\mathrm{K_B}}(\mathrm{K_{A-B}}, \mathrm{ID_A}, \mathrm{ID_B}, \textit{time}_{\mathrm{TGS}}, \text{lifetime})$

$\mathrm{ID_A}, \mathrm{T_B}, \text{authenticator}_2$

$\text{authenticator}_2 = \mathbf{E}_{\mathrm{K_{A-B}}}(\mathrm{ID_A}, \textit{time}_{\mathrm{A}})$

$\mathbf{E}_{\mathrm{K_{A-B}}}(\textit{time}_A + 1)$

Server authentication

Kerberos Overview

- Long-term symmetric keys: used only to derive short-term, session keys
  - K_A of client (Alice), known to client and KDC
  - K_{TGS} of the TGS server, known to TGS and KDC
  - K_B of a network server (Bob), known to this server and TGS
- Short-term symmetric keys: a unique key for each pair of client and server
  - K_{A-TGS}, session key between client and TGS, generated by KDC
  - K_{A-B}, session key between client and network server, generated by TGS
- Tickets are short-term credentials. So, they can be stored in memory or disk.

Kerberos Summary

- It provides a centralized authentication service.
- It can support mutual authentication.
- Entirely based on symmetric cryptography.
- Less keys to remember for clients (only the credential with KDC)
- Less work on servers, a little more work on clients
  - KDC maintains long-term secret keys for clients and servers, but servers don’t.
  - Client requests short-term credentials (ticket + session key) and manages them.
- Less communication overhead (client sends both ticket and authenticator to server, no need to wait)
- More scalable in a large distributed system.