Authentication

Bo Luo
bluo@ku.edu

Authentication

- Authentication
  - Alice talks with Bob
    - How does Bob know it's Alice?
  - Alice logs in to use the system
    - How does the system know it's Alice?
- We consider two authentication scenarios
  - Server authentication
    - Certificate
  - User authentication
    - With OS
    - In a distributed system

Certificate

- Bob gets a message claiming to be from Alice
  - Message is signed with key claiming to be Alice's
    - Still remember MAC?
  - Signature matches the message
  - Is Bob sure that it came from Alice?
  - How could Bob confirm that the public key belongs to Alice?

Certificate

- Related question. can I:
  - Take a small picture of myself
  - Attach it to a card saying that I'm Trump
  - get a free ride on Air Force One?

Certificate

- I tell you my real name
  - How can you verify?
  - I can show you my Driver's License
  - I can show you my KU Card
- I also have a card saying that I'm Trump
  - why do you trust a driver's license but not the ID card that I created saying I'm Trump?

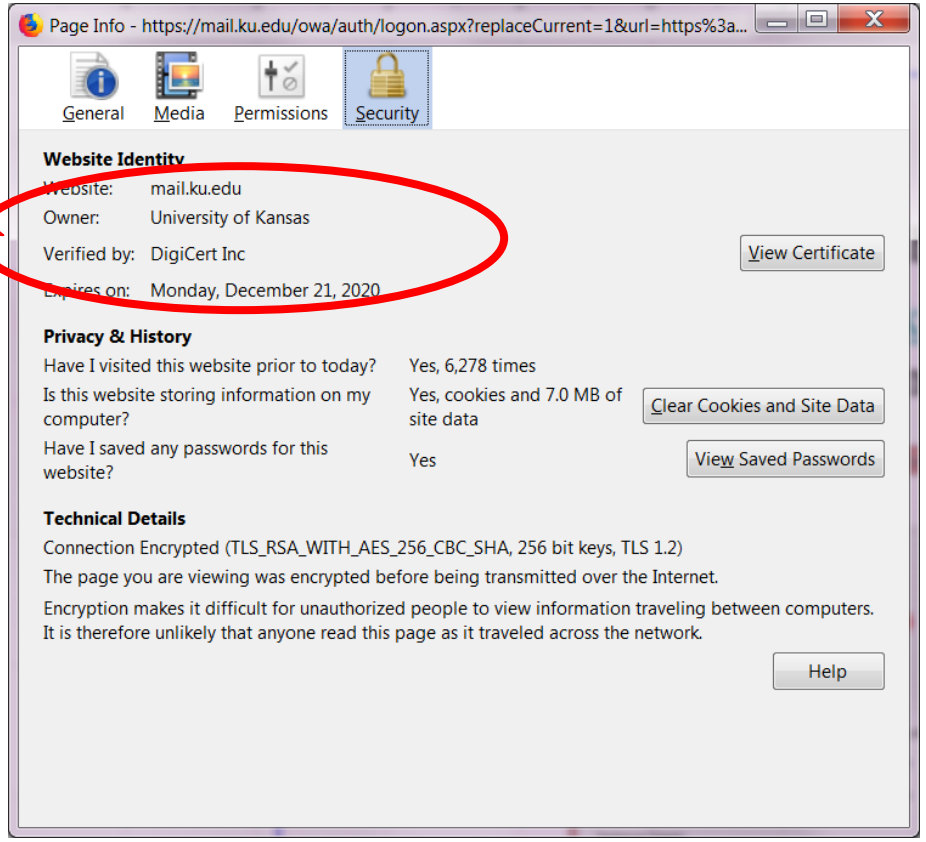
Certificate

- I tell you my real name
  - How can you verify?
  - I can show you my Driver's License
  - I can show you my KU Card
- I also have a card saying that I'm Trump
  - why do you trust a driver's license but not the ID card that I created saying I'm Trump?
  - State of KS vouches the picture matches: the name, address, etc. of the info on the card
  - *If you trust KS, you believe info on license*

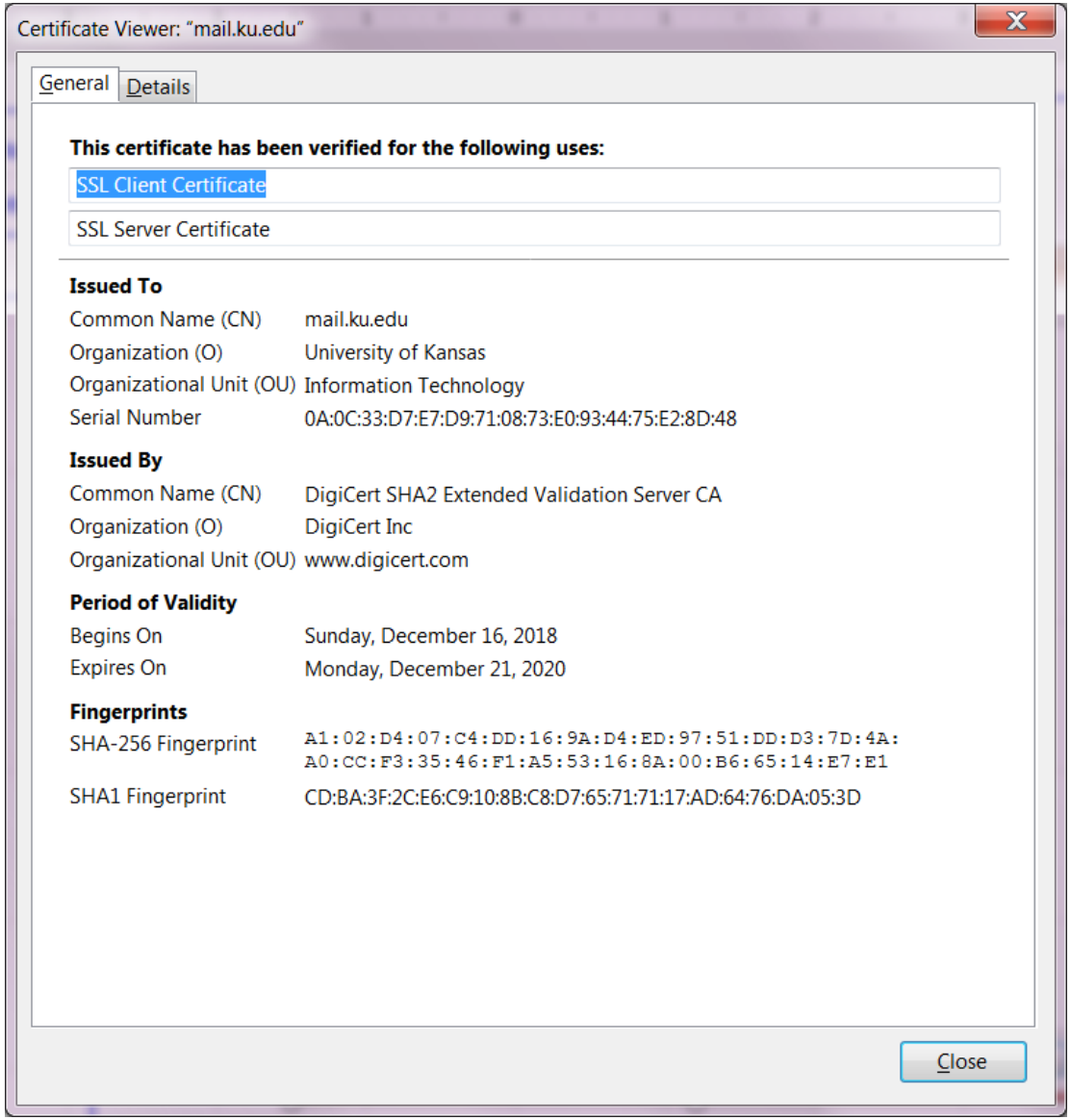
Certificate

- Can we do the same for public keys?
- When we use https://mail.ku.edu/, how do we know if it's really KU mail system?

You trust this is KU, if you trust *DigiCert Inc*



Certificate



Public-Key Infrastructure (PKI)

- Goal of authentication: bind identity to key
- Public key: bind identity to public key
  - Crucial as people will use key to communicate with principal whose identity is bound to key
  - Erroneous binding means no secrecy between principals
  - Assume principal identified by an acceptable name

Public-Key Infrastructure (PKI)

- Certificate: token (message) containing
  - Identity of principal (here, Alice)
  - Corresponding public key
  - Timestamp (when issued)
  - Other information (perhaps identity of signer)
  - Compute hash (message digest) of token
- Hash encrypted by trusted authority (here, Cathy) using private key: called a "signature"

$$C_A = e_A \,||\, \text{Alice} \,||\, T \,||\, \{\text{h}(e_A \,||\, \text{Alice} \,||\, T)\}_{d_C}$$

public key

Public-Key Infrastructure (PKI)

- Alice's certificate:

$$C_A = e_A \,\|\, \text{Alice} \,\|\, T \,\|\, \{\text{h}(e_A \,\|\, \text{Alice} \,\|\, T)\}_{d_C}$$

- Bob gets Alice's certificate
  - If he knows Cathy's public key, he can validate the certificate
    - Decrypt the encrypted hash using Cathy's public key
    - Re-compute hash from certificate and compare
    - Check validity
    - Is the principal Alice?
  - Now Bob has Alice's public key
    - Alice is endorsed by Cathy!

Public-Key Infrastructure (PKI)

- Alice's certificate:

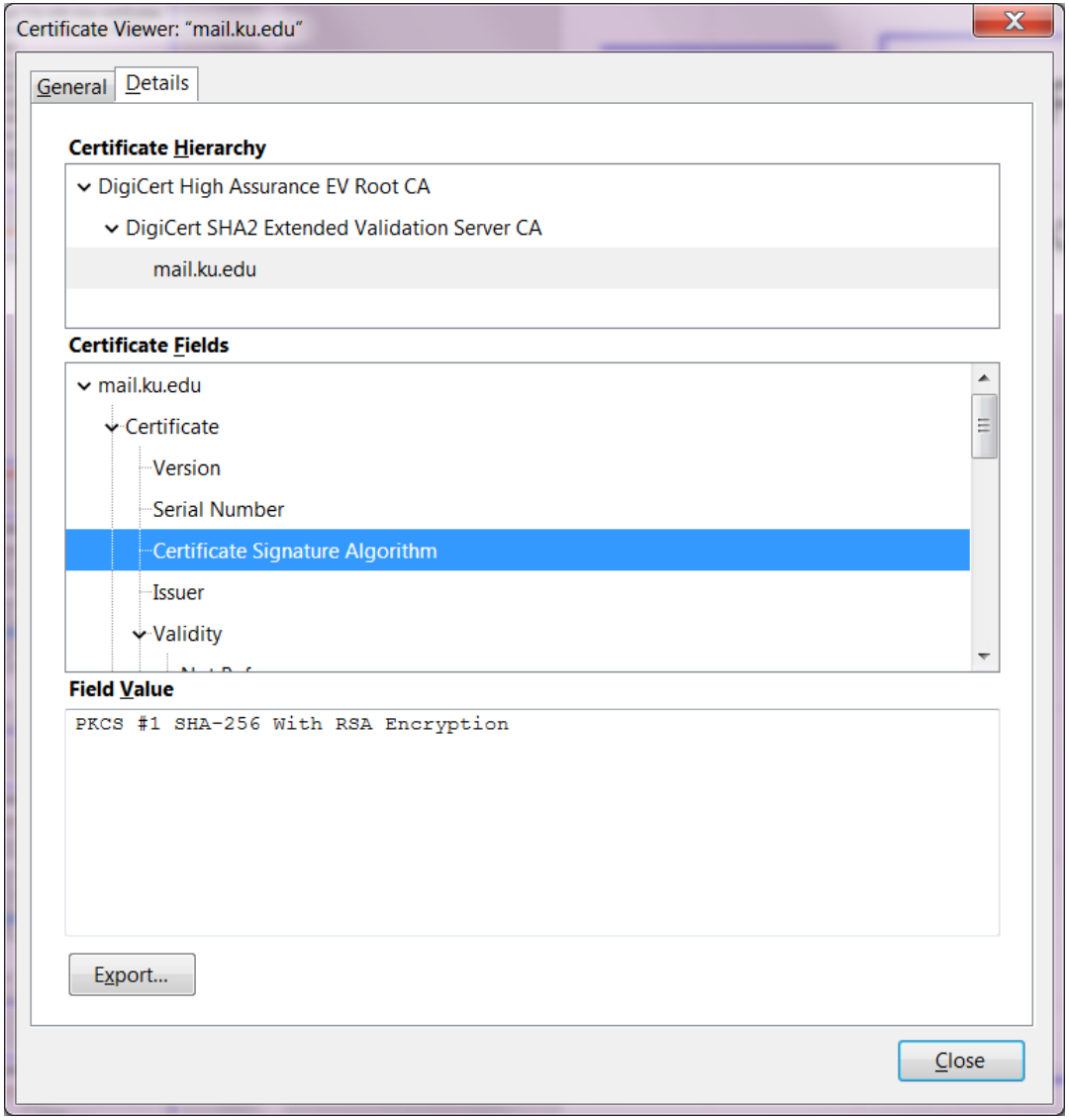
$$C_A = e_A \,\|\, \text{Alice} \,\|\, T \,\|\, \{\text{h}(e_A \,\|\, \text{Alice} \,\|\, T)\}_{d_C}$$

- Problem: Bob needs Cathy's public key to validate certificate
  - That is, secure distribution of public keys
  - Solution: Public Key Infrastructure (PKI) using **trust anchors** called *Certificate Authorities* (CAs) that issue certificates

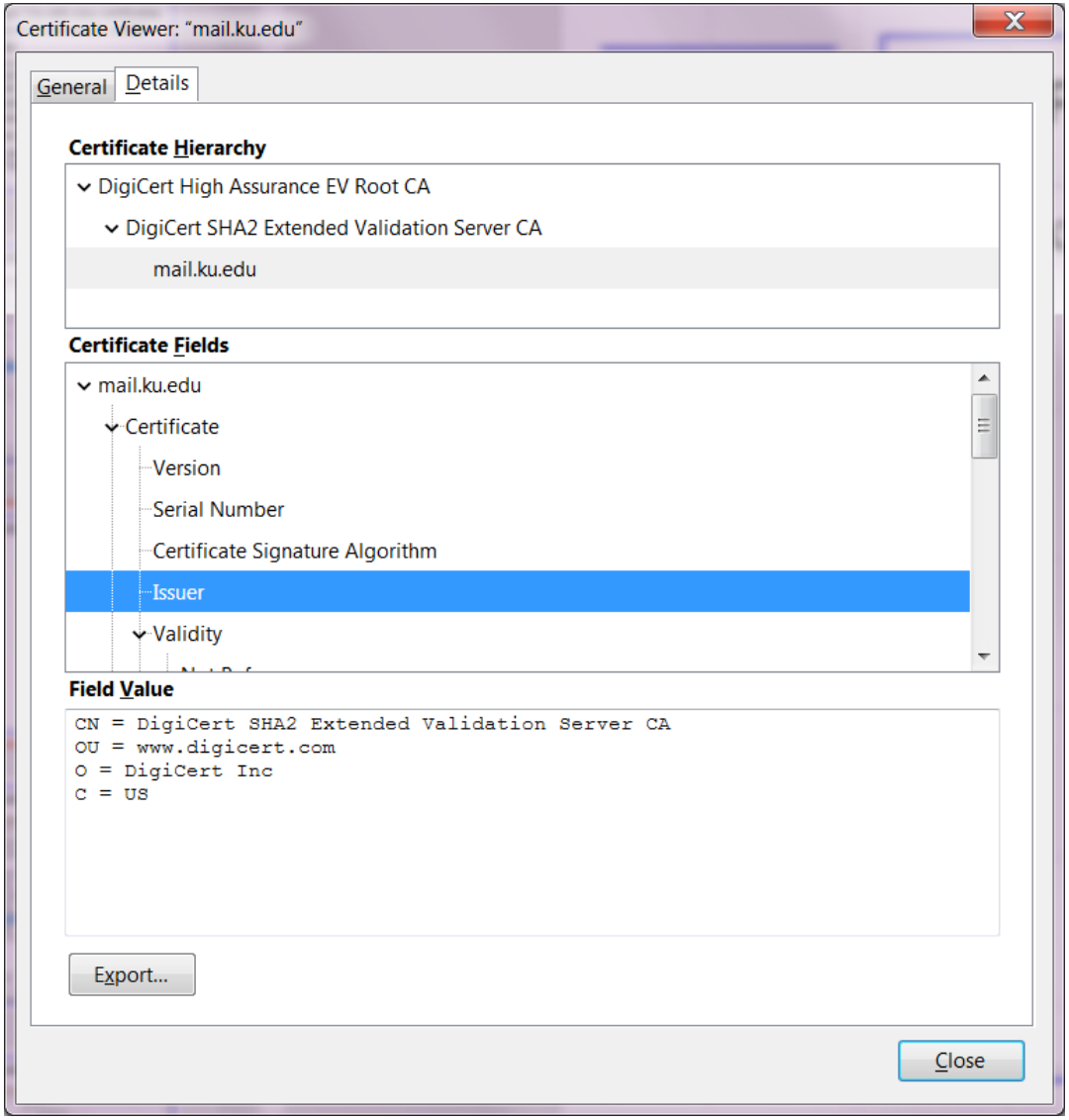
Public-Key Infrastructure (PKI)

- Hierarchical CAs with cross-certification
  - Multiple root CAs that are cross-certified
- Web Model
  - **Browsers or Operating Systems come pre-configured with multiple trust anchor certificates**
  - New certificates can be added (be careful!)
  - Bad certificate can be revoked.
- Distributed model (e.g., PGP)
  - No CA; instead, users certify each other to build a “web of trust”

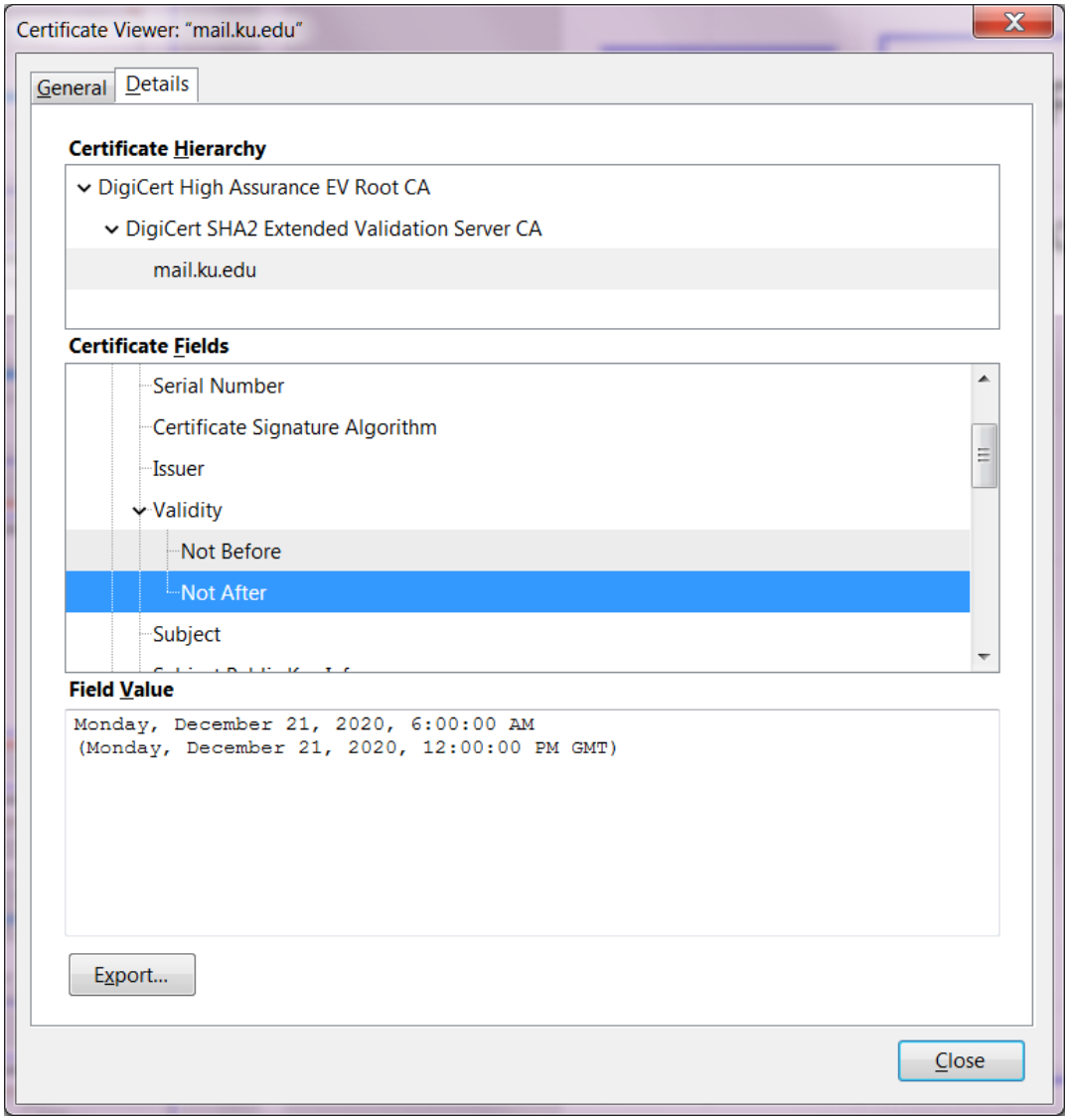
Public-Key Infrastructure (PKI)



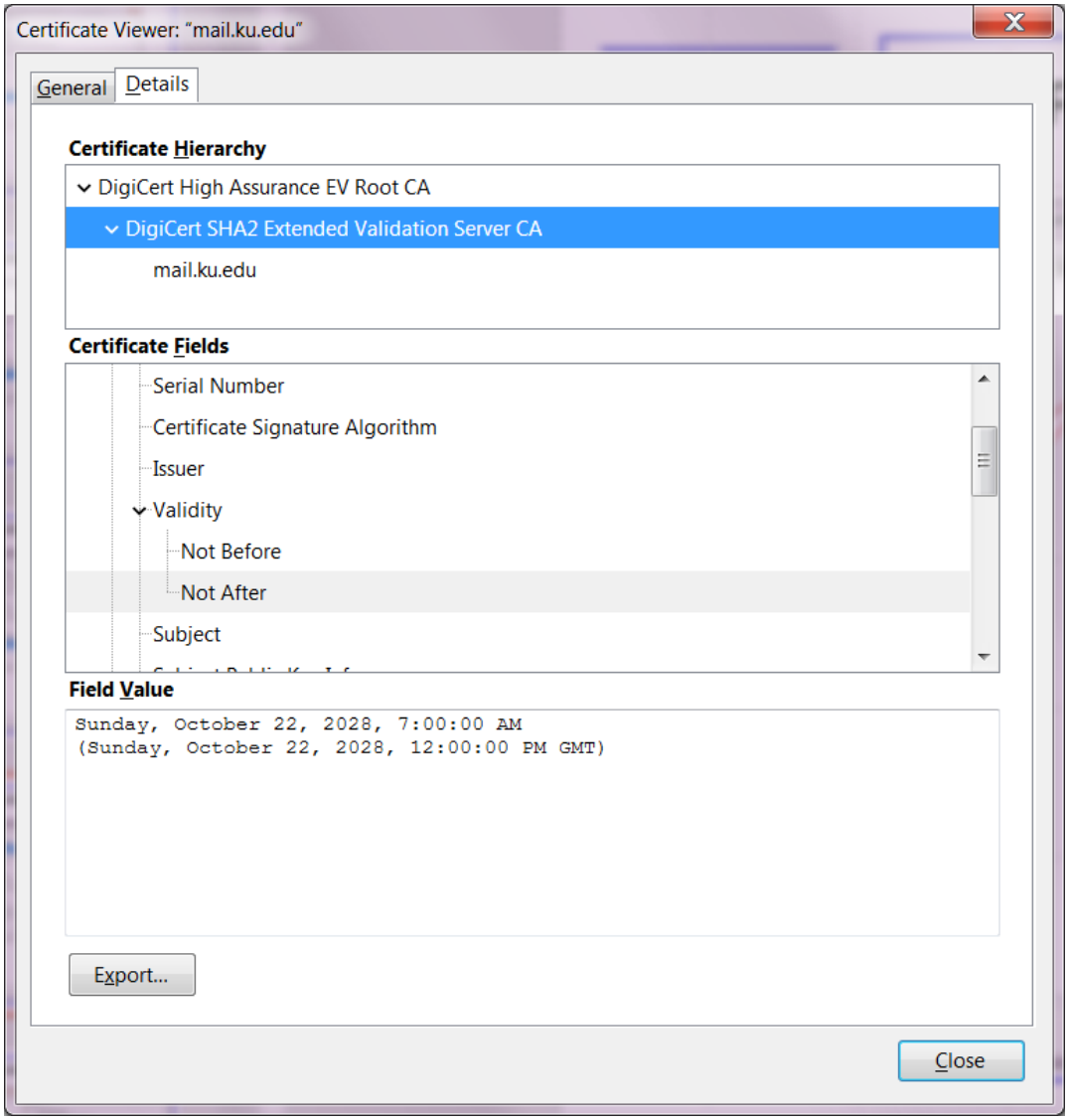
Public-Key Infrastructure (PKI)



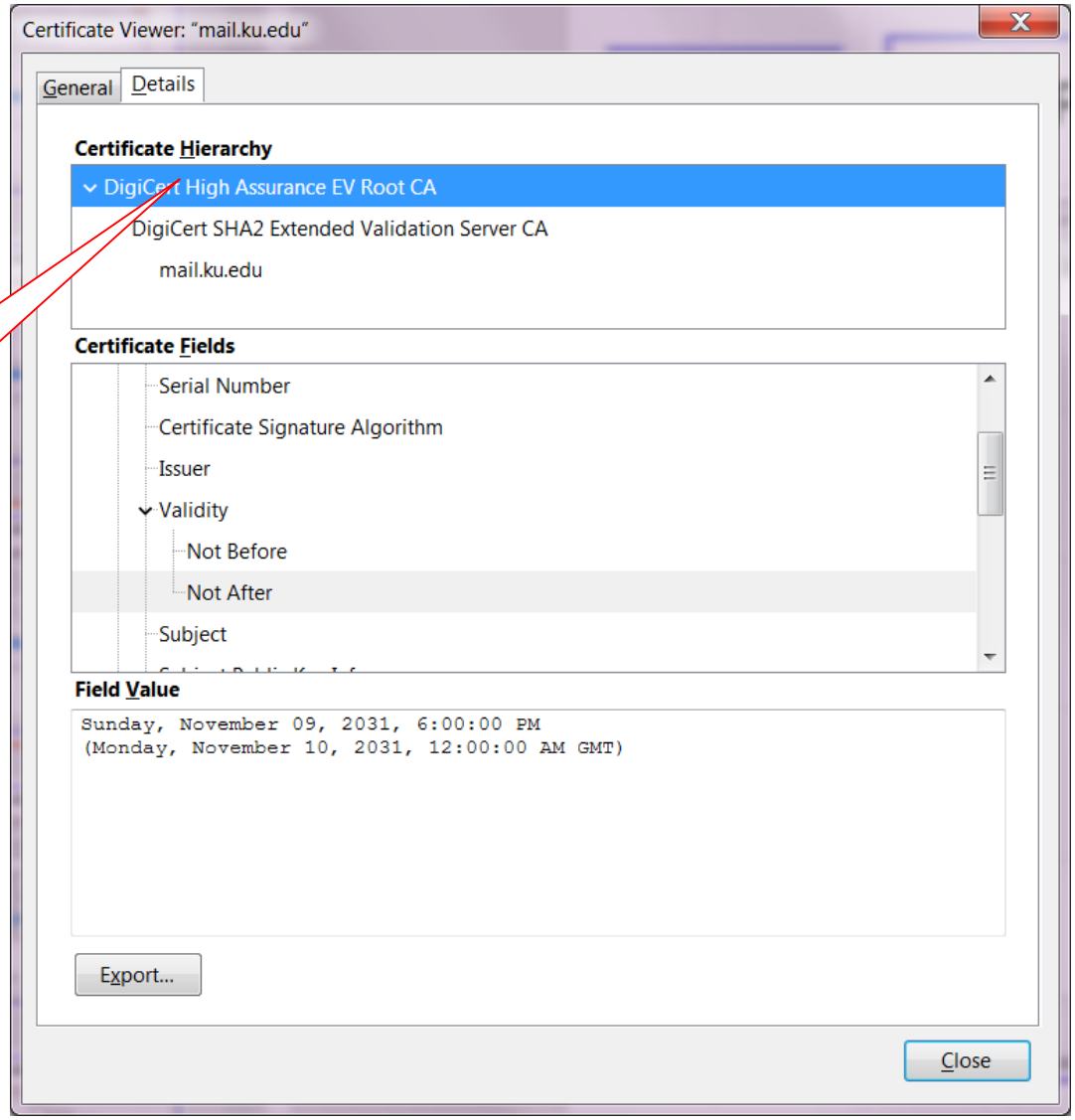
Public-Key Infrastructure (PKI)



Public-Key Infrastructure (PKI)

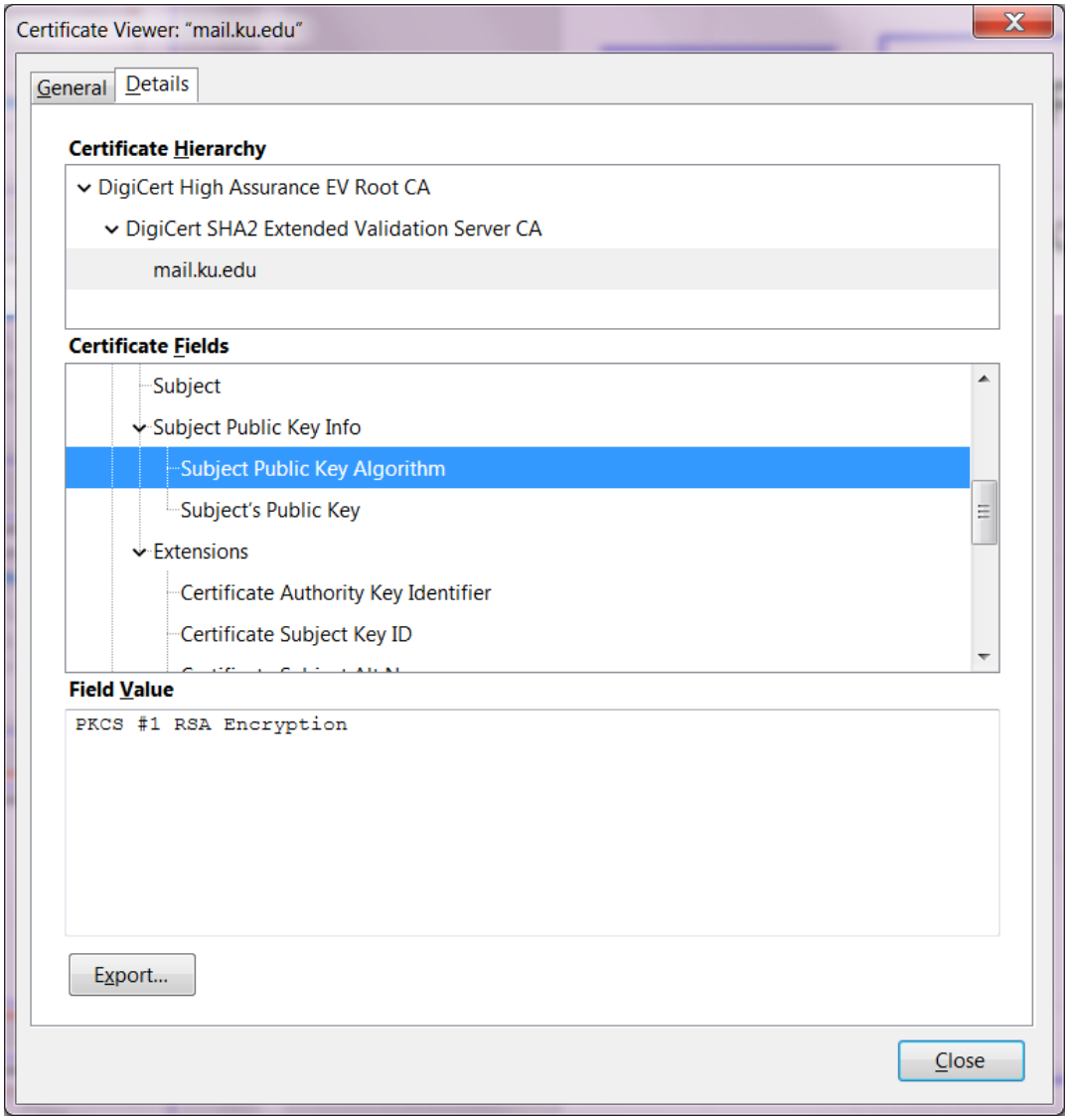


Public-Key Infrastructure (PKI)

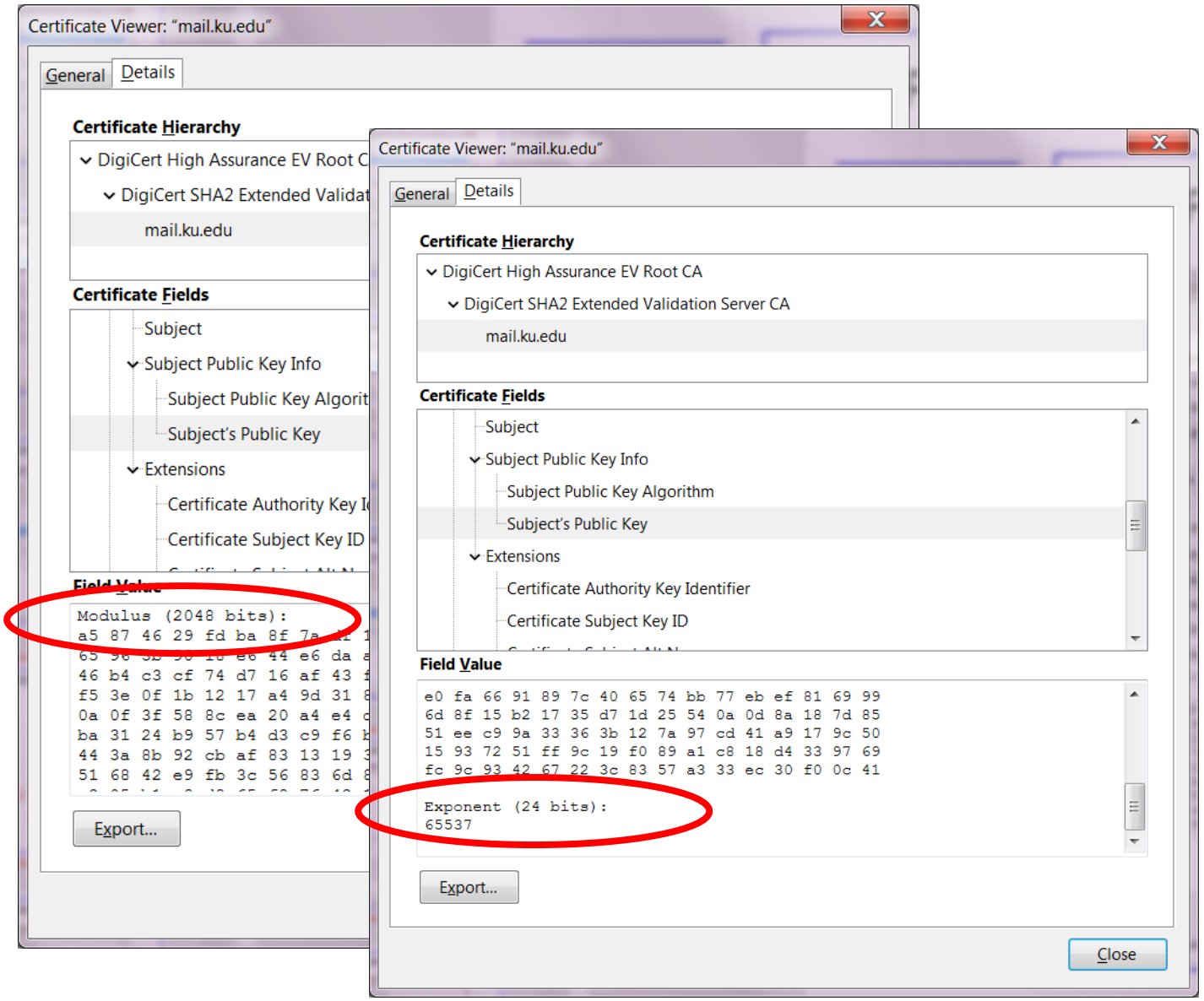


The root CA is good for 25 years.

Public-Key Infrastructure (PKI)



Public-Key Infrastructure (PKI)



Authentication

- Authentication
  - Alice talks with Bob
    - How does Bob know it's Alice?
  - Alice logs in to use the system
    - How does the system know it's Alice?
- We consider two authentication scenarios
  - Server authentication
    - Certificate
  - **User authentication**
    - **With OS**
    - In a distributed system

Authentication

- Credentials can be
  - Something I am
  - Something I have
  - Something I know
- Passwords
  - Used to authenticate users in OS, web, email, etc.

Password Authentication

- Passwords are pretty weak……
- SplashData's Worst Passwords List: most commonly used passwords in 2015 (from 2M+ leaked passwords)

RANK	PASSWORD	CHANGE FROM 2014
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	password	NEW
25	starwars	NEW

Password Authentication

- Passwords are pretty weak……
- SplashData's Worst Passwords List: most commonly used passwords in 2018

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

Password Authentication

- What if the bad guy gets your password file?
  - Aside: this has happened a lot. Many of them through SQL injection attack.
- Instead of storing cleartext passwords
  - Store passwords transformed through some one-way function, *e.g.* the hash of the password.
- When user sends password
  - take hash of the password, H(pass)
  - check H(pass) == what's in password file

Password Authentication

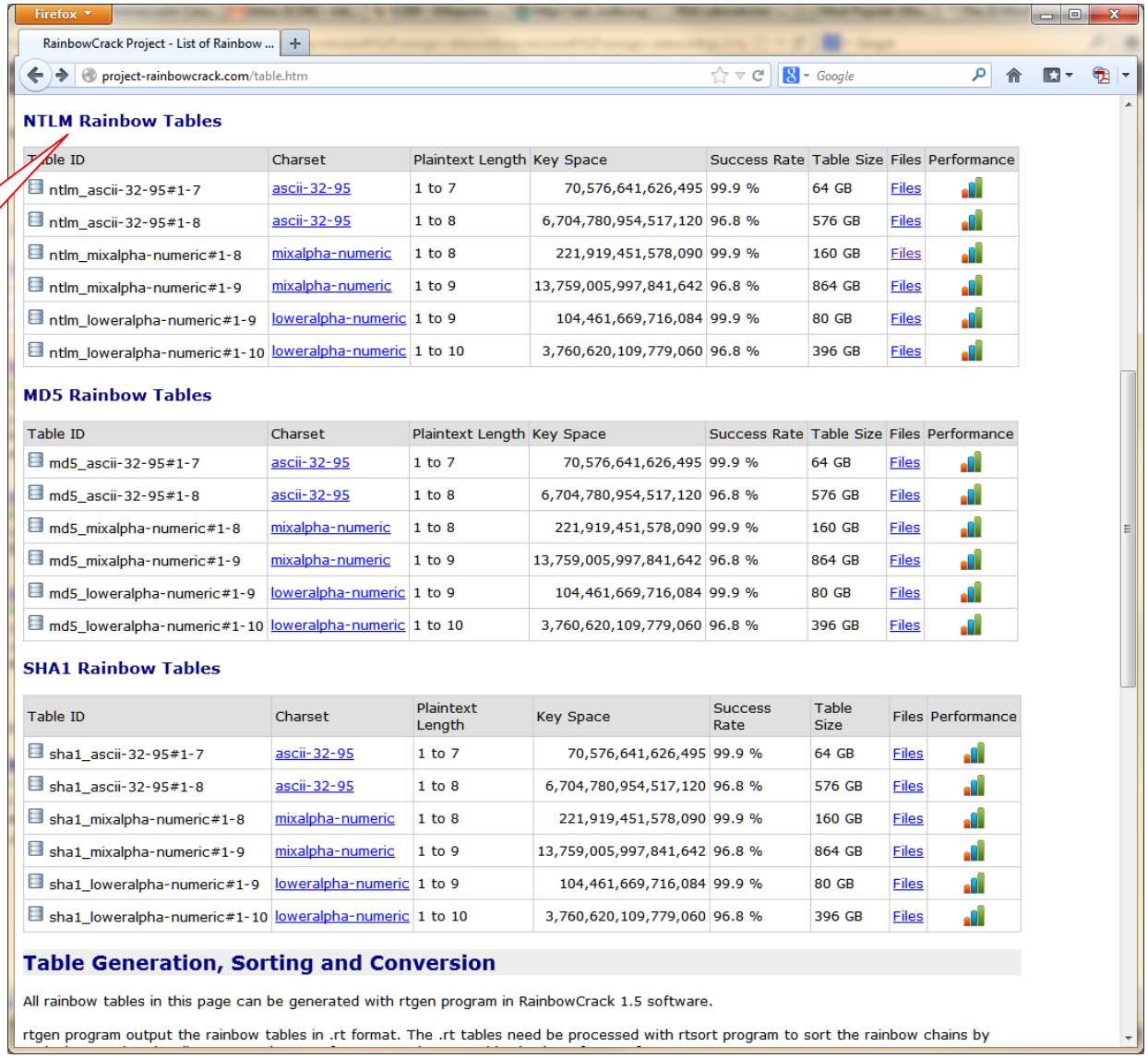
- Password crackers
  - Brute force
  - Dictionary based
- What if the bad guy manages to get the a hashed password *h(p)*
  - Hash the terms in the dictionary, and compare them with *h(p)*
  - Counter measure: we can make the h() function very slow, or hash it many times.
    - If it takes 0.1 second compute hash – doesn't matter in real world applications.
    - However, the adversary could only test 600 passwords in a minute.

Password Authentication

- What if the bad guy pre-computes *h(p)* for all entries in the dictionary?
  - **Rainbow table**
  - Yes, you can purchase from the Internet.

Password Authentication

NTLM: MS's NT Lan Manager



Firefox — RainbowCrack Project - List of Rainbow ...

project-rainbowcrack.com/table.htm

NTLM Rainbow Tables

Table ID	Charset	Plaintext Length	Key Space	Success Rate	Table Size	Files	Performance
ntlm_ascii-32-95#1-7	ascii-32-95	1 to 7	70,576,641,626,495	99.9 %	64 GB	Files	
ntlm_ascii-32-95#1-8	ascii-32-95	1 to 8	6,704,780,954,517,120	96.8 %	576 GB	Files	
ntlm_mixalpha-numeric#1-8	mixalpha-numeric	1 to 8	221,919,451,578,090	99.9 %	160 GB	Files	
ntlm_mixalpha-numeric#1-9	mixalpha-numeric	1 to 9	13,759,005,997,841,642	96.8 %	864 GB	Files	
ntlm_loweralpha-numeric#1-9	loweralpha-numeric	1 to 9	104,461,669,716,084	99.9 %	80 GB	Files	
ntlm_loweralpha-numeric#1-10	loweralpha-numeric	1 to 10	3,760,620,109,779,060	96.8 %	396 GB	Files	

MD5 Rainbow Tables

Table ID	Charset	Plaintext Length	Key Space	Success Rate	Table Size	Files	Performance
md5_ascii-32-95#1-7	ascii-32-95	1 to 7	70,576,641,626,495	99.9 %	64 GB	Files	
md5_ascii-32-95#1-8	ascii-32-95	1 to 8	6,704,780,954,517,120	96.8 %	576 GB	Files	
md5_mixalpha-numeric#1-8	mixalpha-numeric	1 to 8	221,919,451,578,090	99.9 %	160 GB	Files	
md5_mixalpha-numeric#1-9	mixalpha-numeric	1 to 9	13,759,005,997,841,642	96.8 %	864 GB	Files	
md5_loweralpha-numeric#1-9	loweralpha-numeric	1 to 9	104,461,669,716,084	99.9 %	80 GB	Files	
md5_loweralpha-numeric#1-10	loweralpha-numeric	1 to 10	3,760,620,109,779,060	96.8 %	396 GB	Files	

SHA1 Rainbow Tables

Table ID	Charset	Plaintext Length	Key Space	Success Rate	Table Size	Files	Performance
sha1_ascii-32-95#1-7	ascii-32-95	1 to 7	70,576,641,626,495	99.9 %	64 GB	Files	
sha1_ascii-32-95#1-8	ascii-32-95	1 to 8	6,704,780,954,517,120	96.8 %	576 GB	Files	
sha1_mixalpha-numeric#1-8	mixalpha-numeric	1 to 8	221,919,451,578,090	99.9 %	160 GB	Files	
sha1_mixalpha-numeric#1-9	mixalpha-numeric	1 to 9	13,759,005,997,841,642	96.8 %	864 GB	Files	
sha1_loweralpha-numeric#1-9	loweralpha-numeric	1 to 9	104,461,669,716,084	99.9 %	80 GB	Files	
sha1_loweralpha-numeric#1-10	loweralpha-numeric	1 to 10	3,760,620,109,779,060	96.8 %	396 GB	Files	

Table Generation, Sorting and Conversion

All rainbow tables in this page can be generated with rtgen program in RainbowCrack 1.5 software.

rtgen program output the rainbow tables in .rt format. The .rt tables need be processed with rtsort program to sort the rainbow chains by

Password Authentication

- What if the bad guy pre-computes *h(p)* for all entries in the dictionary?
  - Rainbow table
  - Yes, you can purchase from the Internet.
- Counter measure: salt
  - Add randomness to password hash
  - Random data called *salt*

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.		

Password Authentication

- Future of password authentication
  - Graphical passwords?
    - Android: patterns
  - Biometrics?
    - Face recognition
    - Fingerprint
  - Two factor authentication?
    - password + token
    - password + something like SecureID
    - password + biometrics

Distributed Authentication

- In a distributed environment?
  - Key management
  - Secret key: if there are N principals, N^2 shared keys would be needed
  - PKI: if there are N principals, N (public, private) key pairs would be needed
  - A better solution?

Kerberos

- Kerberos
  - Cerberus: in Greek and Roman mythology, is a multi-headed (usually three-headed) dog, or "hellhound" which guards the gates of the Underworld, to prevent those who have crossed the river Styx from ever escaping.

    -- Wiki

- A centralized authentication service
  - Authenticate users to services
  - Authenticate services to users
  - Servers are relieved of the burden of maintaining authentication information

Kerberos

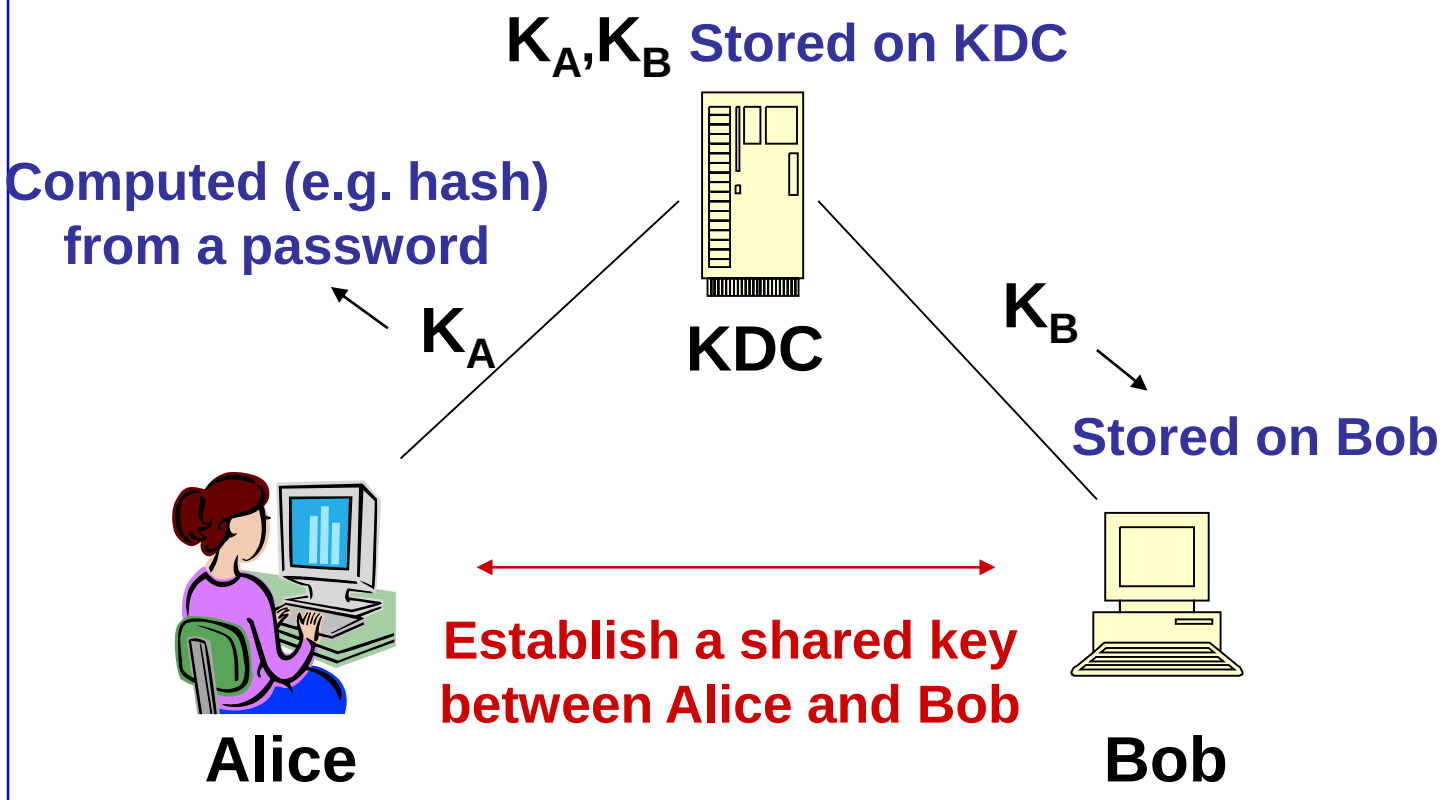
- History
  - Developed as a part of *Project Athena* at MIT
  - Solves: password eavesdropping
  - Online authentication: variant of Needham-Schroeder
  - Easy application integration API
  - First *single sign-on* system
  - Sign-on once, access all services
- Most widely used (non-web) centralized password system in existence
  - Adopted by Windows 2000 and all later versions
  - Also available for Unix/Linux family of OS
  - Again, there are some interesting stories about military adoption and export control…

Kerberos

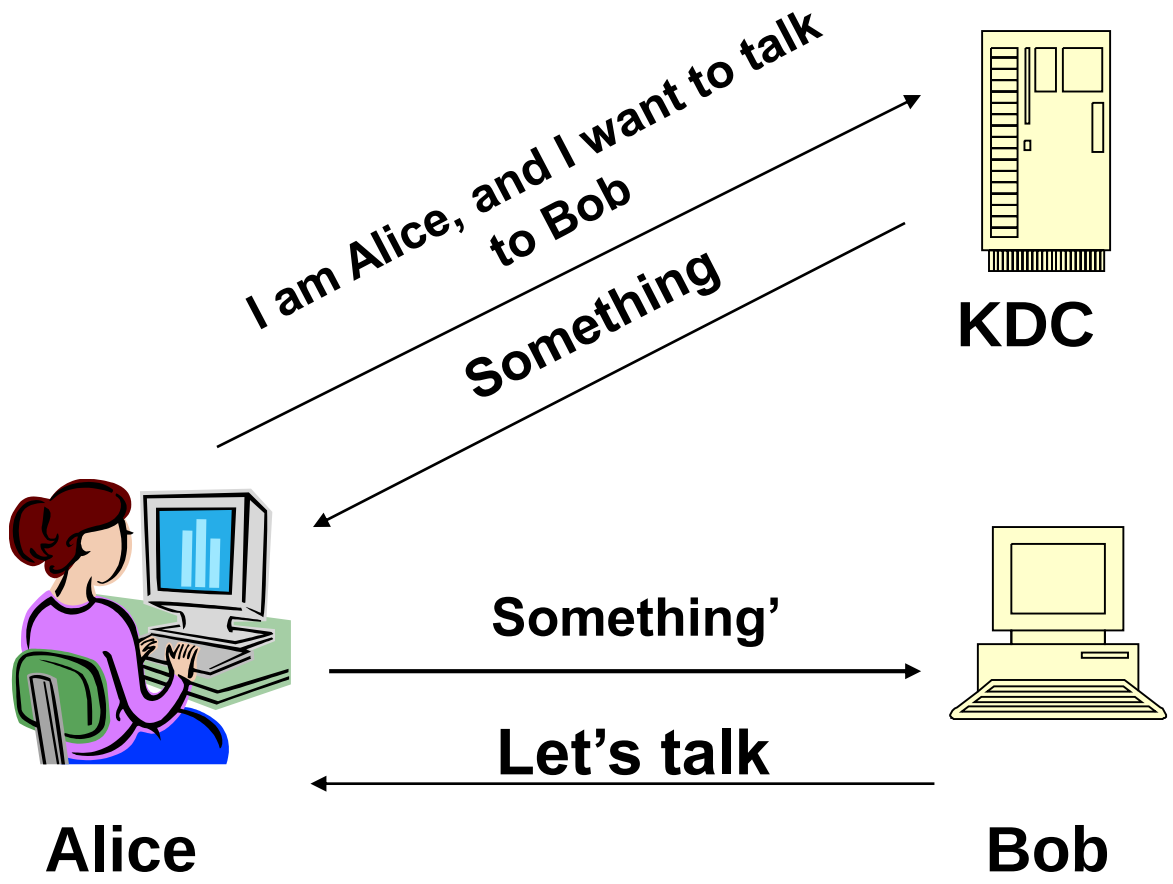
- The design of Kerberos
  - Goal: reduce the amount of information each individual system needs to maintain.
  - Trusted third-party issues certificate
  - Prove that I am the person on the certificate
  - Certificates contain other properties useful for authorization decisions
  - Assumption: the communication channel is insecure.

Kerberos

- The Kerberos architecture
  - KDC: Key Distribution Center
    - K_A: Shared between Alice and KDC



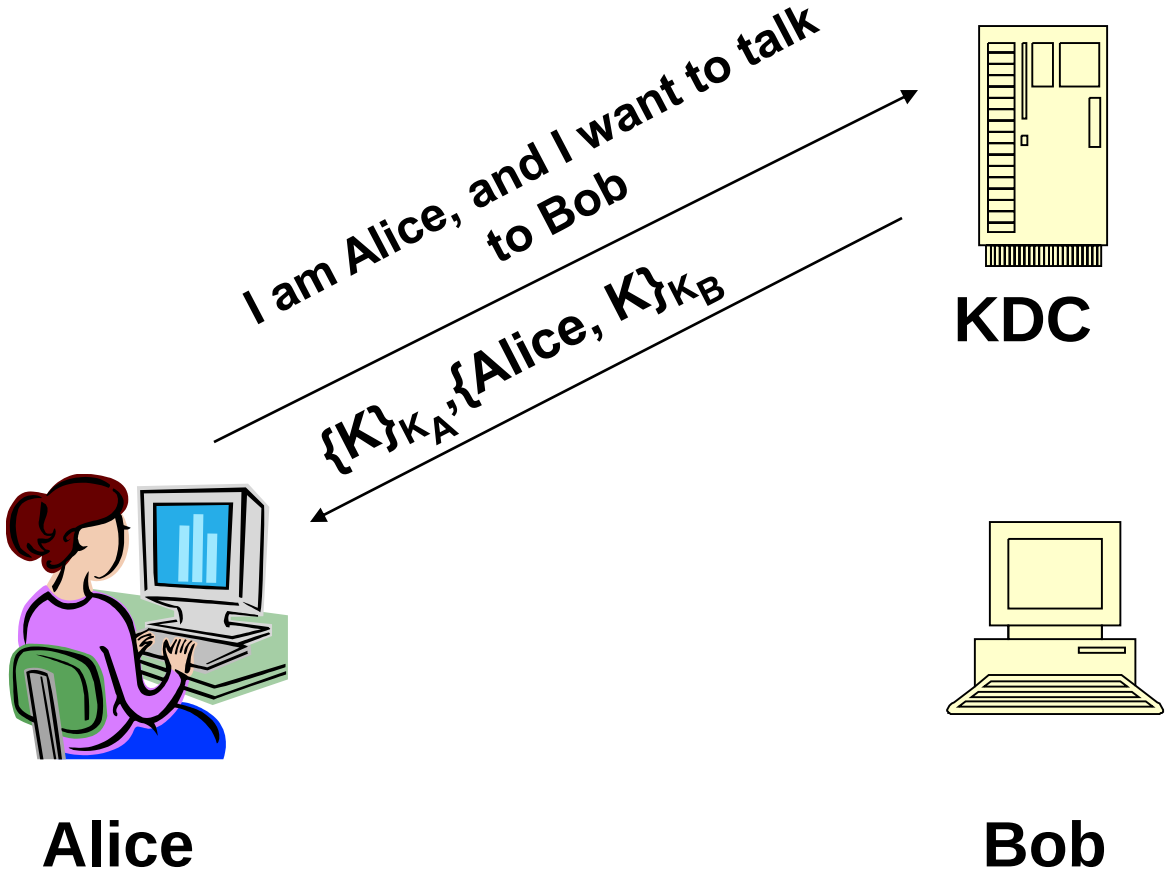
Kerberos



- Q: how to verify Alice? What is the certificate?

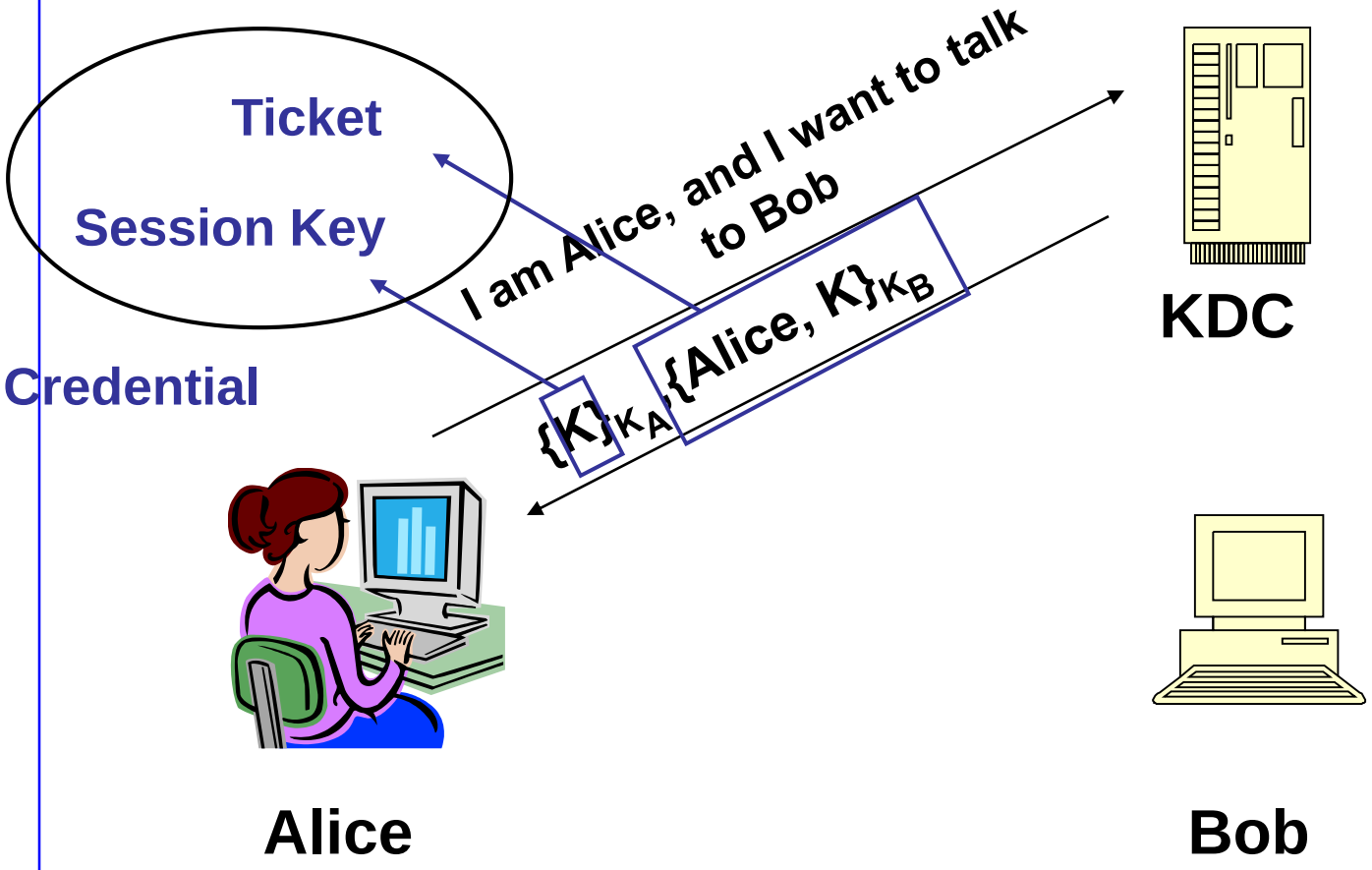
Kerberos

- KDC: only Alice could see K; only Bob could see {Alice, K}.



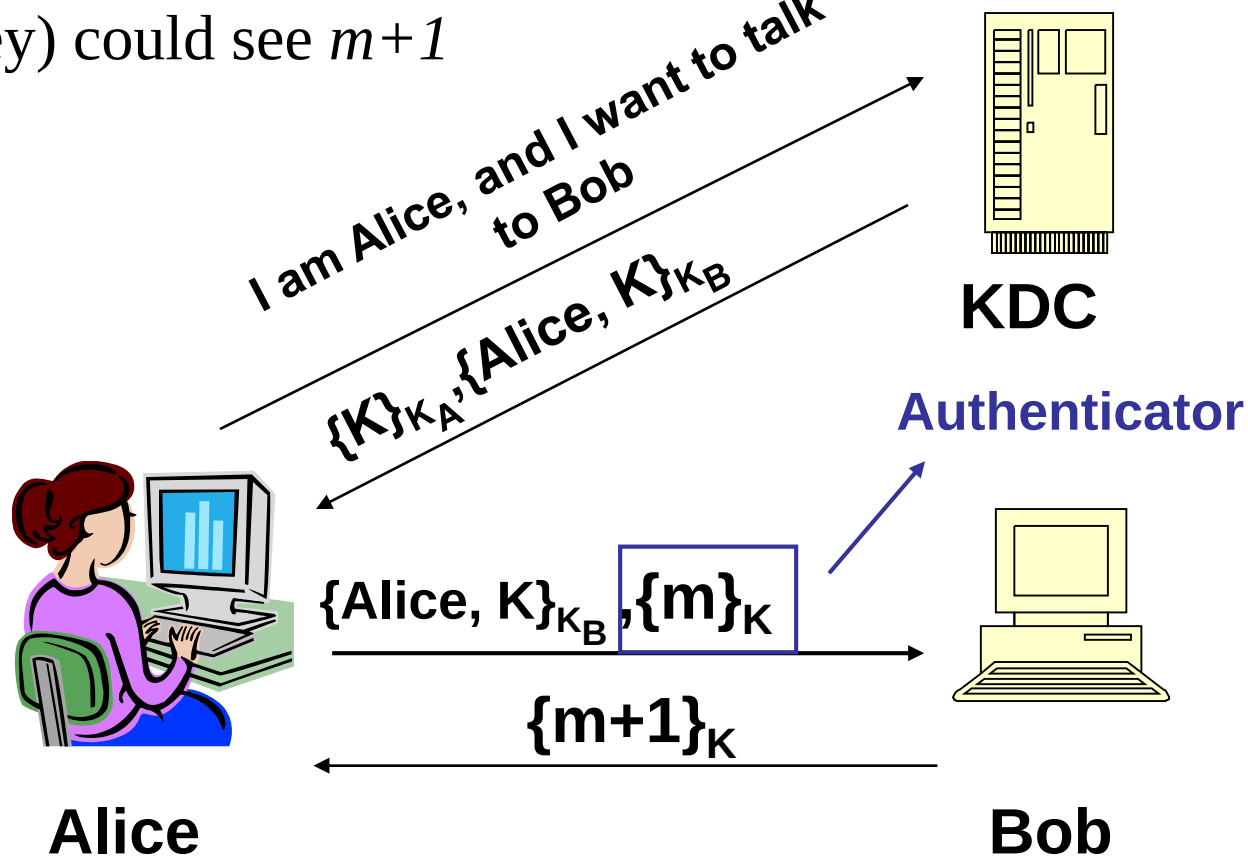
Kerberos

- KDC: only Alice could see the session key; only Bob could see the ticket.



Kerberos

- Alice: only Bob could "understand" the ticket; only Bob could see *m*
- Bob: only Alice (with session key) could see *m+1*

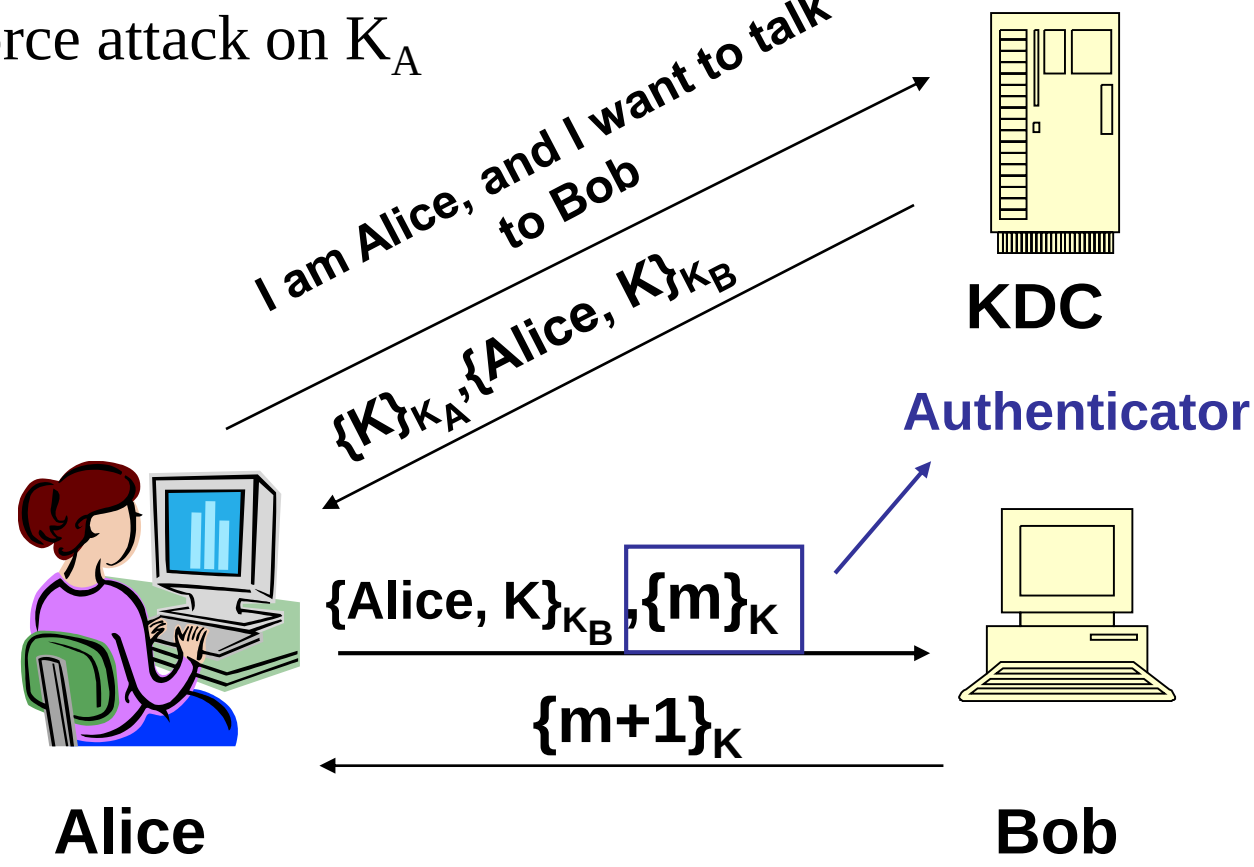


Kerberos

- Design rationales behind Kerberos
  - Less work on servers (KDC and Bob), more work on clients
    - A client requests credentials (ticket + session key) and manage them; KDC and Bob do not maintain states about user credentials (except that it must remember authenticators seen in a time window to avoid replay attacks)
    - More scalable in a large distributed system
  - Less communication overhead
    - Alice sends both the ticket and the authenticator to Bob. Neither Alice nor Bob needs to wait.

Kerberos

- Mallory could claim to be Alice, to get $\{K\}_{K_A}$,$\{Alice, K\}_{K_B}$
- Mallory then launches brute force attack on K_A



Kerberos

- Discussions
  - User Alice in EECS uses a weak password. She logged in from her office to a EECS server using Kerberos.
  - Eve gets Alice's password through social engineering attacks.
  - Later it was discovered that the network communication link was exposed to possible eavesdropping attack.
  - Eve obtains $\{K\}_{K_A}$,$\{Alice, K\}_{K_B}$
  - Alice changes her password to a strong password.
  - What could Eve do?

Kerberos

- Discussions
- A session key can be reused by Alice
  - It really should be called a “multi-session” key
  - Would be disastrous if Alice’s password gets compromised: an attacker can impersonate Alice indefinitely, even after Alice changes her password
  - Kerberos imposes a life-time on a ticket to address this problem

Kerberos

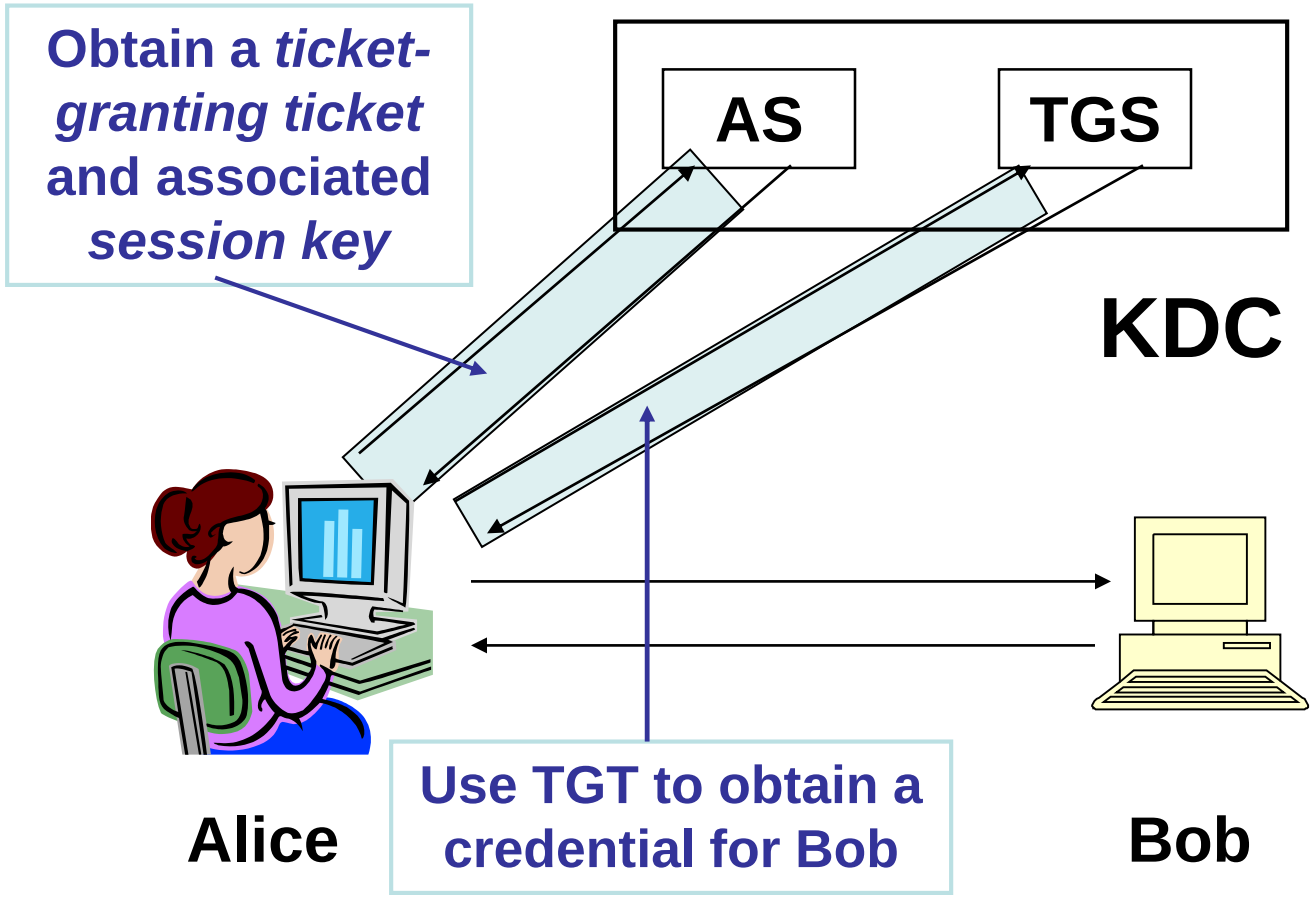
- Discussions
- Short-term credentials
  - Kerberos tickets and session keys are short-term credentials. Unlike the long-term credentials (keys shared with KDC), they can be stored in memory or disk.

Kerberos

- One ticket for one server
  - The ticket is only good between Alice and Bob
  - Alice would have to type in password to obtain a new ticket for a different server
- Single Sign-On (SSO)
  - Ticket-granting Server (TGS)
    - A dedicated server that issues tickets for all other servers
  - User only needs to type in password once to obtain a credential (ticket+session key) for TGS (Ticket-Granting Ticket)
  - Subsequent ticket-granting requests will be conducted through TGS

Kerberos

- Kerberos Single Sign-On (SSO)
  - AS: Authentication server



Kerberos

- Kerberos Single Sign-On (SSO)
  - AS: Authentication server
  - TGS Session Key: $K_{S\text{-}TGS}$
  - Sending $K_{S\text{-}TGS}$ to Alice: $\{K_{S\text{-}TGS}\}_{K_A}$
  - Ticket Granting Ticket (TGT): $\{\text{Alice}, K_{S\text{-}TGS}\}_{K_{TGS}}$

Kerberos

- Discussions
  - EECS uses a single sign-on Kerberos system.
  - User Alice in EECS uses a weak password. She logged in to AS to obtain a ticket-granting ticket.
  - Eve gets Alice's password.
  - Alice uses the TGT to contact the Ticket Granting Server, to obtain a ticket to server X.
  - Later it was discovered that the network communication link between Alice and TGS was exposed to possible eavesdropping attack.
  - Was Alice's password endangered because of this?
  - Eve obtained the TGT. Could Eve use the ticket to impersonate Alice to EECS servers?

Wrap-up

- What we have covered?
  - Cryptography
    - Basic concepts
    - Classic Cryptography
    - Modern symmetric cryptography
    - Asymmetric cryptography
    - Cryptographic hash
  - Authentication
    - Password-based authentication
    - Authentication in a distributed system
    - The Public Key Infrastructure

Pure topic == Basic