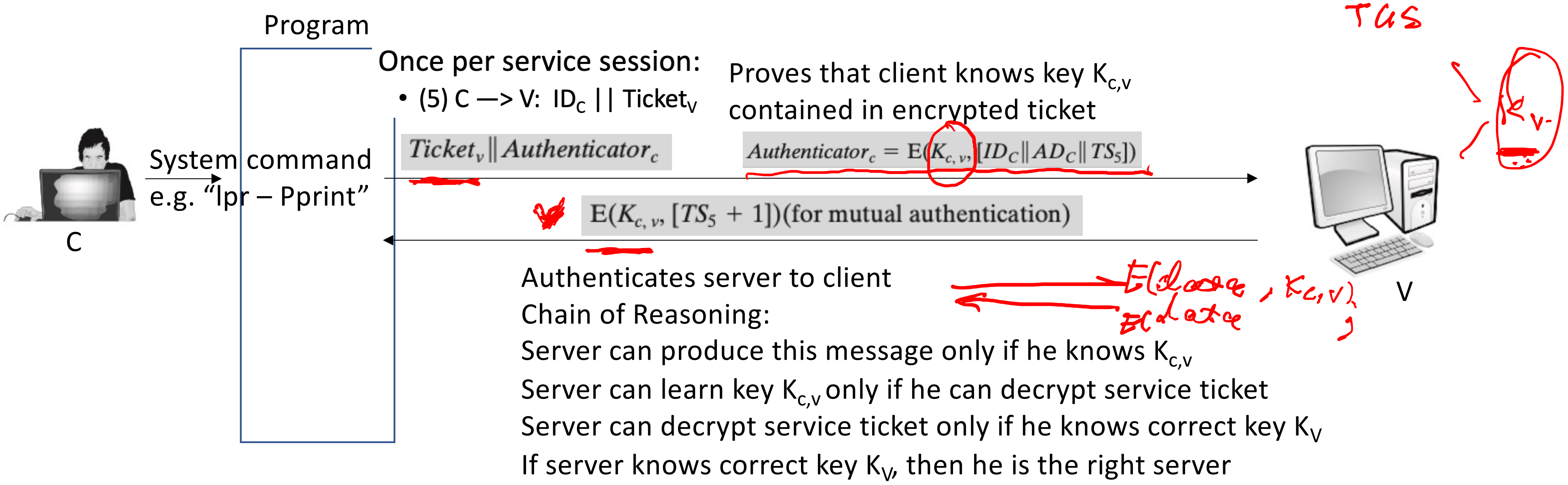
Kerberos v4. - once per service session



Once per service session:

- (5) C —> V: ID_C || $Ticket_V$

$Ticket_v \| Authenticator_c$

Proves that client knows key $K_{c,v}$ contained in encrypted ticket

$Authenticator_c = E(K_{c,v}, [ID_C \| AD_C \| TS_5])$

$E(K_{c,v}, [TS_5 + 1])$(for mutual authentication)

Authenticates server to client

Chain of Reasoning:

Server can produce this message only if he knows $K_{c,v}$

Server can learn key $K_{c,v}$ only if he can decrypt service ticket

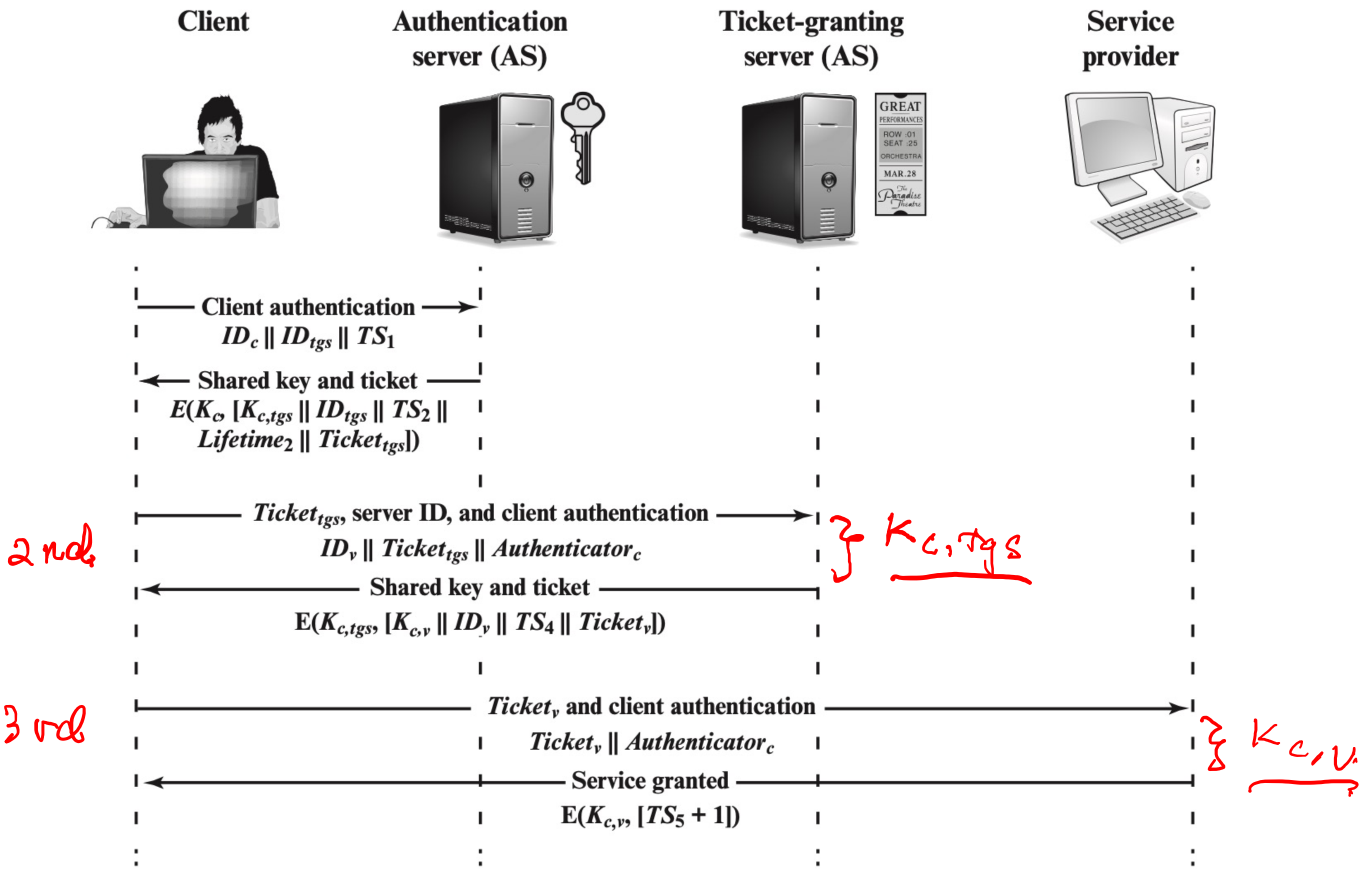
Server can decrypt service ticket only if he knows correct key K_V

If server knows correct key K_V, then he is the right server

For each service request, client uses the short-term key, $K_{c,v}$, for that service and the ticket he received from TGS

$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| \text{Lifetime}_4])$

Overview of Kerberos



Client authentication

$ID_c \,||\, ID_{tgs} \,||\, TS_1$

Shared key and ticket

$E(K_c, [K_{c,tgs} \,||\, ID_{tgs} \,||\, TS_2 \,||\, Lifetime_2 \,||\, Ticket_{tgs}])$

2nd

$Ticket_{tgs}$, server ID, and client authentication

$ID_v \,||\, Ticket_{tgs} \,||\, Authenticator_c$

} $\underline{K_{c,tgs}}$

Shared key and ticket

$E(K_{c,tgs}, [K_{c,v} \,||\, ID_v \,||\, TS_4 \,||\, Ticket_v])$

3rd

$Ticket_v$ and client authentication

$Ticket_v \,||\, Authenticator_c$

} $\underline{K_{c,v}}$

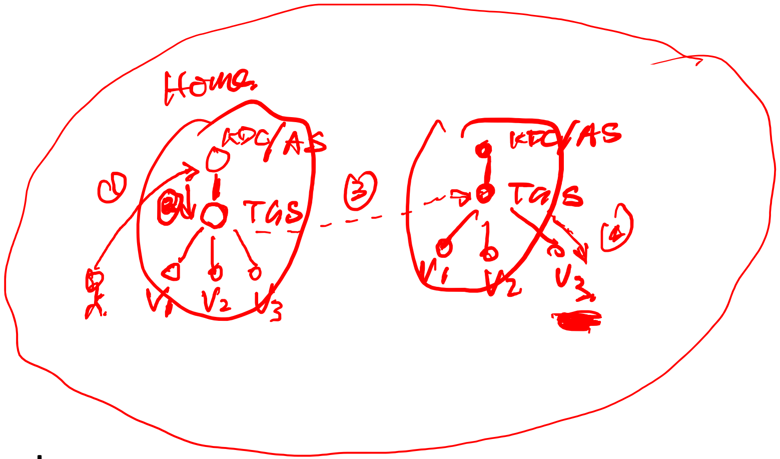
Service granted

$E(K_{c,v}, [TS_5 + 1])$

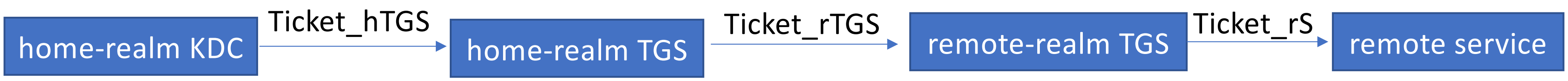
Important Ideas in Kerberos

- Short-term session keys
  - Long-term secrets used only to derive short-term keys
  - Separate session key for each user-server pair
    - Re-used by multiple sessions between same user and server
- Proofs of identity based on authenticators
  - Client encrypts his identity, addr, time with session key; knowledge of key proves client has authenticated to KDC/AS Session key
    - Also prevents replays (if clocks are globally synchronized) time stamp
  - Server learns this key separately (via encrypted ticket that client can't decrypt), then verifies client's ticket authenticator
- Symmetric cryptography only

Kerberos in Large Networks



- One KDC isn't enough for large networks
- Network is divided into realms
  - KDCs in different realms have different key databases
- To access a service in another realm, users must...
  - Get ticket for home-realm TGS from home-realm KDC
  - Get ticket for remote-realm TGS from home-realm TGS
    - As if remote-realm TGS were just another network service
  - Get ticket for remote service from that realm's TGS
  - Use remote-realm ticket to access service



home-realm KDC → Ticket_hTGS → home-realm TGS → Ticket_rTGS → remote-realm TGS → Ticket_rS → remote service

Practical Uses of Kerberos

- Microsoft Windows – Active Directory
- Email, FTP, network file systems, many other applications have been kerberized
  - Use of Kerberos is transparent for the end user
  - Transparency is important for usability!
- Local authentication login_krb5
  - login and su in OpenBSD
- Authentication for network protocols
  - rsh → ssh username @ IP address
- Secure windowing systems → X11 Linux

Readings

- Kerberos: The Network Authentication Protocol
  https://web.mit.edu/kerberos/

Practice – no submission

- William Stallings, “Network Security Essentials”, 6 Edition, 2017
  - Chapter 4’s problems: 4.8, 4.9, 4.10

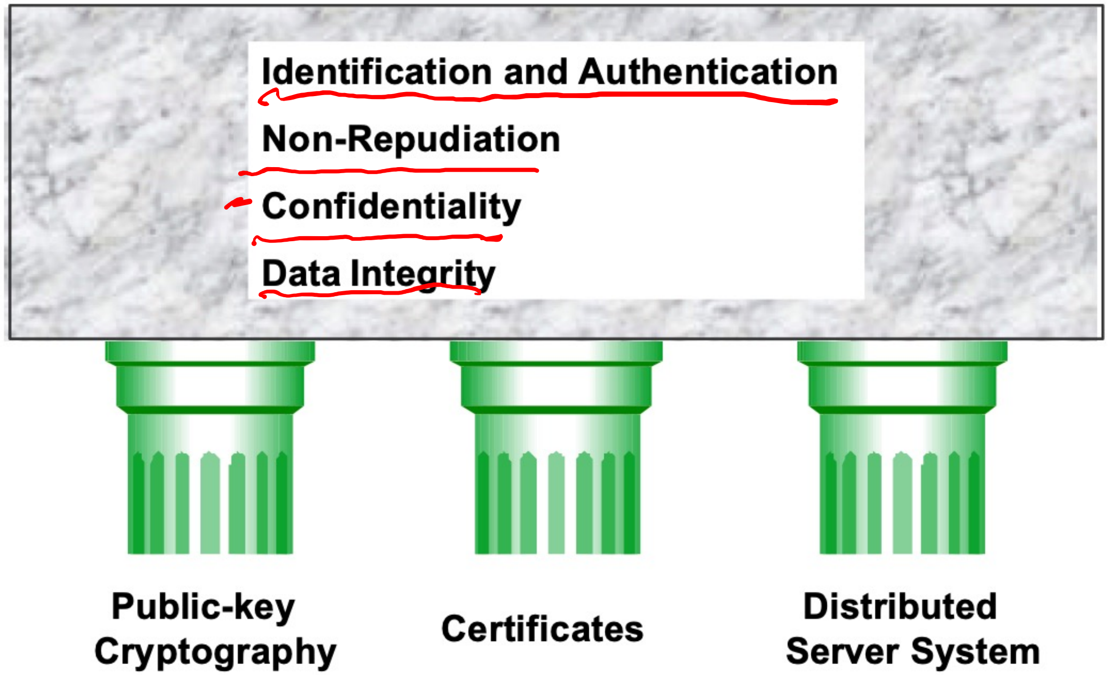
Key Distribution Using Asymmetric Encryption

PKI and Certificates

(Section 4.5)

What is PKI?

- Use of public-key cryptography and X.509 certificates in a distributed server system to establish secure domains and trusted relationships



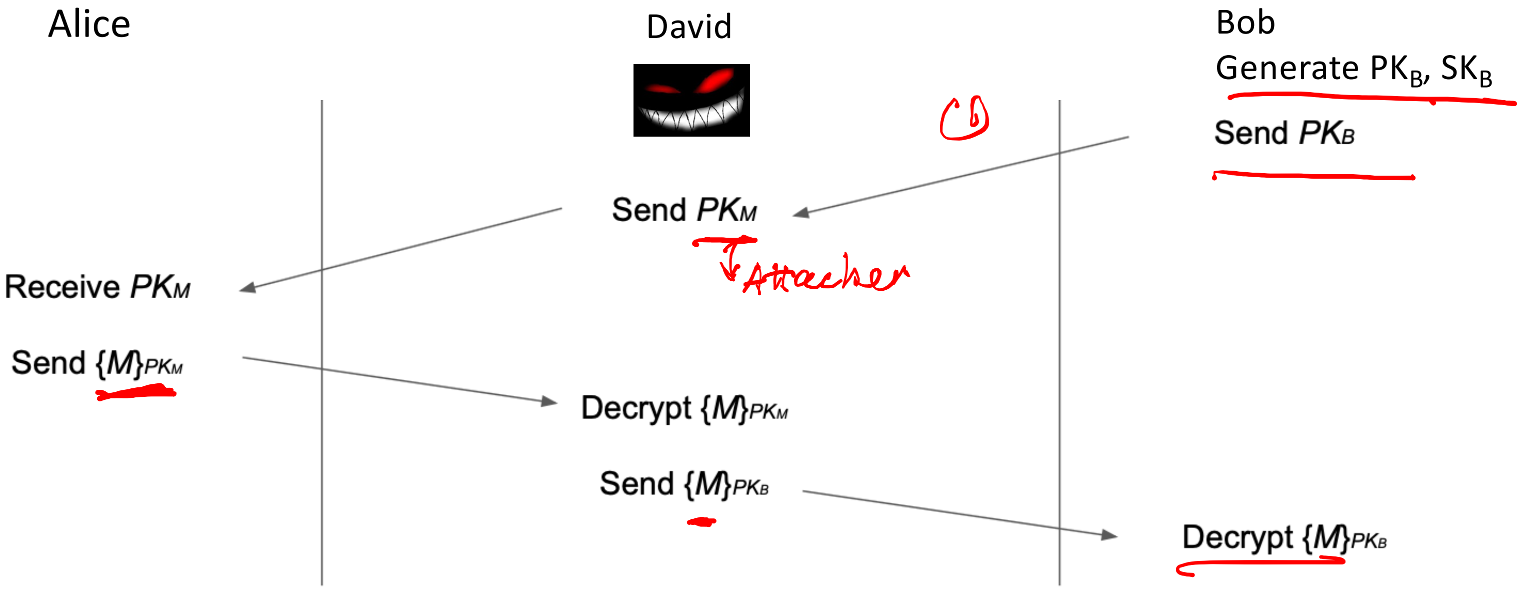
Why use public-key cryptography?

- Review: Public-key cryptography is great! We can communicate securely without a shared secret
  - Public-key encryption: Everybody encrypts with the public key, but only the owner of the private key can decrypt
  - Digital signatures: Only the owner of the private key can sign, but everybody can verify with the public key

Problem: Distributing Public Keys

- Public-key cryptography alone is not secure against man-in-the-middle attacks
- Scenario
  - Alice wants to send a message to Bob
  - Alice asks Bob for his public key
  - Bob sends his public key to Alice
  - Alice encrypts her message with Bob’s public key and sends it to Bob
- What can David do?
  - Replace Bob's public key with David’s public key
  - Now Alice has encrypted the message with David’s public key, and David can read it!

Problem: Distributing Public Keys



Alice

David

Bob

Generate PK_B, SK_B

Send PK_B

Send PK_M

Attacker

Receive PK_M

Send $\{M\}_{PK_M}$

Decrypt $\{M\}_{PK_M}$

Send $\{M\}_{PK_B}$

Decrypt $\{M\}_{PK_B}$

Man-in-the-Middle Attack

Solution: Distributing Public Keys

- Idea: Sign Bob’s public key to prevent tampering
- Problem
  - If Bob signs his public key, we need his public key to verify the signature
  - But Bob’s public key is what we were trying to verify in the first place!
  - Circular problem: Alice can never trust any public key she receives
- You cannot gain trust if you trust nothing. You need a root of trust!
  - **Trust anchor**: Someone that we implicitly trust
  - From our trust anchor, we can begin to trust others

Trust-on-First-Use

- **Trust-on-first-use**: The first time you communicate, trust the public key that is used and warn the user if it changes in the future
  - Used in SSH and a couple other protocols
  - Idea: Attacks aren’t frequent, so assume that you aren’t being attacked the first time communicate

Certificates