TRUST

EN.600.424

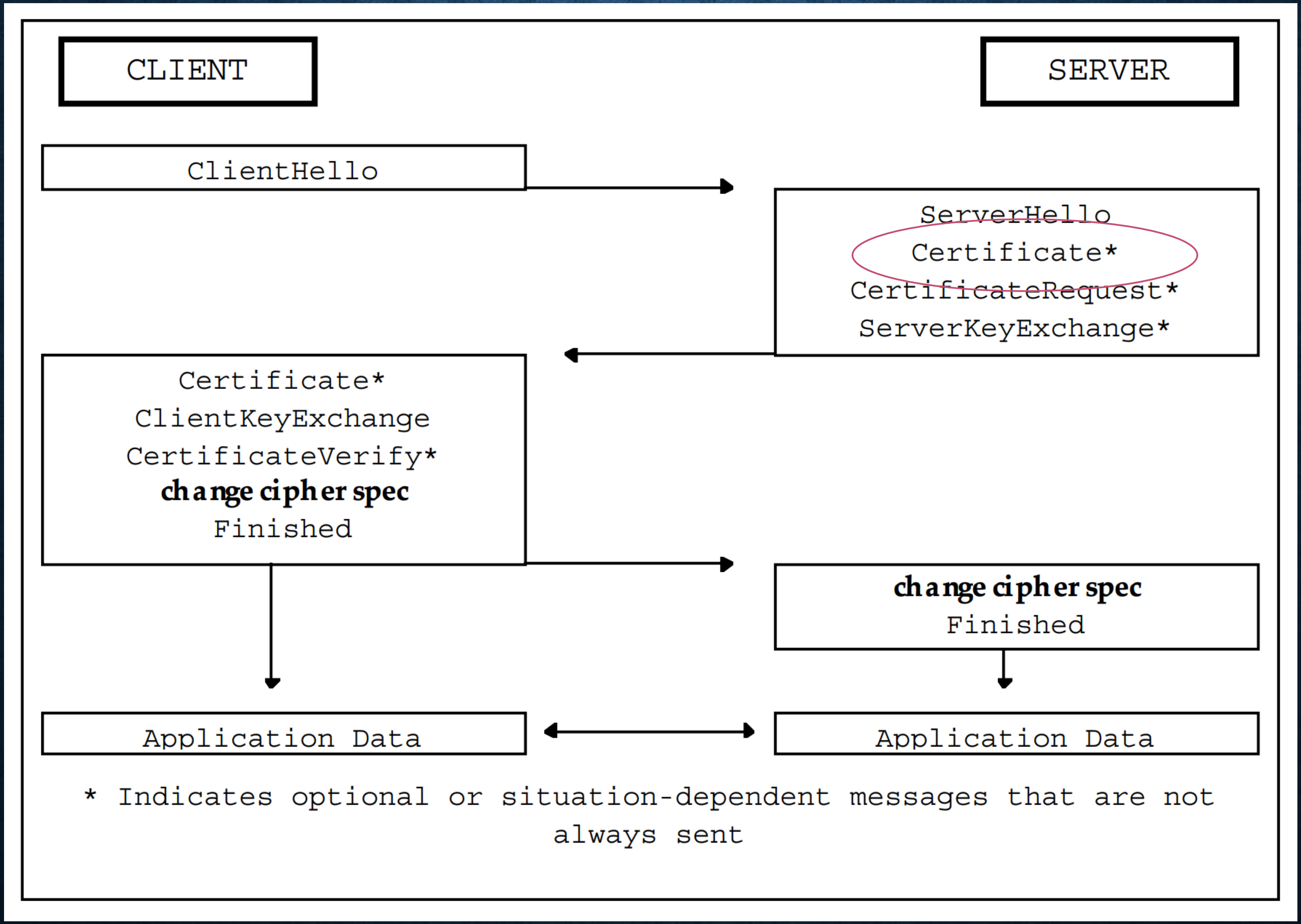
Fall 2018

Lecture Notes

WEAKNESS OF TLS

- TLS is worthless without trust
- If you go to https://yourbank.com, it better be "your bank"
- How do you know it is your bank?

REVIEW TLS HANSHAKS



CLIENT

SERVER

ClientHello

ServerHello
Certificate*
CertificateRequest*
ServerKeyExchange*

Certificate*
ClientKeyExchange
CertificateVerify*
change cipher spec
Finished

change cipher spec
Finished

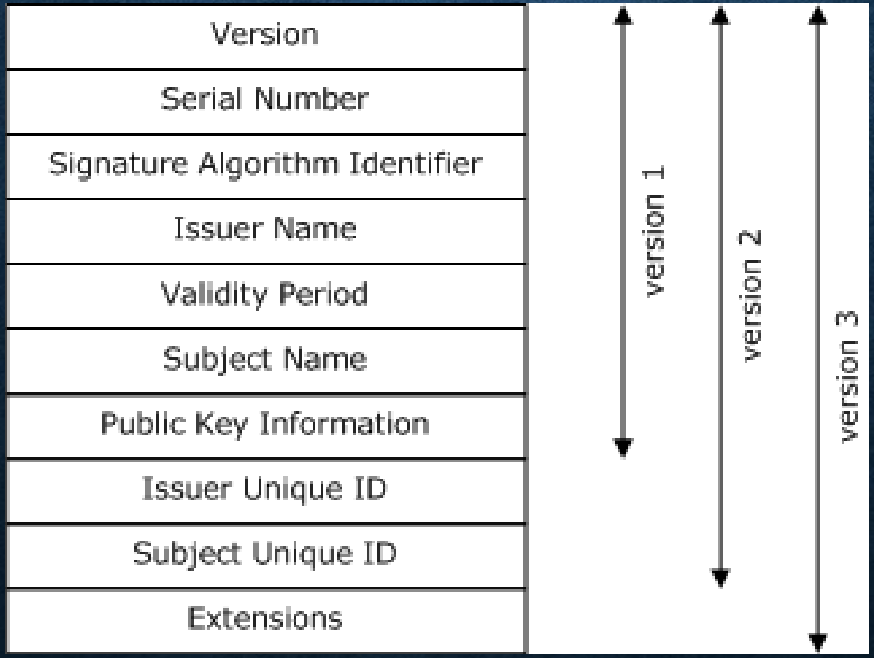
Application Data

Application Data

* Indicates optional or situation-dependent messages that are not always sent

WHAT IS A CERTIFICATE?

- TLS specification (RFC) doesn’t specify cert or cert verification
- The most common is X 509



Version

Serial Number

Signature Algorithm Identifier

Issuer Name

Validity Period

Subject Name

Public Key Information

Issuer Unique ID

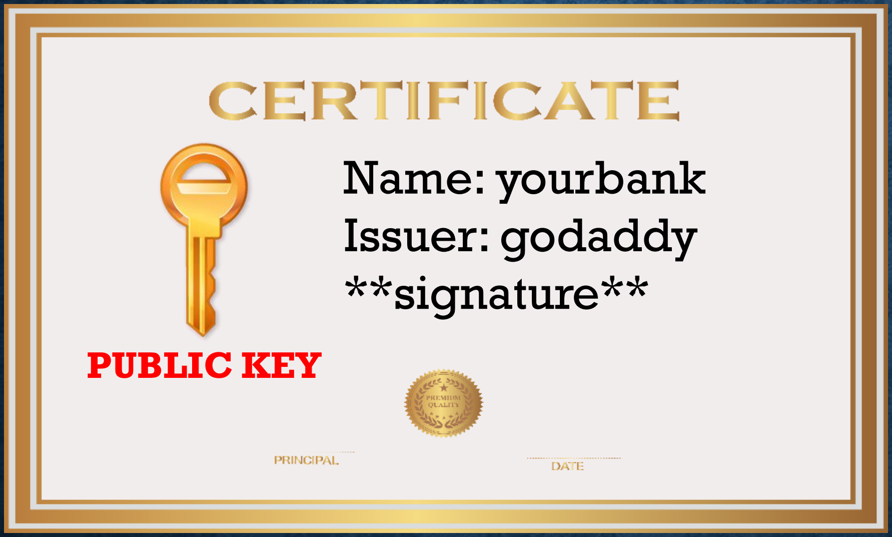
Subject Unique ID

Extensions

KEY ELEMENTS

- Identifying Data
  - Names
  - Serial Number
- Chain Data
  - Who signed the certificate
- Public key
- Signature

PUBLIC KEY PRIVATE KEY



CERTIFICATE

Name: yourbank
Issuer: godaddy
signature

PUBLIC KEY

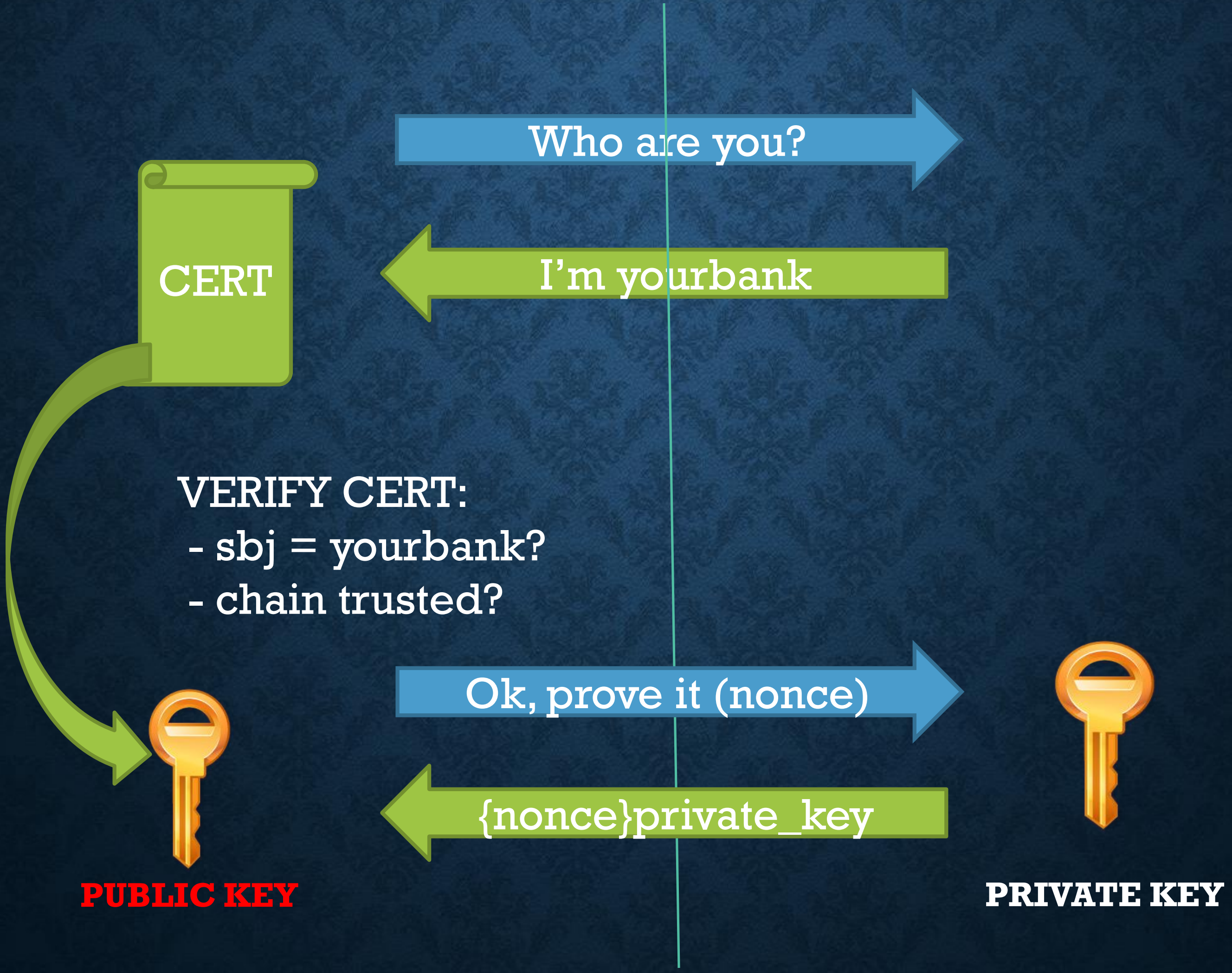
PREMIUM QUALITY

PRINCIPAL

DATE

PRIVATE KEY

PROVING IDENTITY



Who are you?

CERT

I’m yourbank

VERIFY CERT:
- sbj = yourbank?
- chain trusted?

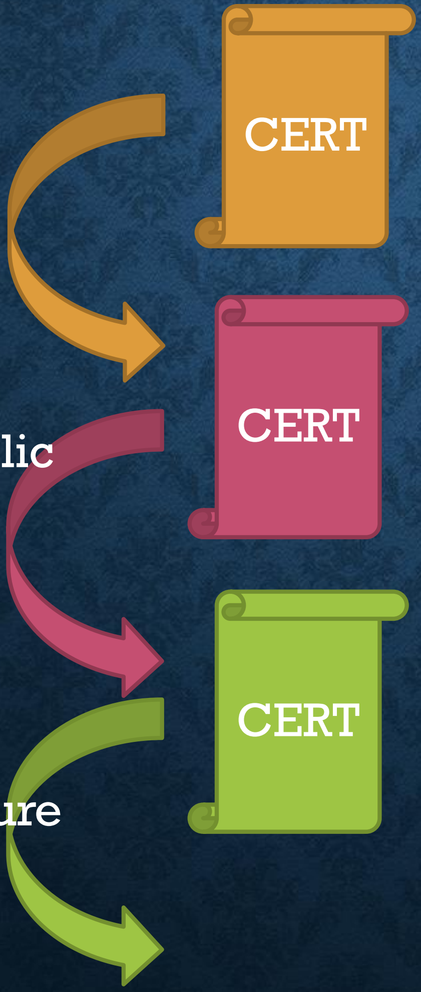
Ok, prove it (nonce)

{nonce}private_key

PUBLIC KEY

PRIVATE KEY

CHAIN OF TRUST



Godaddy public
key verifies cert

Name: godaddy
Signed by godaddy

Banking assoc public
key verifies cert

Name: banking_association
Signed by godaddy

Yourbank public
key verifies signature
on nonce

Name: Yourbank.com
Signed by banking_association

ROOT CERT

- The Root certificate is *self signed*
- It is signed by its own key!
- You have to “trust” somebody *axiomatically!*

CERT CHAIN VERIFICATION (INPUTS)

- The certificate chain
- The current date/time
- Policy information
- Root certificates

CERT CHAIN VERIFICATION (ALGORITHM)

- PKI parameters/algorithms
- Validity of the certificate (time/expiration)
- Revocation status (OCSP, CRL, etc)
- Issuer name matches next subject in path
- Policy checks
- Any intermediate certs are ***CA CERTS!***

CHECKING CA CERTS

- A CA (Certificate Authority) cert should be marked
- Otherwise, you can do this:
  - ROOT
  - Signs, intermediate CA
  - Signs subject (e.g., “yourbank.com”) ***USED AS CA!!!!***
  - Signs fake subject (e.g., “wrongbank.com”)

CERTIFICATE REVOCATION

- How do you revoke a certificate?
- Difficult: so long as the cert is properly signed, it is believed
- You can publish certificate revocation lists:
  - Uses just serial number
  - So make sure your serial numbers are actually unique!
  - But, until the new CRL is received, bad cert still usable

ONLINE CERTIFICATE STATUS PROTOCOL (OCSP)

- Certificates were designed to be used offline
- However, modern security constraints often necessitate OCSP
- Client can ask a server ('OCSP Responder') about a cert
  - Server can respond "Good", "Revoked", "Unknown"
  - Response is signed; however, ***vulnerable to replay attacks!***
  - An extension permits nonces, but often not used for efficiency
  - Also, potential privacy losss
  - But, more efficient and timely than CRL

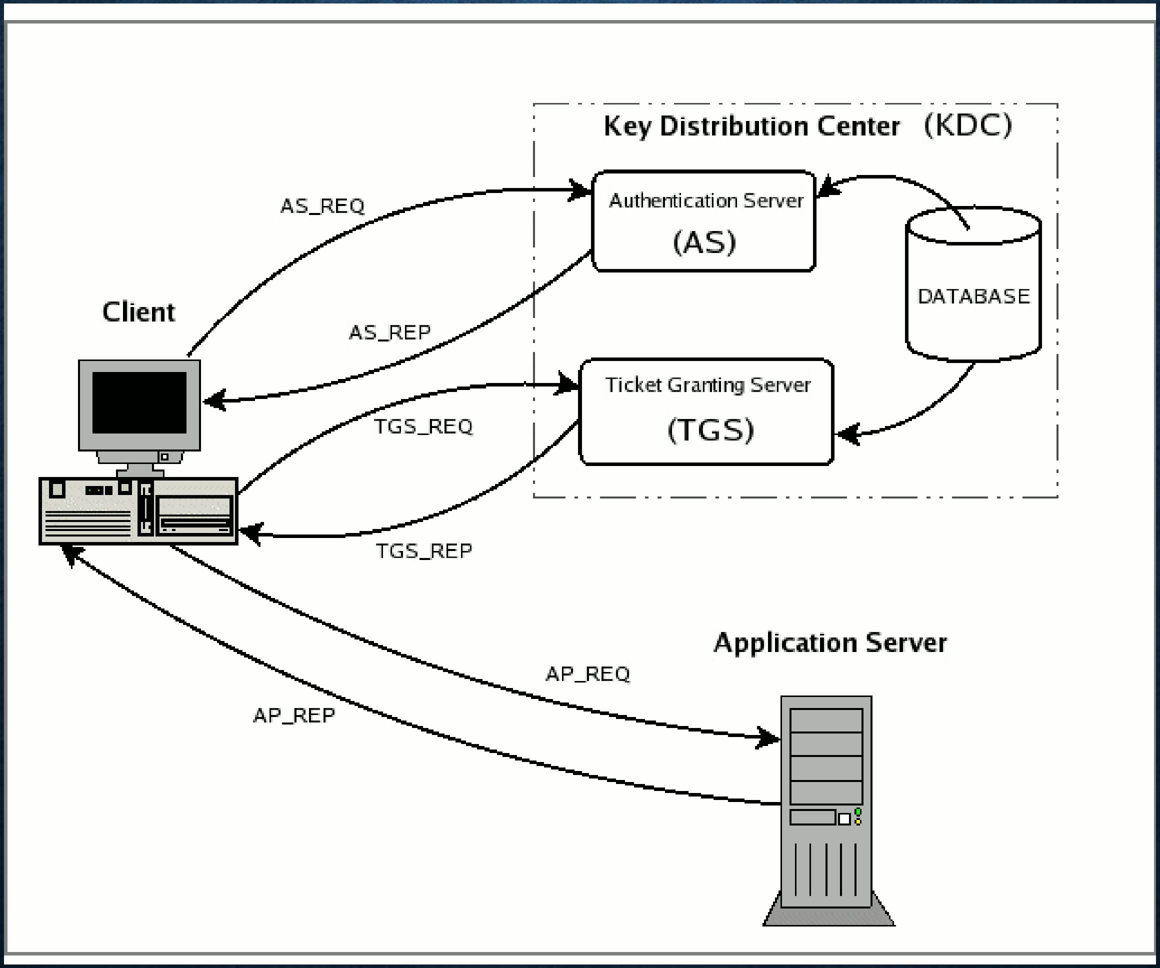
OTHER ALTERNATIVES TO TRUST?

- Sadly, there is no known way to create trust out of thin air
- In almost every case, there must be a trust basis:
  - Out-of-band communication (e.g., in real life)
  - Evolutionary trust over time with long-term identifiers
  - Third parties, including CA’s, authentication/reputation servers
  - Crowds, such as distributed ledger

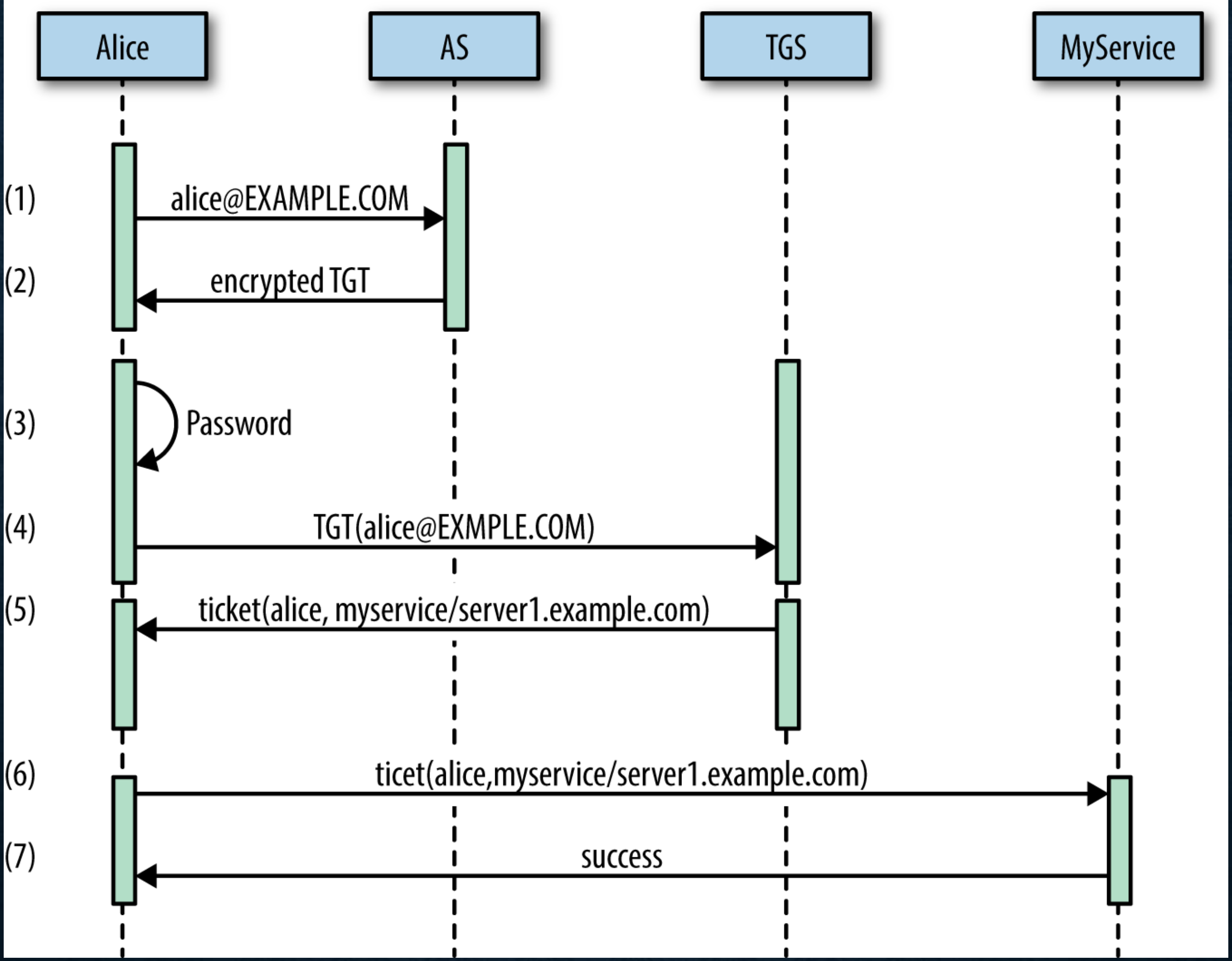
KERBEROS

- Kerberos vs TLS
  - Kerberos uses a trusted authentication server
  - Must be online. And, if compromised, entire system compromised
  - Provides mutual authentication, confidentiality, etc
- Basic components:
  - Authentication Server
  - Key Distribution Server (KDS)
  - Ticket Granting Service
  - Service Server

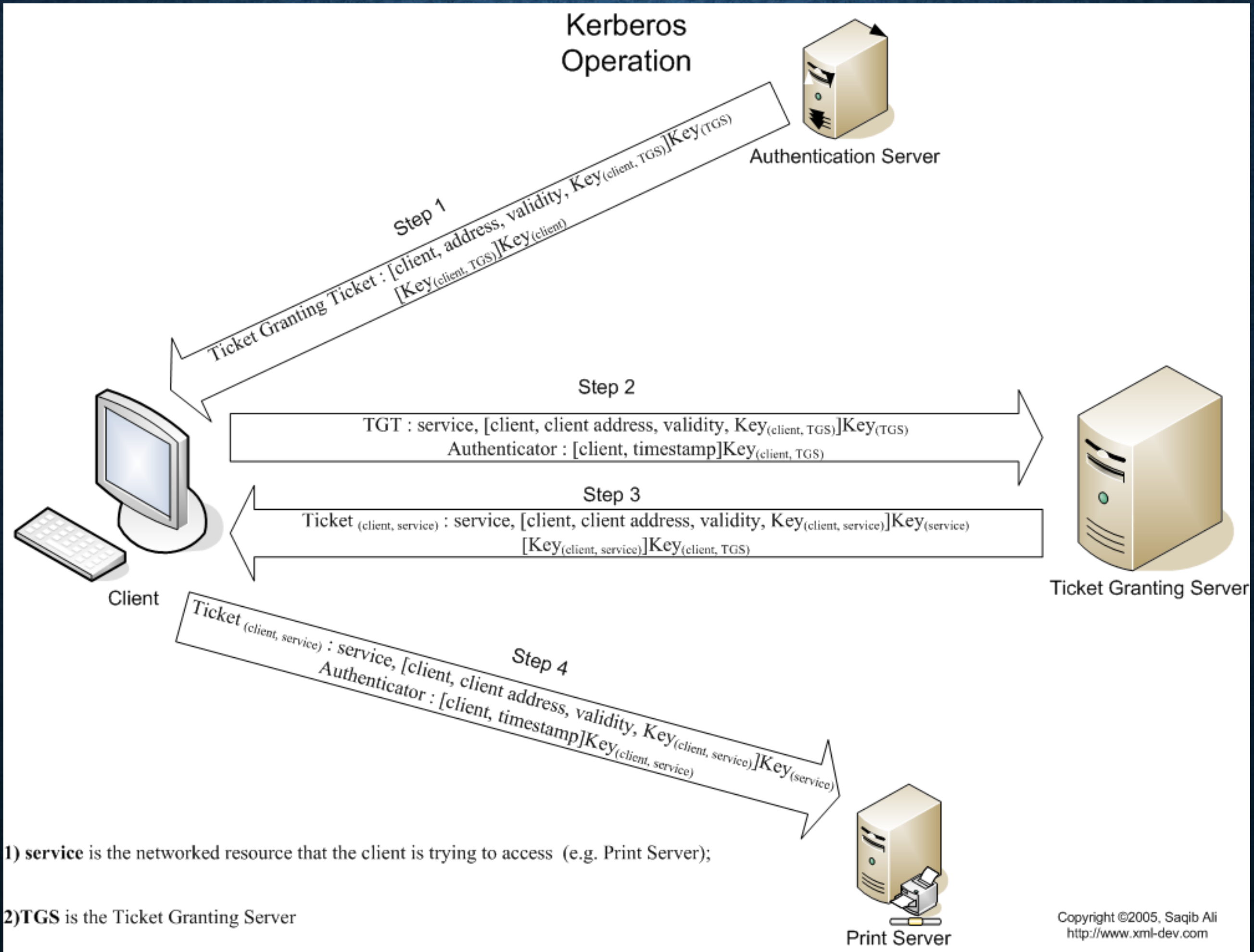
KERBEROS COMMUNICATION



KERBEROS PROTOCOL



KERBEROS PROTOCOL X2


Kerberos Operation
Authentication Server
Step 1
Ticket Granting Ticket : [client, address, validity, Key$_{(\text{client, TGS})}$]Key$_{(\text{TGS})}$
[Key$_{(\text{client, TGS})}$]Key$_{(\text{client})}$
Step 2
TGT : service, [client, client address, validity, Key$_{(\text{client, TGS})}$]Key$_{(\text{TGS})}$
Authenticator : [client, timestamp]Key$_{(\text{client, TGS})}$
Step 3
Ticket $_{(\text{client, service})}$: service, [client, client address, validity, Key$_{(\text{client, service})}$]Key$_{(\text{service})}$
[Key$_{(\text{client, service})}$]Key$_{(\text{client, TGS})}$
Client
Ticket Granting Server
Step 4
Ticket $_{(\text{client, service})}$: service, [client, client address, validity, Key$_{(\text{client, service})}$]Key$_{(\text{service})}$
Authenticator : [client, timestamp]Key$_{(\text{client, service})}$
Print Server
Copyright ©2005, Saqib Ali
http://www.xml-dev.com

1) **service** is the networked resource that the client is trying to access (e.g. Print Server);

2)**TGS** is the Ticket Granting Server

PROTOCOL PRINCIPLES

- Note that the user’s key never goes over the wire
- Note that pre-encrypted messages can be sent.
  - AS sends a message to A that only TGT can decrypt
  - Thus, TGT knows that the message sent by A MUST come from AS
- How scalable is this system?