TLS, CERTS, KERBEROS

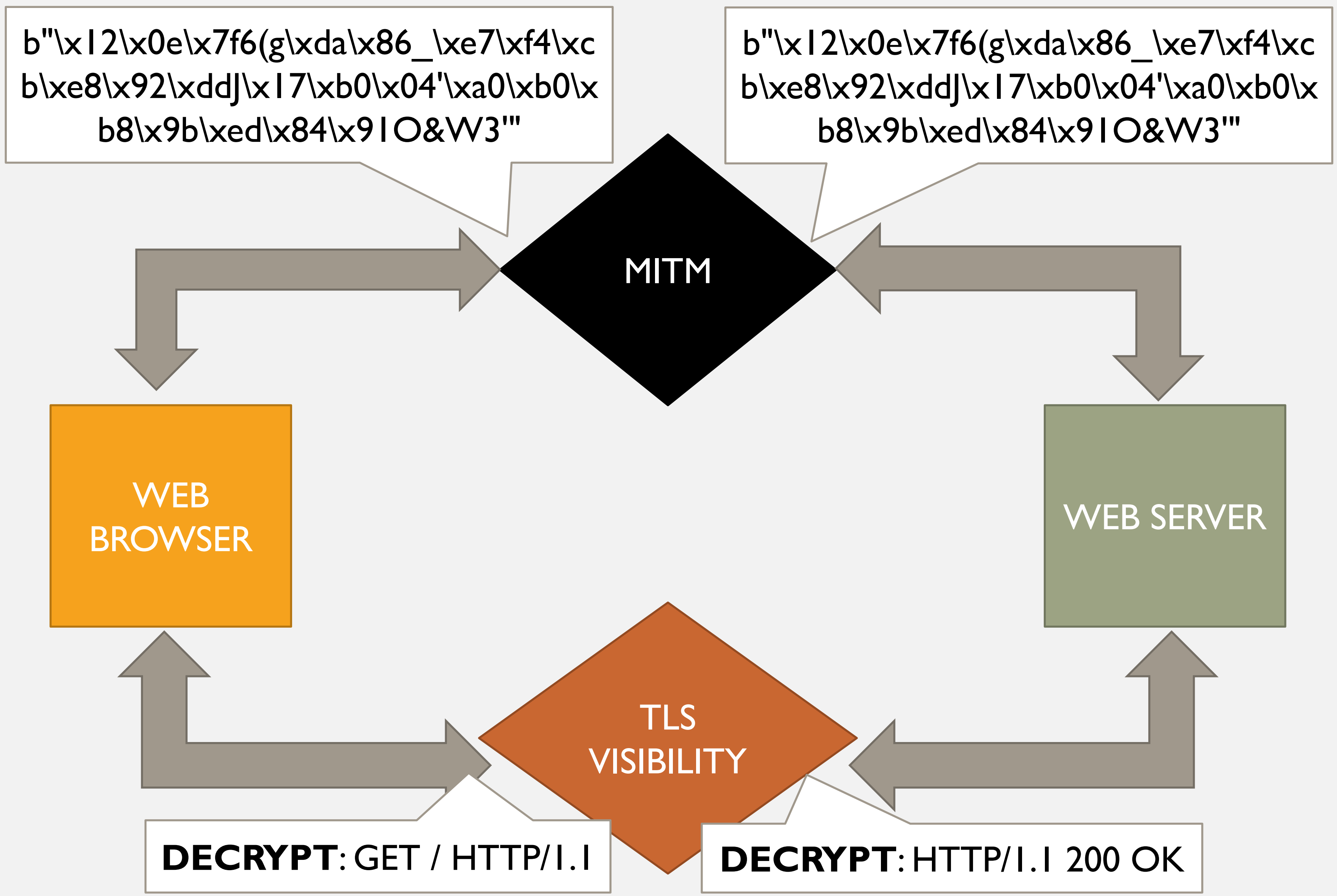
CS 361S

Fall 2020

Seth James Nielson

TLS VISIBILITY

- TLS is designed to provide END-TO-END “security”
- MITM should NOT be able to read/modify/forge data
- TLS Visibility “breaks” this for “authorized” purposes



b"\x12\x0e\x7f6(g\xda\x86_\xe7\xf4\xcb\xe8\x92\xddJ\x17\xb0\x04'\xa0\xb0\xb8\x9b\xed\x84\x91O&W3'"
b"\x12\x0e\x7f6(g\xda\x86_\xe7\xf4\xcb\xe8\x92\xddJ\x17\xb0\x04'\xa0\xb0\xb8\x9b\xed\x84\x91O&W3'"
MITM
WEB BROWSER
WEB SERVER
TLS VISIBILITY
DECRYPT: GET / HTTP/1.1
DECRYPT: HTTP/1.1 200 OK

TLS 1.2 HANDSHAKE

Goal: Agree on a cipher suite for encryption, authentication, etc.

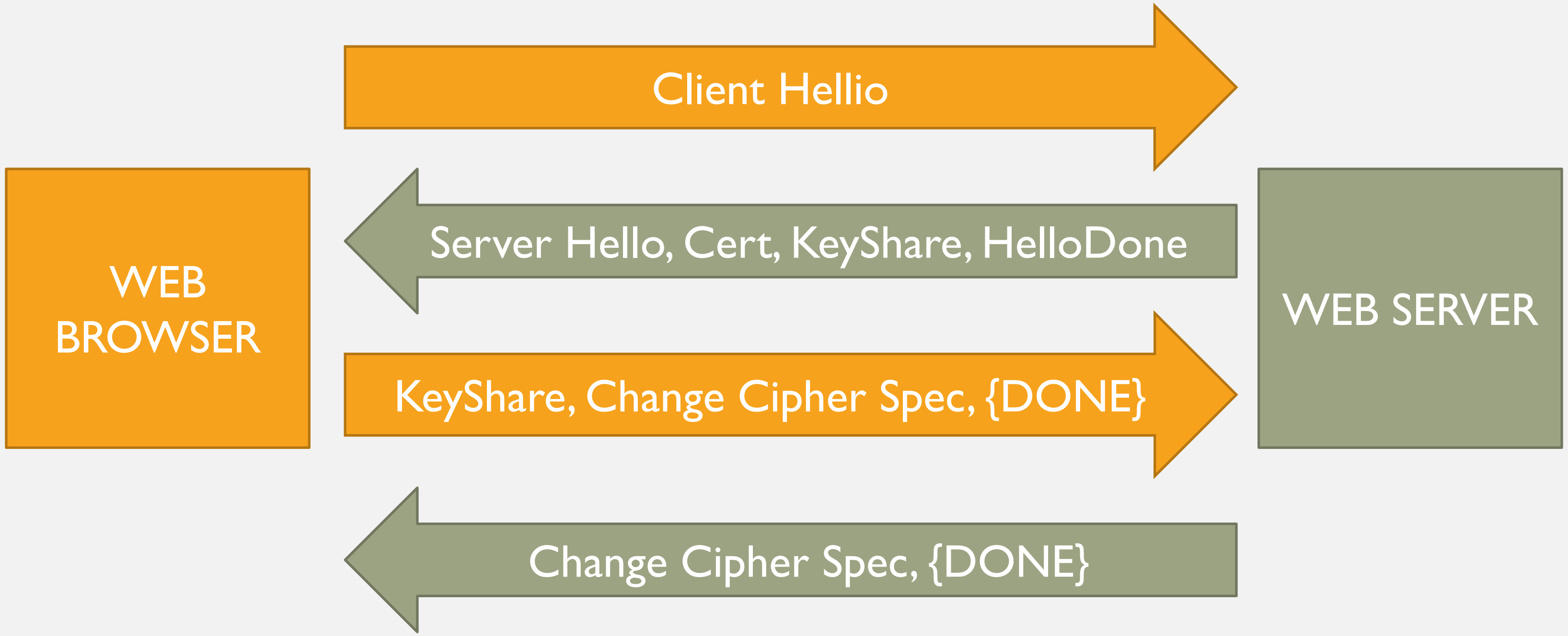
Goal: Identify the server (and optionally the client)

Goal: Create session keys for bi-directional communication

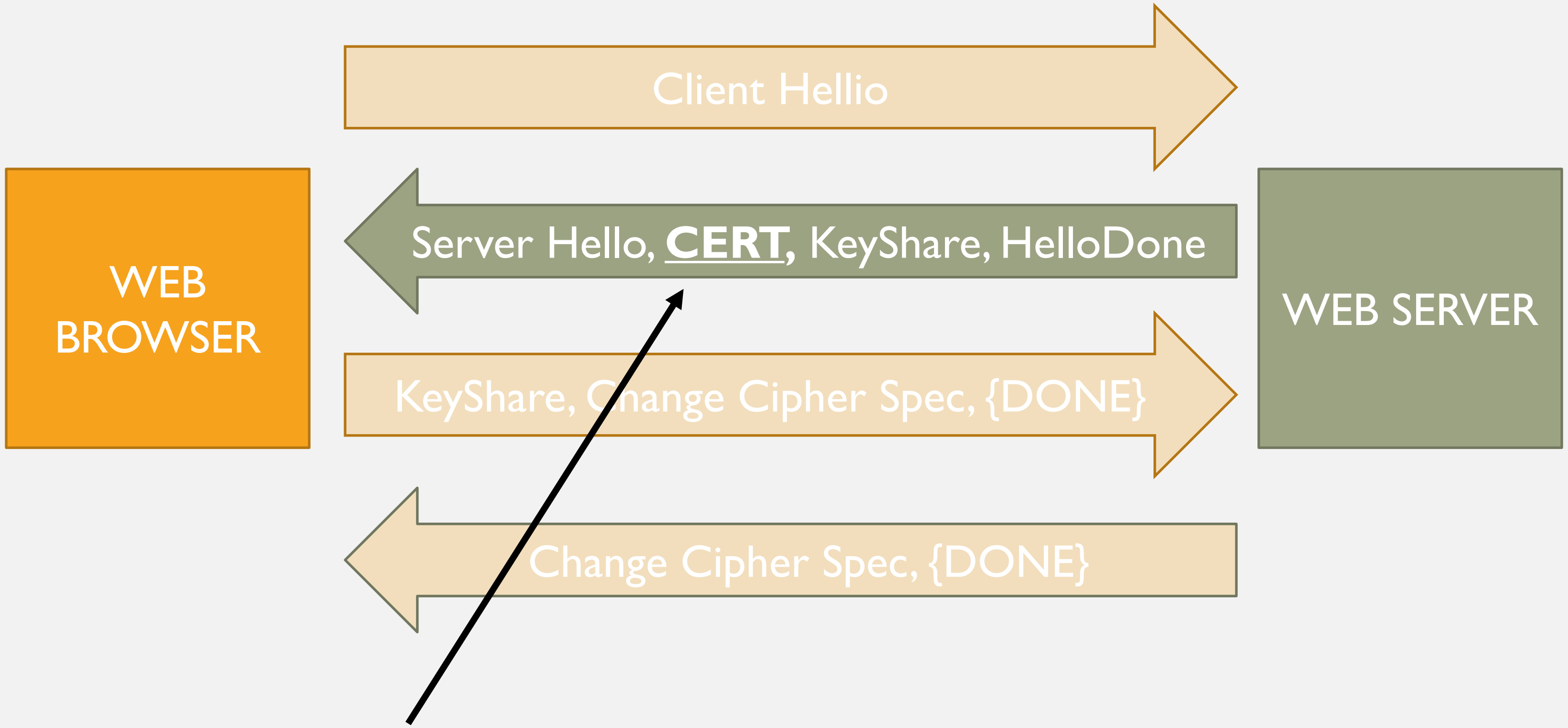
TLS 1.2 HANDSHAKE REVIEW

Step	Client	Direction	Message	Direction	Server
1			Client Hello	>	
2		<	Server Hello		
3		<	Certificate		
4		<	Server Key Exchange		
5		<	Server Hello Done		
6			Client Key Exchange	>	
7			Change Cipher Spec	>	
8			Finished	>	
9		<	Change Cipher Spec		
10		<	Finished		

END-TO-END HANDSHAKE VISUALIZATION #2



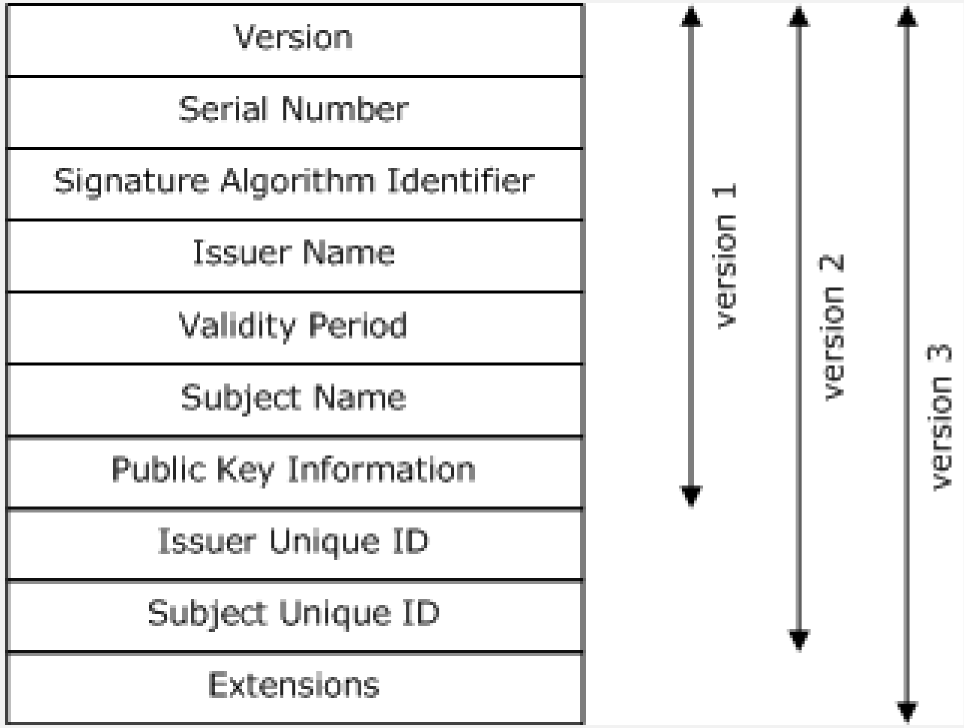
AUTHENTICATION



The “Certificate” message includes ONE OR MORE certificates.

WHAT IS A CERTIFICATE?

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



CERTIFICATE VERIFICATION

WEB BROWSER

Verify “amazon.com” is the URL
Verify the validity period
(Other Verification)
Who issued the cert?

CERTIFICATE

Subject CN:	amazon.com
Not Valid Before:	2001
Not Valid After:	2030
Issued By:	amazon CA
Signature Blob:	<sig>

PUBLIC KEY PRIVATE KEY

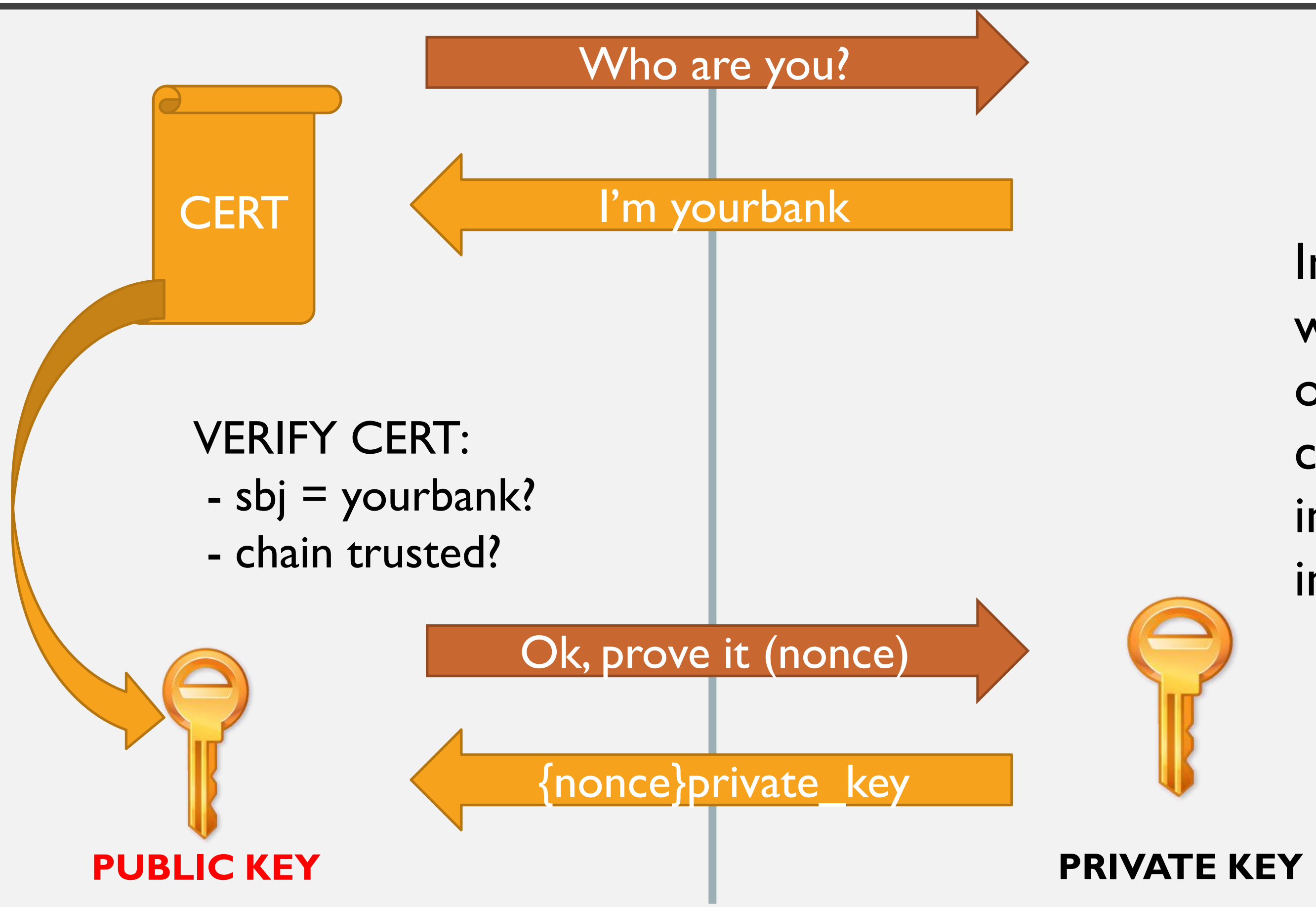
PRIVATE KEY

CERTIFICATE CHAINS

The certificate for the Host may be signed by an INTERMEDIATE Certificate Authority

Because the web browser probably doesn't have this intermediate cert, the TLS handshake includes both certificates.

PROVING IDENTITY



In TLS, the "nonce" is just wrapped up with the other data, such as the client hello, which is all included in the final hash in the finished message.

ROOT CA CERTIFICATES

Certificate chains MUST have a ROOT

A Root Certificate is SELF SIGNED

Browsers trust a set of root certificates AXIOMATICALLY

Certificate chains must have a trust chain to one of these roots.

TRUSTING DIFFIE HELLMAN

- Recall that DH keys are EPHEMERAL
- The Server's cert includes a long-term public key
- The Server's DH key is signed by this key pair
- IF the client trusts the cert, THEN it can validate the DH key

TLS BULK TRANSPORT

- Both Client and Server derive keys
- Encryption keys AND MAC keys
- MAC's ensure continuous authentication

WHEN A TLS MESSAGE IS RECEIVED:

- The sender is “proved” by the MAC
- The MAC is “proved” via MAC key derived from DH
- Server’s DH key “proved” authentic by cert signature
- Certificate “proved” authentic by chain to trusted root

IT ALL DEPENDS ON THE CERT

IF a browser trusts MY certificate to be Amazon's certificate

- THEN the browser will trust my DH public key

IF the browser trusts my DH public key

- THEN the browser will derive the same MAC key I do

IF the browser derives the same MAC key I do

- THEN the browser will believe my messages are from Amazon

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

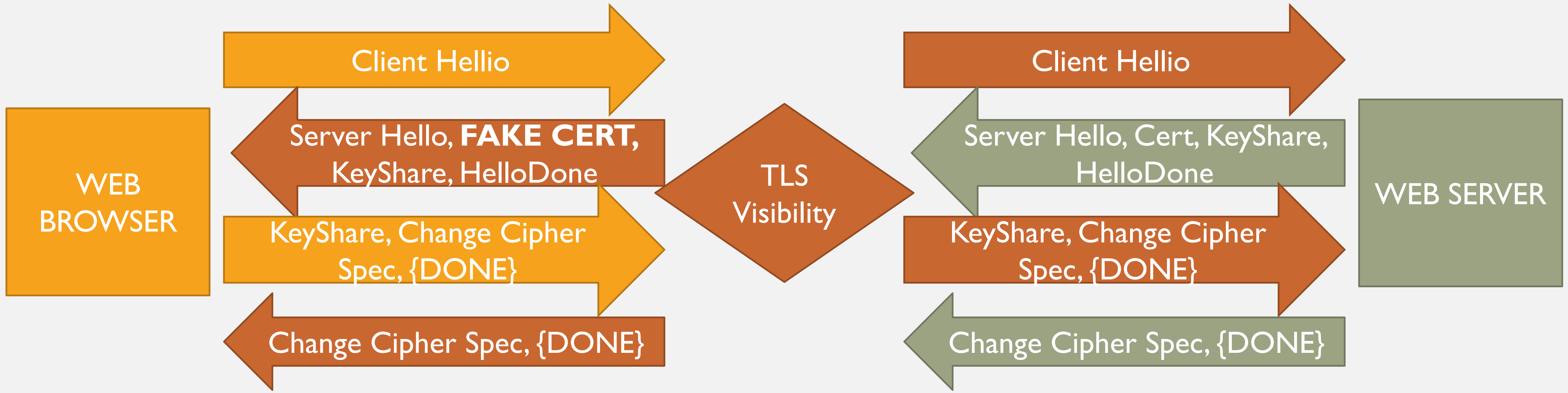
OTHER ALTERNATIVES TO TRUST?

- 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

TLS VISIBILITY

- Typically, a browser/client MUST have a new root CA installed
- This root CA is a self-signed certificate from the Visibility appliance
- The appliance can now generate ANY cert and the browser believes it!
- We will discuss the huge security concerns in a later lecture

TLS VISIBILITY HANDSHAKE VISUALIZATION



KERBEROS

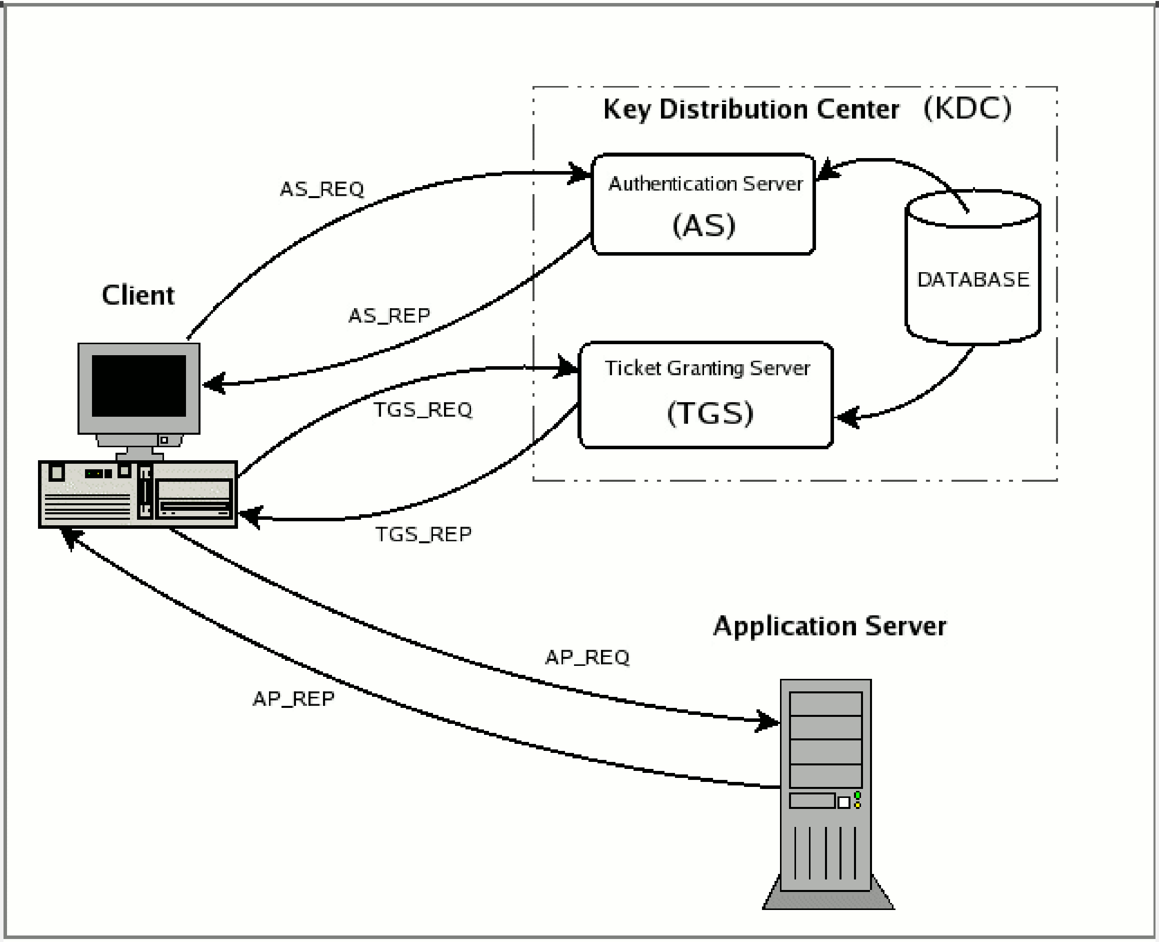
Kerberos vs TLS

- Kerberos uses trusted authentication server
- Must be online.
- If compromised, entire system compromised
- Mutual authentication, confidentiality

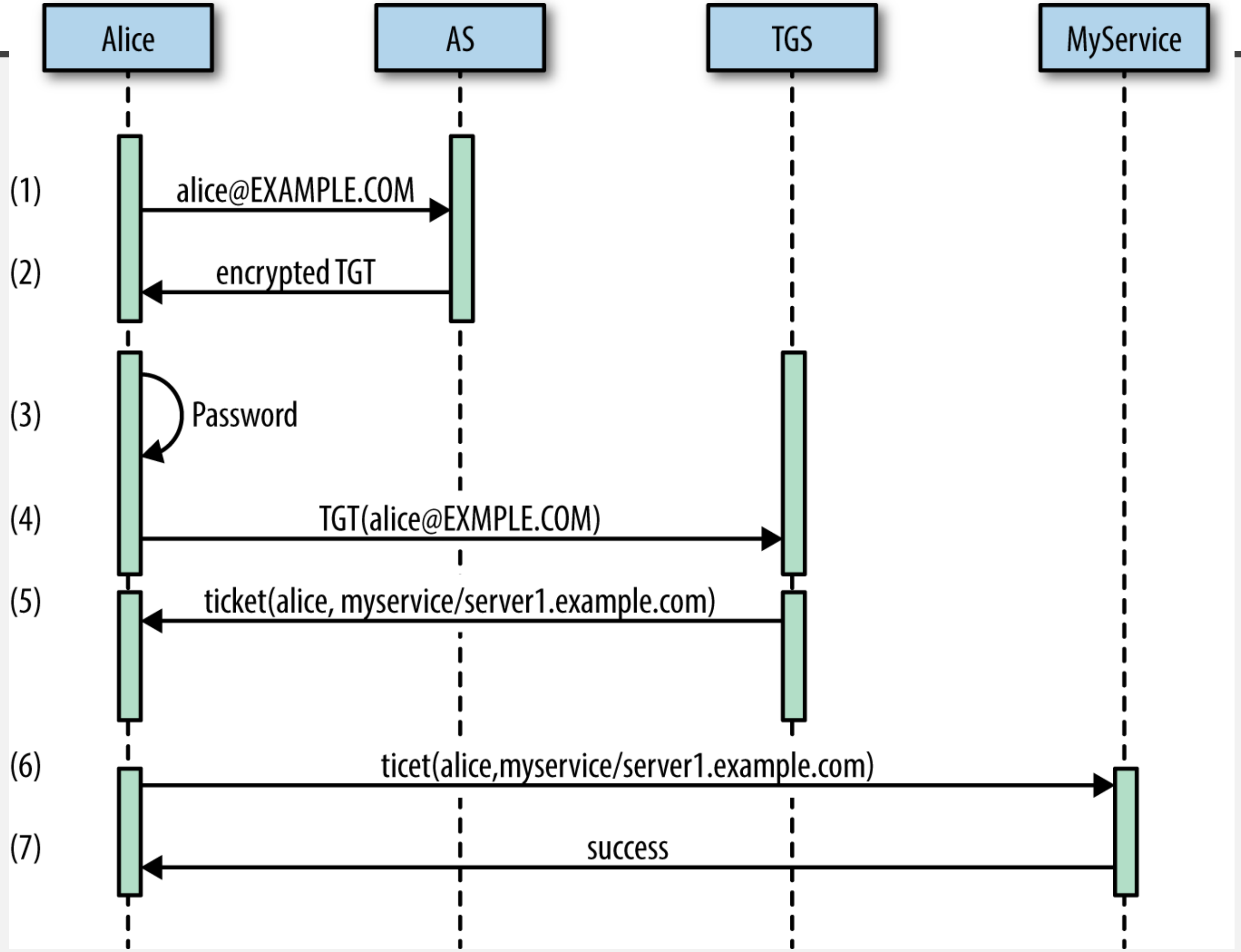
Basic components:

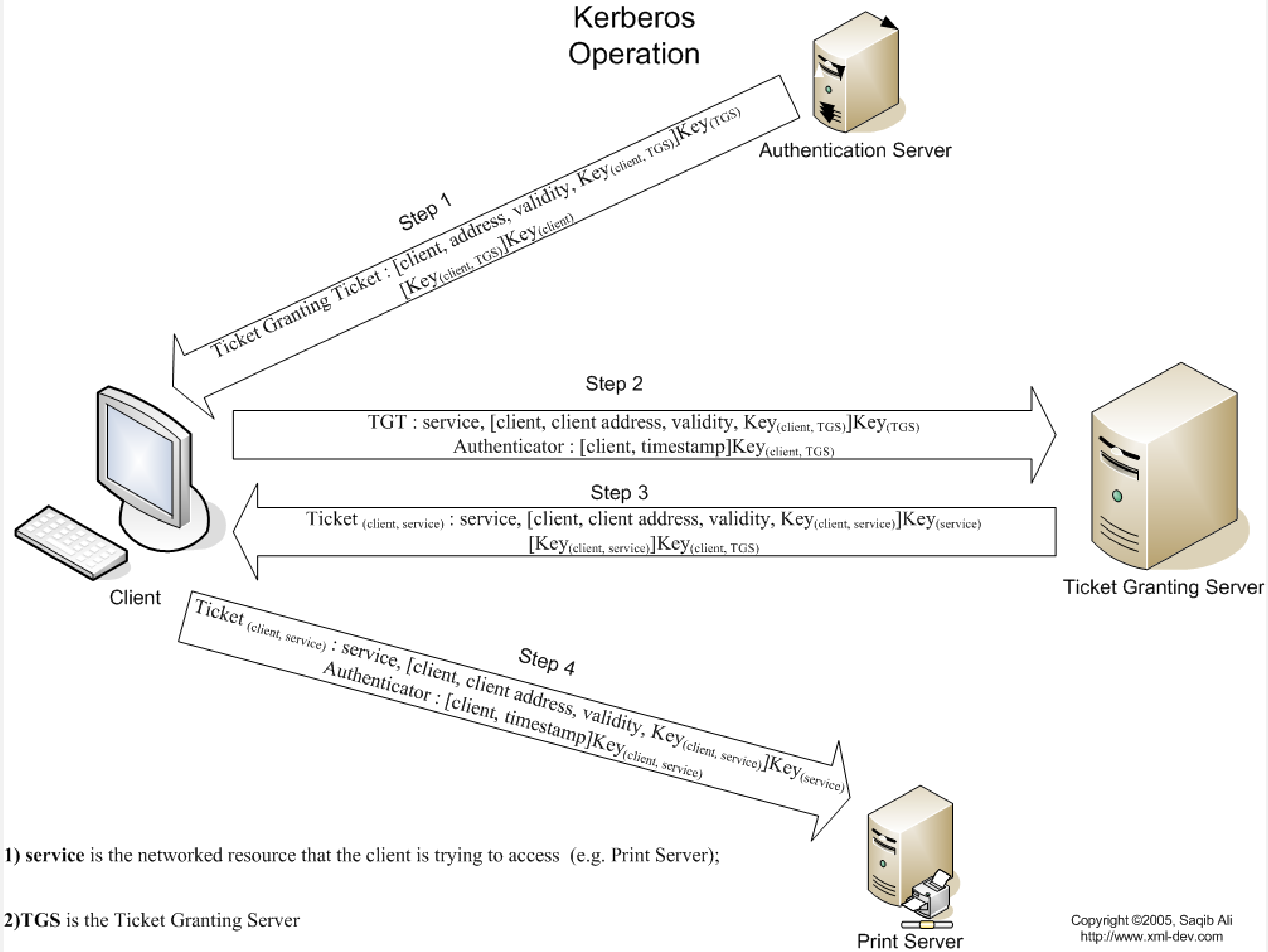
- Authentication Server
- Key Distribution Server (KDS)
- Ticket Granting Service
- Service Server

KERBEROS COMMUNICATION



KERBEROS PROTOCOL





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?