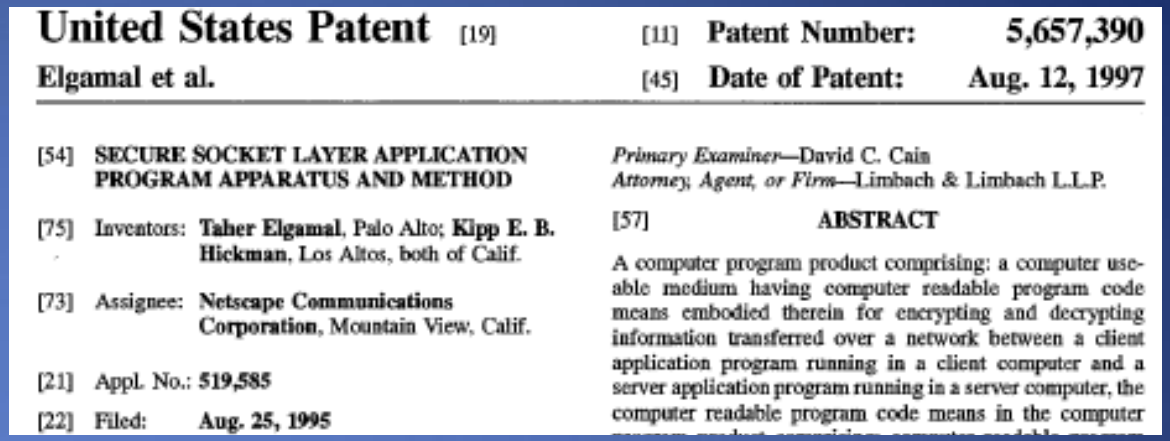
Networks IV: SSL/TLS

CS 1660: Introduction to Computer Systems Security

SSL and TLS

- Secure Socket Layer (SSL)
  - Early protocol for securing web connections
  - Developed in the 90s by team led by Taher Elgamal at Netscape
- Transport Layer Security (TLS)
  - Evolution of SSL
  - Standardized by IETF
  - TLS 1.0 RFC 2246 (1999)
  - TLS 1.2 RFC 5246 (2008)
  - TLS 1.3 RFC 8446 (2018)



United States Patent [19]
Elgamal et al.

[11] **Patent Number: 5,657,390**
[45] **Date of Patent: Aug. 12, 1997**

[54] **SECURE SOCKET LAYER APPLICATION PROGRAM APPARATUS AND METHOD**

[75] Inventors: **Taher Elgamal**, Palo Alto; **Kipp E. B. Hickman**, Los Altos, both of Calif.

[73] Assignee: **Netscape Communications Corporation**, Mountain View, Calif.

[21] Appl. No.: **519,585**

[22] Filed: **Aug. 25, 1995**

Primary Examiner—David C. Cain
Attorney, Agent, or Firm—Limbach & Limbach L.L.P.

[57] **ABSTRACT**

A computer program product comprising: a computer useable medium having computer readable program code means embodied therein for encrypting and decrypting information transferred over a network between a client application program running in a client computer and a server application program running in a server computer, the computer readable program code means in the computer

- Patent issued in 1997
  - … method of encrypting and decrypting information transferred over a network between a client … and a server …

Taher Elgamal

Image source: Alexander Klink via Wikipedia

Overview

Goals of SSL/TLS

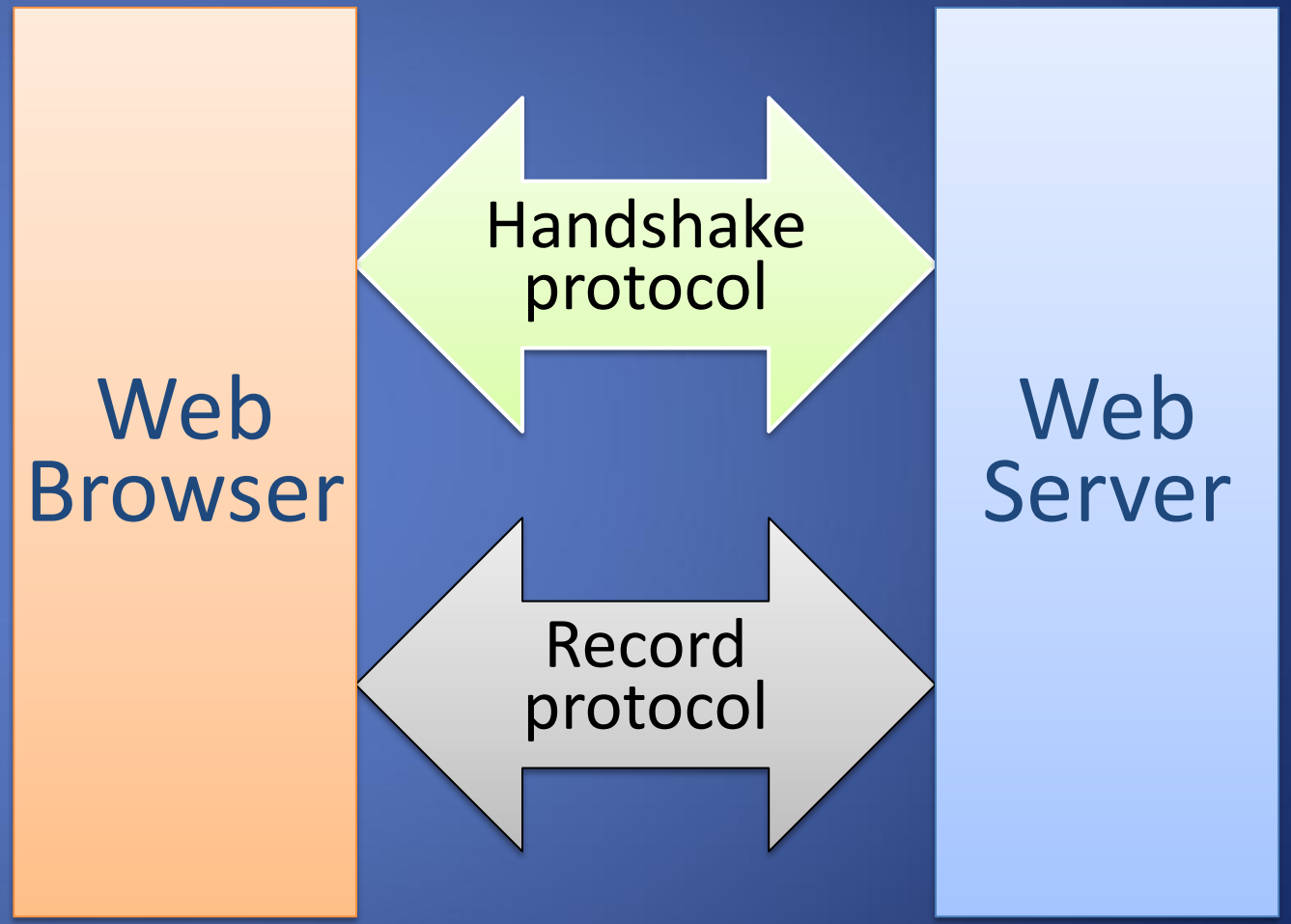
- End-to End Confidentiality
  - Encrypt communication between client and server applications
- End-to-End Integrity
  - Detect corruption of communication between client and server applications
- Required server Authentication
  - Identity of server always proved to client
- Optional client authentication
  - Identity of client optionally proved to server
- Modular deployment
  - Intermediate layer between application and transport layers
  - Handles encryption, integrity, and authentication on behalf of client and server applications

TLS Building Blocks

	Confidentiality	Integrity	Authentication
Setup	Public-key encryption (e.g, RSA)	Public-key digital signature (e.g., RSA)	Public-key digital signature (e.g., RSA)
Data transmission	Symmetric encryption (e.g., AES)	Cryptographic hashing (e.g., SHA256)	

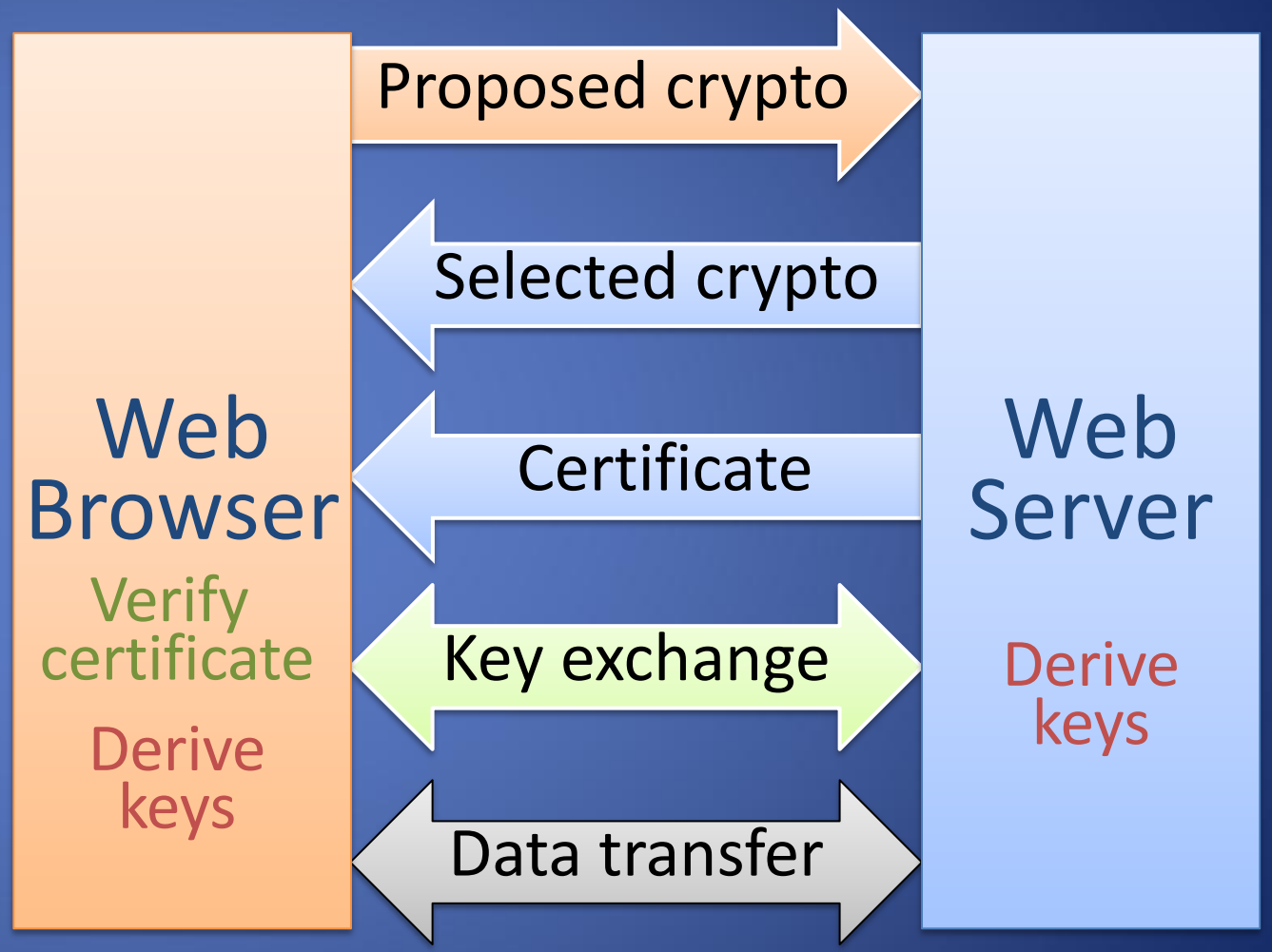
TLS Overview

- Handshake protocol
  - Client authenticates server
  - [Server authenticates client]
  - Client and server agree on crypto algorithms
  - Client and server establish session keys
- Record protocol
  - Encrypt and add integrity protection before sending data
  - Verify integrity and decrypt after receiving data



TLS Overview

- Browser sends supported crypto algorithms (aka cipher suite)
- Server picks strongest algorithms it supports
- Server sends certificate (chain)
- Client verifies certificate (chain)
- Client and server agree on secret value by exchanging messages
- Secret value is used to derive keys for symmetric encryption and hash-based authentication of subsequent data transfer



Example of Cipher Suite

TLS_RSA_WITH_AES_128_GCM_SHA256

- **TLS** defines the protocol
- **RSA** specifies the key exchange algorithm
- **AES_128_GCM** indicates the cipher being used to encrypt the message stream
- **SHA256** identifies the hash algorithm used to authenticate messages

SSL/TLS analysis with Wireshark
https://tls.ulfheim.net/

Clicker Question (1)

- Which of the following is not true about TLS?

A. TLS is a more secure and updated version of SSL

B. Encryption of data takes place during handshake between client and server

C. TLS is not immune from private key theft

D. TLS is faster because it uses fewer resources than SSL

Clicker Question (1) - Answer

- Which of the following is not true about TLS?

A. TLS is a more secure and updated version of SSL

B. Encryption of data takes place during handshake between client and server

C. TLS is not immune from private key theft

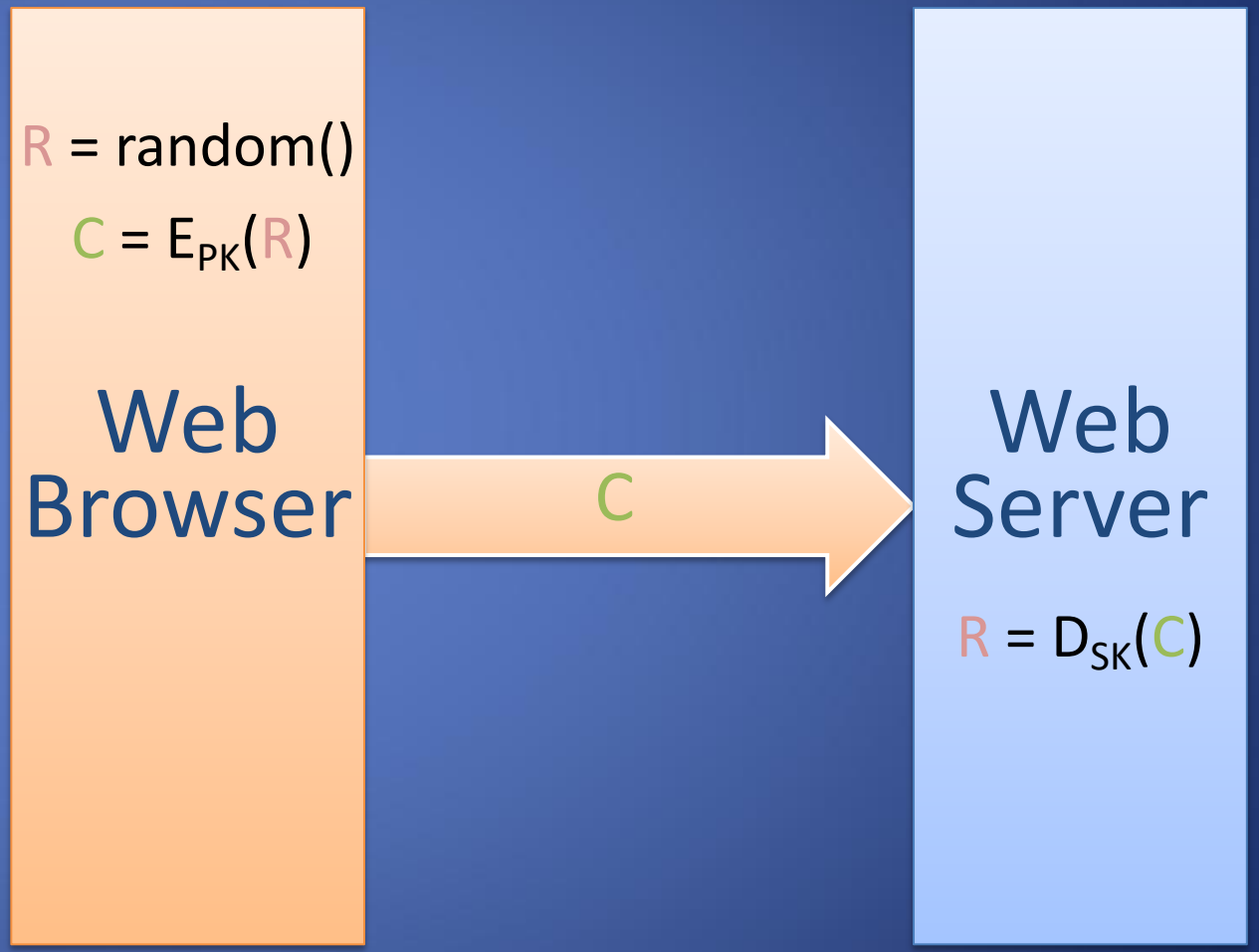
D. TLS is faster because it uses fewer resources than SSL

The handshake simply agrees on crypto algorithm and keys for encryption and integrity checking, but doesn't actually encrypt

Key Exchange and Forward Secrecy

Basic Key Exchange

- Called RSA key exchange for historical reasons
- Client generates random secret value R
- Client encrypts R with public key, PK, of server: $C = E_{PK}(R)$
- Client sends C to server
- Server decrypts C with private key, SK, of server: $R = D_{SK}(C)$

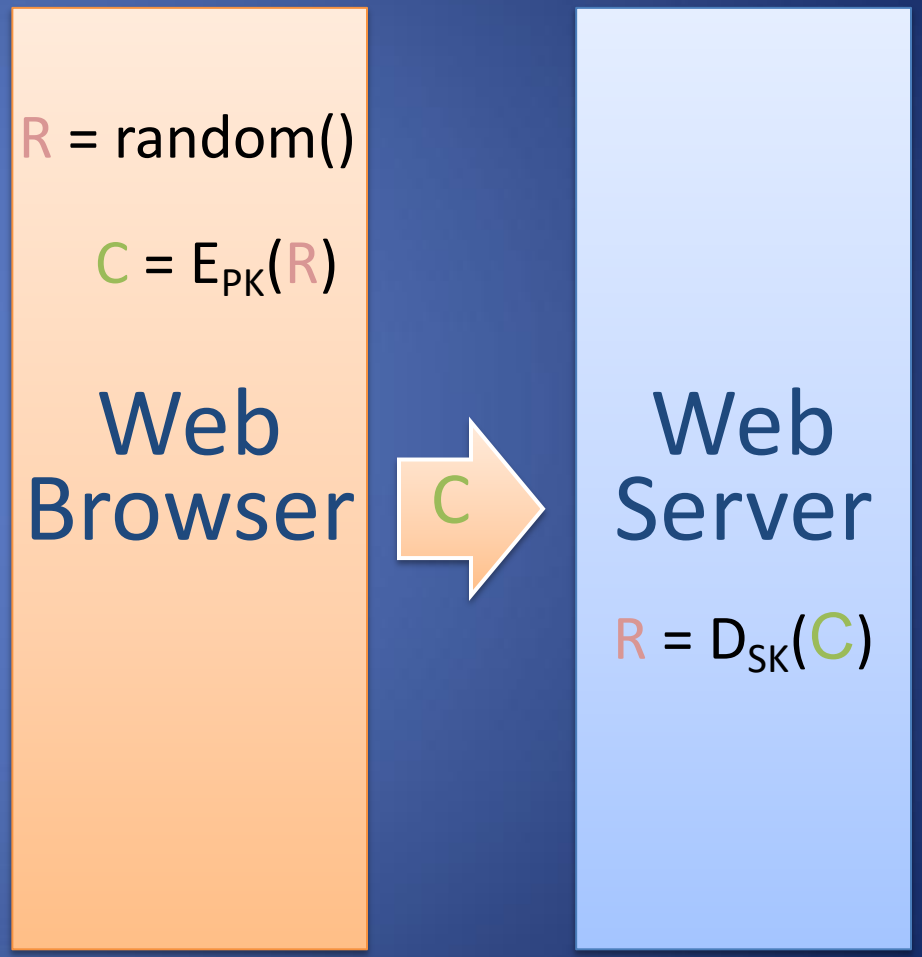


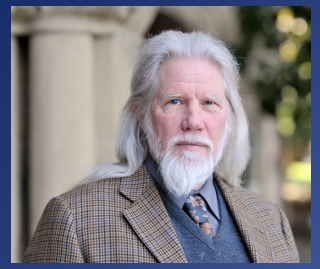
Forward Secrecy

- General concept
  - Compromise of public-key encryption private keys does not break confidentiality of past encrypted messages
- Forward secrecy in the context of TLS
  - Compromise of server's private key (associated with public key in certificate) does not break confidentiality of past TLS sessions
- TLS with basic key exchange (aka RSA key exchange) does not provide forward secrecy

Forward Secrecy

- Compromise of public-key encryption private keys does not break confidentiality of past messages
- TLS with basic key exchange does not provide forward secrecy
  - Attacker eavesdrop and stores all TLS communication
  - If server's private key, SK, is compromised, attacker recovers secret value R in key exchange and derives from R encryption key used in subsequent encrypted TLS communication



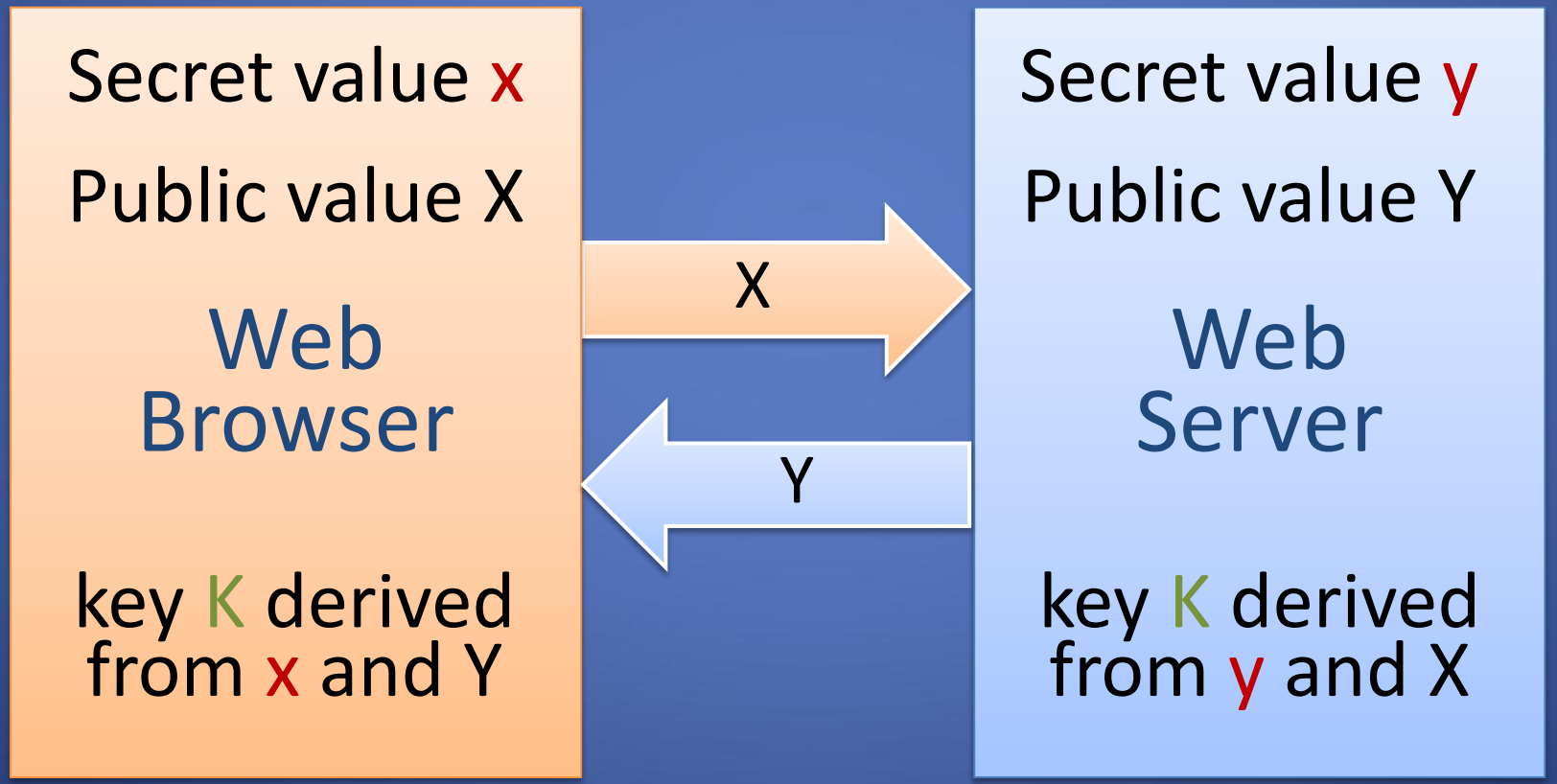


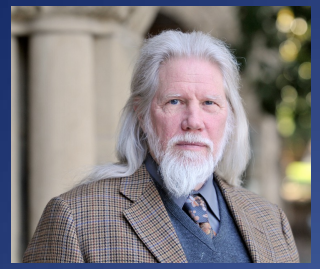
Source: ACM

Diffie Hellman Key Exchange

Achieves forward secrecy

Source: ACM





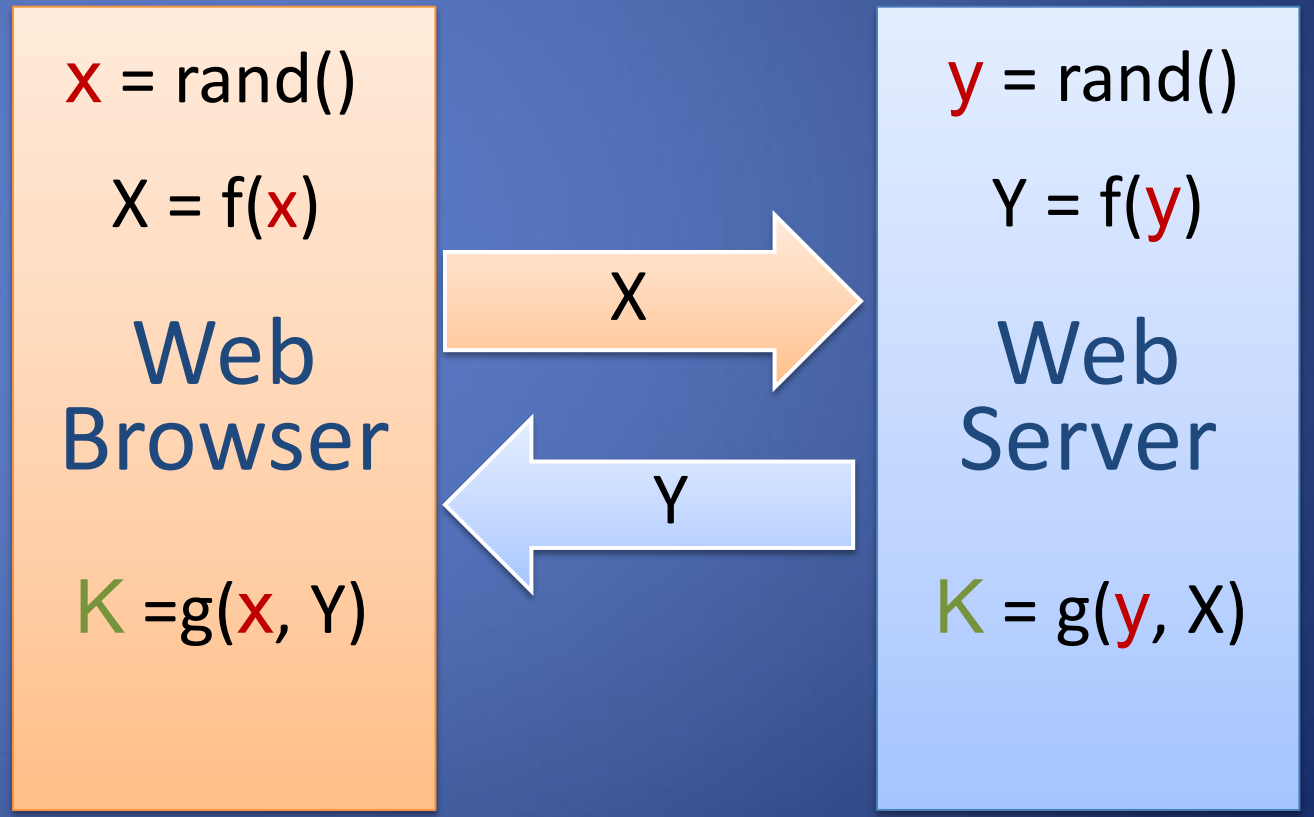
Source: ACM

Diffie Hellman Key Exchange

Achieves forward secrecy

Source: ACM

- Client randomly generates x and derives public value X
- Server randomly generates y and derives public value Y
- Client and server exchange values X and Y
- Client derives key K from x and Y
- Server derives key K from y and X
- Attacker who captures X and Y cannot reconstruct K



Modular Arithmetic

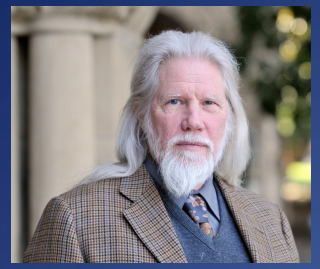
- mod function
  - $x \bmod n$ is the remainder of the division of x by n
  - $x \bmod n$ has has values between 0 and $n - 1$
- Examples
  - $29 \bmod 13 = 3$
  - $13 \bmod 13 = 0$
  - $-1 \bmod 13 = 12$
- Modular arithmetic has properties similar to standard arithmetic
  - E.g., associative and commutative
- Several cryptographic functions are based on modular arithmetic
  - E.g., RSA cryptosystem

Power of a Power Property

- Standard arithmetic
  - $a^{xy} = (a^x)^y = (a^y)^x$
  - Example: $2^{2 \cdot 3} = (2^2)^3 = (2^3)^2 = 64$
- Modular arithmetic
  - $a^{xy} \bmod n = (a^x)^y \bmod n = (a^y)^x \bmod n$

Discrete Logarithm Problem

- Modular power and logarithm
  - $y = a^x \mod n$
  - Assume a and n are fixed public parameters
  - x is the logarithm of y in base a modulo n
- Modular power is easy
  - There is an efficient algorithm to compute y given x
- Modular logarithm is hard
  - No efficient algorithm is known to compute x given y



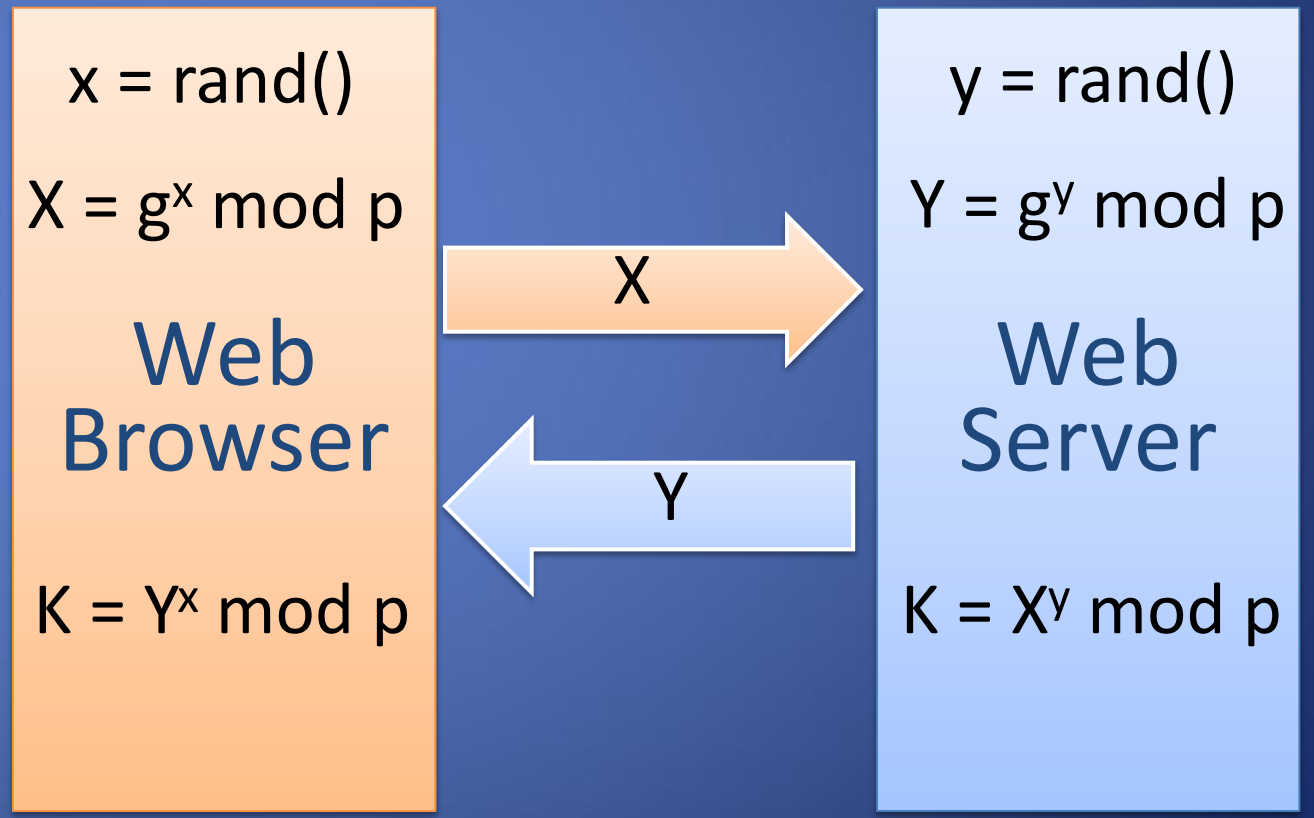
Source: ACM

DH Key Exchange Details

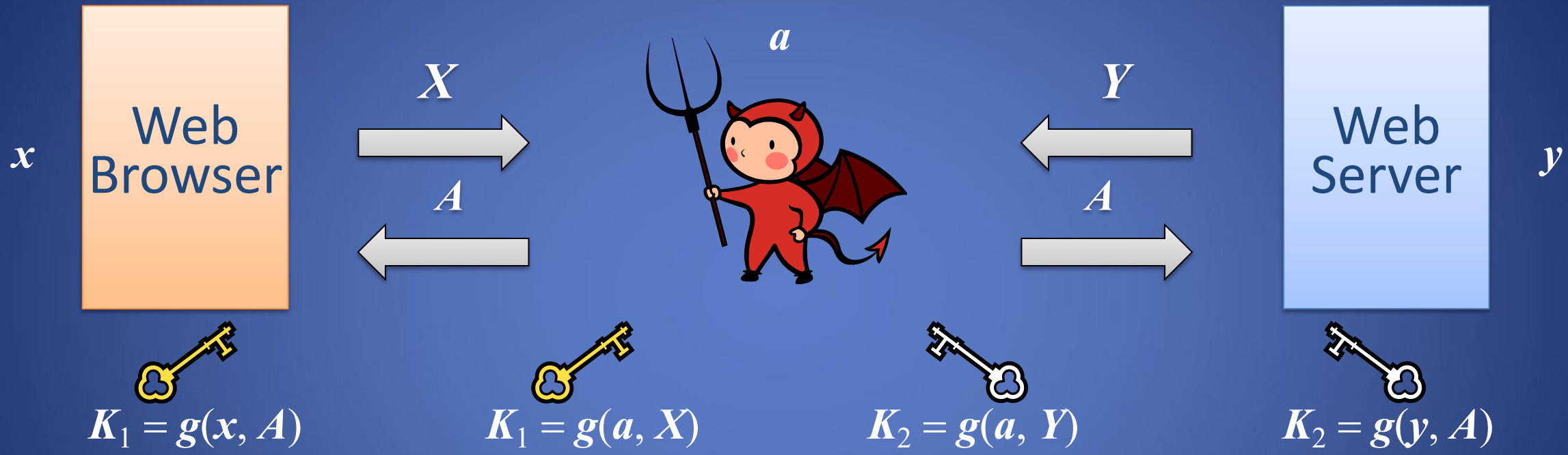
Achieves forward secrecy

Source: ACM

- Public parameters: prime p and generator g
- Client generates random x and computes $X = g^x \bmod p$
- Server generates random y and computes $Y = g^y \bmod p$
- Client sends X to server
- Server sends Y to client
- Client and server compute $K = g^{xy} \bmod p$



Injection Attack



Solution

- Browser and server send signed X and Y respectively
- Requires each to know the public key of the other
- Optional for browser as it usually does not have certificate

Clicker Question (2)

- DH key exchange is prone to man in the middle attack, because it does not provide _____ of participating parties.

A. Security token
B. Authentication
C. One-time pad
D. Password

Clicker Question (2) - Answer

- DH key exchange is prone to man in the middle attack, because it does not provide authentication of participating parties.

A. Security token

B. Authentication

C. One-time pad

D. Password

Certificates and PKI

TLS Goals

- Confidentiality
- Integrity
- Authentication

TLS Goals

Authentication: verifying that the entity on the other end of the connection is who they claim to be

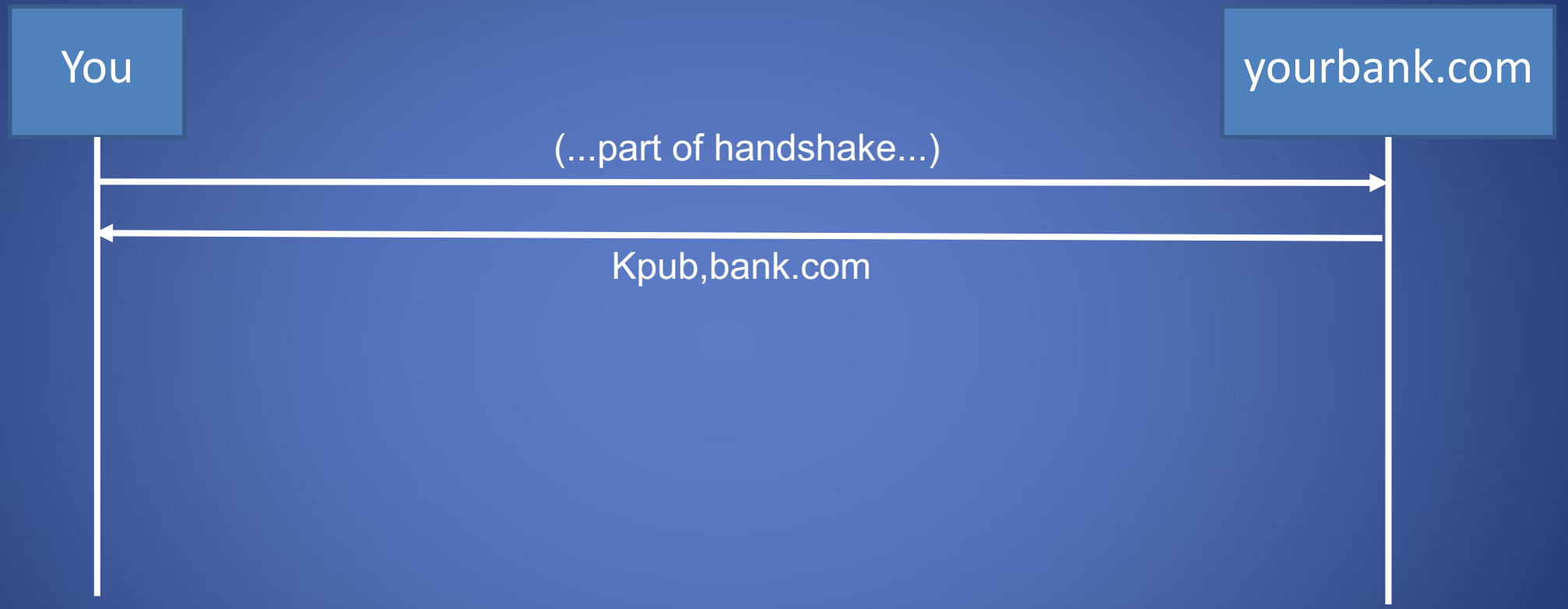
TLS Goals

Authentication: verifying that the entity on the other end of the connection is who they claim to be

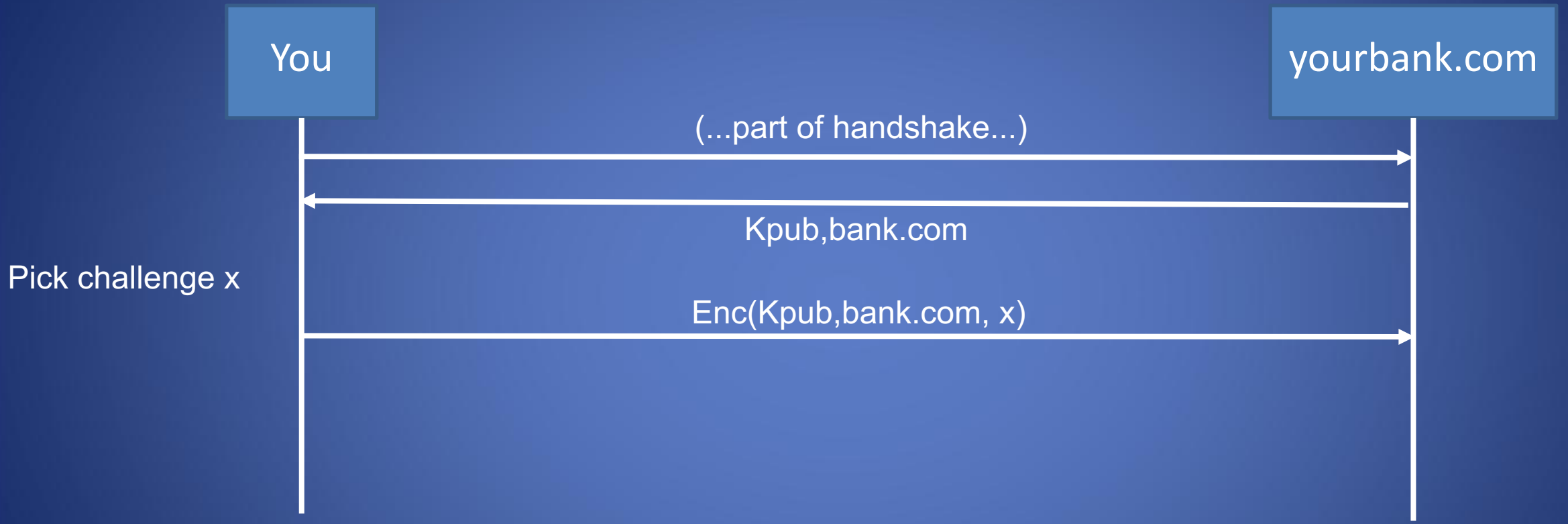
- Technical aspects: crypto
- Social aspects
  - How to distribute keys to entities
  - What to do when things go wrong

TLS: relies on Public Key Infrastructure (PKI) via certificates

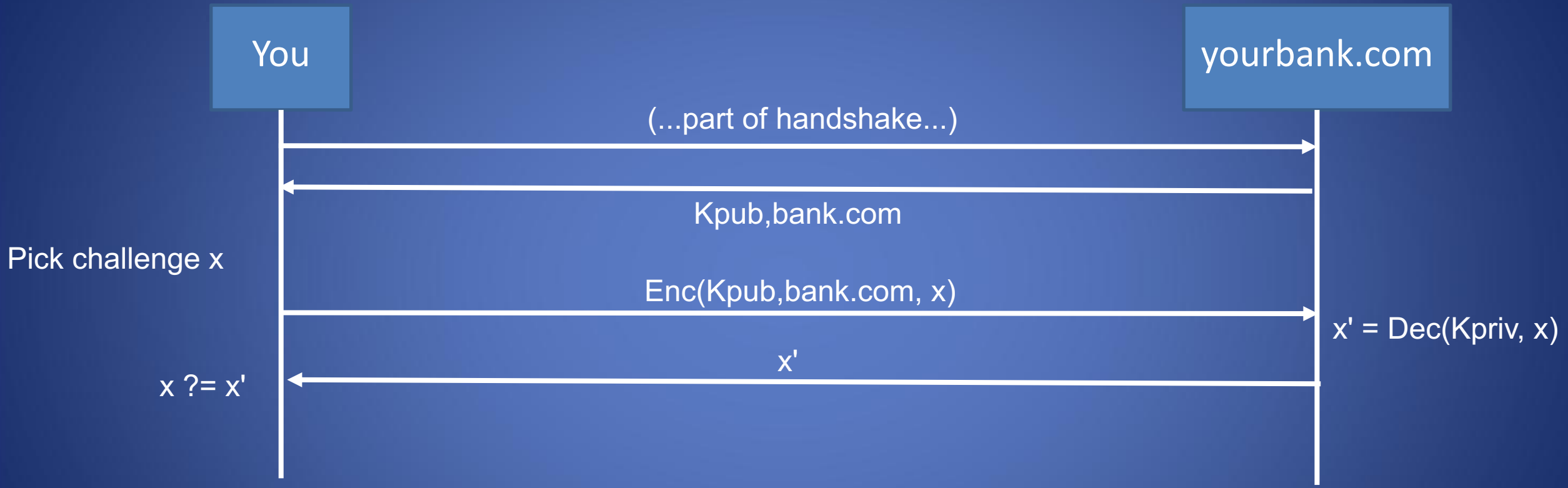
The Challenge



The Challenge



The Challenge



What does this prove?

Authentication challenges

- Challenge proves that the server at yourbank.com holds Kpriv
- Does NOT prove belong to the server belongs to your bank, the real-life bank with your money

Authentication challenges

- Challenge proves that the server at yourbank.com holds Kpriv
- Does NOT prove belong to the server belongs to your bank, the real-life bank with your money

"But I'm visiting yourbank.com!"

Authentication challenges

- Challenge proves that the server at yourbank.com holds Kpriv
- Does NOT prove the server belongs to YourBank, the real-life bank that holds your money

"But I'm visiting yourbank.com!"

- DNS can be spoofed
- Possible active network attacker (redirecting your IP traffic to malicious server)
- Domain names can expire and be re-registered...

Problem: distributing trust

How can we trust Kpub is Your Bank's public key?

Problem: Trust distribution

- Hard to verify real-world identities
- Hard to scale to the whole Internet

Different protocols have different mechanisms

=> TLS (and others): Public Key Infrastructure (PKI) with certificates

PKI: The main idea

Public keys managed by Certificate Authorities (CAs)

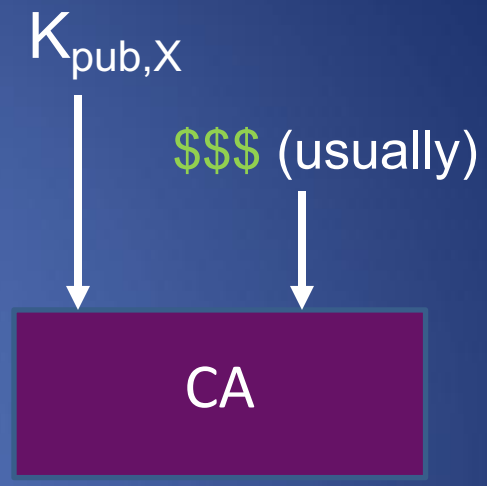
- Everyone knows public key for some root CAs
  - Pre-installed into browser/OS

PKI: The main idea

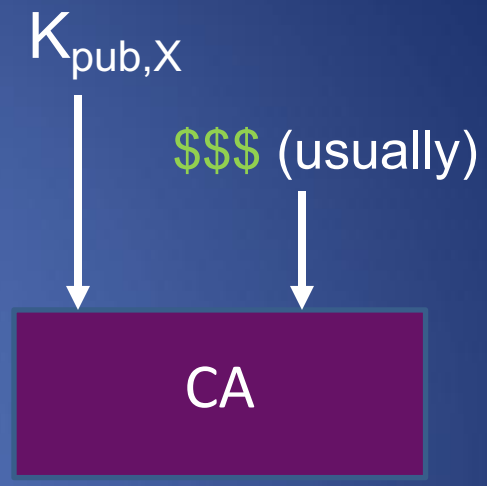
Public keys managed by Certificate Authorities (CAs)

- Everyone knows public key for some root CAs
  - Pre-installed into browser/OS

- If X wants a public key, request from CA
  - CA validates X's identity, then signs X's public key



PKI: The main idea



Public keys managed by Certificate Authorities (CAs)

- Everyone knows public key for some root CAs
  - Pre-installed into browser/OS

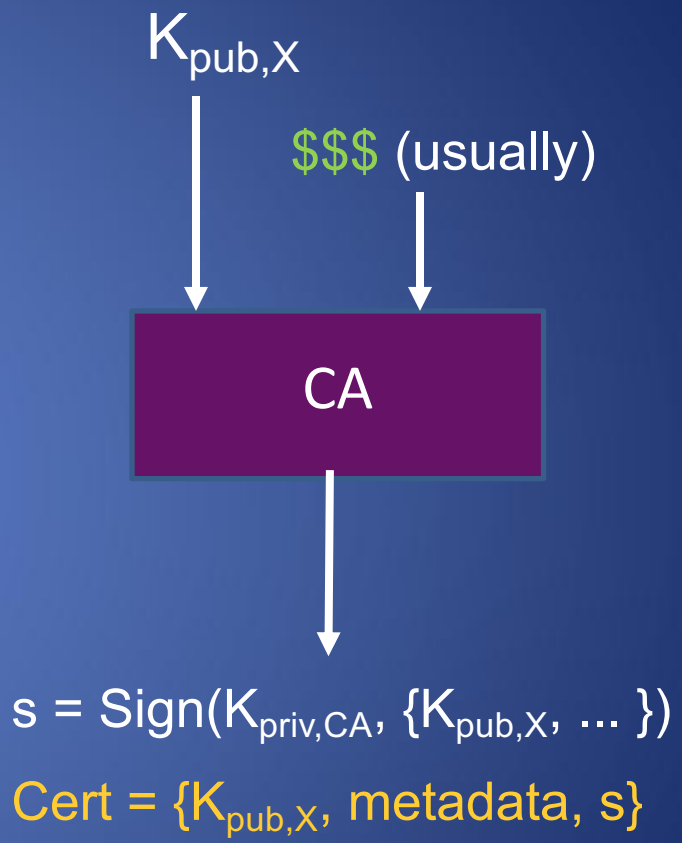
- If X wants a public key, request from CA
  - CA validates X's identity, then signs X's public key
  - Generates certificate

PKI: The main idea

Public keys managed by Certificate Authorities (CAs)

- Everyone knows public key for some root CAs
  - Pre-installed into browser/OS

- If X wants a public key, request from CA
  - CA validates X's identity, then signs X's public key
  - Generates certificate



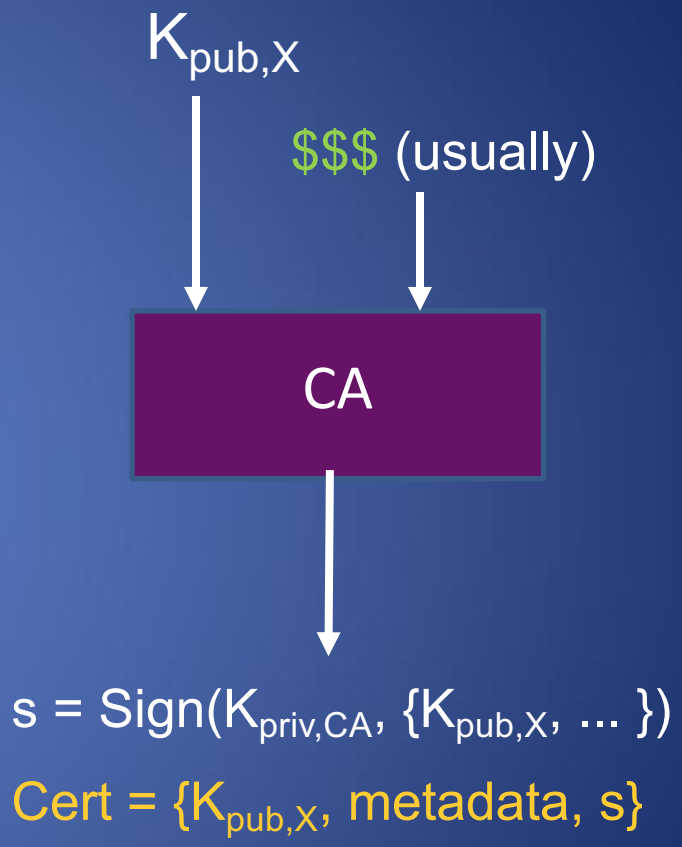
PKI: The main idea

Public keys managed by Certificate Authorities (CAs)

- Everyone knows public key for some root CAs
  - Pre-installed into browser/OS

- If X wants a public key, request from CA
  - CA validates X's identity, then signs X's public key
  - Generates certificate
- Client can verify $K_{pub,X}$ from CA's signature:

  Verify($K_{pub,CA}$ Cert) => True/False



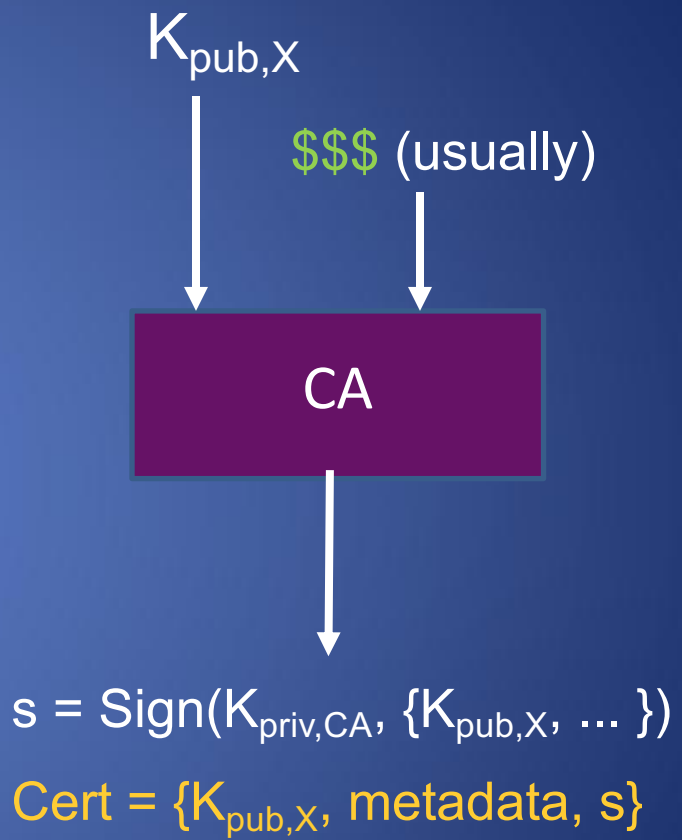
PKI: The main idea

Public keys managed by Certificate Authorities (CAs)

- Everyone knows public key for some root CAs
  - Pre-installed into browser/OS

- If X wants a public key, request from CA
  - CA validates X's identity, then signs X's public key
  - Generates certificate
- Client can verify $K_{pub,X}$ from CA's signature:

  Verify($K_{pub,CA}$ Cert) => True/False



=> Delegates trust for individual entity to a more trusted authority


DigiCert Assured ID Root CA
DigiCert Assured ID Root CA
Root certificate authority
Expires: Sunday, November 9, 2031 at 19:00:00 Eastern Standard Time
This certificate is valid
Trust
Details
Subject Name
Country or Region US
Organization DigiCert Inc
Organizational Unit www.digicert.com
Common Name DigiCert Assured ID Root CA
Issuer Name
Country or Region US
Organization DigiCert Inc
Organizational Unit www.digicert.com
Common Name DigiCert Assured ID Root CA
Serial Number 0C E7 E0 E5 17 D8 46 FE 8F E5 60 FC 1B F0 30 39
Version 3
Signature Algorithm SHA-1 with RSA Encryption (1.2.840.113549.1.1.5)
Parameters None
Not Valid Before Thursday, November 9, 2006 at 19:00:00 Eastern Standard Time
Not Valid After Sunday, November 9, 2031 at 19:00:00 Eastern Standard Time
Public Key Info
Algorithm RSA Encryption (1.2.840.113549.1.1.1)
Parameters None
Public Key 256 bytes : AD 0E 15 CE E4 43 80 5C ...
Exponent 65537
Key Size 2,048 bits
Key Usage Verify

Keychain Access

Search

All Items | Passwords | Secure Notes | My Certificates | Keys | Certificates

Amazon Root CA 1
Root certificate authority
Expires: Saturday, January 16, 2038 at 19:00:00 Eastern Standard Time
This certificate is valid

Name	Kind	Date Modified	Expires	Keychain
AAA Certificate Services	certificate	--	Dec 31, 2028 at 18:59:59	System Roots
AC RAIZ FNMT-RCM	certificate	--	Dec 31, 2029 at 19:00:00	System Roots
Actalis Authentication Root CA	certificate	--	Sep 22, 2030 at 07:22:02	System Roots
AffirmTrust Commercial	certificate	--	Dec 31, 2030 at 09:06:06	System Roots
AffirmTrust Networking	certificate	--	Dec 31, 2030 at 09:08:24	System Roots
AffirmTrust Premium	certificate	--	Dec 31, 2040 at 09:10:36	System Roots
AffirmTrust Premium ECC	certificate	--	Dec 31, 2040 at 09:20:24	System Roots
Amazon Root CA 1	certificate	--	Jan 16, 2038 at 19:00:00	System Roots
Amazon Root CA 2	certificate	--	May 25, 2040 at 20:00:00	System Roots
Amazon Root CA 3	certificate	--	May 25, 2040 at 20:00:00	System Roots
Amazon Root CA 4	certificate	--	May 25, 2040 at 20:00:00	System Roots
ANF Global Root CA	certificate	--	Jun 5, 2033 at 13:45:38	System Roots
Apple Root CA	certificate	--	Feb 9, 2035 at 16:40:36	System Roots
Apple Root CA - G2	certificate	--	Apr 30, 2039 at 14:10:09	System Roots
Apple Root CA - G3	certificate	--	Apr 30, 2039 at 14:19:06	System Roots
Apple Root Certificate Authority	certificate	--	Feb 9, 2025 at 19:18:14	System Roots
Atos TrustedRoot 2011	certificate	--	Dec 31, 2030 at 18:59:59	System Roots
Autoridad de Certificacion Firmaprofesional CIF A62634068	certificate	--	Dec 31, 2030 at 03:38:15	System Roots
Autoridad de Certificacion Raiz del Estado Venezolano	certificate	--	Dec 17, 2030 at 18:59:59	System Roots
Baltimore CyberTrust Root	certificate	--	May 12, 2025 at 19:59:00	System Roots
Buypass Class 2 Root CA	certificate	--	Oct 26, 2040 at 04:38:03	System Roots
Buypass Class 3 Root CA	certificate	--	Oct 26, 2040 at 04:28:58	System Roots
CA Disig Root R1	certificate	--	Jul 19, 2042 at 05:06:56	System Roots
CA Disig Root R2	certificate	--	Jul 19, 2042 at 05:15:30	System Roots
Certigna	certificate	--	Jun 29, 2027 at 11:13:05	System Roots
Certinomis - Autorité Racine	certificate	--	Sep 17, 2028 at 04:28:59	System Roots
Certinomis - Root CA	certificate	--	Oct 21, 2033 at 05:17:18	System Roots
Certplus Root CA G1	certificate	--	Jan 14, 2038 at 19:00:00	System Roots
Certplus Root CA G2	certificate	--	Jan 14, 2038 at 19:00:00	System Roots
certSIGN ROOT CA	certificate	--	Jul 4, 2031 at 13:20:04	System Roots
Certum CA	certificate	--	Jun 11, 2027 at 06:46:39	System Roots
Certum Trusted Network CA	certificate	--	Dec 31, 2029 at 07:07:37	System Roots

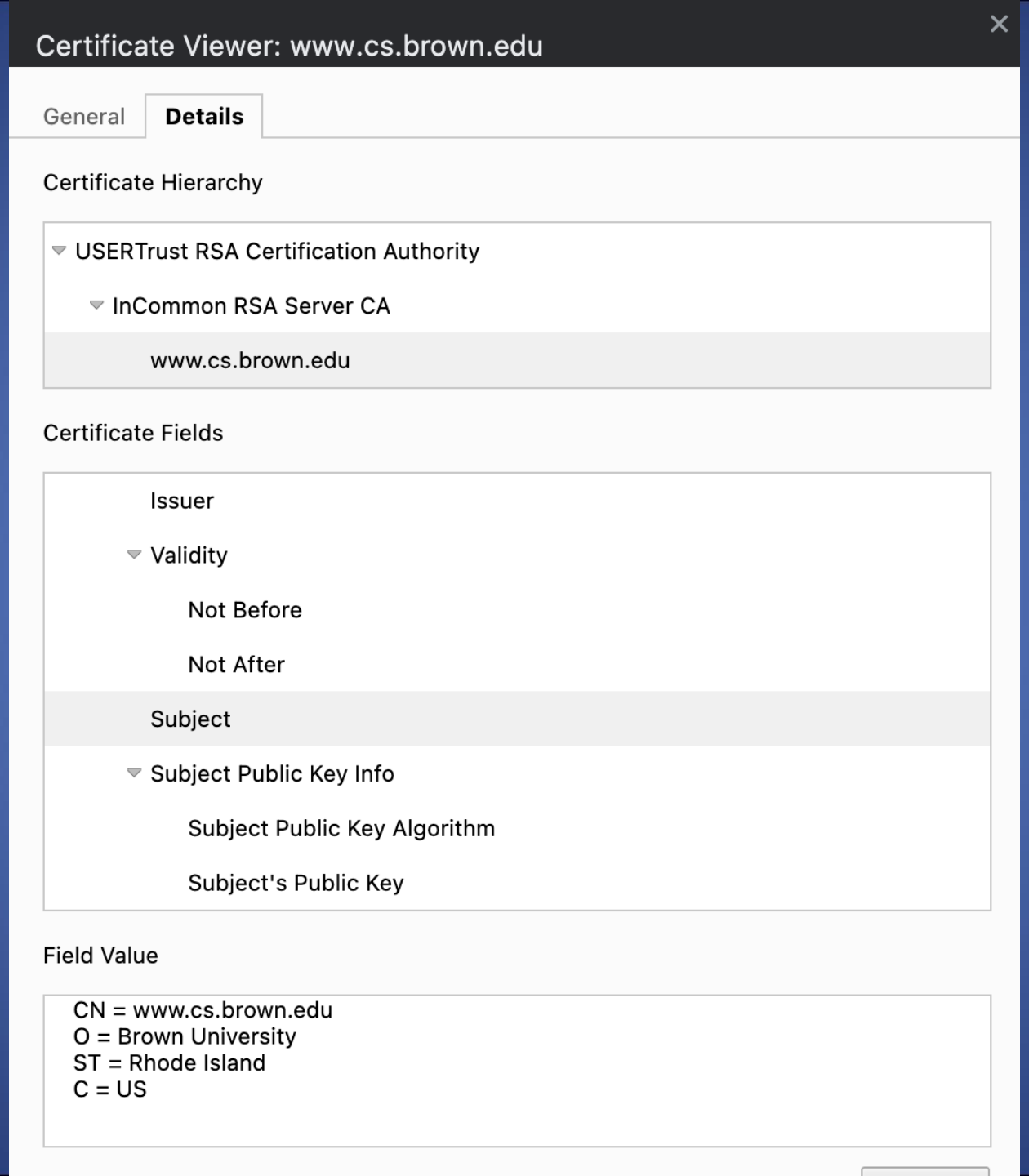
What's in a certificate?

- Public key of entity (eg. yourbank.com)
- Common name: DNS name of server (yourbank.com)
- Contact info for organization

What's in a certificate?

- Public key of entity (eg. yourbank.com)
- Common name: DNS name of server (yourbank.com)
- Contact info for organization
- Validity dates (start date, expire date)
- URL of *revocation center* to check if key has been revoked

All of this is part of the data signed by the CA
=> Critical to check all parts during TLS startup!


Certificate Viewer: www.cs.brown.edu
General
Details
Certificate Hierarchy
USERTrust RSA Certification Authority
InCommon RSA Server CA
www.cs.brown.edu
Certificate Fields
Issuer
Validity
Not Before
Not After
Subject
Subject Public Key Info
Subject Public Key Algorithm
Subject's Public Key
Field Value
CN = www.cs.brown.edu
O = Brown University
ST = Rhode Island
C = US

PKI hierarchy

In reality, PKI creates a hierarchy of trust:

- Root CAs: k_{pub} stored in virtually every browser, OS
  - Private keys protected by most stringent security measures (software, hardware, physical)
- Intermediate CAs: k_{pub} signed by root CA
  - Sign certificates for general use (ie, regular websites)
  - Doesn't require same protections as root
- General-use certificates: for a specific webserver

PKI hierarchy

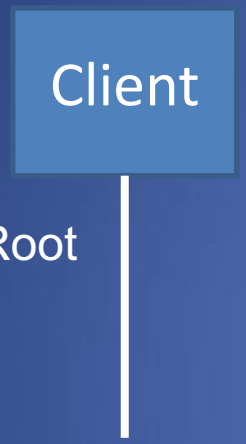
In reality, PKI creates a hierarchy of trust:

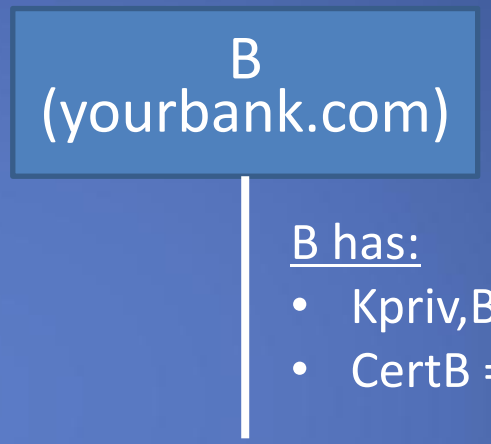
- Root CAs: k_{pub} stored in virtually every browser, OS
  - Private keys protected by most stringent security measures (software, hardware, physical)
- Intermediate CAs: k_{pub} signed by root CA
  - Sign certificates for general use (ie, regular websites)
  - Doesn't require same protections as root
- General-use certificates: for a specific webserver

What happens if a root is compromised?

How the hierarchy works

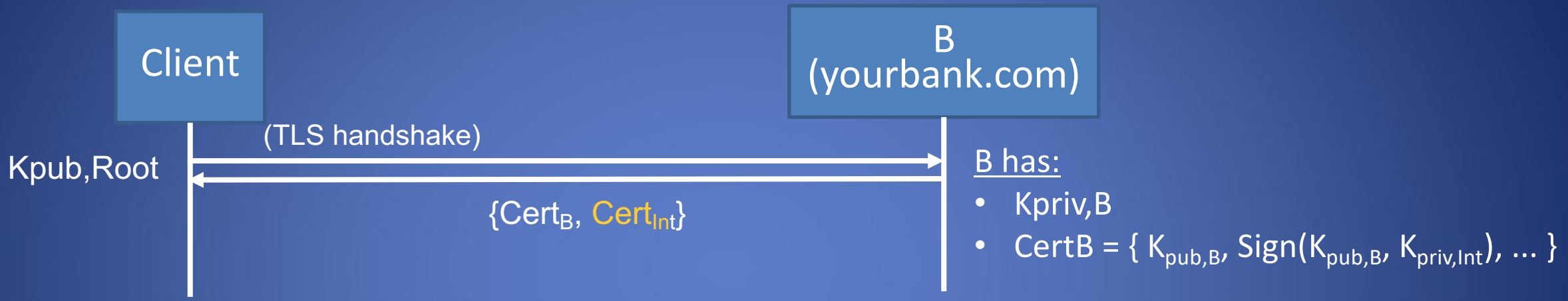
Ex. Server has certificate from Intermediate CA_{Int}





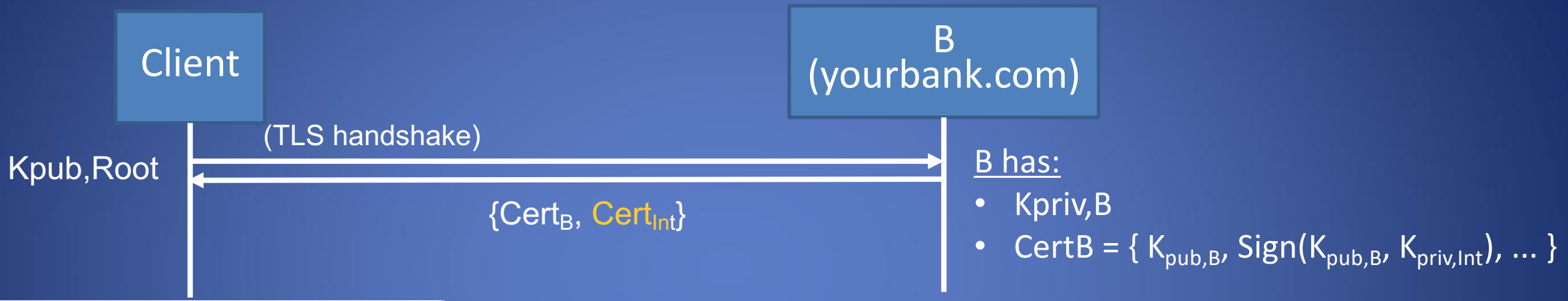
How the hierarchy works

Ex. Server has certificate from Intermediate CA_{Int}



How the hierarchy works

Ex. Server has certificate from Intermediate CA_{Int}

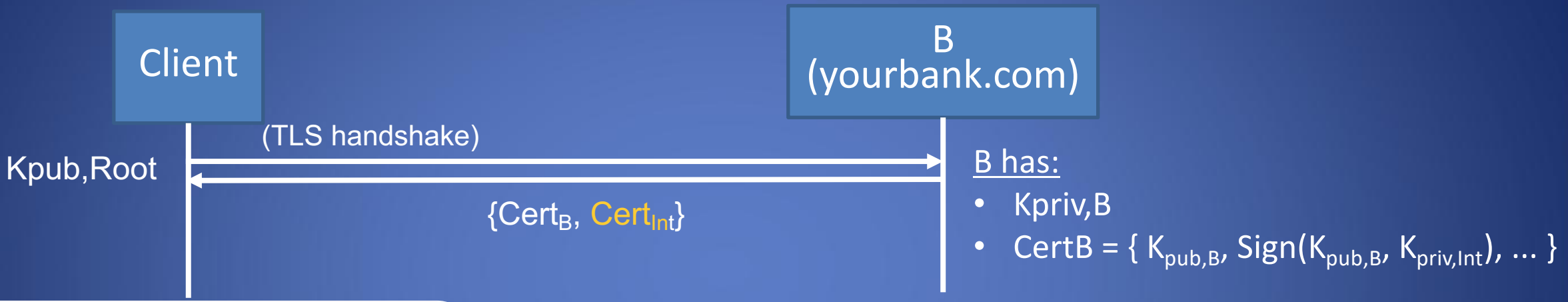


Client's workflow:

- Checks metadata ✅
- Verify($Cert_B$, $K_{pub,Int}$) ✅
- Verify($Cert_{Int}$, $K_{pub,Root}$) ✅

How the hierarchy works

Ex. Server has certificate from Intermediate CA_{Int}

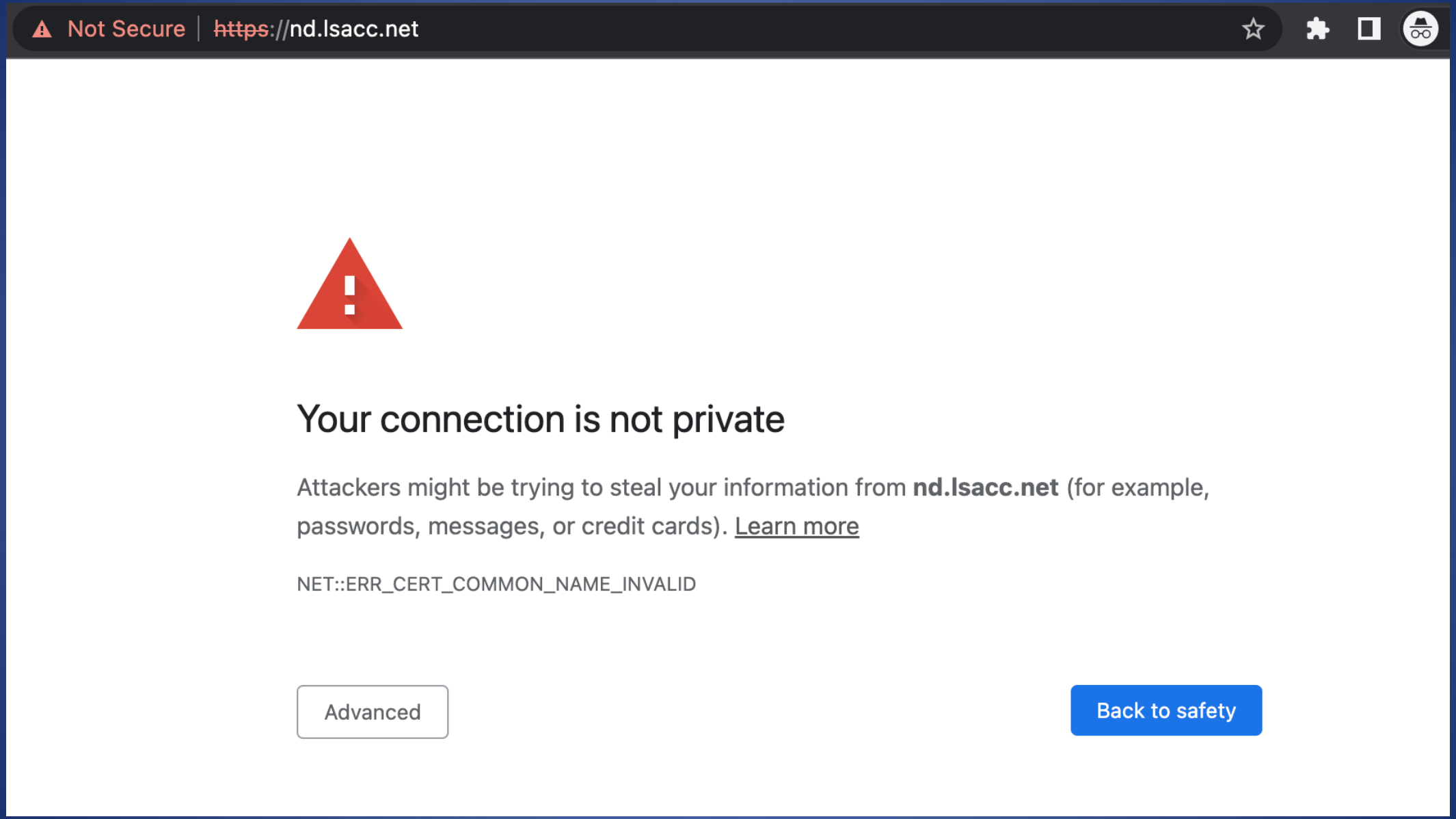


Client's workflow:

- Checks metadata ✅
- Verify($Cert_B$, $K_{pub,Int}$) ✅
- Verify($Cert_{Int}$, $K_{pub,Root}$) ✅

=> To verify integrity, need to verify certificates back to (trusted) root certificate

=> OK if verification passes and metadata correct: 🔒


Not Secure | https://nd.lsacc.net
Your connection is not private
Attackers might be trying to steal your information from nd.lsacc.net (for example, passwords, messages, or credit cards). Learn more
NET::ERR_CERT_COMMON_NAME_INVALID
Advanced
Back to safety

Most common TLS errors you might see

- Common name invalid
- Self-signed
- Certificate expired

When is it okay to click "proceed"? What happens if you do?

Most common TLS errors you might see

- Common name invalid
- Self-signed
- Certificate expired

When is it okay to click "proceed"? What happens if you do?

=> Might occur if webserver configured improperly, or if you're setting up a system

Rogue Certificates?

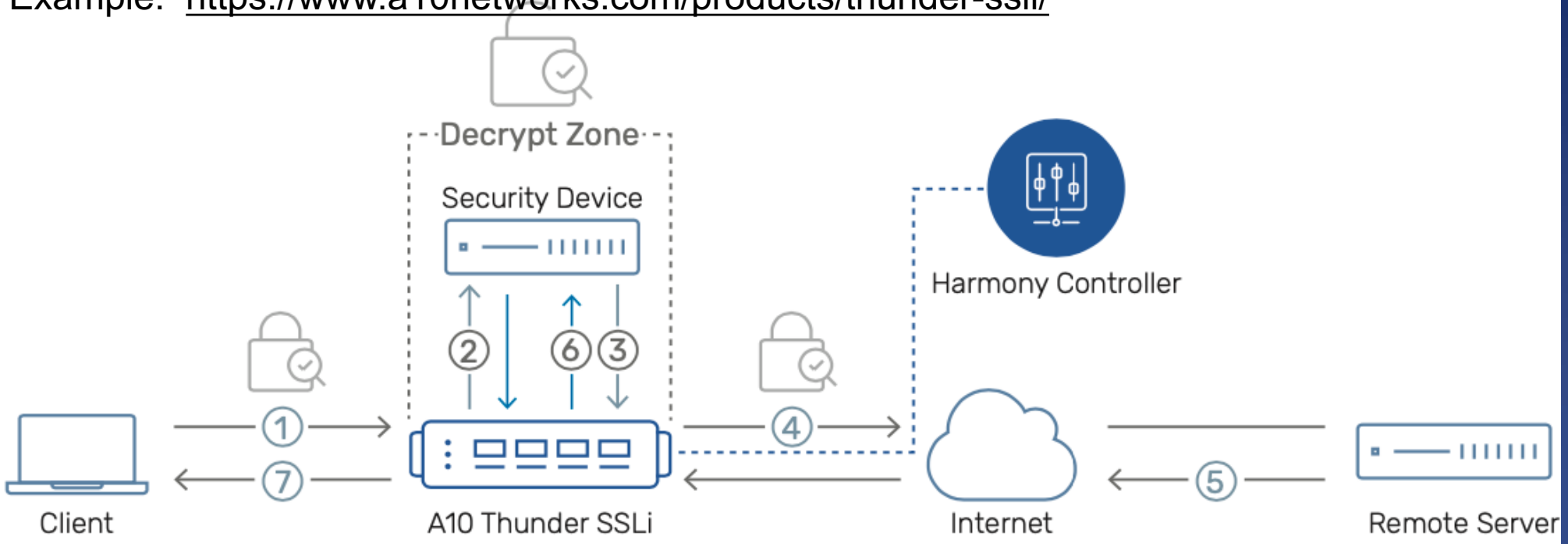
- In 2011, DigiNotar, a Dutch root certificate authority, was compromised
- The attacker created rogue certificates for popular domains like google.com and yahoo.com
- DigiNotar was distrusted by browsers and filed for bankruptcy
- See the incident investigation report by Fox-IT

- In 2017, Google questioned the certificate issuance policies and practices of Symantec
- Google's Chrome would start distrusting Symantec's certificates unless certain remediation steps were taken
- See back and forth between Ryan Sleevi (Chromium team) and Symantec
- The matter was settled with DigiCert acquiring Symantec's certificate business

TLS decryption

What happens when an organization wants to view TLS traffic on its network?

Example: https://www.a10networks.com/products/thunder-ssli/



1. Encrypted traffic from the client is intercepted by Thunder SSLi and decrypted.
2. Thunder SSLi sends the decrypted traffic to a security device, which inspects it in clear-text.
3. The security device, after inspection, sends the traffic back to Thunder SSLi, which intercepts and re-encrypts it.
4. Thunder SSLi sends the re-encrypted traffic to the server.
5. The server processes the request and sends an encrypted response to Thunder SSLi.
6. Thunder SSLi decrypts the response traffic and forwards it to the same security device for inspection.
7. Thunder SSLi receives the traffic from the security device, re-encrypts it and sends it to the client.

View SSL Certificates

- Browser can show certificate chain
- View OS's certificate keystore
  - MacOS: Keychain Access app
- Linux tools: openssl
  - E.g., inspect the brown.edu certificate

```
openssl s_client -connect brown.edu:443
```

What We Have Learned

- Goals of the SSL/TLS protocol
- SSL certificates, chain of trust, and revocation
- Overview of the SSL protocol
- DNSSEC