Lecture 28: Transport Layer Security (TLS)

Professor Adam Bates

CS 461 / ECE 422

Fall 2019

Goals for Today

- Learning Objectives:
  - Understand the fundamental building blocks of the Internet, specifically the Application Layer
  - Understand TLS
- Announcements, etc:
  - Vote on favorite u-pick-em lecture topic (Piazza)
  - Networking CP2: Due Nov 11
  - Final Exam *is* on Dec 13 at 7pm
  - Lecture slides from this week will be up soon
  - Midterm regrade requests, if you haven't already heard from me, will also be processed soon

Reminder: Please put away devices at the start of class

TCP Security Properties

	Passive	Off-Path	MitM
Availability	–	✗	✗
Confidentiality	✗	–	✗
Integrity	–	–	✗
Authenticity	–	✔	✗

- Availability is hard to achieve
  - No protocol-level defenses against bandwidth exhaustion
- Authenticity against off-path attacks depends on integrity of Internet routing *(not covered in this class)*

TCP Security Properties

	Passive	Off-Path	MitM
Availability	–	✗	✗
Confidentiality	✗	–	✗
Integrity	–	–	✗
Authenticity	–	✔	✗

- Can we use cryptography to defend even against the strongest MitM attacker?

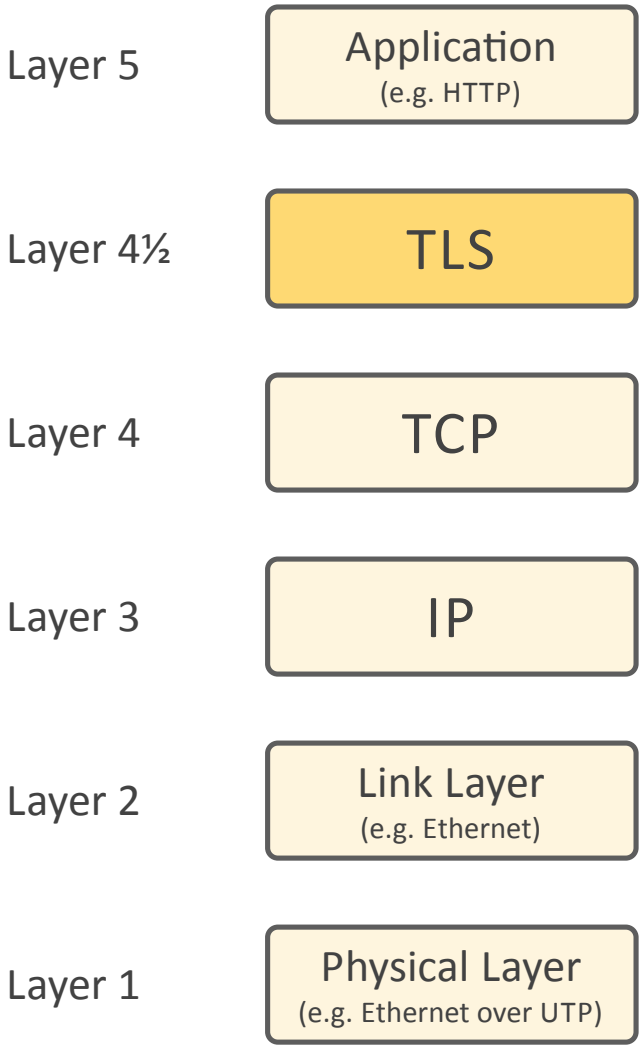
TCP Security Properties

	Passive	Off-Path	MitM
Availability	–	✗	✗
Confidentiality	✗	–	✗
Integrity	–	–	✗
Authenticity	–	✔	✗

- Can we use cryptography to defend even against the strongest MitM attacker? **Yes!**

Transport Layer Security (TLS)

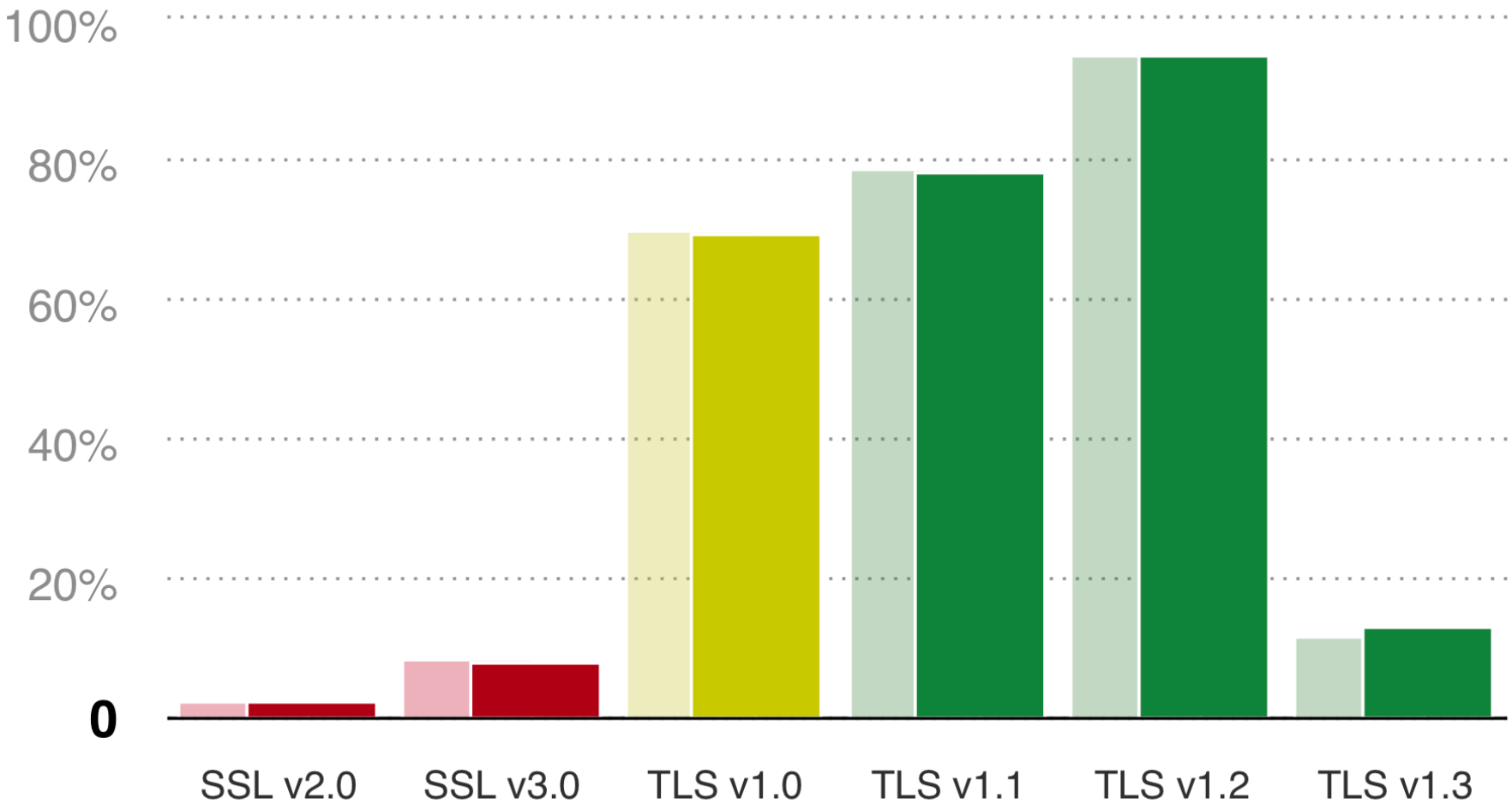
- TLS provides confidentiality, integrity, and authenticity of data sent over TCP
- Provides a data stream interface to application
  - Easy to add TLS support to applications that use TCP
- Uses TCP for transport
  - Datagram TLS (DTLS) runs on UDP, not used very much



History of SSL/TLS

- Secure Sockets Later (SSL) developed by Netscape designed to allow secure Web sessions
- Transport Layer Security (TLS) evolved from SSL and now replaces it
  - Current version of TLS is 1.3
  - Most widely used is TLS 1.2

Web Site Protocol Support



https://www.ssllabs.com/ssl-pulse/

Cryptography Toolbox

- Cryptographic hash functions
- Message Authentication Codes
- Symmetric encryption and decryption
- Asymmetric encryption and decryption
- Digital signatures
- Key exchange

Symmetric

Asymmetric

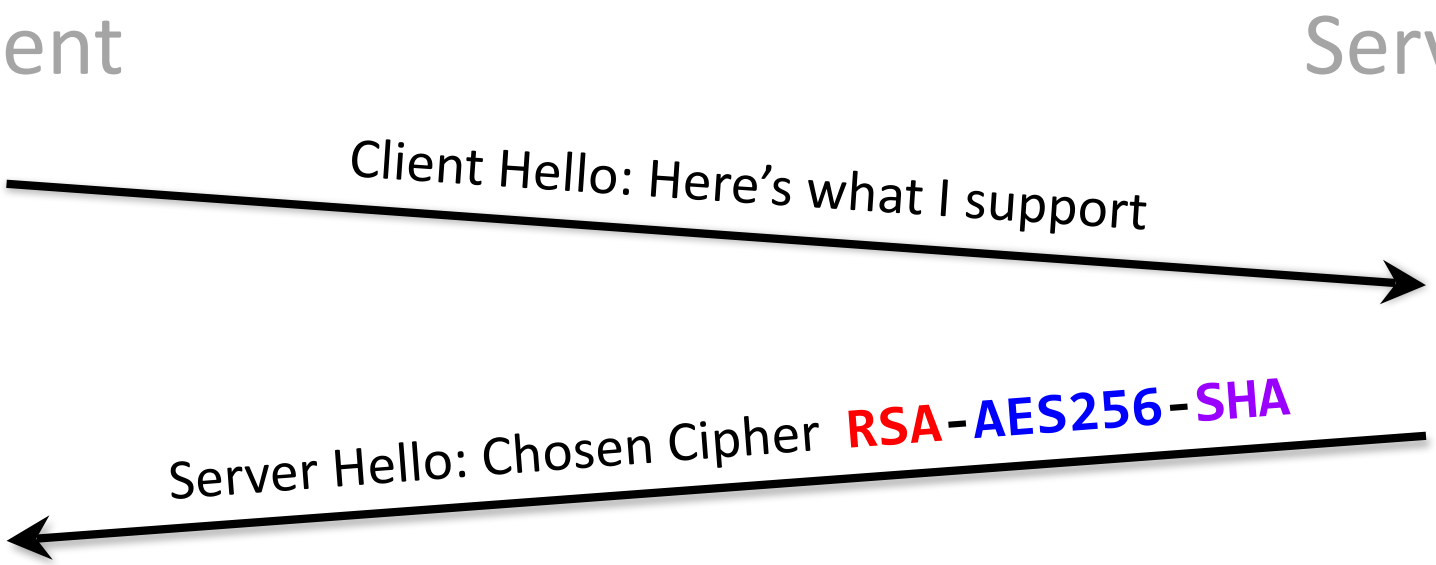
Basic Idea

- Based on *key agreement*
  - Establish a shared secret key using key agreement (e.g. Diffie-Hellman)
  - Authenticate shared secret (digital signatures scheme)
- Use shared secret for symmetric cryptography
  - Symmetric encryption and decryption
  - MACs

- Based on *key exchange*
  - Generate random session key, encrypt using peer's public key, and send to peer
  - Authenticate encrypted key (digital signature scheme)
- Use shared secret for symmetric cryptography
  - Symmetric encryption and decryption
  - MACs

Client Hello includes a nonce *(random number generated by client)*

Illustrated TLS connection with explanations:
`https://tls.ulfheim.net/`

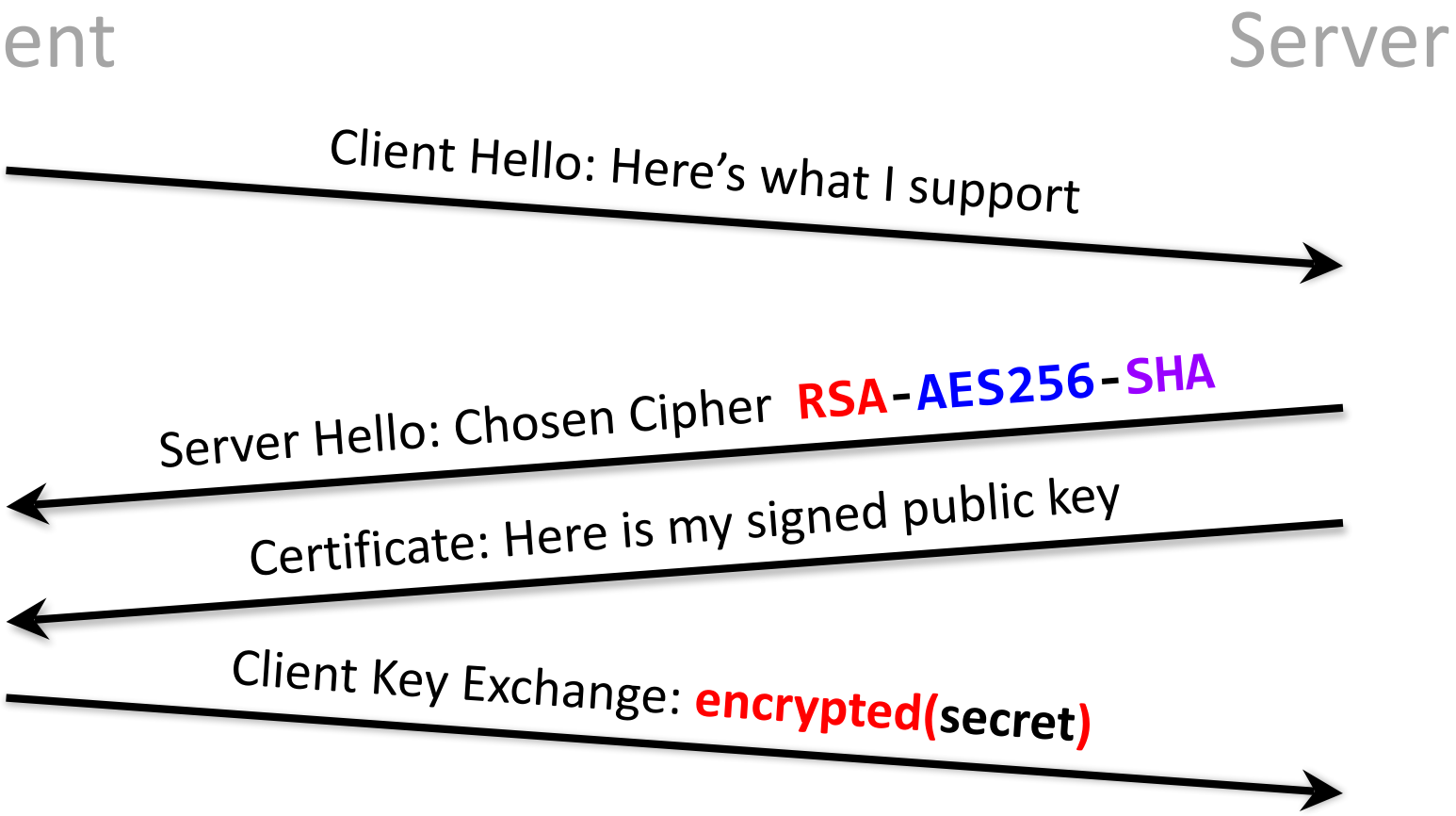


RSA-AES256-SHA

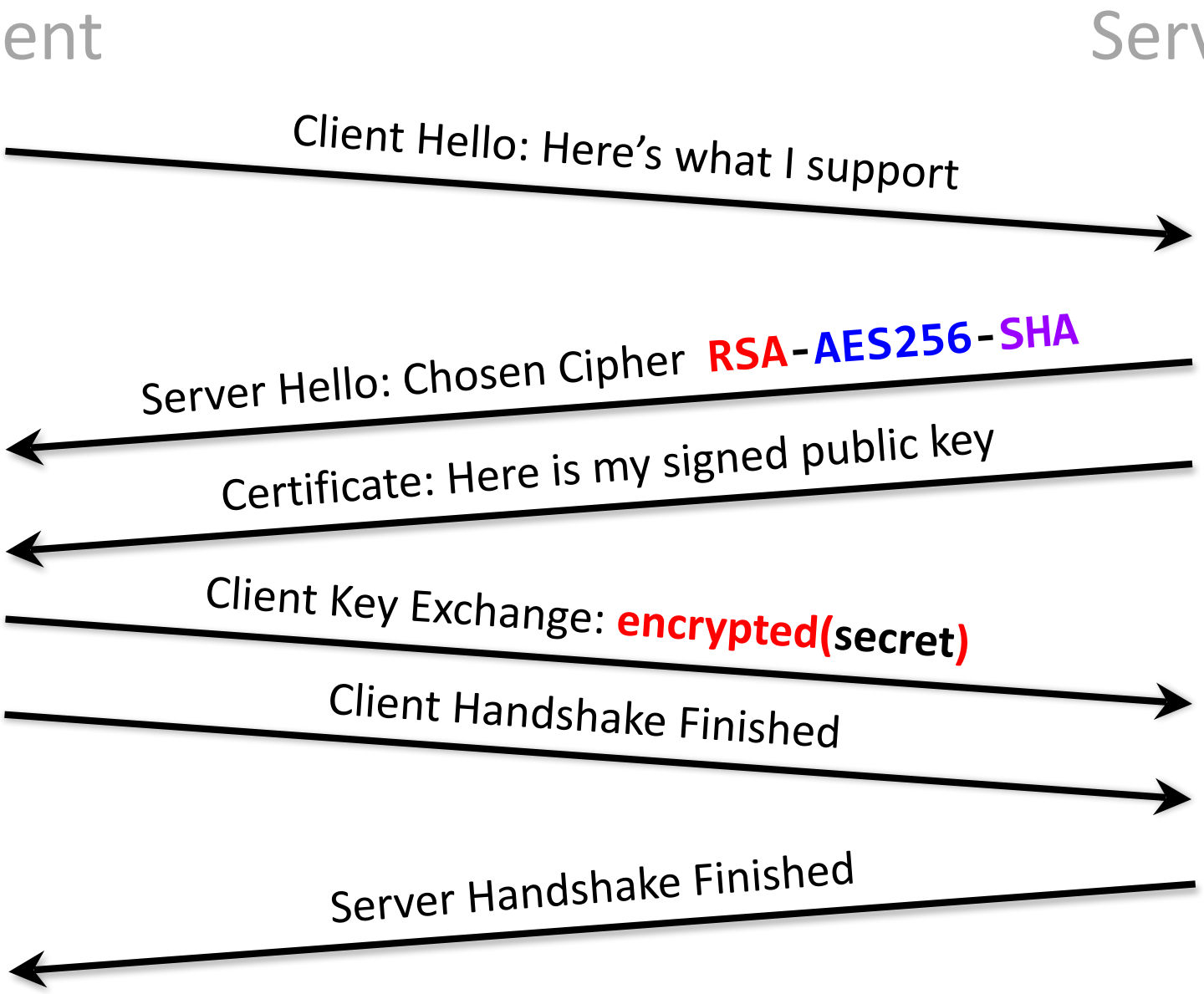
Key Exchange

Data Transfer Cipher

Message Digest / Authentication Code



Encrypted using Server's public key *(the public key from the Certificate)*
This means that *only the server* can decrypt the secret!



The Handshake Finished message contains a MAC of the handshake so far

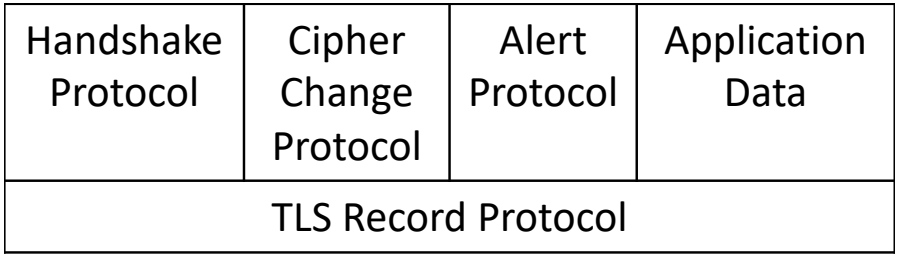
Why?

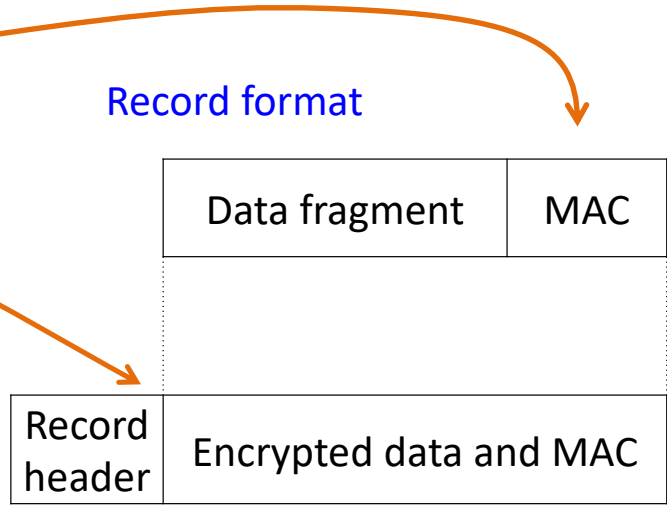
Session Keys

- Shared secret used to derive session keys
  - Symmetric key for data encryption and decryption
  - Symmetric key for MACs
- Session keys must be hard to guess
  - Pick shared secret uniformly at random
- All communication after handshake is encrypted and MAC'd using negotiated suite

TLS Protocol Stack

- All TLS data sent in discrete records handled by Record Protocol
  - Handshake protocol built on Record Protocol
  - Application data also sent via Record Protocol
- Each record has MAC
- Data + MAC encrypted
- During handshake *null* encryption is used

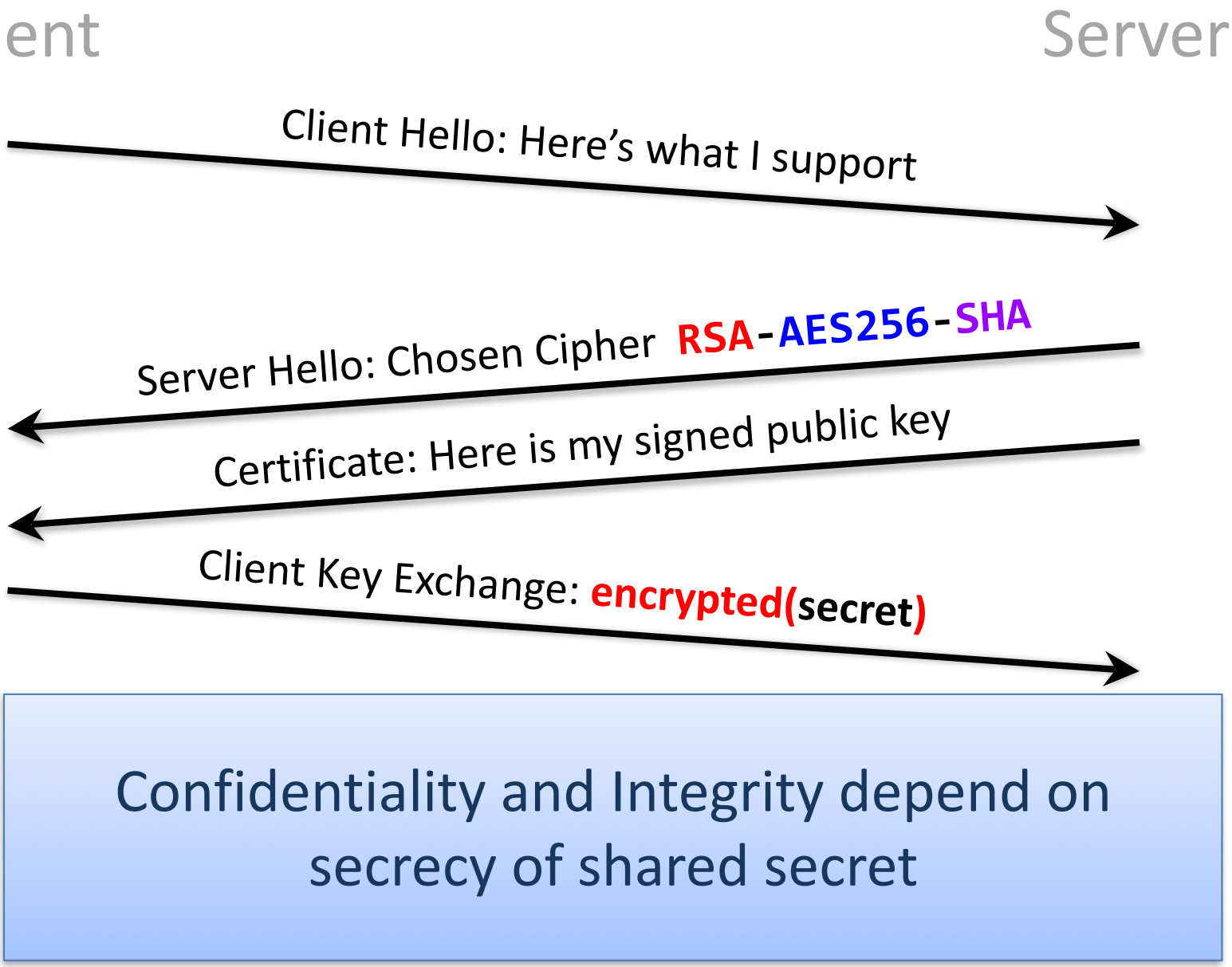




TLS Security Properties

	Passive	Off-Path	MitM
Availability	–	✗	✗
Confidentiality	✔	–	
Integrity	–	–	
Authenticity	–	✔	

- Encryption protects against passive adversary
- What about Man-in-the-Middle?



Encrypted using Server's public key *(the public key from the Certificate)*
This means only the server can decrypt the secret!

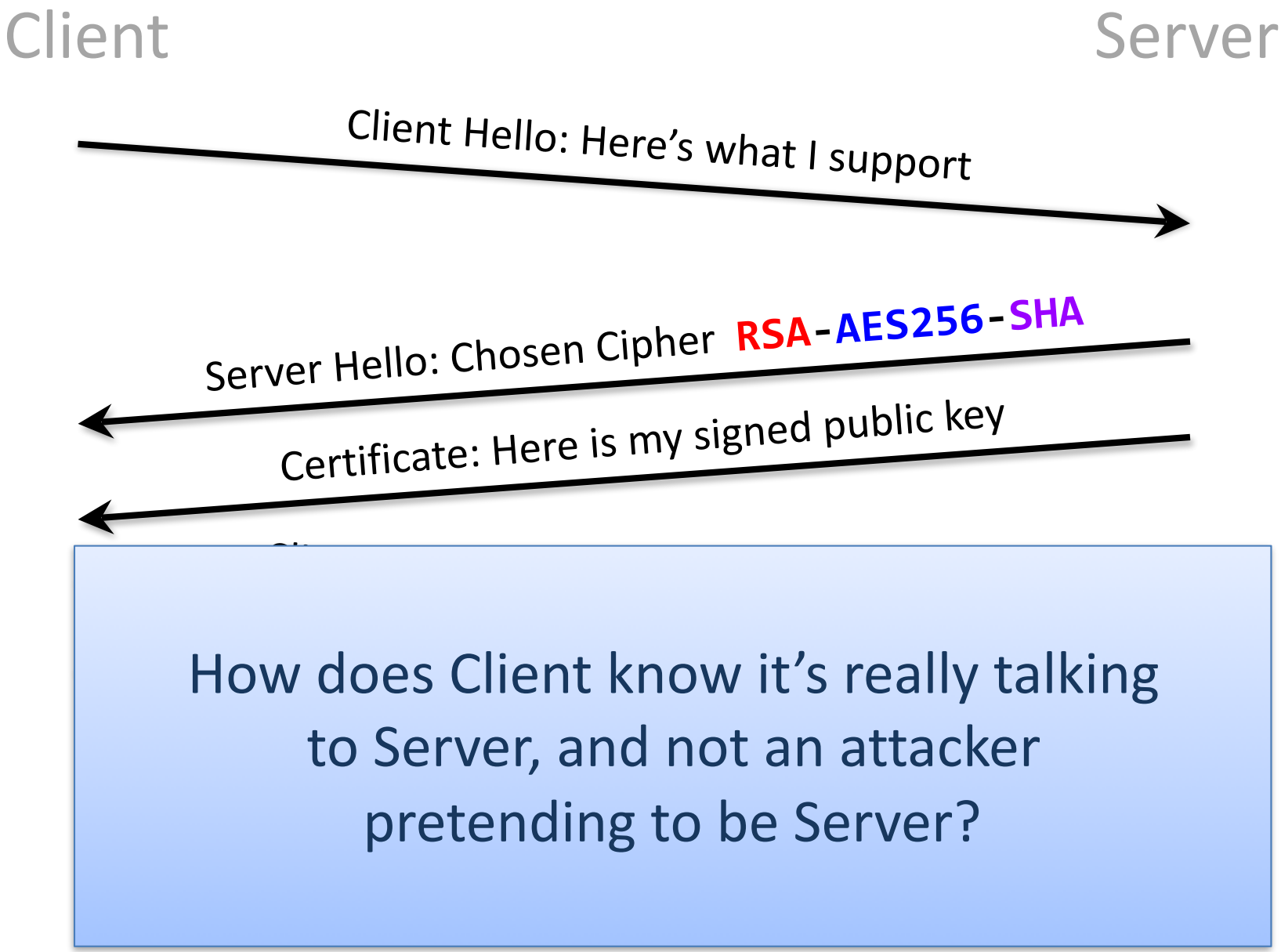
TLS Security Properties

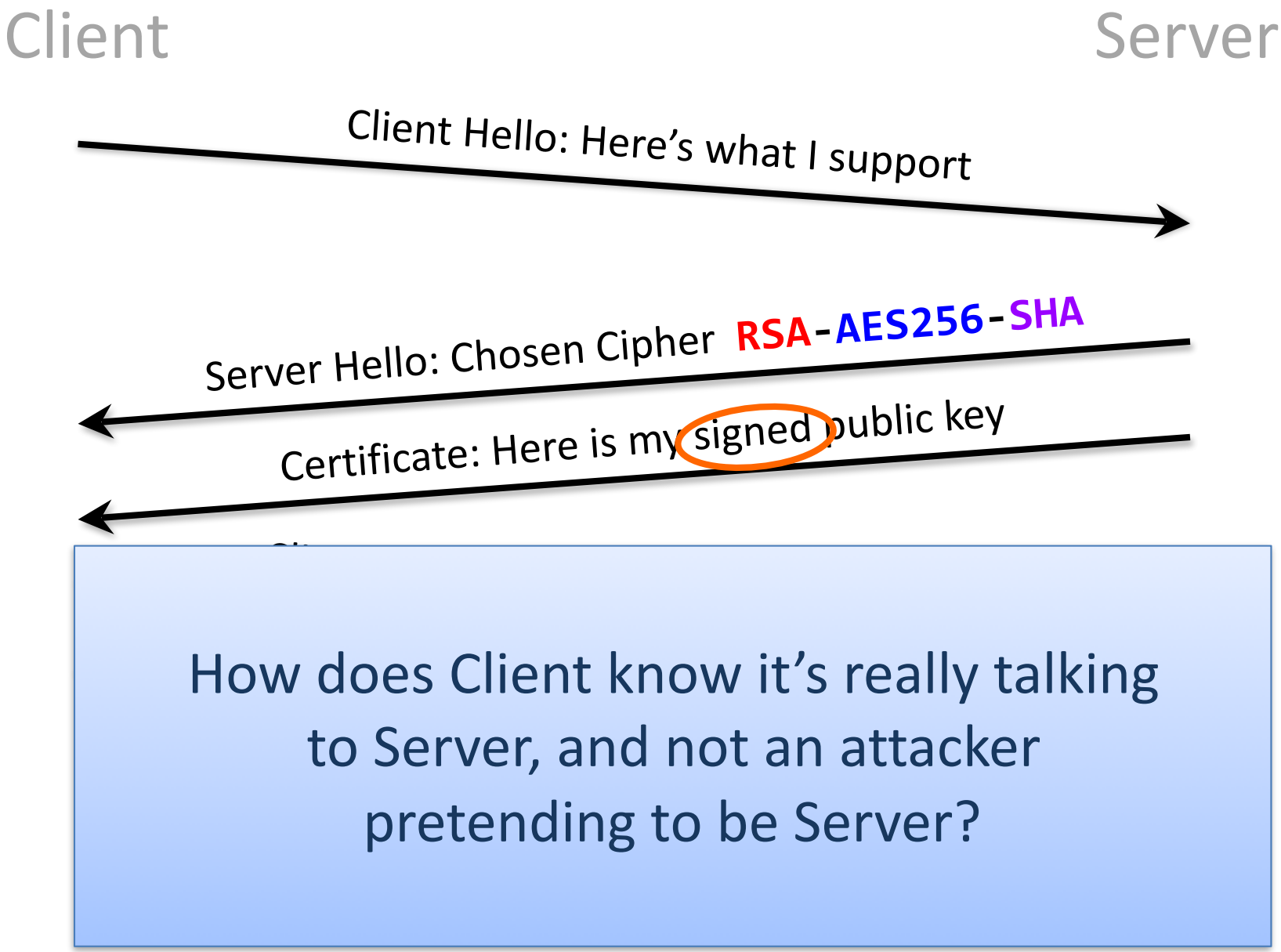
	Passive	Off-Path	MitM
Availability	–	✗	✗
Confidentiality	✔	–	✔
Integrity	–	–	✔
Authenticity	–	✔	

- Encryption protects against passive adversary
- What about Man-in-the-Middle?
- Shared secret is encrypted to server's public key, so only server can decrypt secret

Authenticity

- Cannot have integrity without *authenticity*
  - *Integrity without authenticity is meaningless!*
- In a key exchange/agreement scenario, cannot have confidentiality without *authenticity*
- *If Attacker can impersonate Server, then attacker can learn shared secret*
  - Can forge or modify messages from Server
  - Can decrypt messages intended for Server


Client
Server
Client Hello: Here's what I support
Server Hello: Chosen Cipher RSA-AES256-SHA
Certificate: Here is my signed public key
How does Client know it's really talking to Server, and not an attacker pretending to be Server?


Client
Server
Client Hello: Here's what I support
Server Hello: Chosen Cipher RSA-AES256-SHA
Certificate: Here is my signed public key
How does Client know it's really talking to Server, and not an attacker pretending to be Server?

Authenticating the Server

- Server *digitally signs* public key sent to Client
- Client can verify that signature is correct

Authenticating the Server

- Server *digitally signs* public key sent to Client
- Client can verify that signature is correct …

  … but how does client know correct public signing key of the server?

Authenticating the Server

- Server digitally signs public key sent to Client
- Client can verify that signature is correct if it knows the real public signing key of Server
- Server also sends public signing key to client
  - Key is not secret, so no risk if attacker sees it

Authenticating the Server

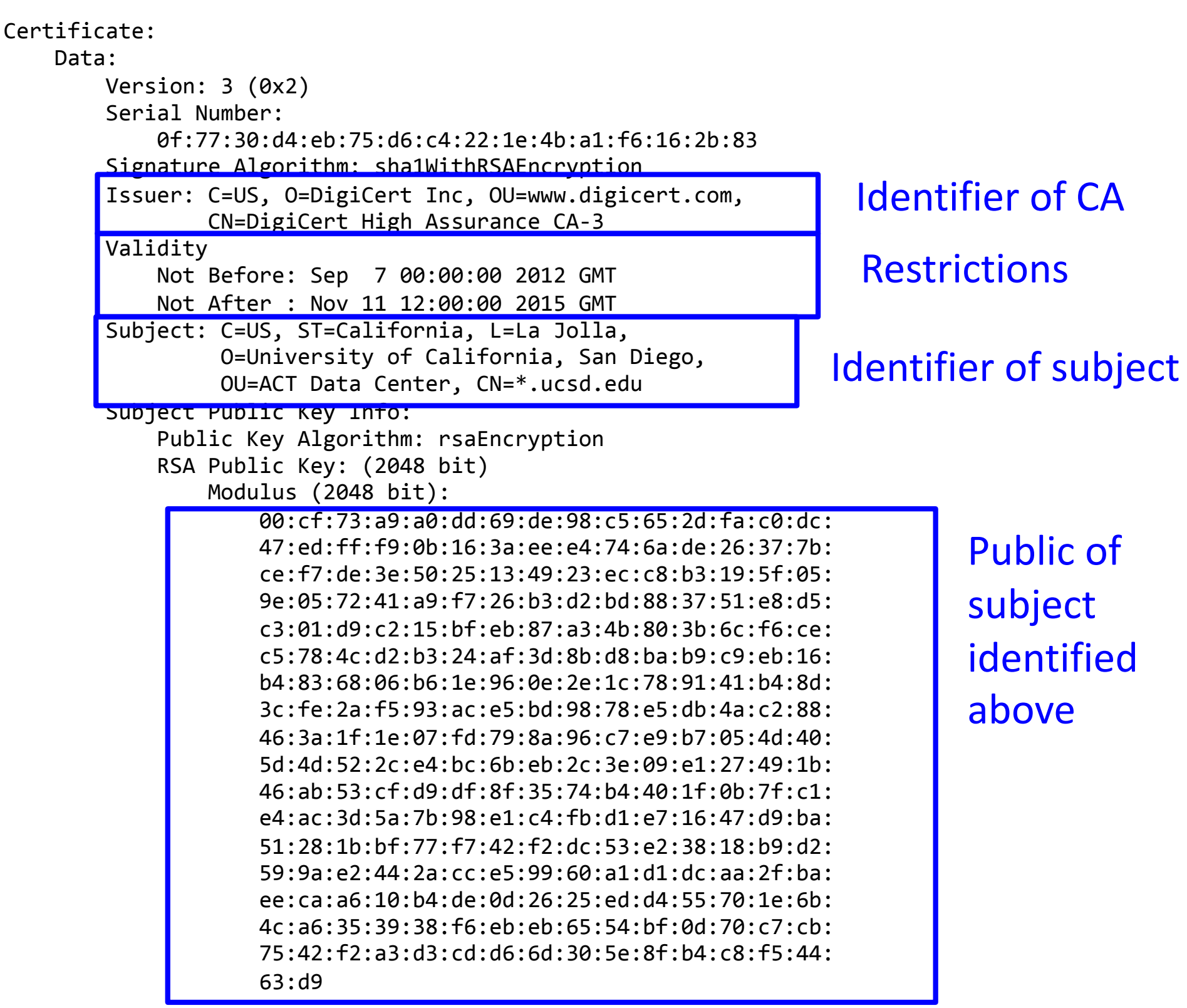
- Server digitally signs public key sent to Client
- Client can verify that signature is correct if it knows the real public signing key of Server
- Server also sends public signing key to client
  - Key is not secret, so no risk if attacker sees it
- But what if MitM attacker sends his/her own public signing key instead of server key?
- *Need to authenticate the public signing key*

Certificate Authority

- Server's public signing key is signed by a *trusted third party*, a Certificate Authority (CA)
- CA *verifies identity* of Server and signs Server's signing key using its own signing key
- Client must know CA's public signing key to verify validity of CA's signature. How?
  - Baked into Operating Systems
  - Baked into Web Browsers

Certificates

- A certificate is an identifier, a public key linked to the identifier, and restrictions on validity, all signed by a Certificate Authority
- TLS certificates follow X.509 standard
- In most TLS usage (e.g. for HTTPS), identifier is a host or domain name
  - Application must ensure that identifier in certificate identifies the host it wants to talk to
- Restrictions include expiration date


Certificate:
Data:
Version: 3 (0x2)
Serial Number:
0f:77:30:d4:eb:75:d6:c4:22:1e:4b:a1:f6:16:2b:83
Signature Algorithm: sha1WithRSAEncryption
Issuer: C=US, O=DigiCert Inc, OU=www.digicert.com,
CN=DigiCert High Assurance CA-3
Validity
Not Before: Sep 7 00:00:00 2012 GMT
Not After : Nov 11 12:00:00 2015 GMT
Subject: C=US, ST=California, L=La Jolla,
O=University of California, San Diego,
OU=ACT Data Center, CN=*.ucsd.edu
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public Key: (2048 bit)
Modulus (2048 bit):
00:cf:73:a9:a0:dd:69:de:98:c5:65:2d:fa:c0:dc:
47:ed:ff:f9:0b:16:3a:ee:e4:74:6a:de:26:37:7b:
ce:f7:de:3e:50:25:13:49:23:ec:c8:b3:19:5f:05:
9e:05:72:41:a9:f7:26:b3:d2:bd:88:37:51:e8:d5:
c3:01:d9:c2:15:bf:eb:87:a3:4b:80:3b:6c:f6:ce:
c5:78:4c:d2:b3:24:af:3d:8b:d8:ba:b9:c9:eb:16:
b4:83:68:06:b6:1e:96:0e:2e:1c:78:91:41:b4:8d:
3c:fe:2a:f5:93:ac:e5:bd:98:78:e5:db:4a:c2:88:
46:3a:1f:1e:07:fd:79:8a:96:c7:e9:b7:05:4d:40:
5d:4d:52:2c:e4:bc:6b:eb:2c:3e:09:e1:27:49:1b:
46:ab:53:cf:d9:df:8f:35:74:b4:40:1f:0b:7f:c1:
e4:ac:3d:5a:7b:98:e1:c4:fb:d1:e7:16:47:d9:ba:
51:28:1b:bf:77:f7:42:f2:dc:53:e2:38:18:b9:d2:
59:9a:e2:44:2a:cc:e5:99:60:a1:d1:dc:aa:2f:ba:
ee:ca:a6:10:b4:de:0d:26:25:ed:d4:55:70:1e:6b:
4c:a6:35:39:38:f6:eb:eb:65:54:bf:0d:70:c7:cb:
75:42:f2:a3:d3:cd:d6:6d:30:5e:8f:b4:c8:f5:44:
63:d9
Identifier of CA
Restrictions
Identifier of subject
Public of subject identified above

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            0f:77:30:d4:eb:75:d6:c4:22:1e:4b:a1:f6:16:2b:83
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, O=DigiCert Inc, OU=www.digicert.com,
                CN=DigiCert High Assurance CA-3
        Validity
            Not Before: Sep  7 00:00:00 2012 GMT
            Not After : Nov 11 12:00:00 2015 GMT
        Subject: C=US, ST=California, L=La Jolla,
                 O=University of California, San Diego,
                 OU=ACT Data Center, CN=*.ucsd.edu
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:cf:73:a9:a0:dd:69:de:98:c5:65:2d:fa:c0:dc:
                    47:ed:ff:f9:0b:16:3a:ee:e4:74:6a:de:26:37:7b:
                    ce:f7:de:3e:50:25:13:49:23:ec:c8:b3:19:5f:05:
                    9e:05:72:41:a9:f7:26:b3:d2:bd:88:37:51:e8:d5:
                    c3:01:d9:c2:15:bf:eb:87:a3:4b:80:3b:6c:f6:ce:
                    c5:78:4c:d2:b3:24:af:3d:8b:d8:ba:b9:c9:eb:16:
                    b4:83:68:06:b6:1e:96:0e:2e:1c:78:91:41:b4:8d:
                    3c:fe:2a:f5:93:ac:e5:bd:98:78:e5:db:4a:c2:88:
                    46:3a:1f:1e:07:fd:79:8a:96:c7:e9:b7:05:4d:40:
                    5d:4d:52:2c:e4:bc:6b:eb:2c:3e:09:e1:27:49:1b:
                    46:ab:53:cf:d9:df:8f:35:74:b4:40:1f:0b:7f:c1:
                    e4:ac:3d:5a:7b:98:e1:c4:fb:d1:e7:16:47:d9:ba:
                    51:28:1b:bf:77:f7:42:f2:dc:53:e2:38:18:b9:d2:
                    59:9a:e2:44:2a:cc:e5:99:60:a1:d1:dc:aa:2f:ba:
                    ee:ca:a6:10:b4:de:0d:26:25:ed:d4:55:70:1e:6b:
                    4c:a6:35:39:38:f6:eb:eb:65:54:bf:0d:70:c7:cb:
                    75:42:f2:a3:d3:cd:d6:6d:30:5e:8f:b4:c8:f5:44:
                    63:d9
```

Public of subject identified above

Meaning of Digital Signatures

- **Signing** is a mechanical operation that has *no meaning in itself.*
- What Cryptography promises:
  Only someone who knows the private key can create a signature that verifies using the corresponding public key
- Meaning of a signature is a matter of convention
- Both signer and verifier must agree on meaning
  - What is the meaning of a signature in a certificate?

Meaning of Certificates

- **Certificate meaning:** Issuer attests that subject identified in certificate is one that has private key corresponding to public key in certificate
  - How does CA check that requester has private key?

Certificate Signing

- **Certificate meaning:** Issuer attests that subject identified in certificate is one that has private key corresponding to public key in certificate
  - How does CA check that requester has private key?
- Subject sends Certificate Signing Request (CSR) to CA
  - CSR contains info (including subject identifier) that will be included certificate
  - **CSR is signed by requester:** prevents someone who does not have the corresponding private key from requesting a certificate with their own name

Certificate Chaining

- A certificate may be signed by an intermediate CA whose public key is signed by another CA
- Certificates can be chained
  - An initial trusted public signing key can be used to sign a certificate X containing public key K_X that is then used to sign another certificate Y, and so on
- Application using TLS must have a set of trusted CA's public keys, the *root of trust*
  - Root of trust supplied by application, not TLS!

Hypothetical Scheme

- What if CA only checks that requester has public key (but still verifies identity)?

Hypothetical Scheme

- What if CA only checks that requester has public key (but still verifies identity)?
- Anyone would be able to request a certificate with their own name but someone else's public key — What's wrong with this?

Hypothetical Scheme

- What if CA only checks that requester has public key (but still verifies identity)?
- Anyone would be able to request a certificate with their own name but someone else's public key — What's wrong with this?
- Such an attacker would **not** be able to decrypt messages sent to real owner of private key
  - Attacker does not actually have private key
  - Would not be able to complete TLS handshake

Hypothetical Scheme

- What if anyone could request a certificate with their own name but someone else's public key?
- Attacker would **not** be able to decrypt messages sent to real owner of corresponding private key
- Attacker would **not** be able to sign messages that appear to be from real owner

Hypothetical Scheme

- What if anyone could request a certificate with their own name but someone else's public key?
- Attacker would **not** be able to decrypt messages sent to real owner of corresponding private key
- Attacker would **not** be able to sign messages that appear to be from real owner
- *Attacker could claim that messages signed by real owner were signed by him*

Hypothetical Scheme

- **What if someone could request a certificate with their someone else's name but their own public key?**

Hypothetical Scheme

- **What if someone could request a certificate with their someone else's name but their own public key?**
- Attacker **would** be able to decrypt messages sent to real owner of corresponding private key
- Attacker **would** be able to sign messages that appear to be from real owner

Hypothetical Scheme

- **What if someone could request a certificate with their someone else's name but their own public key?**
- Attacker **would** be able to decrypt messages sent to real owner of corresponding private key
- Attacker **would** be able to sign messages that appear to be from real owner

[2011]

[2011]

[2013]

TLS Security Properties

	Passive	Off-Path	MitM
Availability	–	✗	✗
Confidentiality	✔	–	✔
Integrity	–	–	✔
Authenticity	–	✔	✔

- Encryption protects against passive adversary
- Confidentiality, Integrity, and Authenticity depend on integrity of the Certificate Authority system *and* correct certificate validation by TLS implementation and application

Conclusions

- TLS is a highly effective tool for providing communications security, but there are caveats:
  - Weaknesses in underlying cryptographic protocols
  - Weaknesses in protocol implementation
  - Weaknesses in Public Key Infrastructure (i.e., CAs) that make it all possible.
    - Very limited accountability
    - CA's disincentivized to do job well
    - Extremely difficult to walk back a mistake