Lecture 23: Security
Authentication, TLS/SSL

COMP 332, Spring 2018

Victoria Manfredi

WESLEYAN UNIVERSITY

Acknowledgements: materials adapted from Computer Networking: A Top Down Approach 7th edition: ©1996-2016, J.F Kurose and K.W. Ross, All Rights Reserved as well as from slides by Abraham Matta at Boston University, and some material from Computer Networks by Tannenbaum and Wetherall.

Today

1. Announcements
   – hw9 due Wed. at 11:59p

2. Network security
   – message integrity

3. Transport layer security
   – overview
   – toy tls/ssl
   – real tls/ssl

Network Security

MESSAGE INTEGRITY

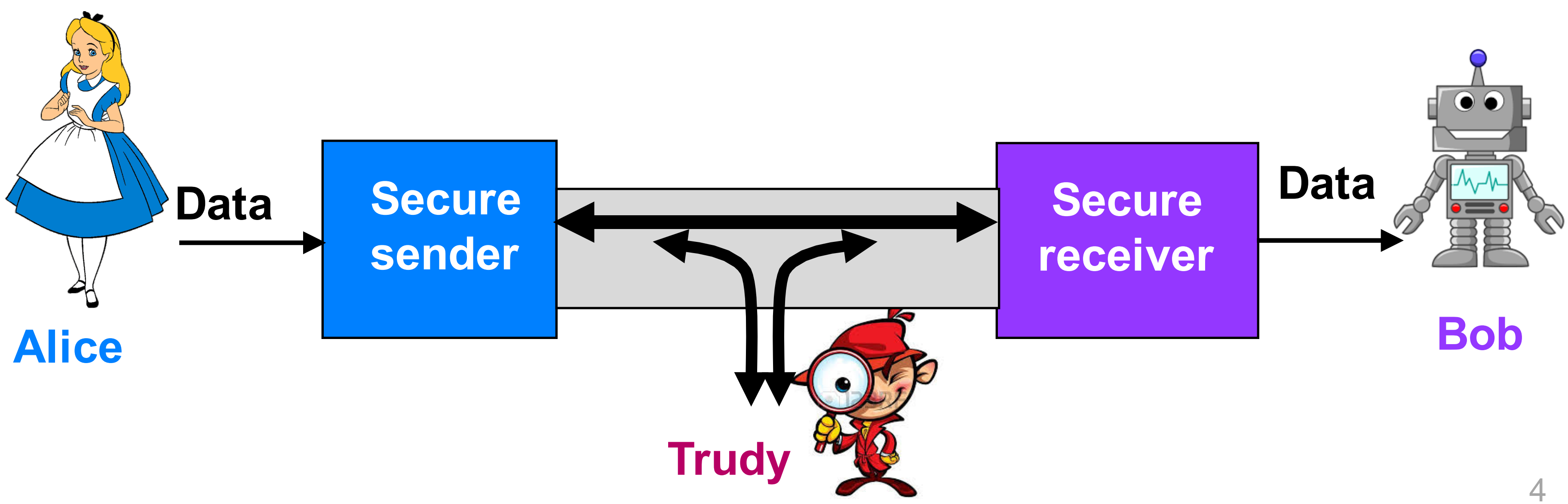
Message integrity

Alice and Bob must be able to detect whether msg changed

1. verify msg originated from Alice
2. verify msg not tampered with on way to Bob

Solution

– digital signatures: cryptographic technique like hand-written signature



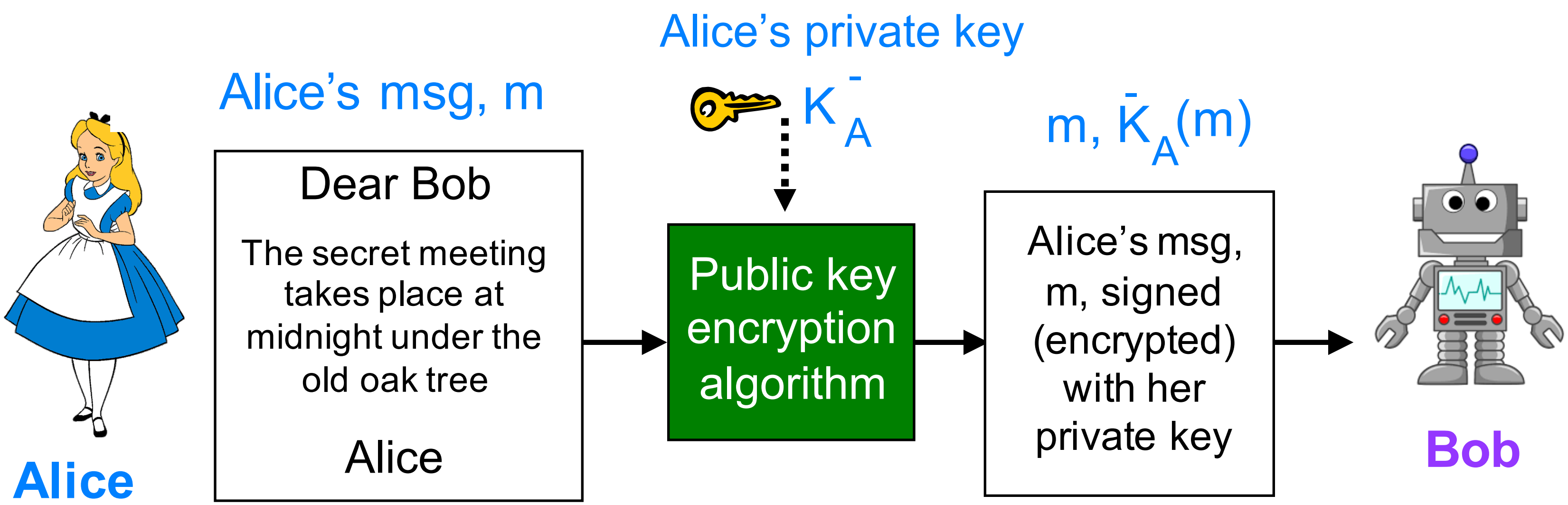
Simple digital signature for message, m

Sender (Alice)

- encrypts message m with her private key K_A^-
- creates "signed" message, $K_A^-(m)$
- proves she is owner/creator

Recipient (Bob)

- applies Alice's public key K_A^- to $K_A^+(m)$
- if $K_A^+(K_A^-(m)) = m$ whoever signed m must have used Alice's private key
- can prove only Alice could have signed document



Problem

Public key cryptography is expensive

- more expensive the longer the message is

Solution

- sign digital ``fingerprint” of msg rather than msg itself

Message digest

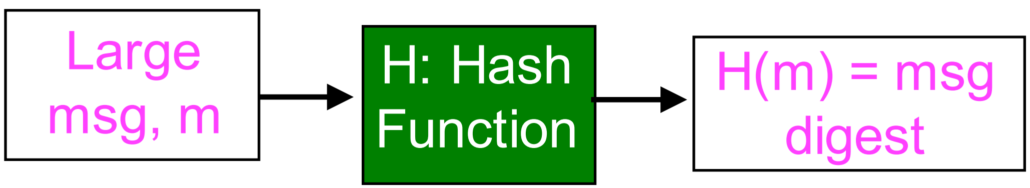
Message digest

Desired features

- fixed-length
- easy- to-compute
- 2 msgs unlikely to have same digest

Use a hash function

Apply hash function H to m



Hash function properties

- many-to-1 function
- produces fixed-size msg digest, H(m)
- given message digest H(m), computationally infeasible to find m’ such that H(m) = H(m’)

Some hash function standards

MD5 hash function (RFC 1321)

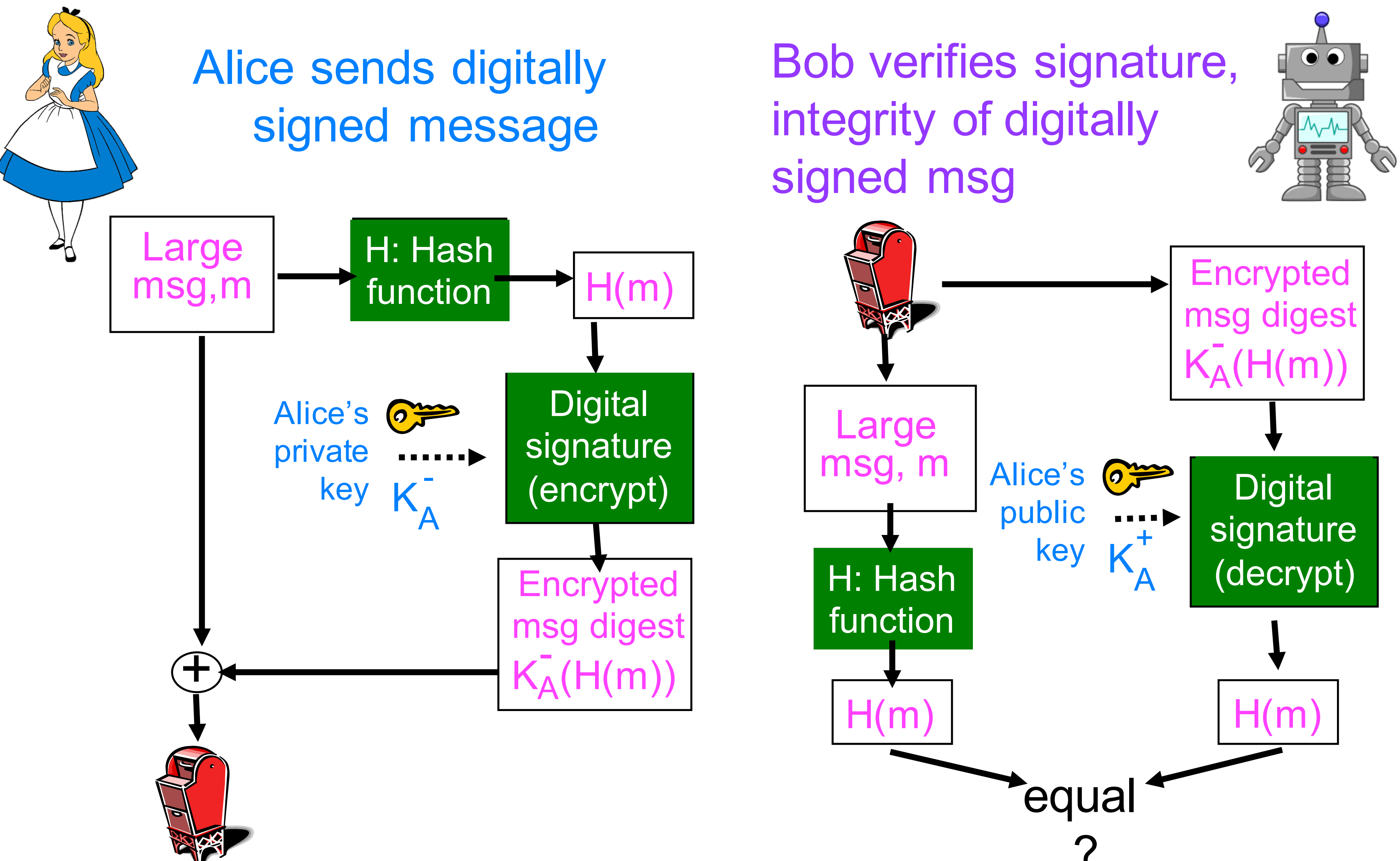
- computes 128-bit message digest in 4-step process.
- “cryptographically broken and unsuitable for further use”
  - CMU Software engineering Institute

SHA-1

- 160-bit message digest
- many vulnerabilities, browsers will no longer use/accept

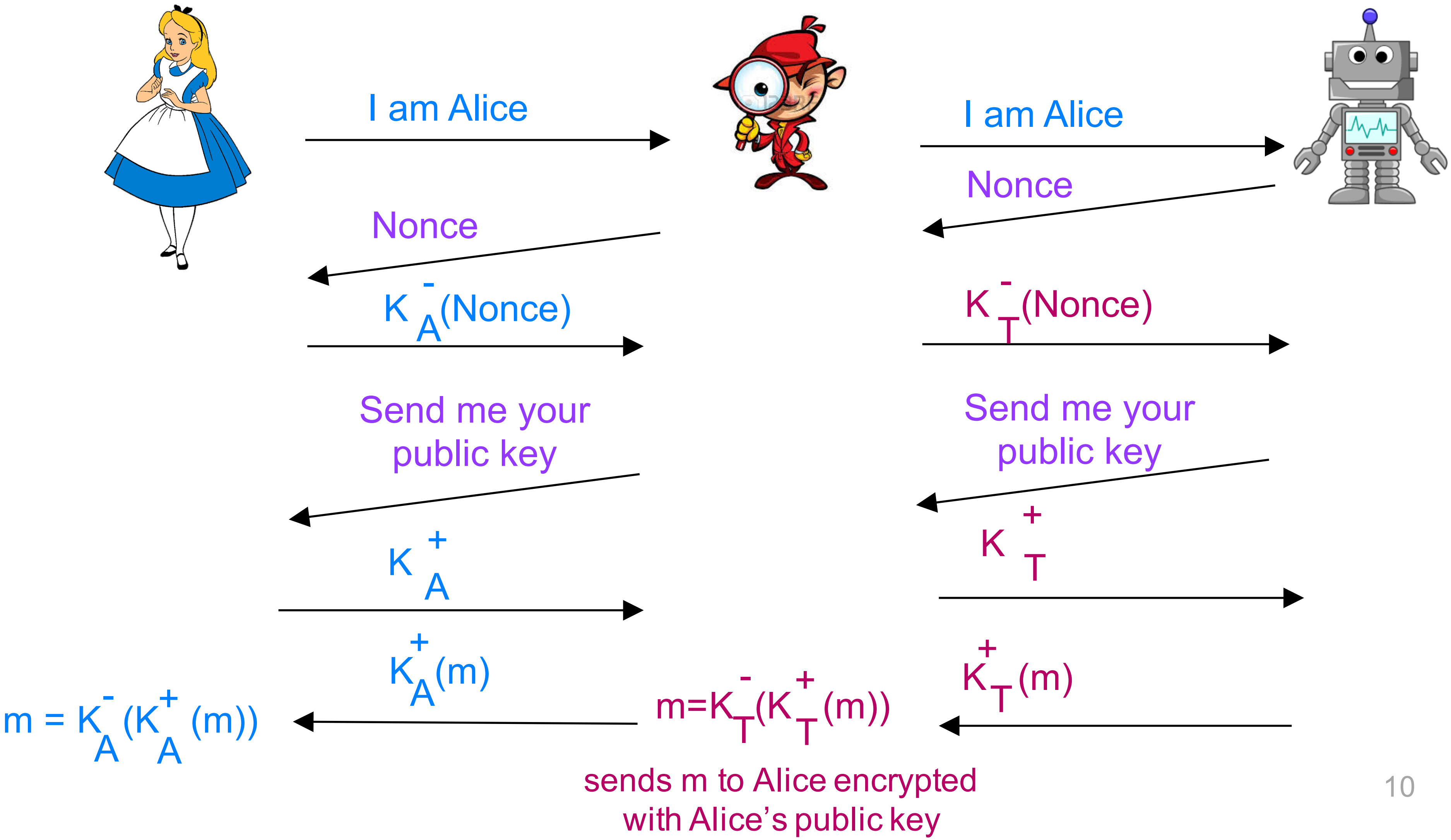
SHA-2, SHA-3

Use signed message digest as digital signature



Recall: ap5.0 man-in-the-middle attack

Trudy poses as Alice (to Bob) and as Bob (to Alice)



Problem

How do we make sure Bob can distinguish Alice's public key from Trudy's public key?

Use certification authority (CA)

- binds public key to particular entity
  - e.g., Alice, Bob, website, …
- 100s of certification authorities

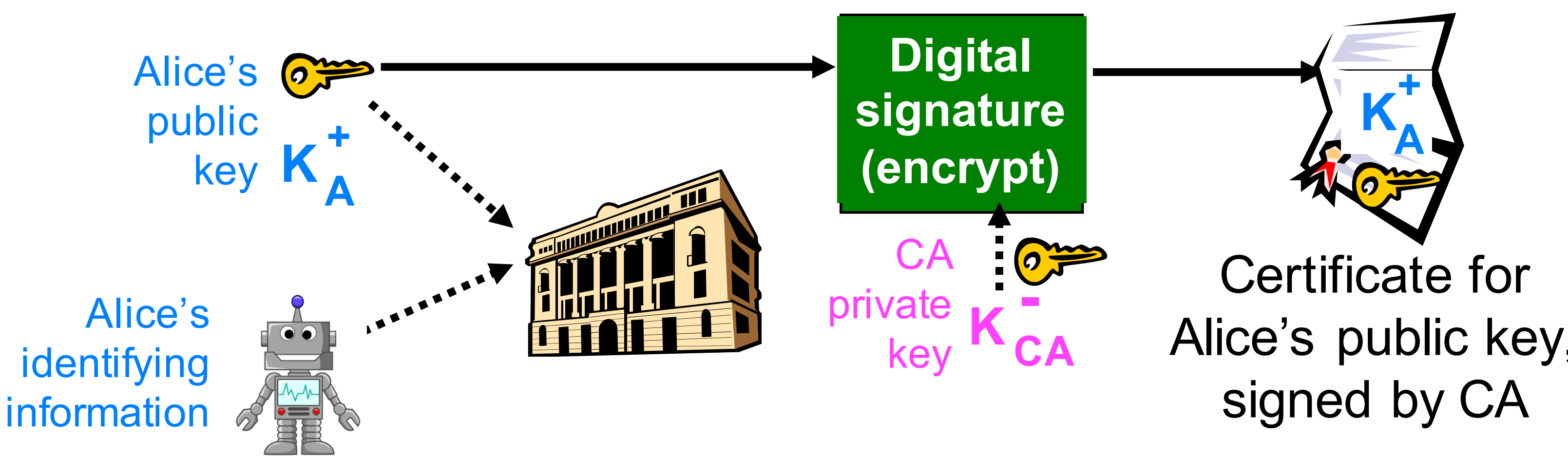
Aside

- CAs are critical but potentially weak link …

How certification authorities work

Alice registers her public key with CA

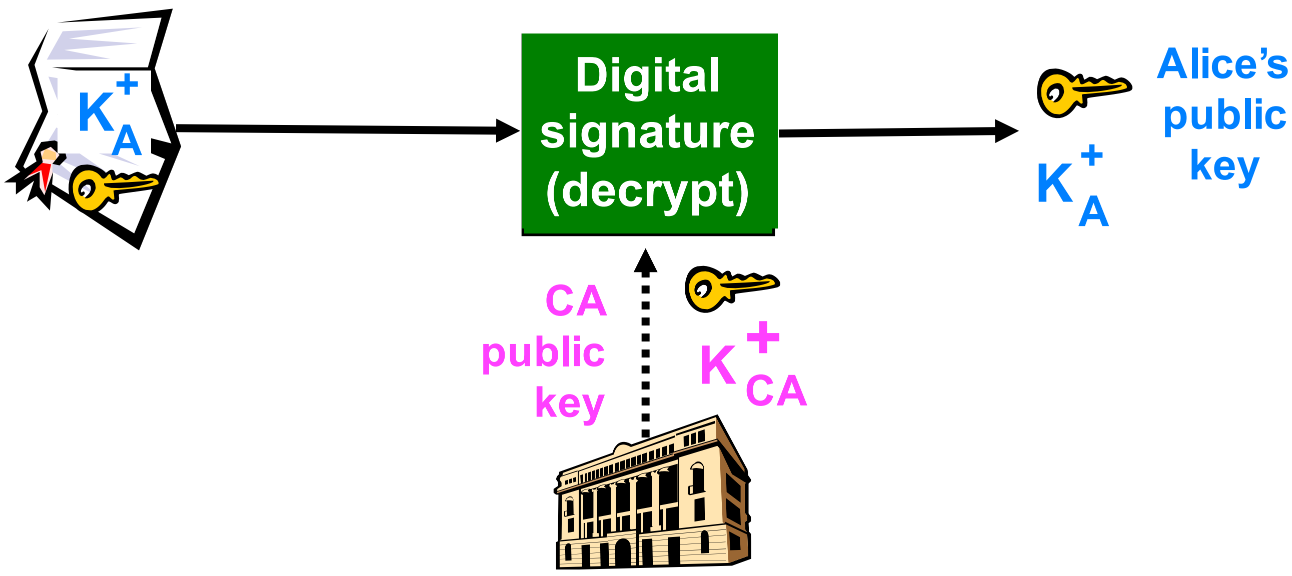
- Alice provides "proof of identity" to CA
- CA creates certificate binding Alice to its public key
- certificate containing Alice's public key digitally signed by CA
  - CA says "this is Alice's public key"



Certification authorities

When Bob wants Alice's public key

- gets Alice's certificate (from Alice or elsewhere)
- applies CA's public key to Alice's certificate, gets Alice's public key



Example

VeriSign Class 3 Public Primary Certification Authority - G5
↳ Symantec Class 3 EV SSL CA - G3
↳ www.bankofamerica.com

www.bankofamerica.com
Issued by: Symantec Class 3 EV SSL CA - G3
Expires: Thursday, July 26, 2018 at 7:59:59 PM Eastern Daylight Time
This certificate is valid

▼ **Details**

Subject Name	
Inc. Country	US
Inc. State/Province	Delaware
Business Category	Private Organization
Serial Number	2927442
Country	US
Postal Code	60603
State/Province	Illinois
Locality	Chicago
Street Address	135 S La Salle St
Organization	Bank of America Corporation
Organizational Unit	eComm Network Infrastructure
Common Name	www.bankofamerica.com
Issuer Name	
Country	US
Organization	Symantec Corporation
Organizational Unit	Symantec Trust Network
Common Name	Symantec Class 3 EV SSL CA - G3
Serial Number	4E 49 91 F1 B7 6A 9D 8D 16 23 5F 38 81 DD F5 E1
Version	3
Signature Algorithm	SHA-256 with RSA Encryption (1.2.840.113549.1.1.11)
Parameters	none
Not Valid Before	Monday, July 24, 2017 at 8:00:00 PM Eastern Daylight Time
Not Valid After	Thursday, July 26, 2018 at 7:59:59 PM Eastern Daylight Time
Public Key Info	
Algorithm	RSA Encryption (1.2.840.113549.1.1.1)

Transport Layer Security

OVERVIEW

TLS aka SSL

Secures data at and above transport layer

- SSL: Secure Sockets Layer, predecessor to TLS
- TLS: Transport Layer Security

Available to all TCP applications

- first setup TCP connection, then run TLS as appliction

Widely deployed

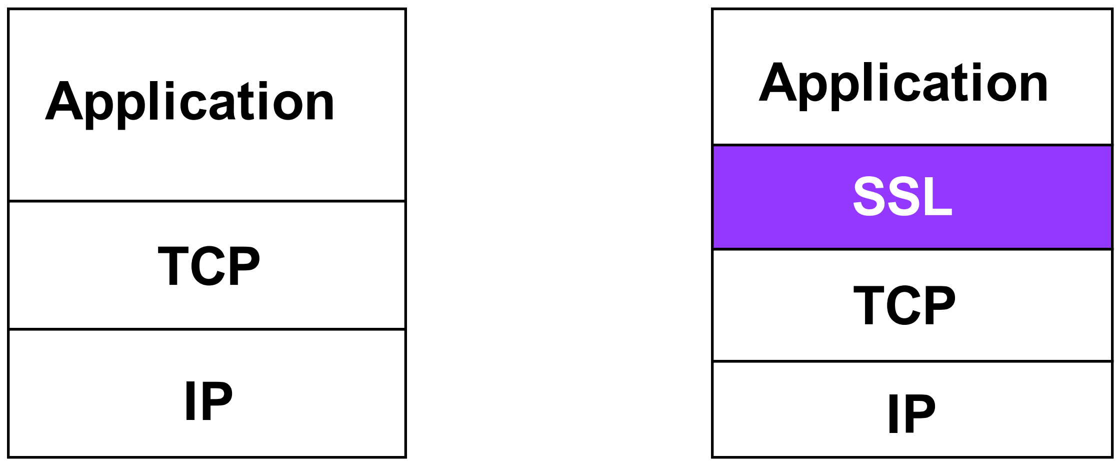
- supported by almost all browsers, web servers
- billions $/year over SSL
- HTTP + SSL = https

Provides

- confidentiality, integrity, authentication

Where SSL sits in Internet stack

SSL provides application programming interface to apps



Normal application

Application with SSL

Very likely your operating system using open source library

- https://www.openssl.org/
- https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS

SSL goals

Send byte streams & interactive data

- why?

Want set of secret keys for entire connection

- why?

Want certificate exchange as part of protocol handshake phase

- why?

Transport Layer Security

TOY TLS/SSL

A simple secure channel

Handshake

– Alice and Bob use their certificates, private keys to authenticate each other and exchange shared secret

Key derivation

– Alice and Bob use shared secret to derive set of keys

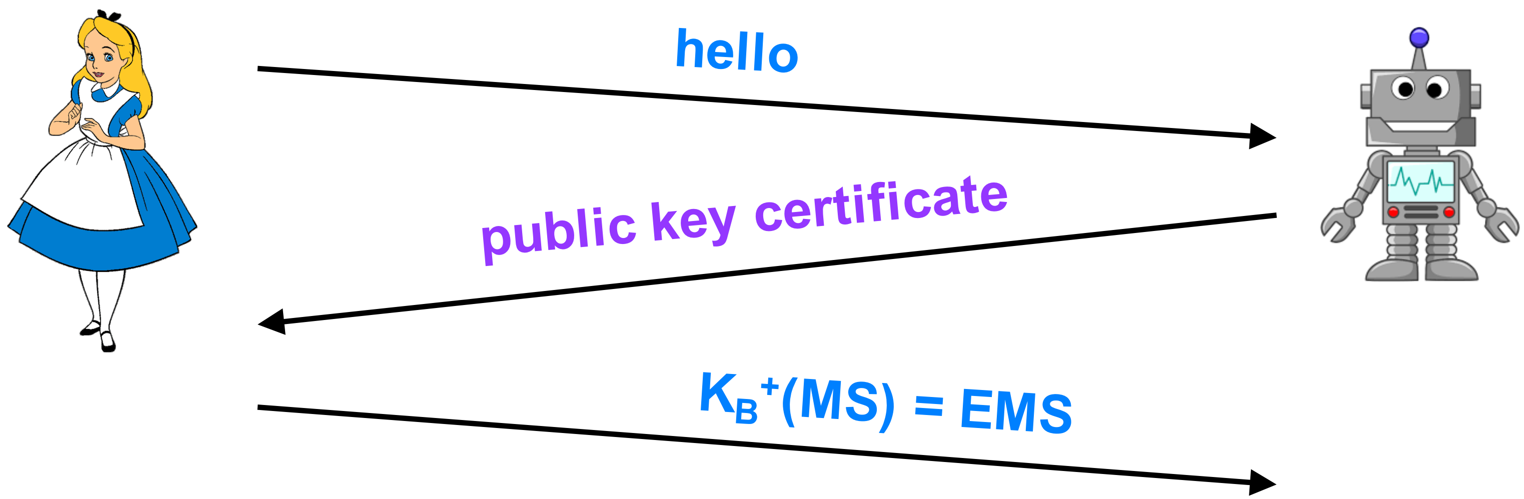
Data transfer

– data to be transferred is broken up into series of records

Connection closure

– special messages to securely close connection

A simple handshake



MS: master secret

EMS: encrypted master secret

Key derivation

Don't use same key for more than one cryptographic operation

Use different keys

- message authentication code (MAC): like hash
- encryption

4 keys

- K_c = encryption key for data sent from client to server
- M_c = MAC key for data sent from client to server
- K_s = encryption key for data sent from server to client
- M_s = MAC key for data sent from server to client

Keys derived from master secret

- use key derivation function (KDF)
  - takes master secret and additional random data and creates keys

Data records

Why not encrypt data in constant stream as we write it to TCP?

- where to put MAC?
  - if at end, no message integrity until all data processed
- e.g., instant messaging
  - how can we do integrity check over all bytes sent before displaying?

Solution: break stream in series of records

- each record carries MAC
- receiver can act on each record as it arrives

What if attacker replays or re-orders records?

Solution: put sequence number into MAC

- note: no sequence number field

$$MAC = MAC(M_x, sequence \,||\, data)$$

What if attacker replays all records

- Solution: use nonce

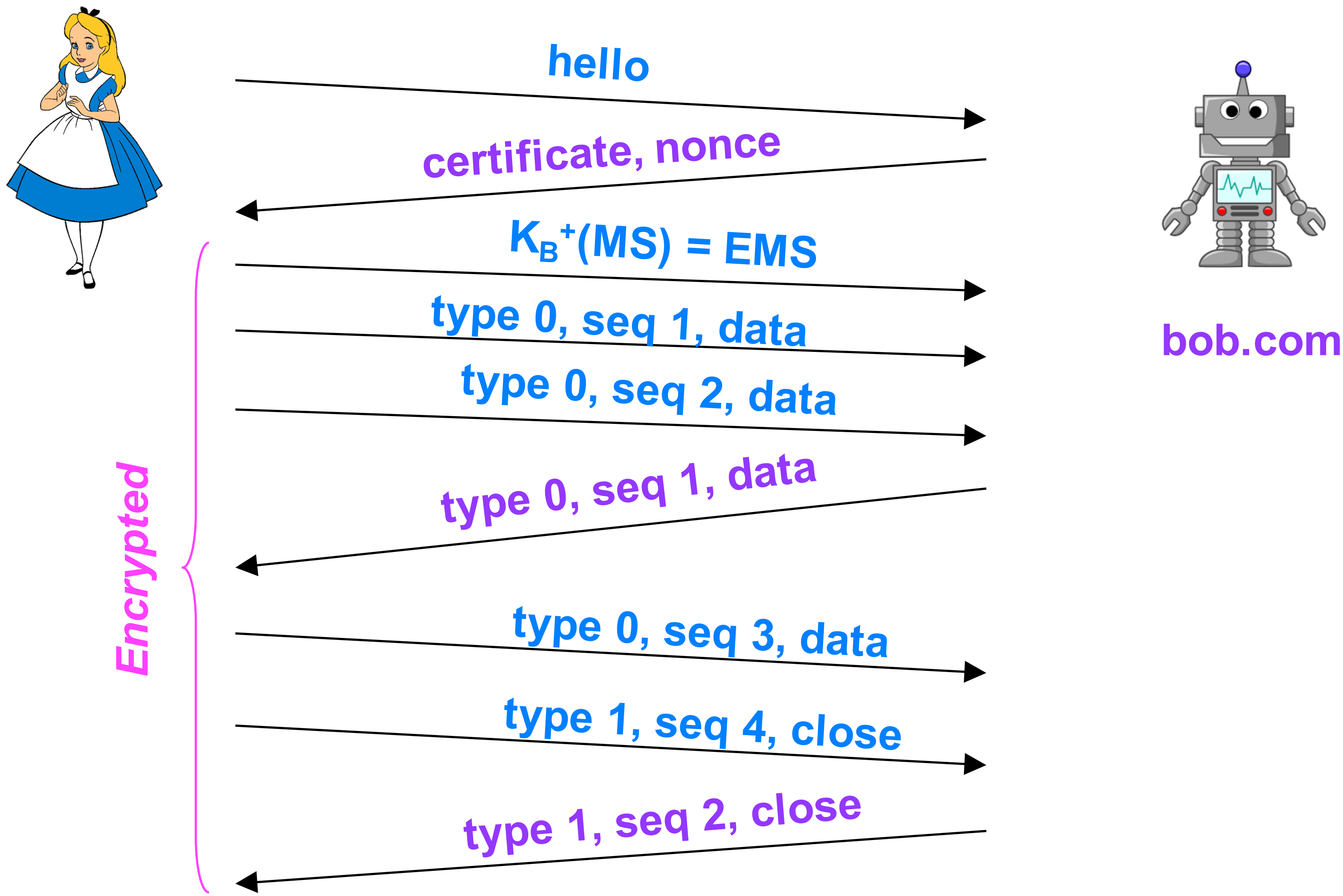
What if attacker forges TCP connection close?

Solution: have record types, with one type for closure

- type 0 for data
- type 1 for closure

$$MAC = MAC(M_x, \text{sequence} \parallel \text{type} \parallel \text{data})$$

Summary



hello
certificate, nonce
$K_B^+(MS)$ = EMS
type 0, seq 1, data
type 0, seq 2, data
type 0, seq 1, data
type 0, seq 3, data
type 1, seq 4, close
type 1, seq 2, close
bob.com
Encrypted

Transport Layer Security

REAL TLS/SSL

Toy TLS/SSL is incomplete

How long are fields? Which encryption protocols? How do client and server negotiate encryption algorithms?

TLS/SSL Handshake

- confidentiality
  - client and server negotiate encryption algorithms before data transfer
    - i.e., negotiate ciphersuite
  - derive keys used in data exchange
- integrity
  - check if handshake tampered with based on hash of handshake msgs
- authentication
  - using public key and server's certificate
  - optional client authentication

TLS/SSL cipher suite

Negotiation: client, server agree on cipher suite

- client offers choice server picks one

TLS_RSA_WITH_3DES_EDE_CBC_SHA

Key exchange algorithm: public-key	Symmetric encryption algorithm: block cipher to encrypt msg stream	MAC algorithm

Which supported depends on version of TLS

- TLS 1.2 supports many cipher suites
- TLS 1.3 supports many fewer cipher suites

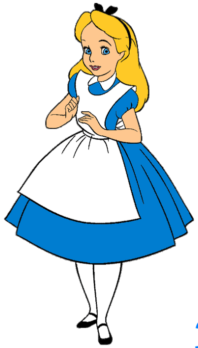
Cipher suites

```
▼ TLSv1 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 144
  ▼ Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 140
      Version: TLS 1.0 (0x0301)
    ▶ Random: 5ae5dac626d5483a3ea908c593979d44170f3e628f26688d...
      Session ID Length: 32
      Session ID: e84d0000076240b35c57828829153be712af150acb327e17...
      Cipher Suites Length: 32
    ▼ Cipher Suites (16 suites)
        Cipher Suite: TLS_EMPTY_RENEGOTIATION_INFO_SCSV (0x00ff)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (0xc024)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (0xc023)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
        Cipher Suite: TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc008)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
        Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
        Cipher Suite: TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (0xc012)
        Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA256 (0x003d)
        Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA256 (0x003c)
        Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
        Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
        Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
```

TLS Client Hello

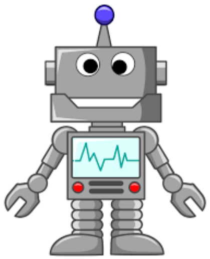
```
▶ Frame 50: 203 bytes on wire (1624 bits), 203 bytes captured (1624 bits) on interface 0
▶ Ethernet II, Src: Apple_73:43:26 (78:4f:43:73:43:26), Dst: JuniperN_1e:18:01 (3c:8a:b0:1e:18:01
▶ Internet Protocol Version 4, Src: vmanfredismbp2.wireless.wesleyan.edu (129.133.187.174), Dst:
▶ Transmission Control Protocol, Src Port: 63173, Dst Port: 443, Seq: 41885059, Ack: 3555367379,
▼ Secure Sockets Layer
  ▼ TLSv1 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 144
    ▼ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 140
        Version: TLS 1.0 (0x0301)
      ▶ Random: 5ae5dac626d5483a3ea908c593979d44170f3e628f26688d...
        Session ID Length: 32
        Session ID: e84d0000076240b35c57828829153be712af150acb327e17...
        Cipher Suites Length: 32
      ▶ Cipher Suites (16 suites)
        Compression Methods Length: 1
      ▶ Compression Methods (1 method)
        Extensions Length: 35
      ▶ Extension: supported_groups (len=8)
      ▶ Extension: ec_point_formats (len=2)
      ▶ Extension: status_request (len=5)
      ▶ Extension: signed_certificate_timestamp (len=0)
      ▶ Extension: extended_master_secret (len=0)
```

SSL handshake



Alice

Bob



1. Client hello →
   client nonce, ciphersuites

← 2. Server hello
   server nonce, chosen ciphersuite, RSA certificate

3. Verifies certificate
   generates premaster secret

4. Premaster secret →
   encrypted with Bob's public key from certificate

5. Generate symmetric keys
   client nonce, server nonce, premaster, ciphersuite

6. Generate symmetric keys
   client nonce, server nonce, premaster, ciphersuite

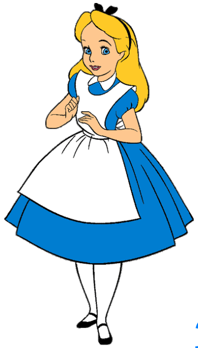
← 7. Server hello done
   MAC of all handshake msgs encrypted with server session keys

8. Client hello done →
   MAC of all handshake msgs encrypted with client symmetric key

7. Encrypted data →

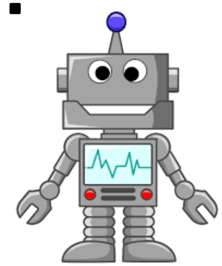
← 8. Encrypted data

What if Trudy modifies ciphersuite list?



Alice

Bob



1. Client hello →
   client nonce, ciphersuites

← 2. Server hello
   server nonce, chosen ciphersuite, RSA certificate

3. Verifies certificate
   generates premaster secret

4. Premaster secret →
   encrypted with Bob's public key from certificate

5. Generate symmetric keys
   client nonce, server nonce, premaster, ciphersuite

6. Generate symmetric keys
   client nonce, server nonce, premaster, ciphersuite

← 7. Server hello done
   MAC of all handshake msgs
   encrypted with server symmetric keys

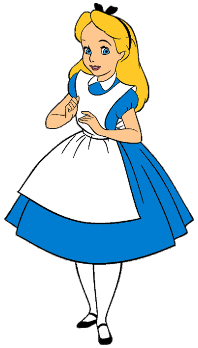
8. Client hello done →
   MAC of all handshake msgs
   encrypted with client symmetric key

Protect handshake from tampering

7. Encrypted data →

← 8. Encrypted data

Why 2 random nonces?

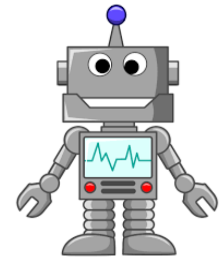


Alice

1. Client hello →

client nonce, ciphersuites

Bob



← 2. Server hello

server nonce, chosen ciphersuite, RSA certificate

Suppose Trudy sniffs all messages between Alice & Bob

– next day, Trudy sets up TCP connection with Bob
  - replays sequence of records
  - Bob (Amazon) thinks Alice made two separate orders for same thing

Solution:

– Bob sends different random nonce for each connection
  - causes encryption keys to be different on the 2 days
  - Trudy's messages will fail Bob's integrity check