Sensitive Information in a Wired World

CPSC 457/557, Fall 2013

Lecture 11, October 3, 2013

1:00-2:15 pm; AKW 400

http://zoo.cs.yale.edu/classes/cs457/fall13/

Motivation

- How do I know the web site I'm talking to is really who I think it is?
- Is it safe to view to give sensitive information over the Web?
  - What keeps my CC#, SSN, financial information or medical records out of the hands of the bad guys?
- How do I know that the information I'm looking at hasn't been malicious modified?
  - Has someone tampered with it?

Securing Internet Traffic

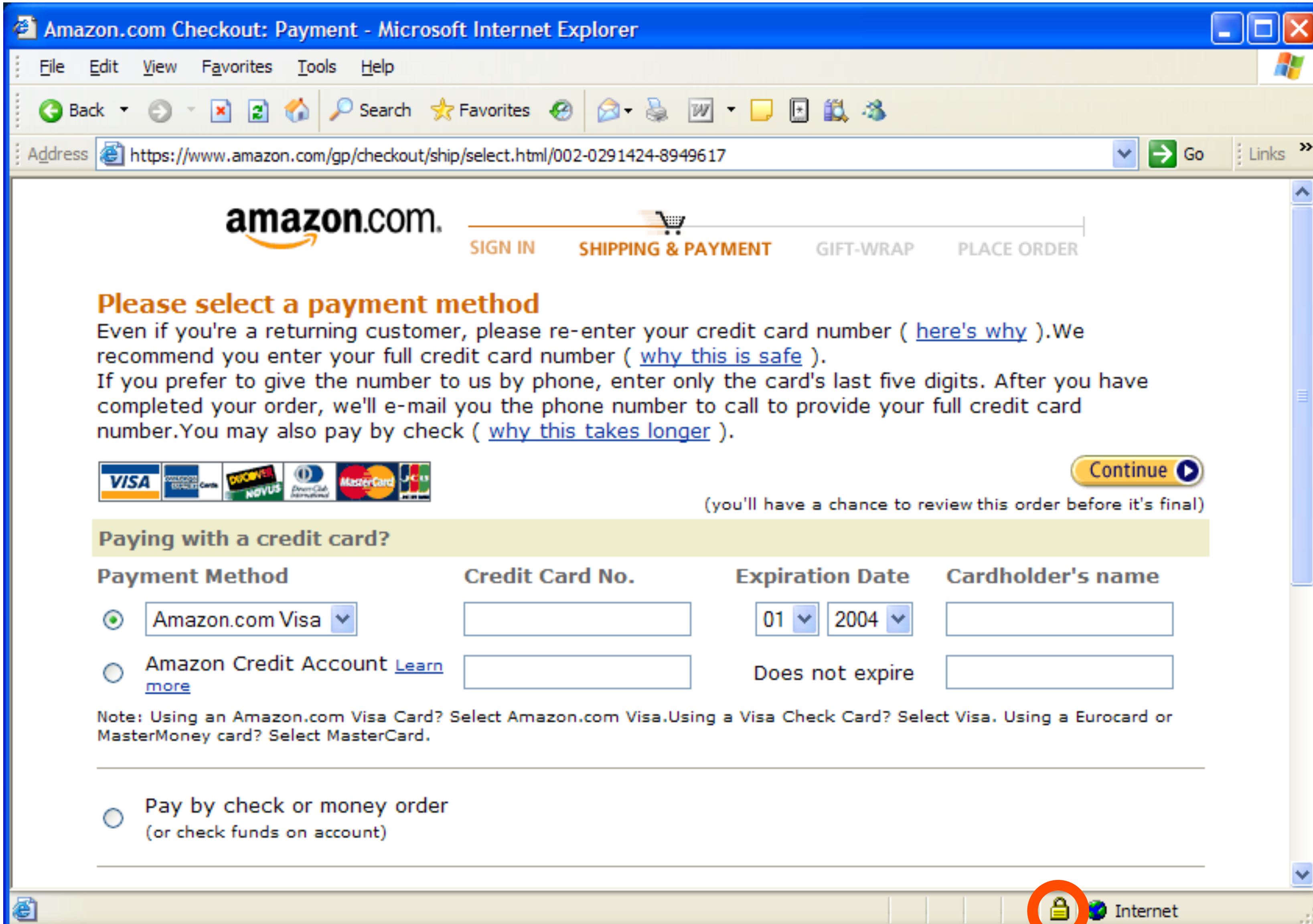
- Application-level security
  - Secure the traffic between two communicating applications
  - Application-specific protocols
  - Example: SSL/TLS for web traffic
- IP-level security
  - Secure traffic at the Internet Protocol layer (low-level wire format)
  - Applications don't have to know about security specifically; they "get it for free"
  - Example: IPSEC

Common Themes

- Three phases
  - Authentication
    - Verify the other party is someone you want to talk to
  - Key agreement
    - Agree on data encryption and integrity protection keys
  - Encrypted data exchange
    - Communicate over the encrypted channel

SSL/TLS

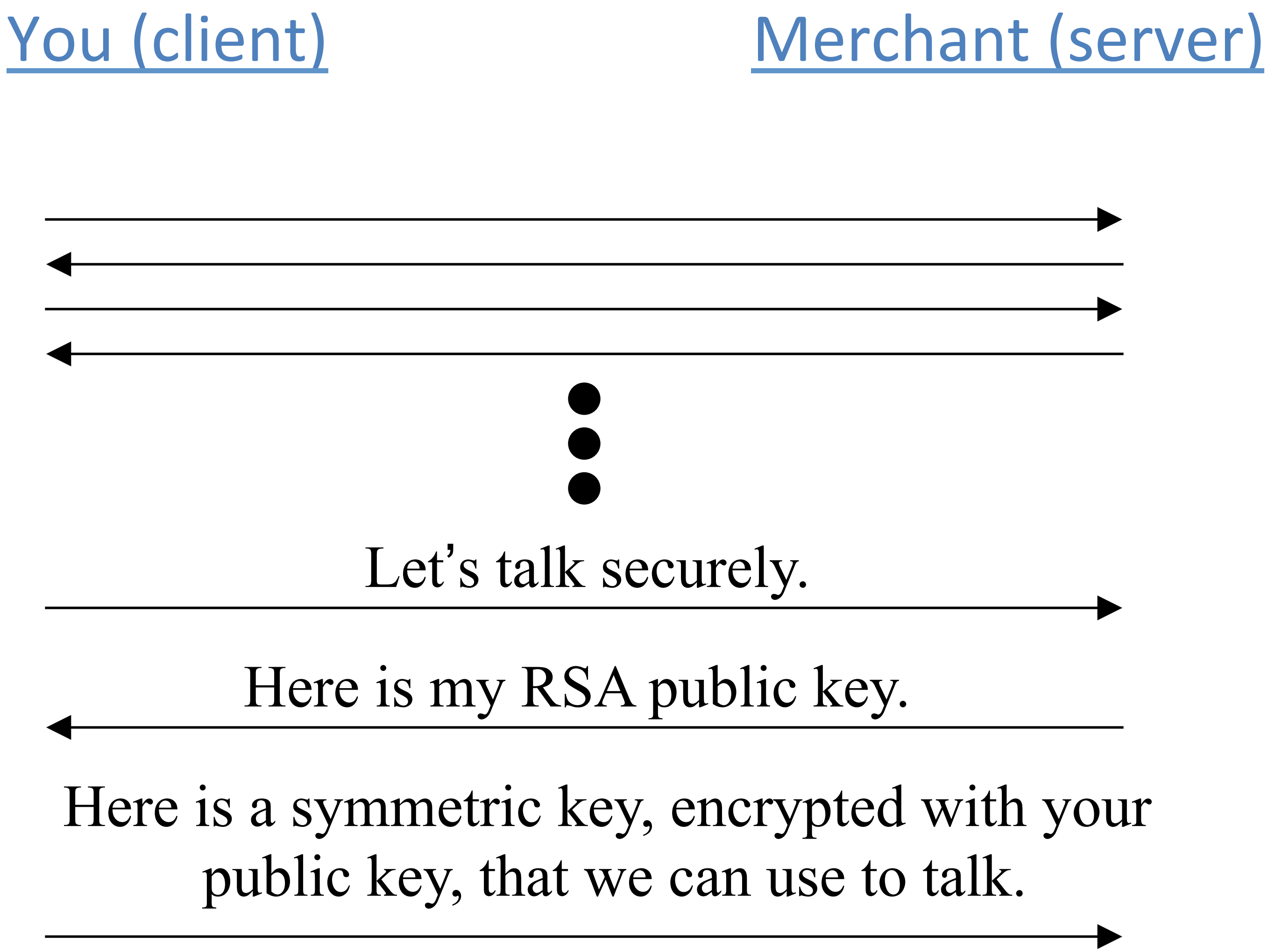
App-Level Security: SSL/TLS



SSL/PCT/TLS History

- 1994: Secure Sockets Layer (SSL) V2.0
- 1995: Private Communication Technology (PCT) V1.0
- 1996: Secure Sockets Layer (SSL) V3.0
- 1997: Private Communication Technology (PCT) V4.0
- 1999: Transport Layer Security (TLS) V1.0
- 2005/2006: TLS V1.1 (currently in the RFC Editor's Queue awaiting publication)

Typical Scenario



SSL/TLS

You (client) Merchant (server)

Let's talk securely. →

← Here is my RSA public key.

Here is a symmetric key, encrypted with your public key, that we can use to talk. →

SSL/TLS

You (client) | Merchant (server)

Let's talk securely.
Here are the protocols and ciphers I understand.

→

Here is my RSA public key.

←

Here is a symmetric key, encrypted with your public key, that we can use to talk.

→

SSL/TLS

You (client) | Merchant (server)

Let's talk securely.
Here are the protocols and ciphers I understand.
→

I choose this protocol and ciphers.
Here is my public key and
some other stuff.
←

Here is a symmetric key, encrypted with your public key, that we can use to talk.
→

SSL/TLS

You (client) | Merchant (server)

Let's talk securely.
Here are the protocols and ciphers I understand.

→

I choose this protocol and ciphers.
Here is my public key and
some other stuff.

←

Using your public key, I've encrypted
a random symmetric key to you.

→

SSL/TLS

All subsequent secure messages are sent using the symmetric key and a keyed hash for message authentication.

The five phases of SSL/TLS

1. Negotiate the ciphersuite to be used
2. Establish the shared session key
3. Client authenticates the server ("server auth")
   - Optional, but almost <u>always</u> done
4. Server authenticates the client ("client auth")
   - Optional, and almost <u>never</u> done
5. Authenticate previously exchanged data

Phase 1: Ciphersuite Negotiation

- Client hello (client→server)
  - "Hi! I speak these n ciphersuites, and here's a 28-byte random number (nonce) I just picked"
- Server hello (client←server)
  - "Hello. We're going to use this particular ciphersuite, and here's a 28-byte nonce I just picked."
- Other info can be passed along (we'll see why a little later...)

TLS V1.0 ciphersuites

TLS_NULL_WITH_NULL_NULL
TLS_RSA_WITH_NULL_MD5
TLS_RSA_WITH_NULL_SHA
TLS_RSA_EXPORT_WITH_RC4_40_MD5
TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
TLS_RSA_WITH_IDEA_CBC_SHA
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_DSS_WITH_DES_CBC_SHA
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_RSA_WITH_DES_CBC_SHA
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_DSS_WITH_DES_CBC_SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_RSA_WITH_DES_CBC_SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DH_anon_EXPORT_WITH_RC4_40_MD5
TLS_DH_anon_WITH_RC4_128_MD5
TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_anon_WITH_DES_CBC_SHA
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA

More defined in other specs

TLS-With-AES ciphersuites

(RFC 3268)

TLS_RSA_WITH_AES_128_CBC_SHA	RSA
TLS_DH_DSS_WITH_AES_128_CBC_SHA	DH_DSS
TLS_DH_RSA_WITH_AES_128_CBC_SHA	DH_RSA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA	DHE_DSS
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	DHE_RSA
TLS_DH_anon_WITH_AES_128_CBC_SHA	DH_anon

TLS_RSA_WITH_AES_256_CBC_SHA	RSA
TLS_DH_DSS_WITH_AES_256_CBC_SHA	DH_DSS
TLS_DH_RSA_WITH_AES_256_CBC_SHA	DH_RSA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA	DHE_DSS
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	DHE_RSA
TLS_DH_anon_WITH_AES_256_CBC_SHA	DH_anon

Phase 2: Establish shared session key

- Client key exchange
  - Client chooses a 48-byte “pre-master secret”
  - Client encrypts the pre-master secret with the server’s RSA public key
  - Client➔server encrypted pre-master secret
- Client and server both compute
  - PRF (pre-master secret, “master secret,” client nonce + server nonce)
  - PRF is a pseudo-random function
  - First 48 bytes output from PRF form master secret

TLS's PRF

- PRF(secret, label, seed) = P_MD5(S1, label + seed) XOR P_SHA-1(S2, label + seed); where S1, S2 are the two halves of the secret
- P_hash(secret, seed) = HMAC_hash(secret, A(1) + seed) + HMAC_hash (secret, A(2) + seed) + HMAC_hash(secret, A(3) + seed) + ...
- A(0) = seed
  A(i) = HMAC_hash(secret, A(i-1))

Phases 3 and 4: Authentication

More on this in a moment...

Phase 5: Authenticate previously exchanged data

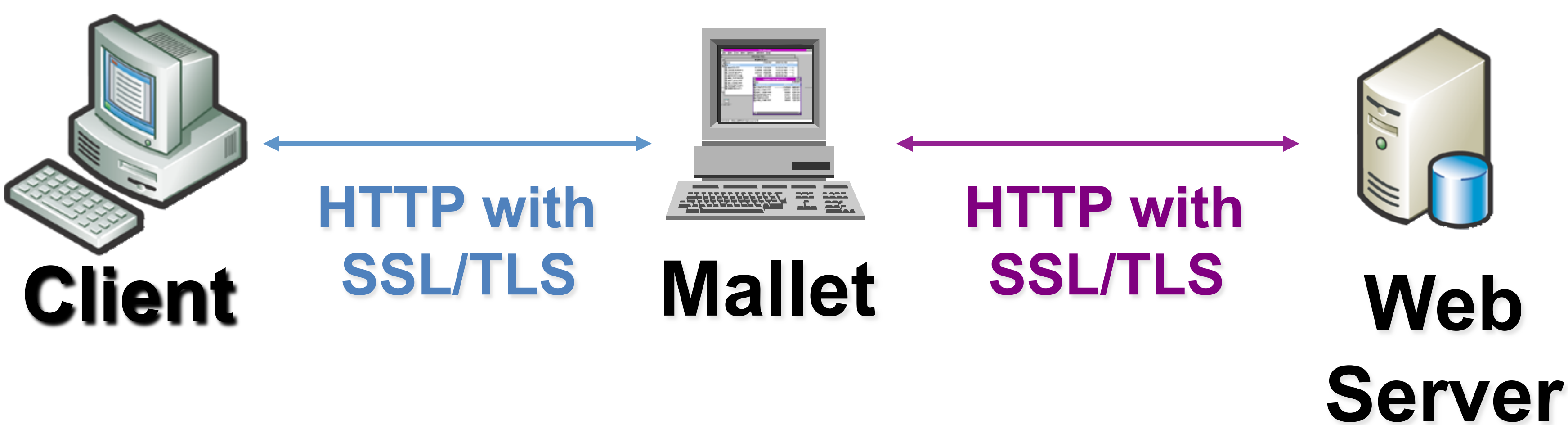
- “Change ciphersuites” message
  - Time to start sending data for real...
- “Finished” handshake message
  - First protected message, verifies algorithm parameters for the encrypted channel
  - 12 bytes from:
    PRF(master_secret, “client finished,” MD5 (handshake_messages) + SHA-1(handshake_messages))

Why do I trust the server key?

- How do I know I'm really talking to Amazon.com?
- What defeats a man-in-the-middle attack?

Why do I trust the server key?

- How do I know I'm really talking to Amazon.com?
- What defeats a man-in-the-middle attack?



SSL/TLS

You (client) | Merchant (server)

Let’s talk securely.
Here are the protocols and ciphers I understand.
→

I choose this protocol and ciphers.
Here is my public key and
some other stuff that will make you
trust this key is mine.
←

Here is a fresh key encrypted with your key.
→

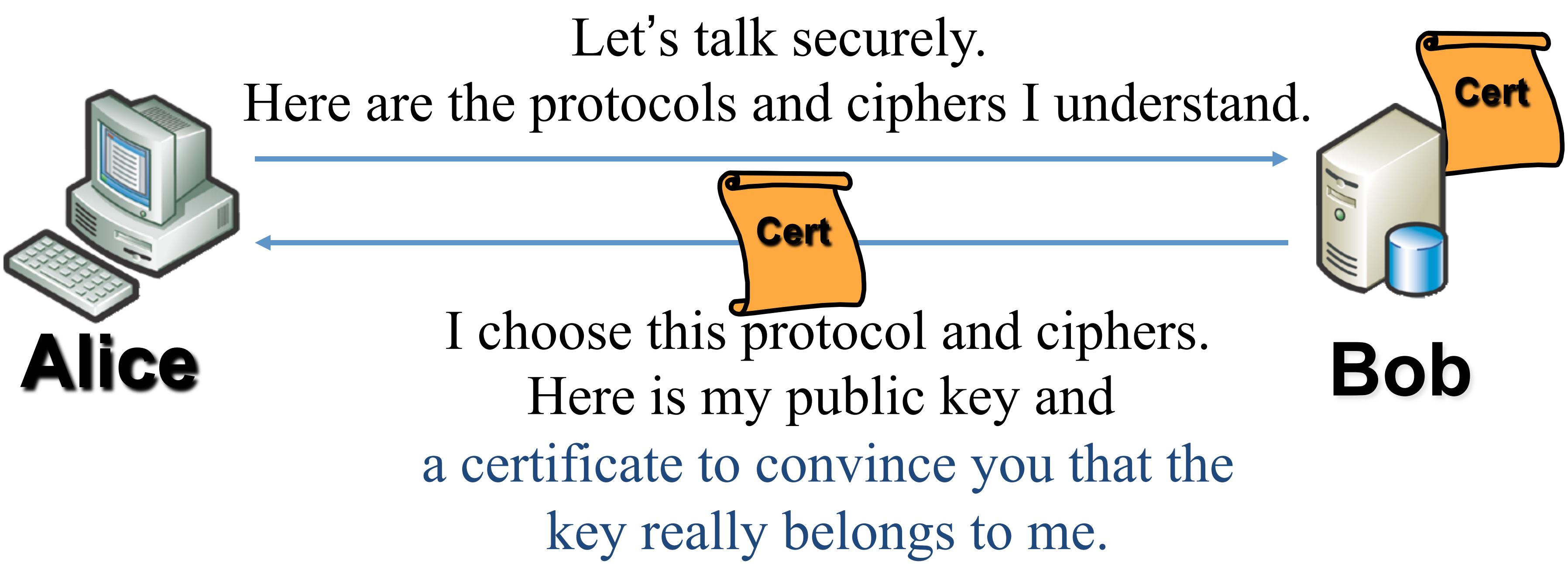
What’s the “some other stuff”?

How can we convince Alice that some key belongs to Bob?

- Alice and Bob could have met previously and exchanged keys directly.
  - *Jeff Bezos isn’t going to shake hands with everyone he’d like to sell to...*
- Someone Alice trusts could vouch to her for Bob and Bob’s key
  - *A third party can certify Bob’s key in a way that convinces Alice.*

Defeating Mallet

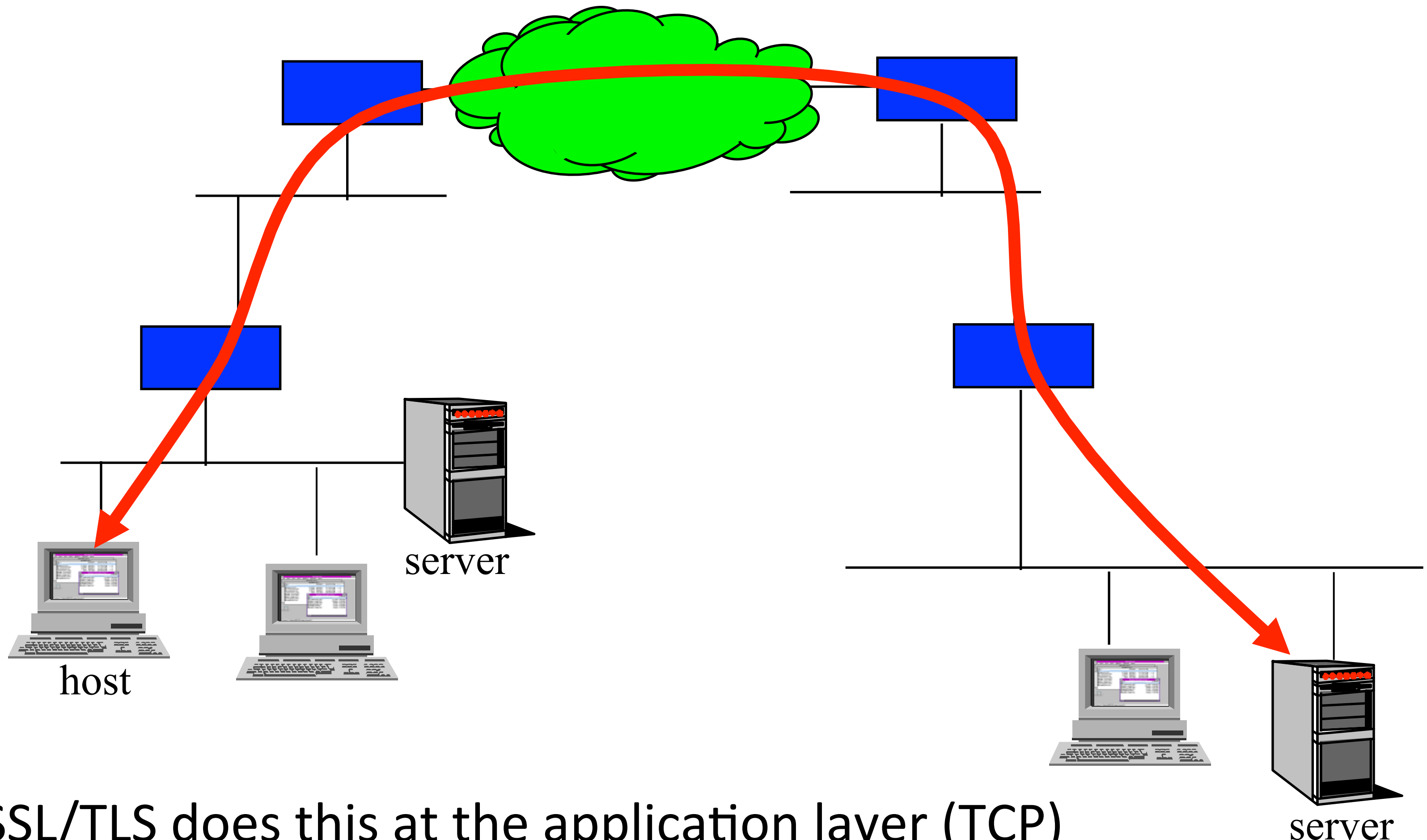
Bob can convince Alice that his key really does belong to him if he can also send along a digital certificate Alice will believe and trust



Protocol-Level Security: IPSEC

- Application-level security protocols work great for particular applications
  - But they <u>only</u> work for that application
- SSL/TLS requires lots of infrastructure to work; how many protocols can we do that for?
- Ideally, we'd like all the security features of SSL/TLS available for <u>every</u> Internet protocol/application
  - "Security at the IP layer"

Ideal Protection: End-to-End



- SSL/TLS does this at the application layer (TCP)
- IPSEC does this for any IP packet, at network layer
- Apps must be aware of/control SSL, don't have to be for IPSec