Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity, authentication

8.4 Securing e-mail

8.5 Securing TCP connections: SSL

8.6 Network layer security: IPsec

8.7 Securing wireless LANs

8.8 Operational security: firewalls and IDS

Digital signatures

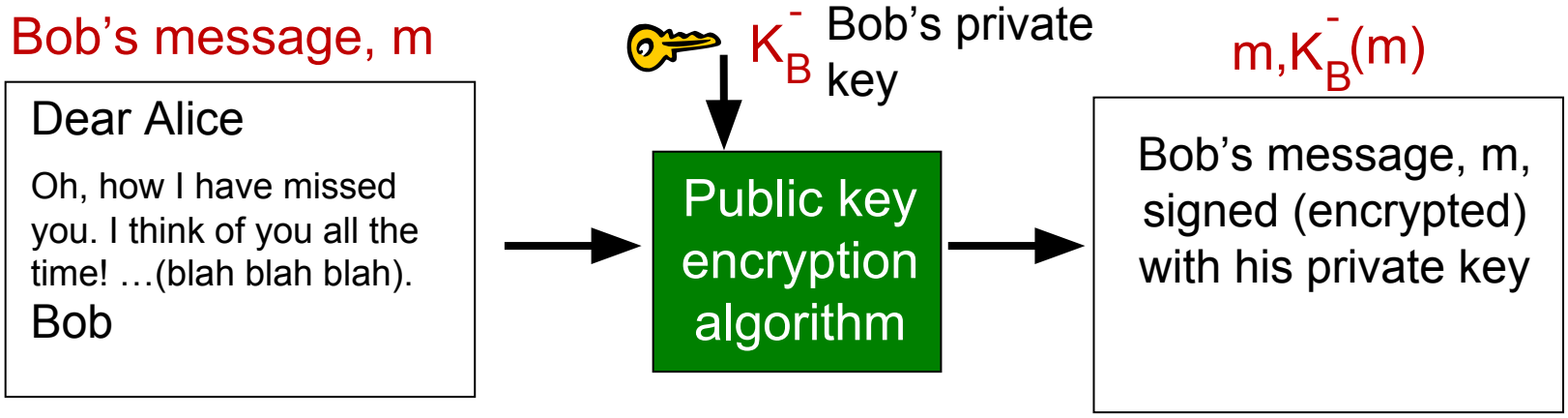
cryptographic technique analogous to hand-written signatures:

- sender (Bob) digitally signs document, establishing he is document owner/creator.
- *verifiable, nonforgeable:* recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document

Digital signatures

simple digital signature for message m:

- Bob signs m by encrypting with his private key $\bar{K}_B$, creating "signed" message, $\bar{K}_B(m)$



Digital signatures

- suppose Alice receives msg m, with signature: m, $K_B^-(m)$
- Alice verifies m signed by Bob by applying Bob's public key K_B^+ to $K_B^-(m)$ then checks $K_B^+(K_B^-(m)) = m$.
- If $K_B^+(K_B^-(m)) = m$, whoever signed m must have used Bob's private key.

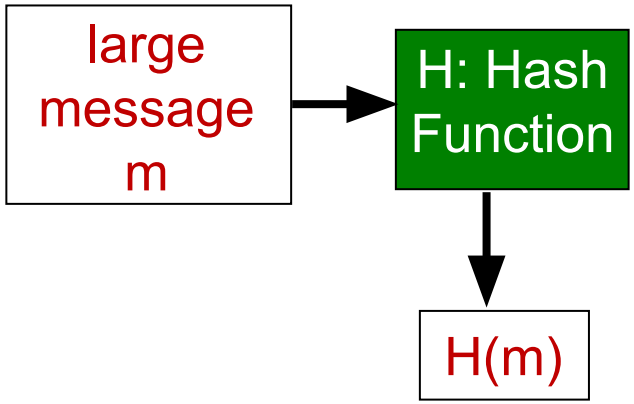
Alice thus verifies that:

- Bob signed m
- no one else signed m
- Bob signed m and not m'

non-repudiation:

- Alice can take m, and signature $K_B(m)$ to court and prove that Bob signed m

Message digests



computationally expensive to public-key-encrypt long messages

goal: fixed-length, easy- to-compute digital "fingerprint"

- apply hash function H to *m*, get fixed size message digest, *H(m)*.

Hash function properties:

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

Internet checksum: poor crypto hash function

Internet checksum has some properties of hash function:

- produces fixed length digest (16-bit sum) of message
- is many-to-one

But given message with given hash value, it is easy to find another message with same hash value:

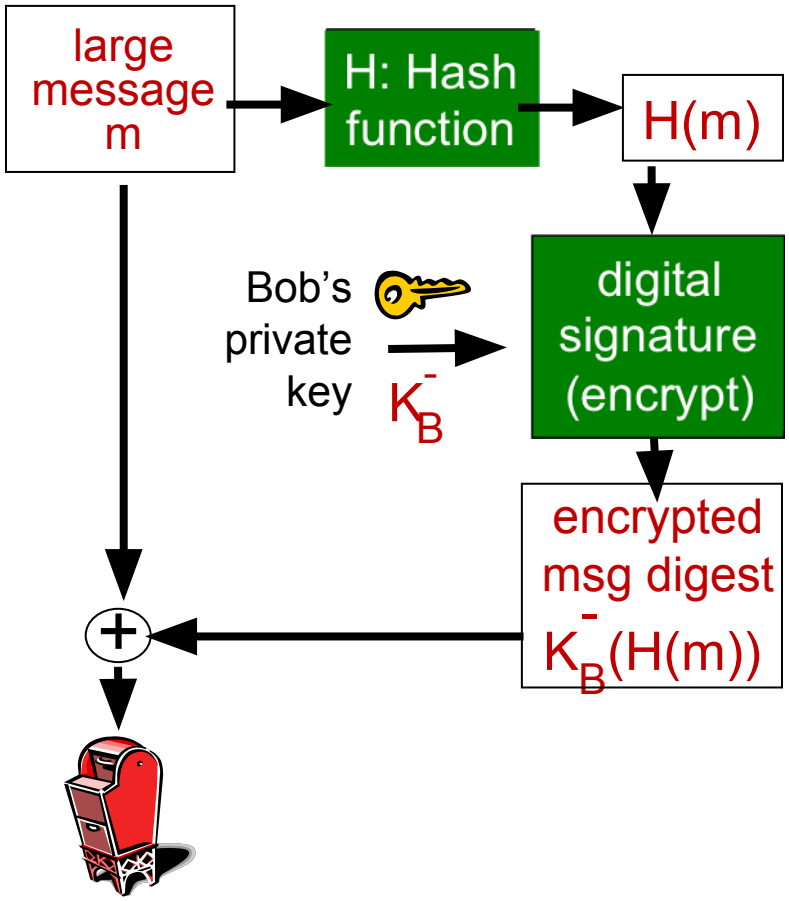
message	ASCII format
I O U 1	49 4F 55 31
0 0 . 9	30 30 2E 39
9 B O B	39 42 D2 42
	B2 C1 D2 AC

different messages but identical checksums!

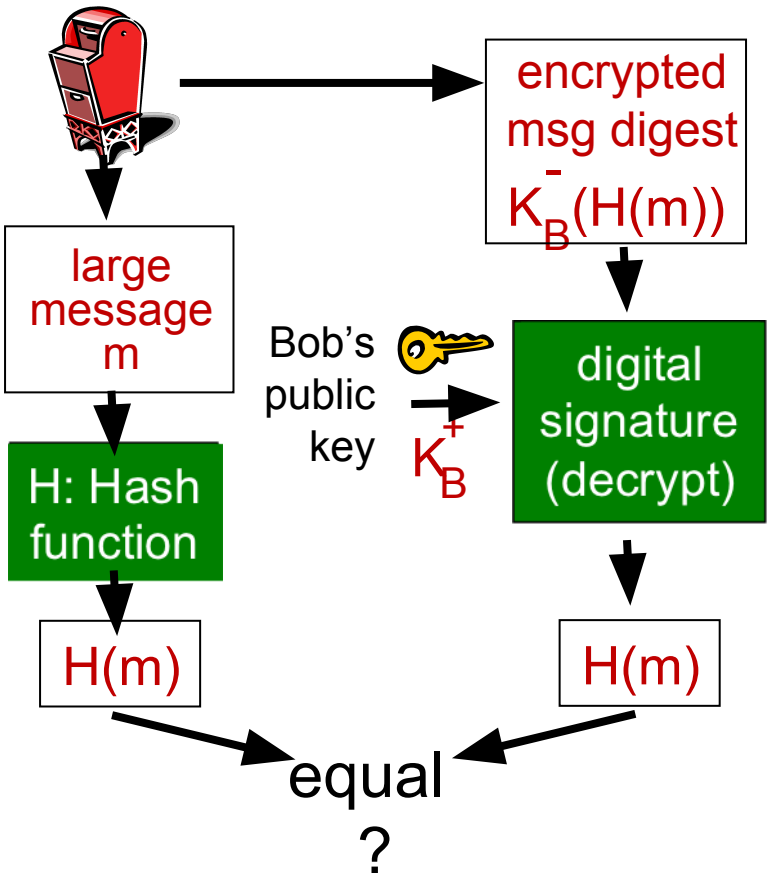
message	ASCII format
I O U 9	49 4F 55 39
0 0 . 1	30 30 2E 31
9 B O B	39 42 D2 42
	B2 C1 D2 AC

Digital signature = signed message digest

Bob sends digitally signed message:



Alice verifies signature, integrity of digitally signed message:

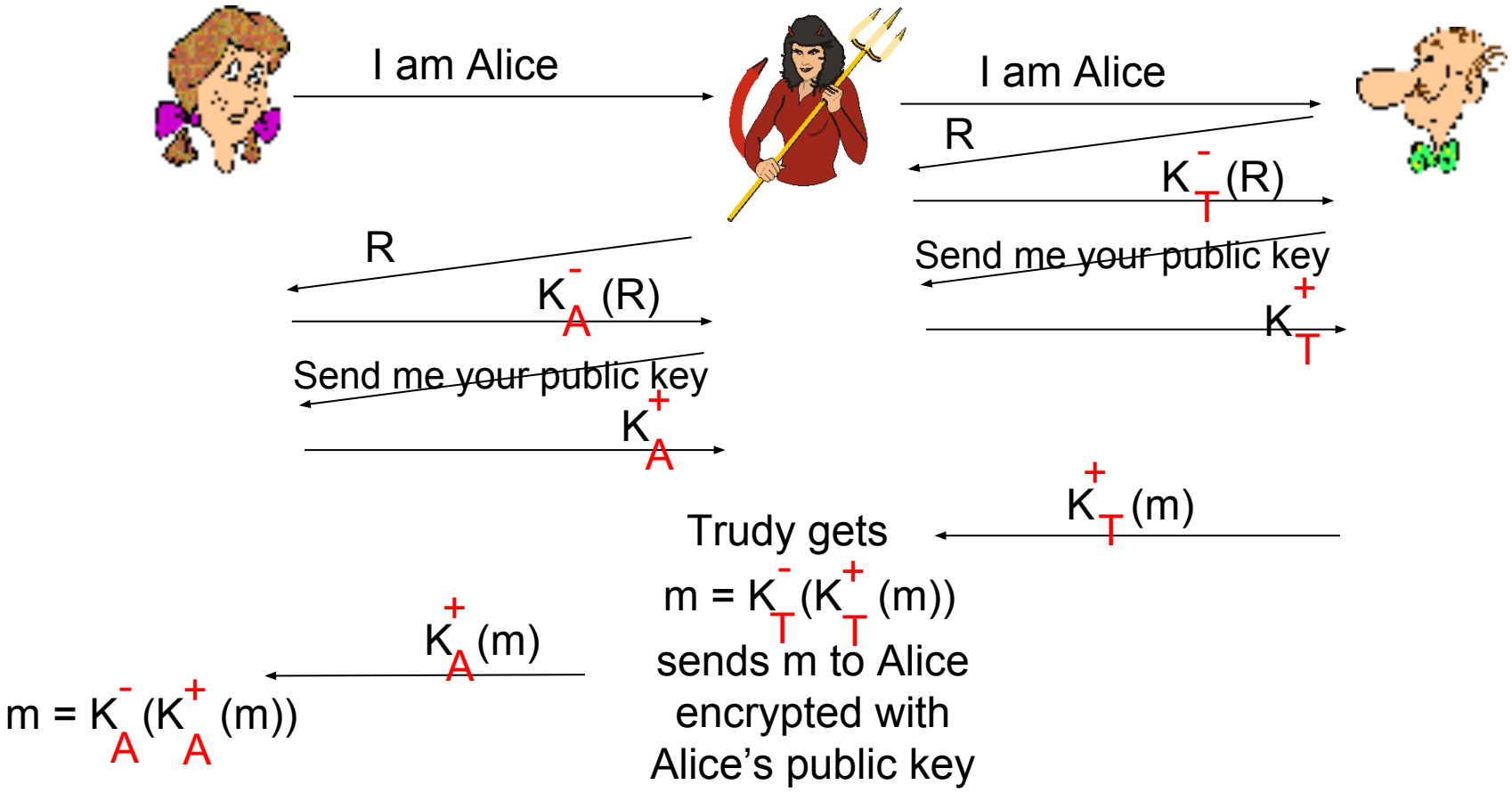


Hash function algorithms

- MD5 hash function widely used (RFC 1321)
  - computes 128-bit message digest in 4-step process.
  - arbitrary 128-bit string x, appears difficult to construct msg m whose MD5 hash is equal to x
- SHA-1 is also used
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest

Recall: ap5.0 security hole

man (or woman) in the middle attack: Eve poses as Alice (to Bob) and as Bob (to Alice)

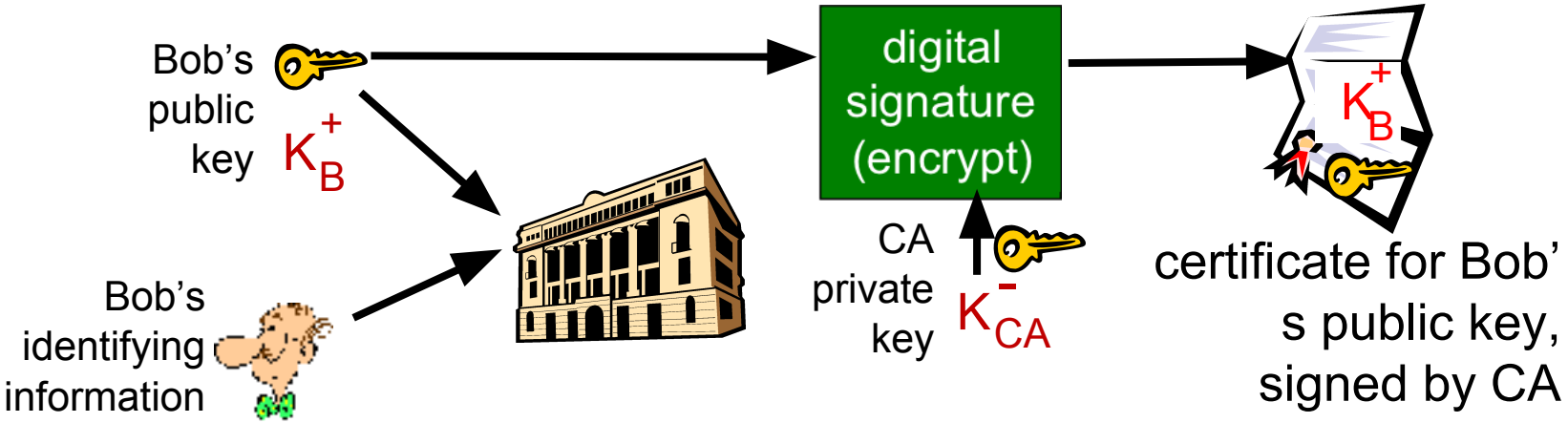


Public-key certification

- motivation: Eve plays pizza prank on Bob
  - Eve creates e-mail order:
    Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob
  - Eve signs order with her private key
  - Eve sends order to Pizza Store
  - Eve sends to Pizza Store her public key, but says it's Bob's public key
  - Pizza Store verifies signature; then delivers four pepperoni pizzas to Bob
  - Bob doesn't even like pepperoni

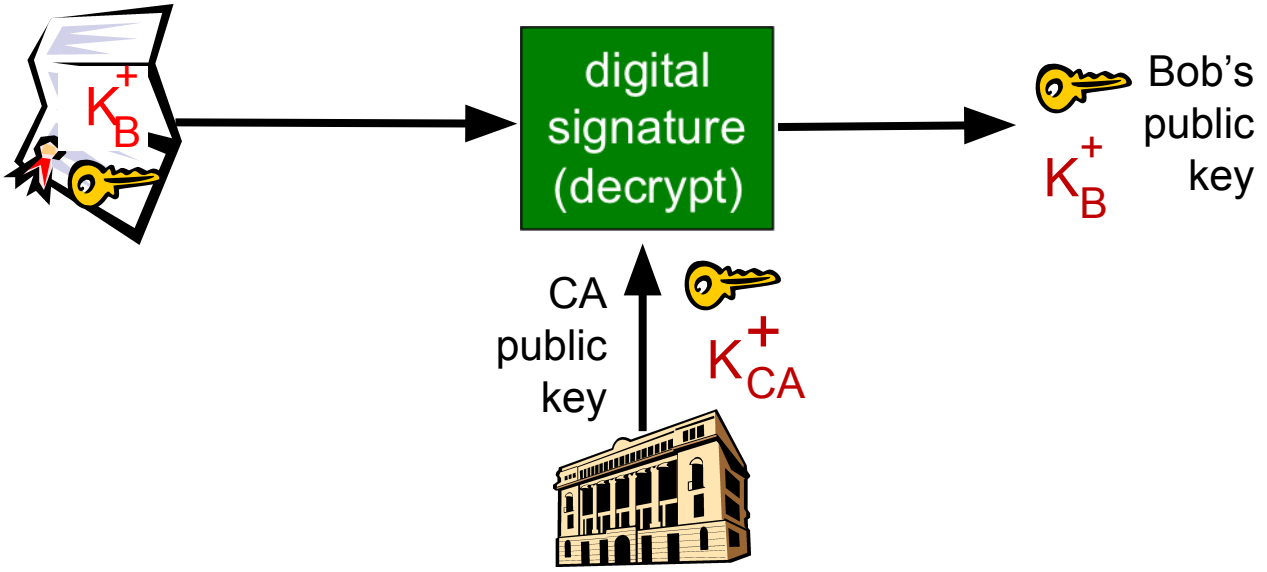
Certification authorities

- *certification authority (CA):* binds public key to particular entity, E.
- E (person, router) registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E's public key digitally signed by CA – CA says "this is E's public key"



Certification authorities

- when Alice wants Bob's public key:
  - gets Bob's certificate (Bob or elsewhere).
  - apply CA's public key to Bob's certificate, get Bob's public key



Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity, authentication

8.4 Securing e-mail

8.5 Securing TCP connections: SSL

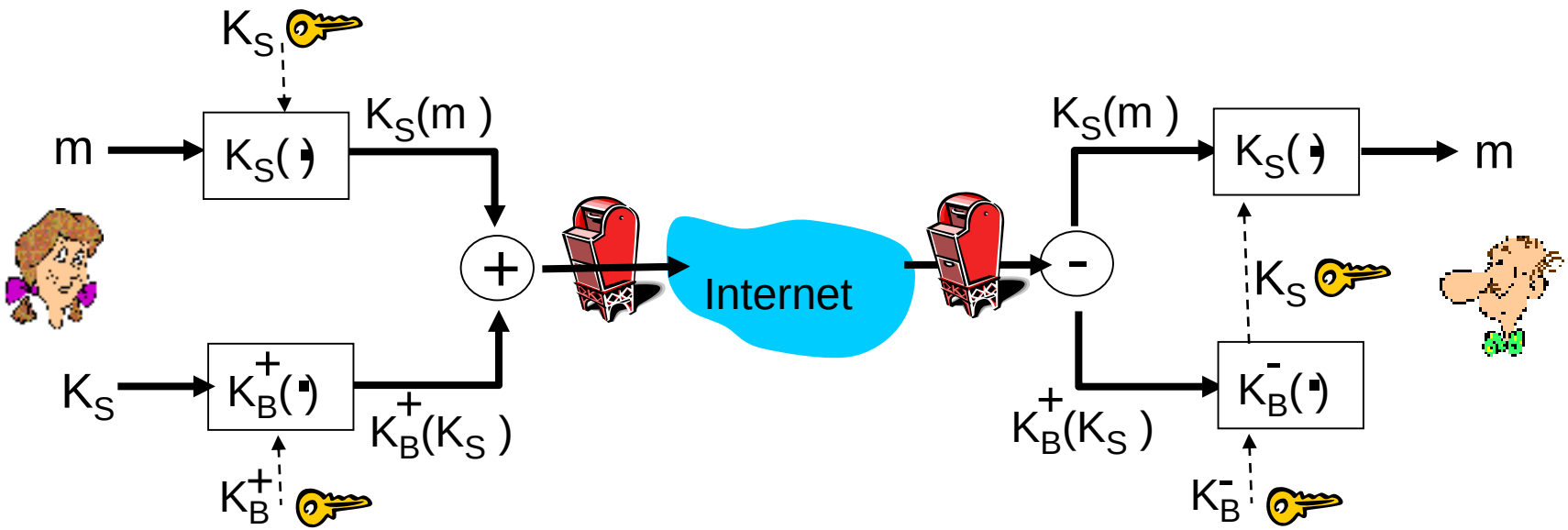
8.6 Network layer security: IPsec

8.7 Securing wireless LANs

8.8 Operational security: firewalls and IDS

Secure e-mail

- Alice wants to send confidential e-mail, m, to Bob.

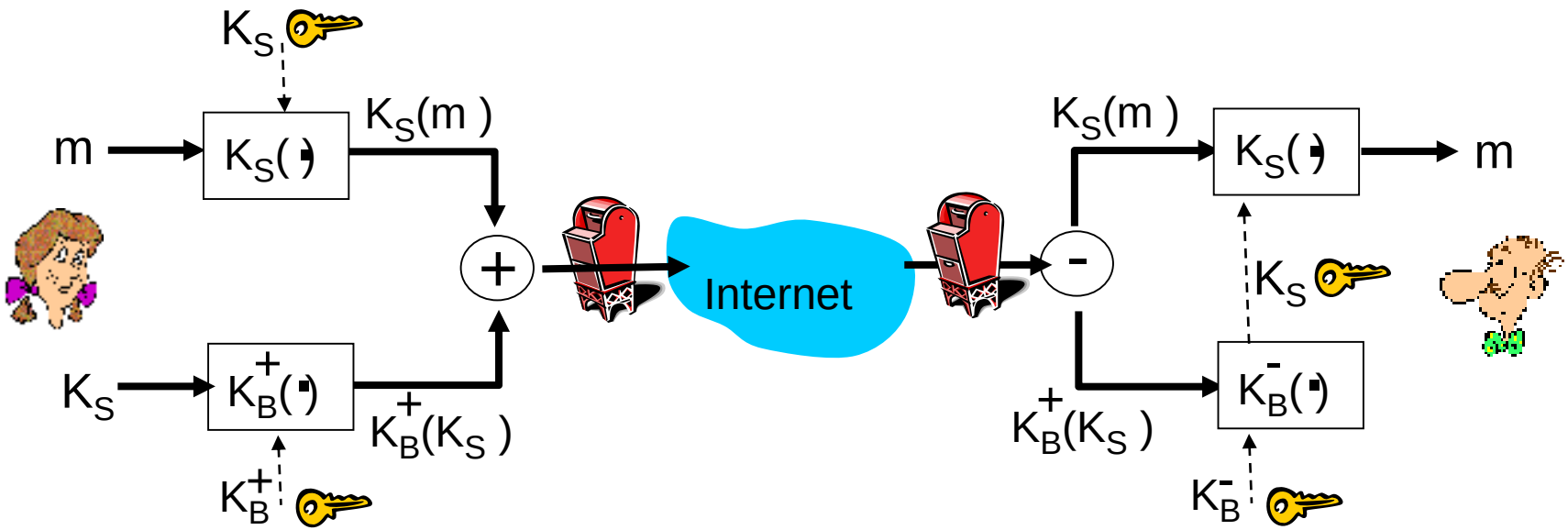


Alice:

- generates random *symmetric* private key, K_S
- encrypts message with K_S (for efficiency)
- also encrypts K_S with Bob's public key
- sends both $K_S(m)$ and $K_B(K_S)$ to Bob

Secure e-mail

- Alice wants to send confidential e-mail, m, to Bob.

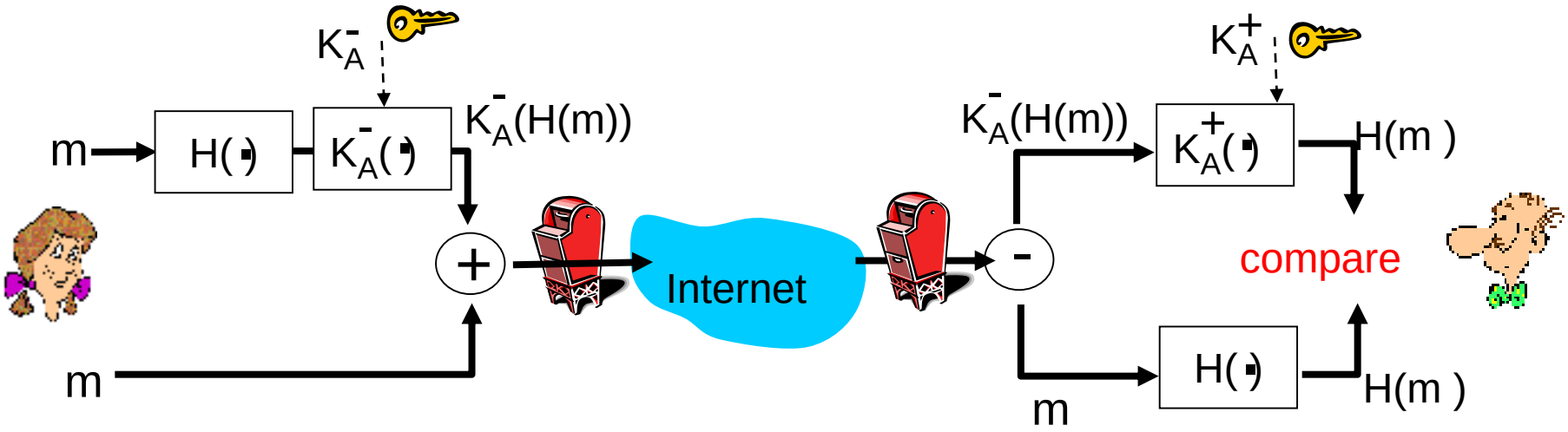


Bob:

- uses his private key to decrypt and recover K_S
- uses K_S to decrypt $K_S(m)$ to recover m

Secure e-mail (continued)

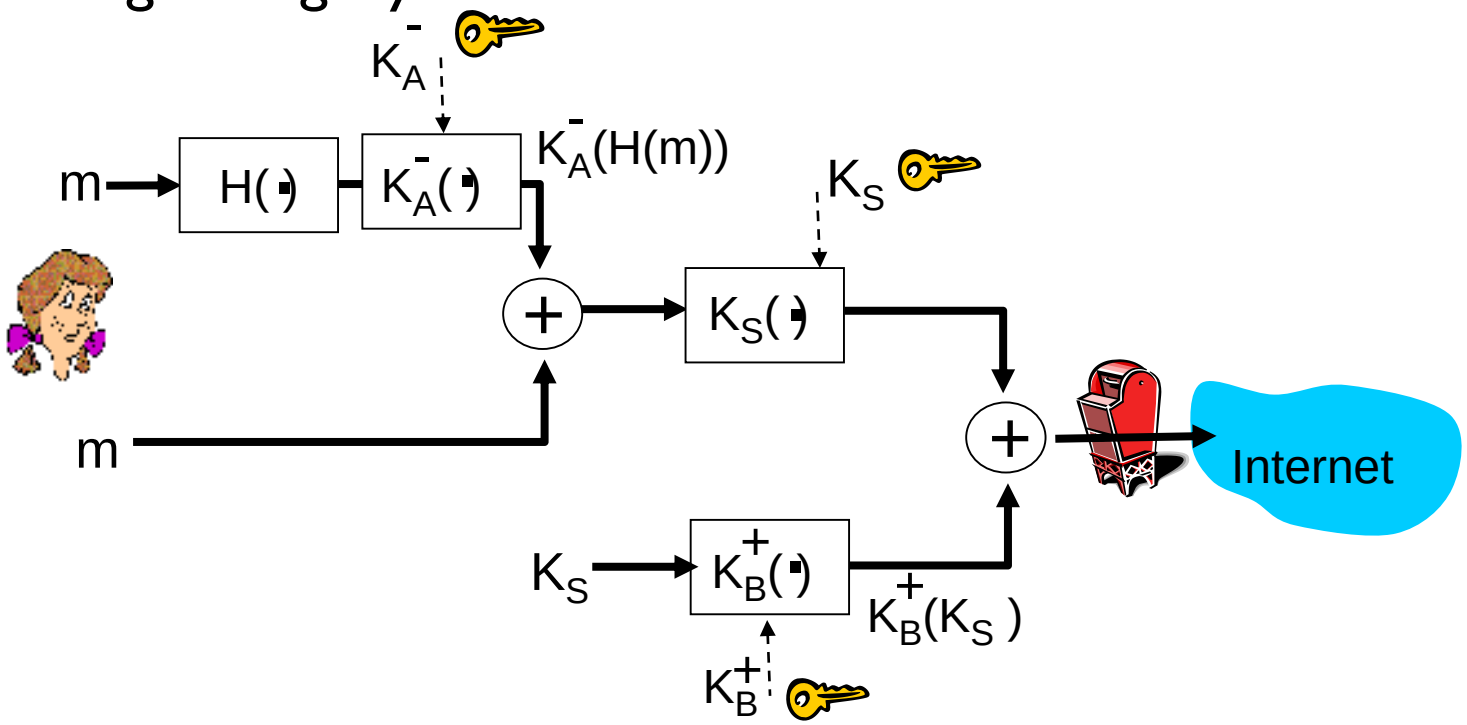
- Alice wants to provide sender authentication message integrity



- Alice digitally signs message
- sends both message (in the clear) and digital signature

Secure e-mail (continued)

- Alice wants to provide secrecy, sender authentication, message integrity.



Alice uses three keys: her private key, Bob's public key, newly created symmetric key

Chapter 8 roadmap

8.1 What is network security?

8.2 Principles of cryptography

8.3 Message integrity

8.4 Securing e-mail

8.5 Securing TCP connections: SSL

8.6 Network layer security: IPsec

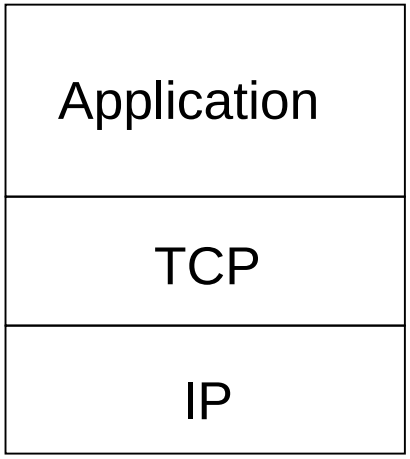
8.7 Securing wireless LANs

8.8 Operational security: firewalls and IDS

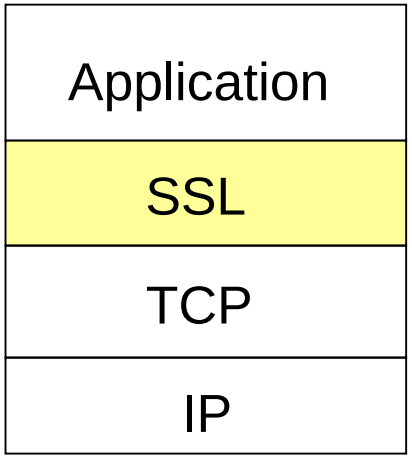
SSL: Secure Sockets Layer

- widely deployed security protocol
  - supported by almost all browsers, web servers
  - https
  - billions $/year over SSL
- mechanisms: [Woo 1994], implementation: Netscape
- variation -TLS: transport layer security, RFC 2246
- provides
  - *confidentiality*
  - *integrity*
  - *authentication*
- original goals:
  - Web e-commerce transactions
  - encryption (especially credit-card numbers)
  - Web-server authentication
  - optional client authentication
  - minimum hassle in doing business with new merchant
- available to all TCP applications
  - secure socket interface

SSL and TCP/IP



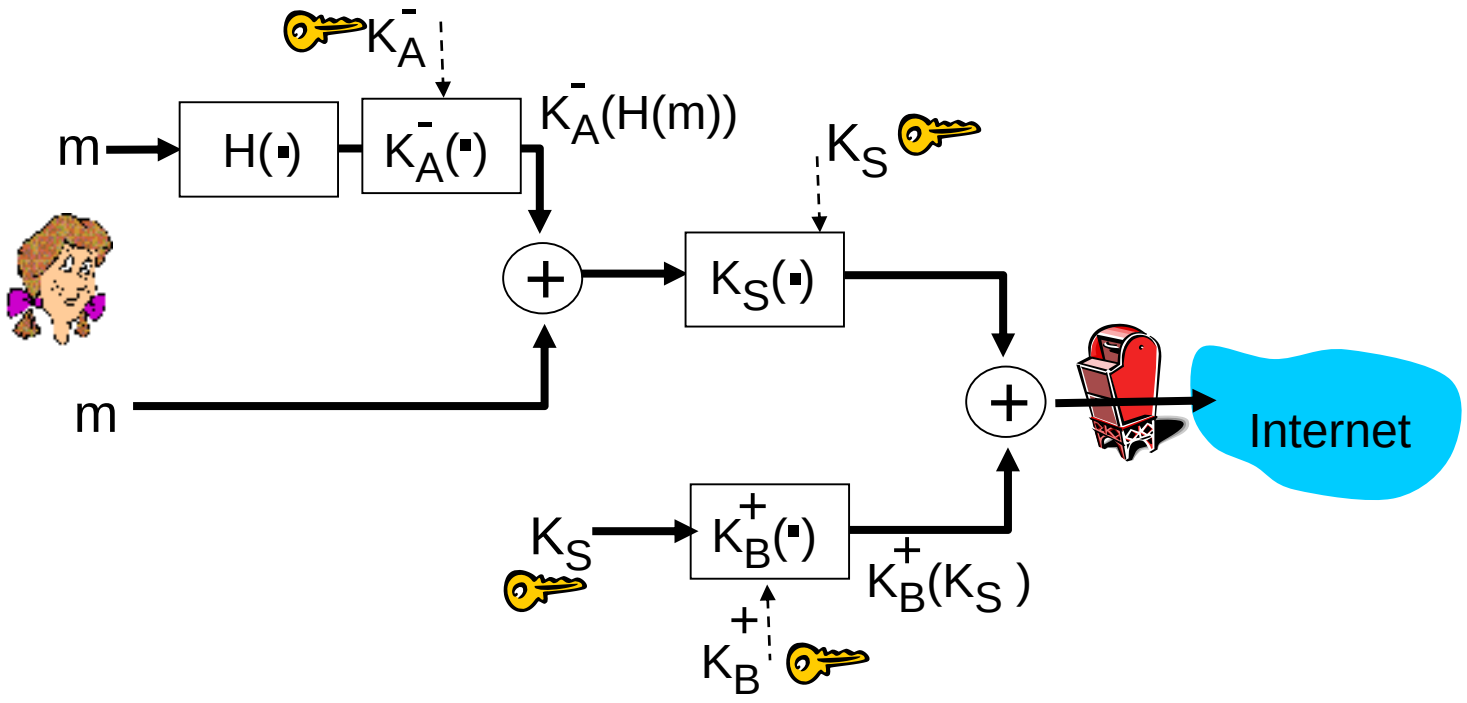
normal application



application with SSL

- SSL provides application programming interface (API) to applications
- C and Java SSL libraries/classes readily available

Could do something like PGP:

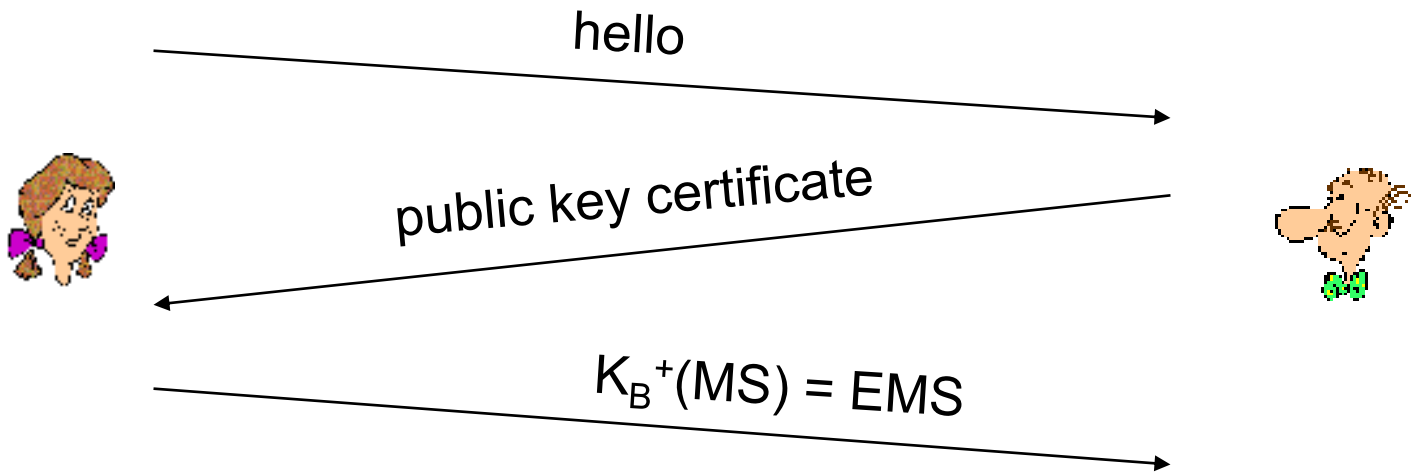


- but want to send byte streams & interactive data
- want set of secret keys for entire connection
- want certificate exchange as part of protocol: handshake phase

Toy 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

Toy: a simple handshake



MS: master secret

EMS: encrypted master secret

Toy: key derivation

- considered bad to use same key for more than one cryptographic operation
  - use different keys for message authentication code (MAC) and encryption
- four 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 key derivation function (KDF)
  - takes master secret and (possibly) some additional random data and creates the keys

Toy: data records

- why not encrypt data in constant stream as we write it to TCP?
  - where would we put the MAC? If at end, no message integrity until all data processed.
  - e.g., with instant messaging, how can we do integrity check over all bytes sent before displaying?
- instead, break stream in series of records
  - each record carries a MAC
  - receiver can act on each record as it arrives
- issue: in record, receiver needs to distinguish MAC from data
  - want to use variable-length records

Toy: sequence numbers

- *problem:* attacker can capture and replay record or re-order records
- *solution:* put sequence number into MAC:
  - MAC = MAC(M_x, sequence||data)
  - note: no sequence number field

- *problem:* attacker could replay all records
- *solution:* use nonce

Toy: control information

- *problem:* truncation attack:
  - attacker forges TCP connection close segment
  - one or both sides thinks there is less data than there actually is.
- *solution:* record types, with one type for closure
  - type 0 for data; type 1 for closure
- MAC = MAC(M_x, sequence||type||data)

Toy SSL: summary

bob.com

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

encrypted

Toy SSL isn't complete

- how long are fields?
- which encryption protocols?
- want negotiation?
  - allow client and server to support different encryption algorithms
  - allow client and server to choose together specific algorithm before data transfer

SSL cipher suite

- cipher suite
  - public-key algorithm
  - symmetric encryption algorithm
  - MAC algorithm
- SSL supports several cipher suites
- negotiation: client, server agree on cipher suite
  - client offers choice
  - server picks one

common SSL symmetric ciphers

- DES – Data Encryption Standard: block
- 3DES – Triple strength: block
- RC2 – Rivest Cipher 2: block
- RC4 – Rivest Cipher 4: stream

SSL Public key encryption

- RSA

Real SSL: handshake (1)

Purpose

1. server authentication
2. negotiation: agree on crypto algorithms
3. establish keys
4. client authentication (optional)

Real SSL: handshake (2)

1. client sends list of algorithms it supports, along with client nonce
2. server chooses algorithms from list; sends back: choice + certificate + server nonce
3. client verifies certificate, extracts server's public key, generates pre_master_secret, encrypts with server's public key, sends to server
4. client and server independently compute encryption and MAC keys from pre_master_secret and nonces
5. client sends a MAC of all the handshake messages
6. server sends a MAC of all the handshake messages

Real SSL: handshaking (3)

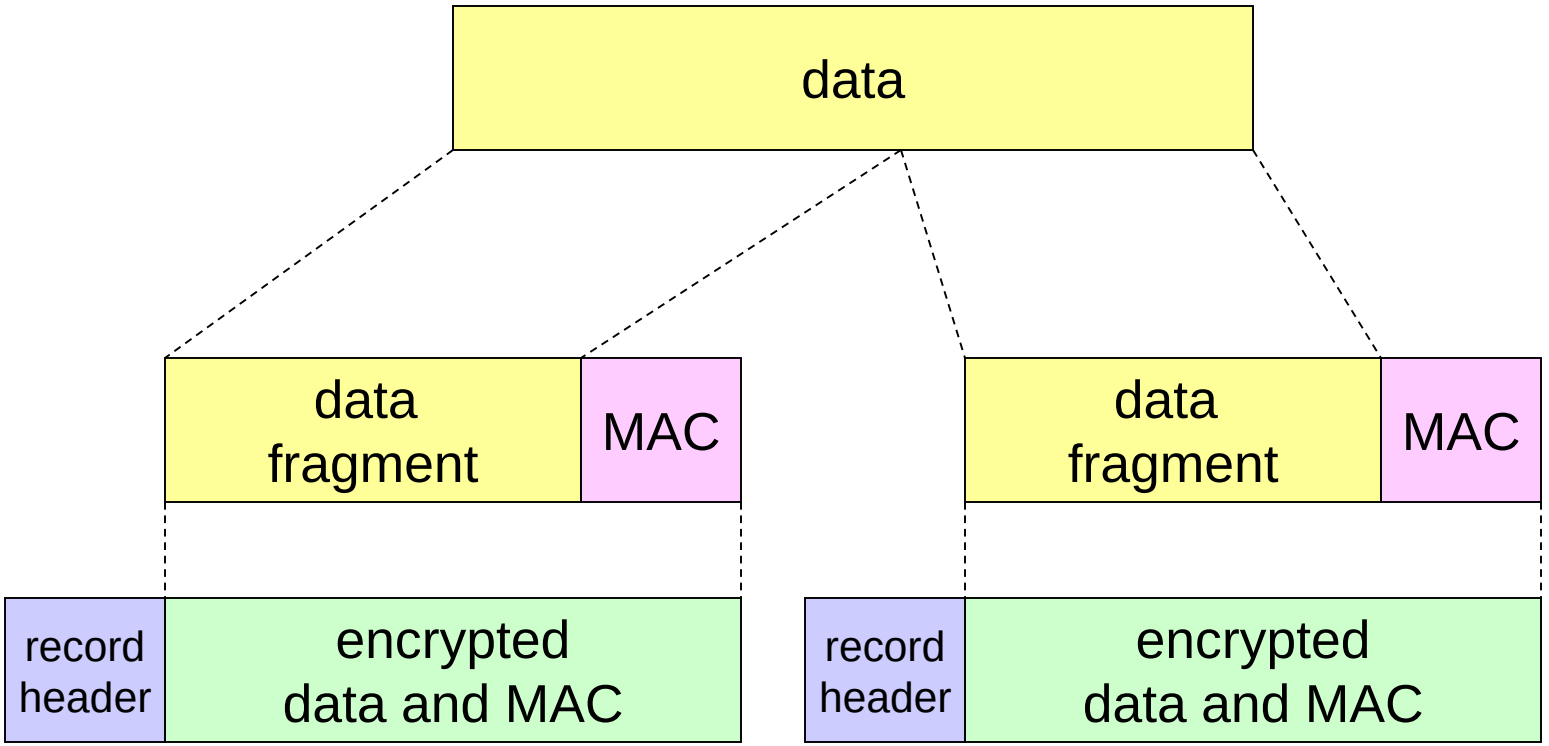
last 2 steps protect handshake from tampering

- client typically offers range of algorithms, some strong, some weak
- man-in-the middle could delete stronger algorithms from list
- last 2 steps prevent this
  - last two messages are encrypted

Real SSL: handshaking (4)

- why two random nonces?
- suppose Trudy sniffs all messages between Alice & Bob
- next day, Trudy sets up TCP connection with Bob, sends exact same sequence of records
  - Bob (Amazon) thinks Alice made two separate orders for the same thing
  - solution: Bob sends different random nonce for each connection. This causes encryption keys to be different on the two days
  - Trudy's messages will fail Bob's integrity check

SSL record protocol

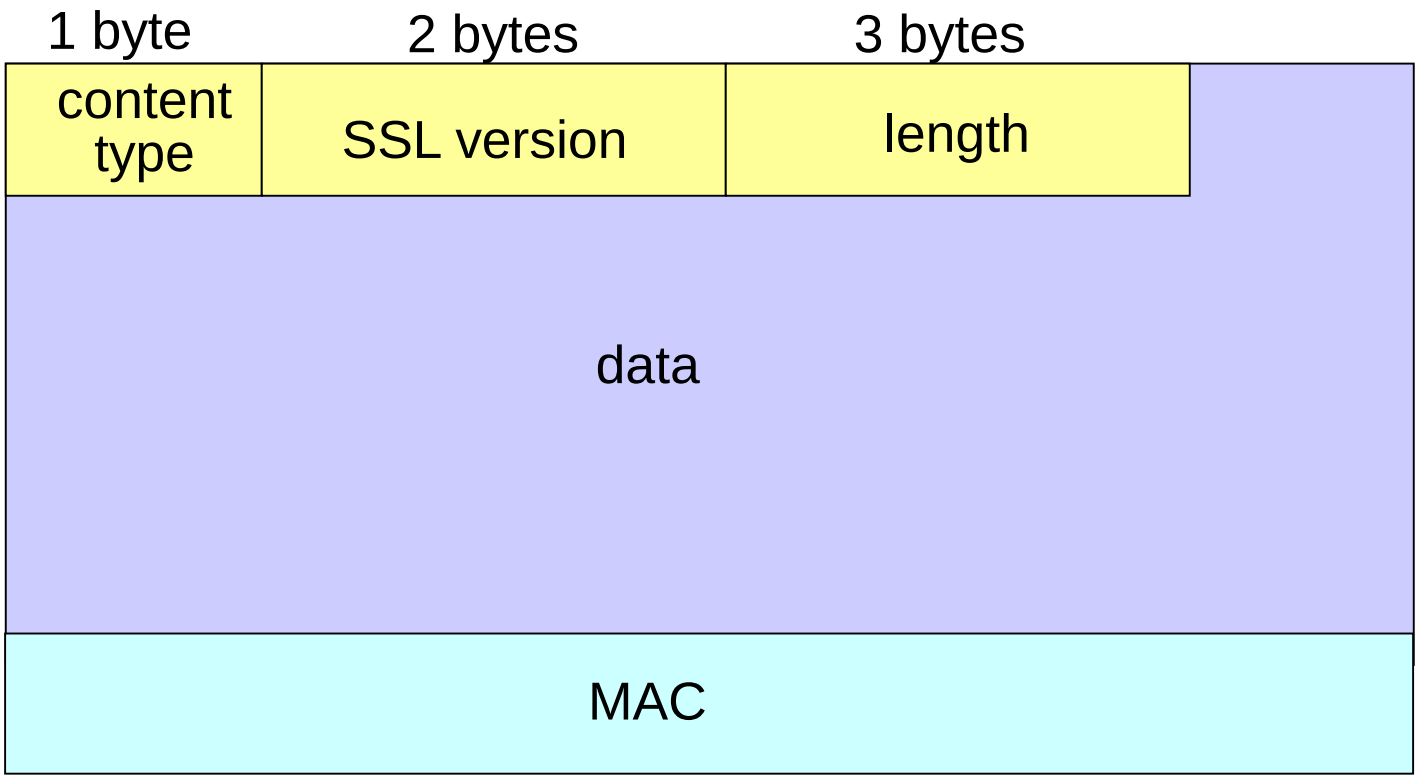


record header: content type; version; length

MAC: includes sequence number, MAC key M_x

fragment: each SSL fragment 2^{14} bytes (~16 Kbytes)

SSL record format



data and MAC encrypted (symmetric algorithm)

Real SSL connection

handshake: ClientHello

handshake: ServerHello

handshake: Certificate

handshake: ServerHelloDone

handshake: ClientKeyExchange

ChangeCipherSpec

handshake: Finished

everything henceforth is encrypted

ChangeCipherSpec

handshake: Finished

application_data

application_data

Alert: warning, close_notify

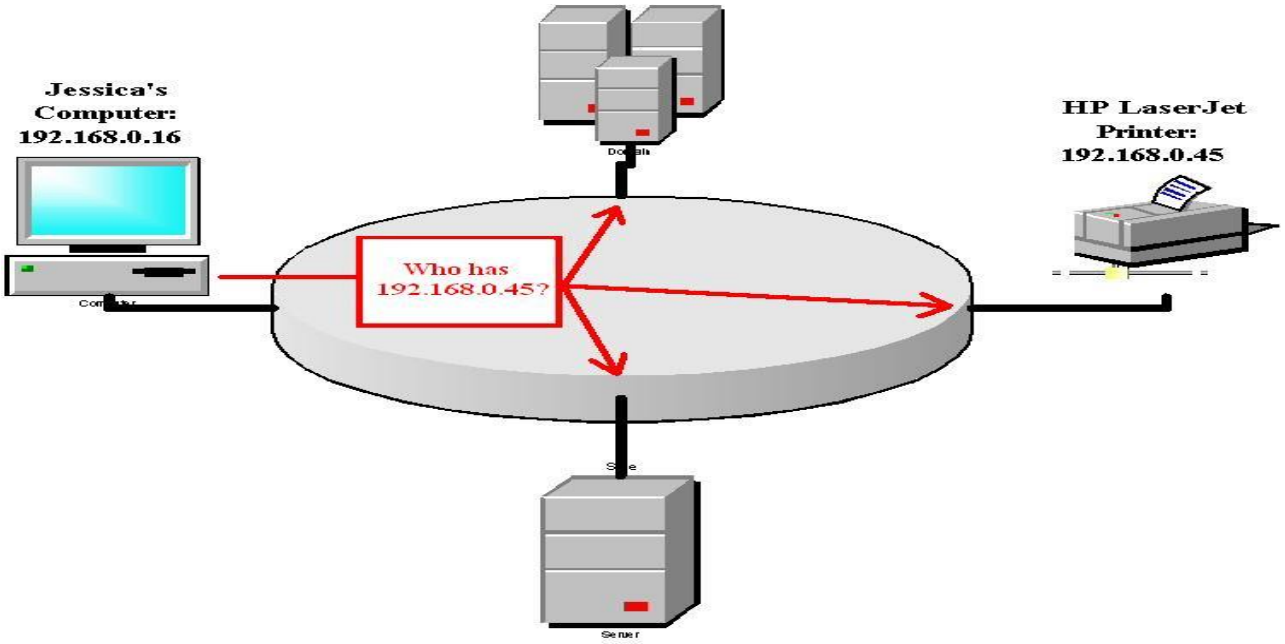
TCP FIN follows

Key derivation

- client nonce, server nonce, and pre-master secret input into pseudo random-number generator.
  - produces master secret
- master secret and new nonces input into another random-number generator: "key block"
  - because of resumption: TBD
- key block sliced and diced:
  - client MAC key
  - server MAC key
  - client encryption key
  - server encryption key
  - client initialization vector (IV)
  - server initialization vector (IV)

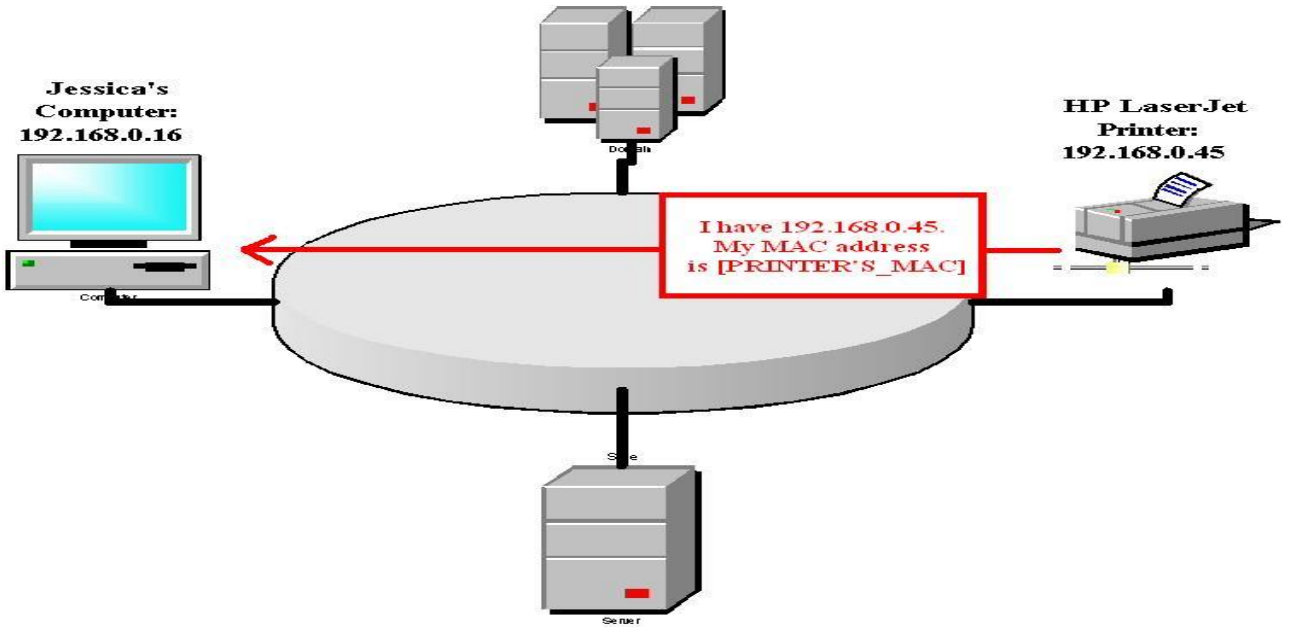
ARP

- ARP request
  - Computer A asks the network, "Who has this IP address?“



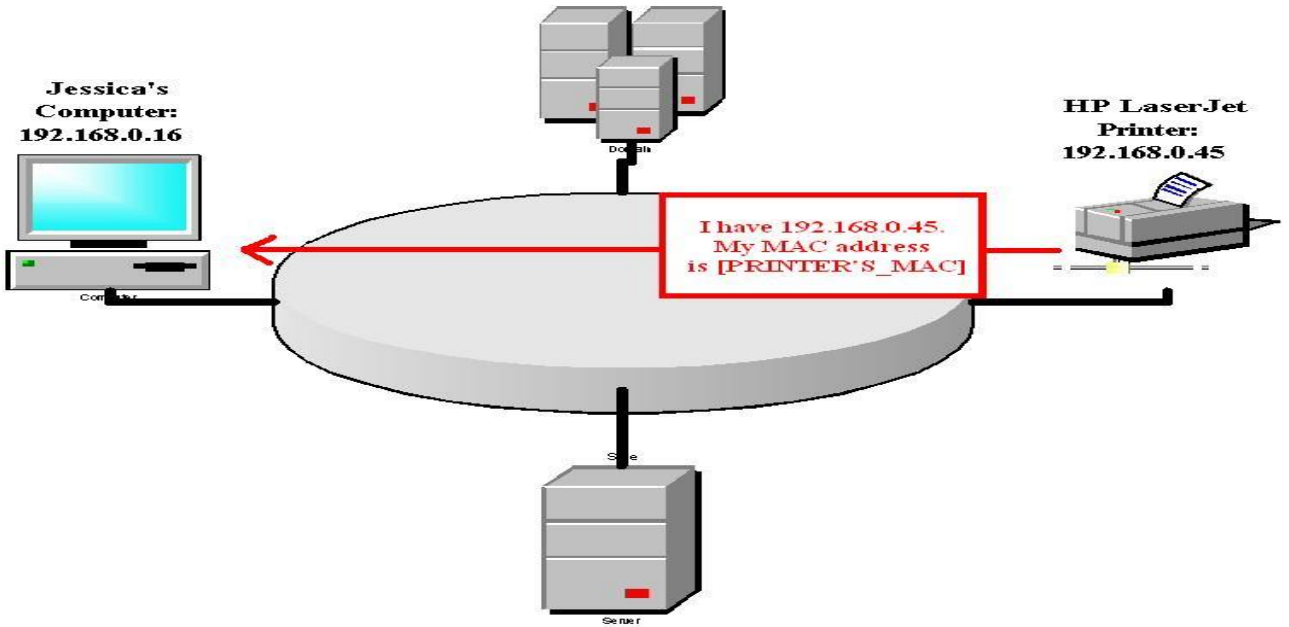
ARP

- ARP reply
  - Computer B tells Computer A, "I have that IP. My Physical Address is [whatever it is].“



ARP

- ARP reply
  - Computer B tells Computer A, "I have that IP. My Physical Address is [whatever it is]."



Cache table

- A short-term memory of all the IP addresses and Physical addresses
- Ensures that the device doesn't have to repeat ARP Requests for devices it has already communicated with
- Implemented as an array of entries
- Entries are updated

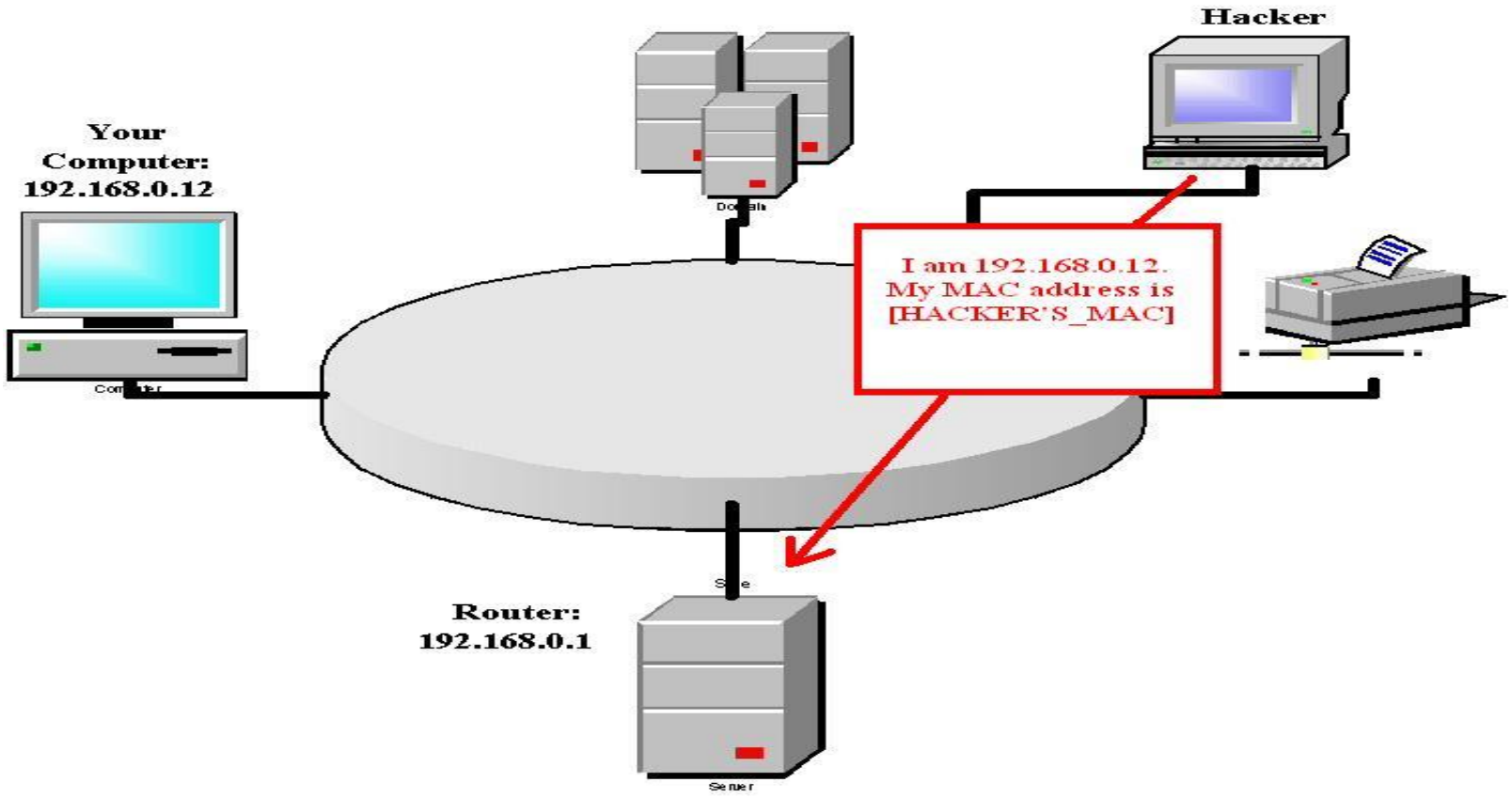
Cache table

State	Queue	Attempt	Time-out	IP Address	Physical Address
R	5		900	180.3.6.1	ACAE32457342
P	2	2		129.34.4.8	
P	14	5		201.11.56.7	
R	8		450	114.5.7.89	457342ACAE32
P	12	1		220.55.5.7	
F					
R	9		60	19.1.7.82	4573E3242ACA
P	18	3		188.11.8.71	

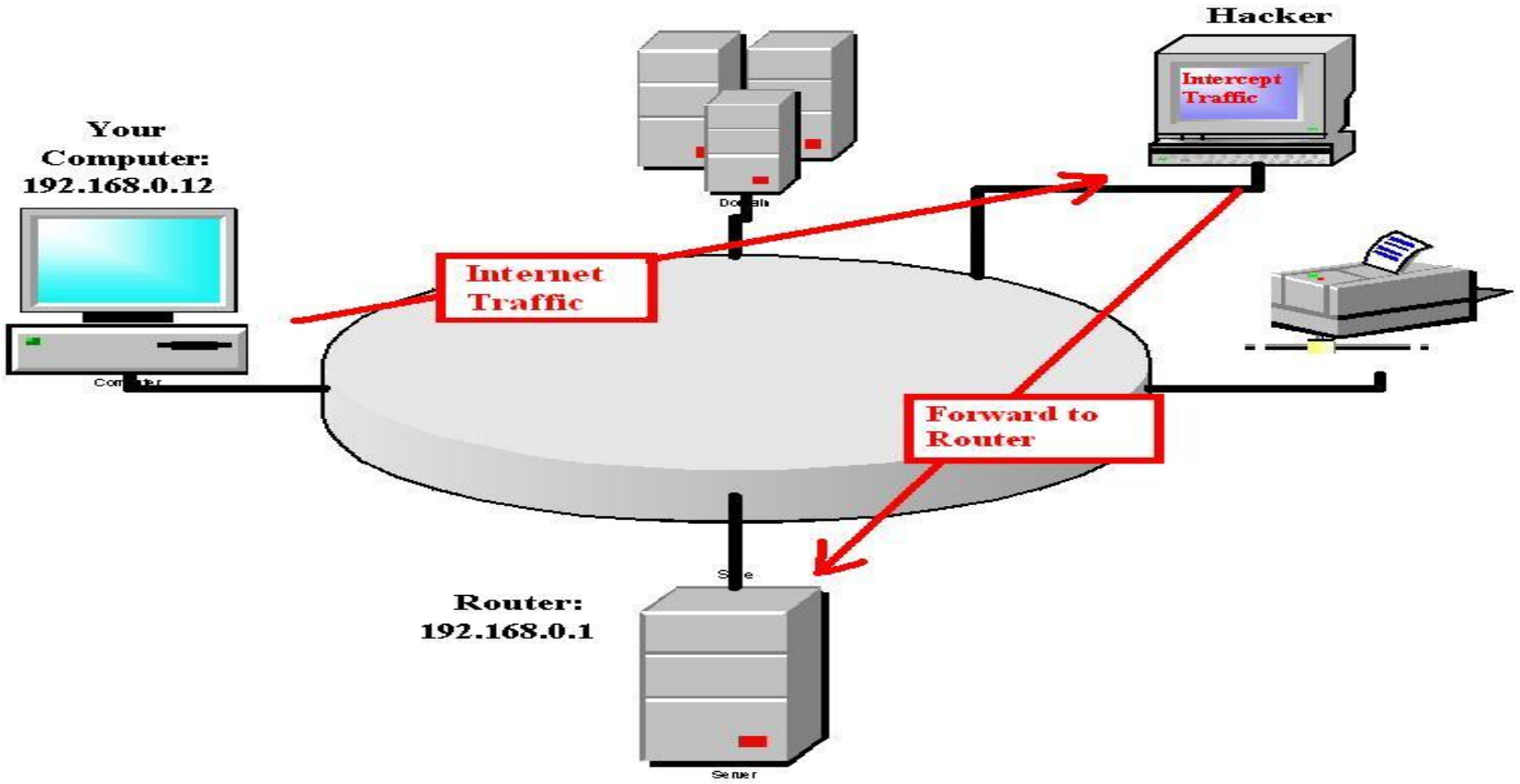
ARP spoofing

- Simplicity also leads to major insecurity
  - No Authentication
    - ARP provides no way to verify that the responding device is really who it says it is
    - Stateless protocol
      - Updating ARP Cache table
- Attacks
  - DOS
    - Hacker can easily associate an operationally significant IP address to a false MAC address
  - Man-in-the-Middle
    - Intercept network traffic between two devices in your network

ARP spoofing (MITM)



ARP spoofing (MITM)



Prevent ARP spoofing

- For Small Network
  - Static Arp Cache table
- For Large Network
  - Arpwatch
- As an administrator, check for multiple Physical addresses responding to a given IP address