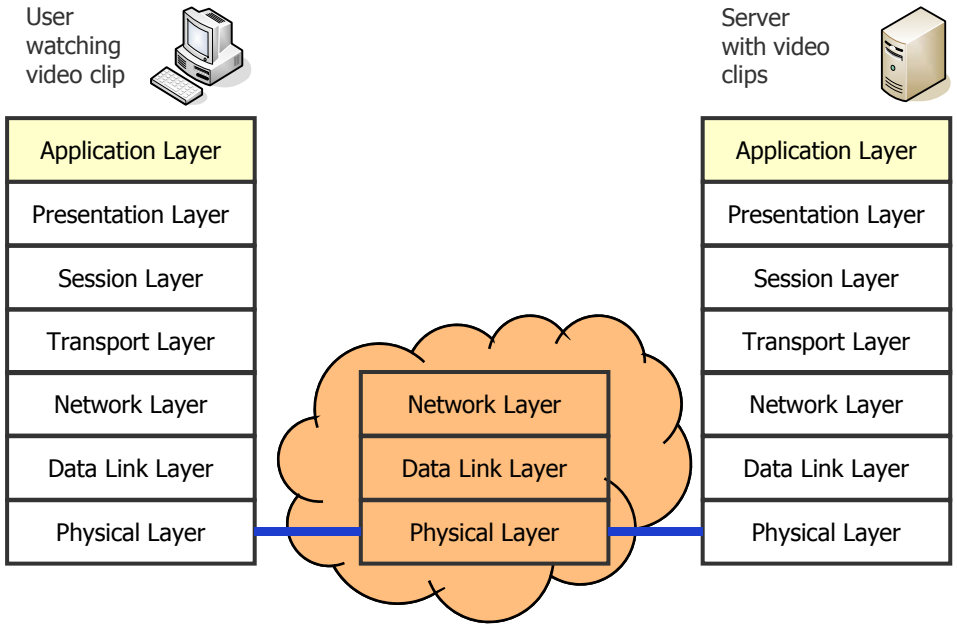
Telematics
Chapter 11: Network Security

Prof. Dr. Mesut Güneş
Computer Systems and Telematics (CST)
Distributed, embedded Systems
Institute of Computer Science
Freie Universität Berlin
http://cst.mi.fu-berlin.de



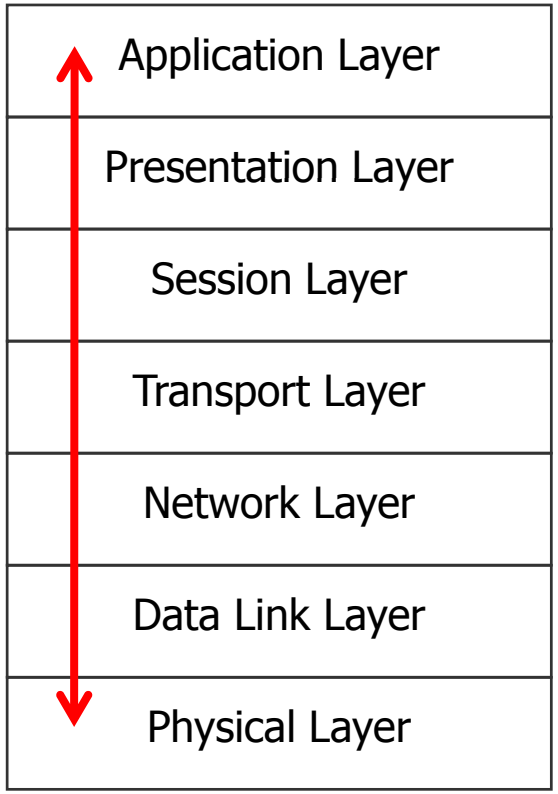
Contents

- What is network security?
- Principles of cryptography
- Message Integrity
- Securing e-mail
- Securing TCP connections: SSL
- Network Layer security: IPsec
- Securing Wireless LANs
- Operational security: Firewalls and IDS

Design Issues

- Understand principles of network security:
  - cryptography and its many uses beyond "confidentiality"
  - authentication
  - message integrity
- Security in practice:
  - firewalls and intrusion detection systems
  - security in application, transport, network, link layers

OSI Reference Model

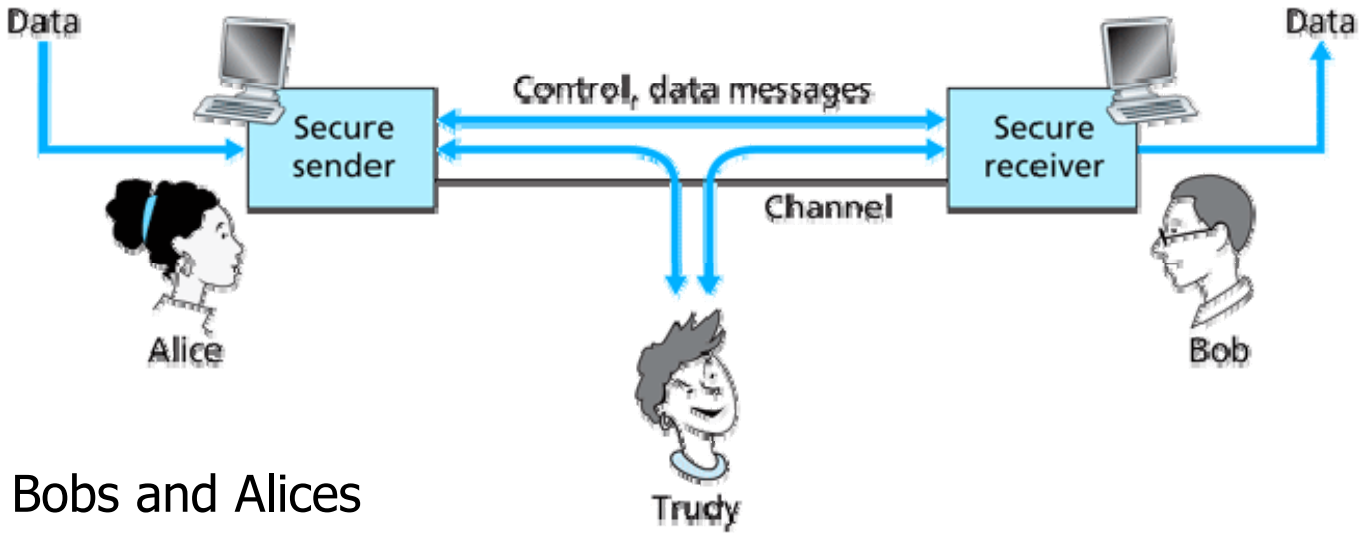


What is network security?

- Confidentiality: only sender and intended receiver should "understand" message contents
  - Sender encrypts message
  - Receiver decrypts message
- Authentication: sender and receiver want to confirm identity of each other
- Message integrity: sender and receiver want to ensure that the message can not altered (in transit, or afterwards) without detection
- Access and availability: services must be accessible and available to users

Friends and enemies: Alice, Bob, Trudy

- Well-known in network security world
  - Bob, Alice (lovers!) want to communicate "securely"
  - Trudy (intruder) may intercept, delete, add messages



- Real-life Bobs and Alices
  - Telnet client and server
  - Web browser/server for electronic transactions
  - On-line banking client/server
  - DNS servers
  - Routers exchanging routing table updates

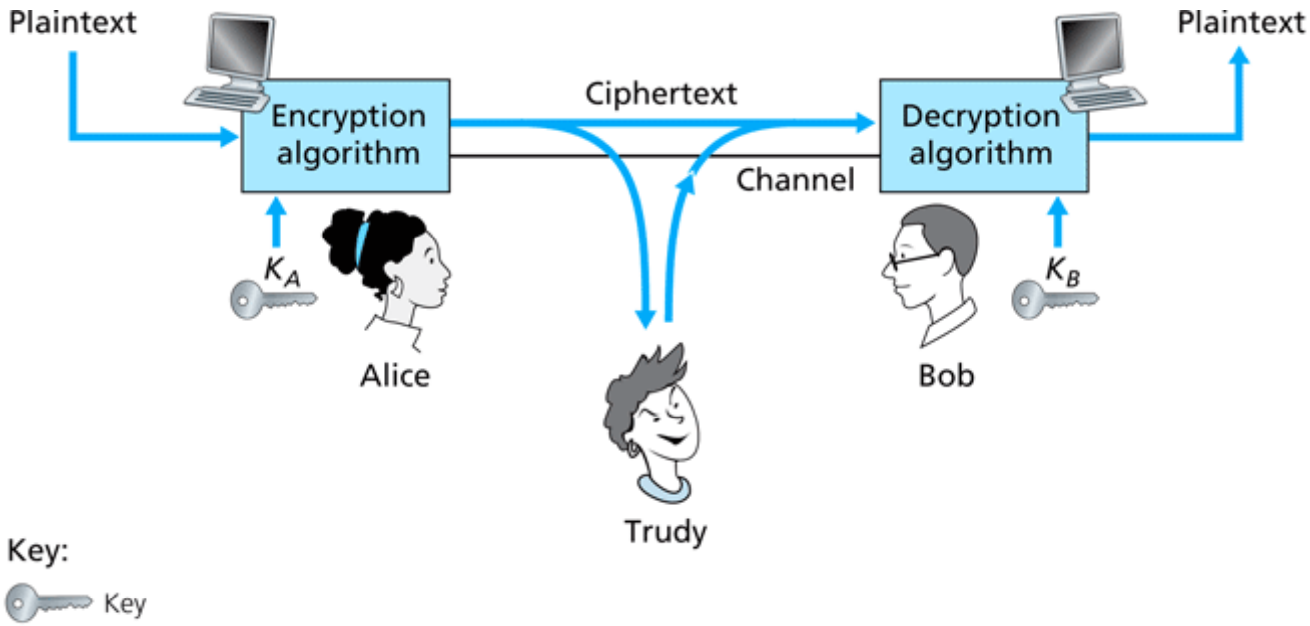
There are bad guys (and girls) out there!

- What can a "bad guy" do?
- A lot!
  - Eavesdrop: intercept messages
  - Actively insert messages into connection
  - Impersonation: can fake (spoof) source address in packet (or any field in packet)
  - Hijacking: "take over" ongoing connection by removing sender or receiver, inserting himself in place
  - Denial of service: prevent service from being used by others (e.g., by overloading resources)

Principles of cryptography

The language of cryptography

- Plaintext message m
- Keys K_A, K_B
- Encryption: ciphertext $c=K_A(m)$, encrypted with key K_A
- Decryption: plaintext $m = K_B(c) = K_B(K_A(m))$



Simple encryption scheme

- Substitution cipher: substituting one thing for another
  - Monoalphabetic cipher: substitute one letter for another

```
plaintext:   abcdefghijklmnopqrstuvwxyz

ciphertext:  mnbvcxzasdfghjklpoiuytrewq
```

e.g.:

```
plaintext: bob i love you alice
ciphertext: nkn s gktc wky mgsbc
```

- Key: the mapping from the set of 26 letters to the set of 26 letters

Polyalphabetic encryption

- n monoalphabetic ciphers: M_1, M_2, …, M_n
- Cycling pattern:
  - n=4
  - M_1,M_3,M_4,M_3,M_2; M_1,M_3,M_4,M_3,M_2
- For each new plaintext symbol, use subsequent monoalphabetic pattern in cyclic pattern
  - dog: d from M1, o from M3, g from M4
- Key: the n ciphers and the cyclic pattern

Breaking an encryption scheme

1) Cipher-text only attack
   - Trudy has ciphertext that she can analyze

- Two approaches:
  - Search through all keys: must be able to differentiate resulting plaintext from gibberish
  - Statistical analysis

2) Known-plaintext attack
   - Trudy has some plaintext corresponding to some ciphertext
   - e.g., in monoalphabetic cipher, Trudy determines pairings for a, l, i, c, e, b, o

3) Chosen-plaintext attack
   - Trudy can get the ciphertext for some chosen plaintext

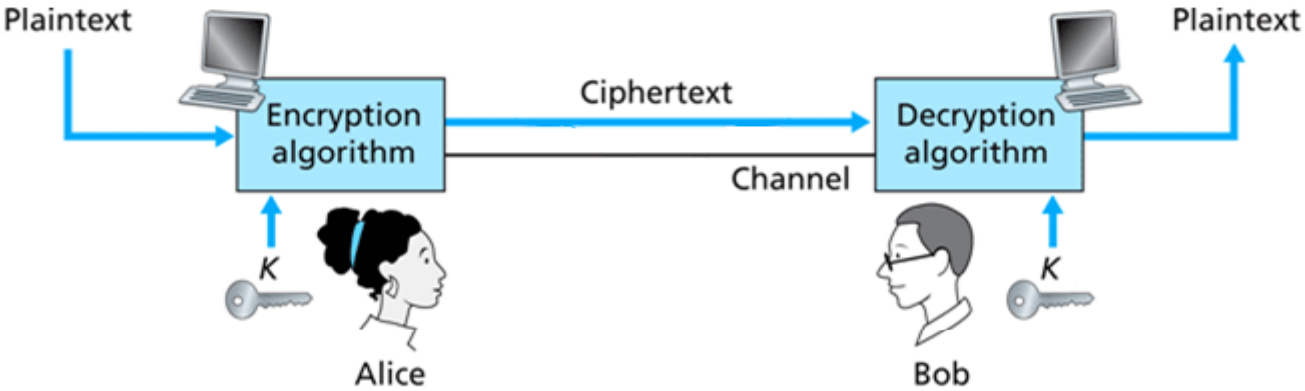
Types of Cryptography

- Cryptography often uses keys:
  - Algorithm is known to everyone
  - Only "keys" are secret
- Symmetric key cryptography
  - Involves the use one key
- Public key cryptography
  - Also known as asymmetric key cryptography
  - Involves the use of two keys
- Hash functions
  - Involves the use of no keys
  - Nothing secret: How can this be useful?

Symmetric Cryptography

Symmetric key cryptography

- Symmetric key cryptography
  - Bob and Alice share same (symmetric) key: K
  - e.g., key is knowing substitution pattern in monoalphabetic substitution cipher
- How do Bob and Alice agree on a key?

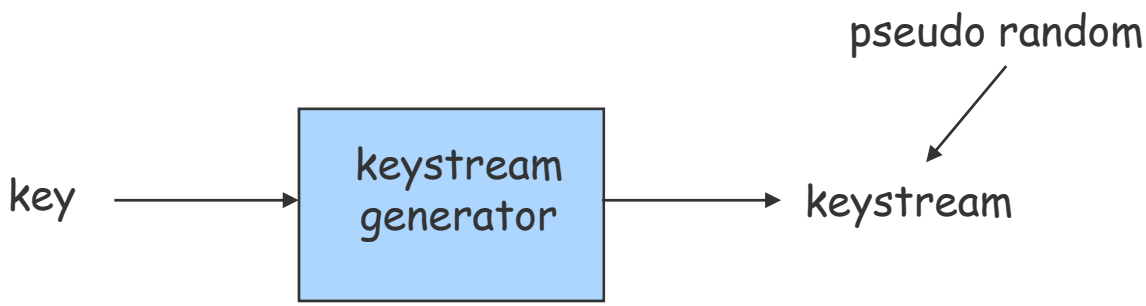


- Two types of symmetric ciphers
  - Stream ciphers: encrypt one bit at time
  - Block ciphers
    - Break plaintext message in equal-size blocks
    - Encrypt each block as a unit

Stream Ciphers

- Combine each bit of keystream with bit of plaintext to get bit of ciphertext
  - $m(i)$ = i-th bit of message
  - $K_s(i)$ = i-th bit of keystream
  - $c(i)$ = i-th bit of ciphertext

  - $c(i) = K_s(i) \oplus m(i)$ ($\oplus$ = exclusive or)
  - $m(i) = K_s(i) \oplus c(i)$



RC4 Stream Cipher

- RC4 is a popular stream cipher
  - Extensively analyzed and considered good
  - Key can be from 1 to 256 bytes
  - Used in WEP for IEEE 802.11
  - Can be used in SSL

Block ciphers

- Message to be encrypted is processed in blocks of k bits
  - e.g., 64-bit blocks
- 1-to-1 mapping is used to map k-bit block of plaintext to k-bit block of ciphertext
- Example with k=3:

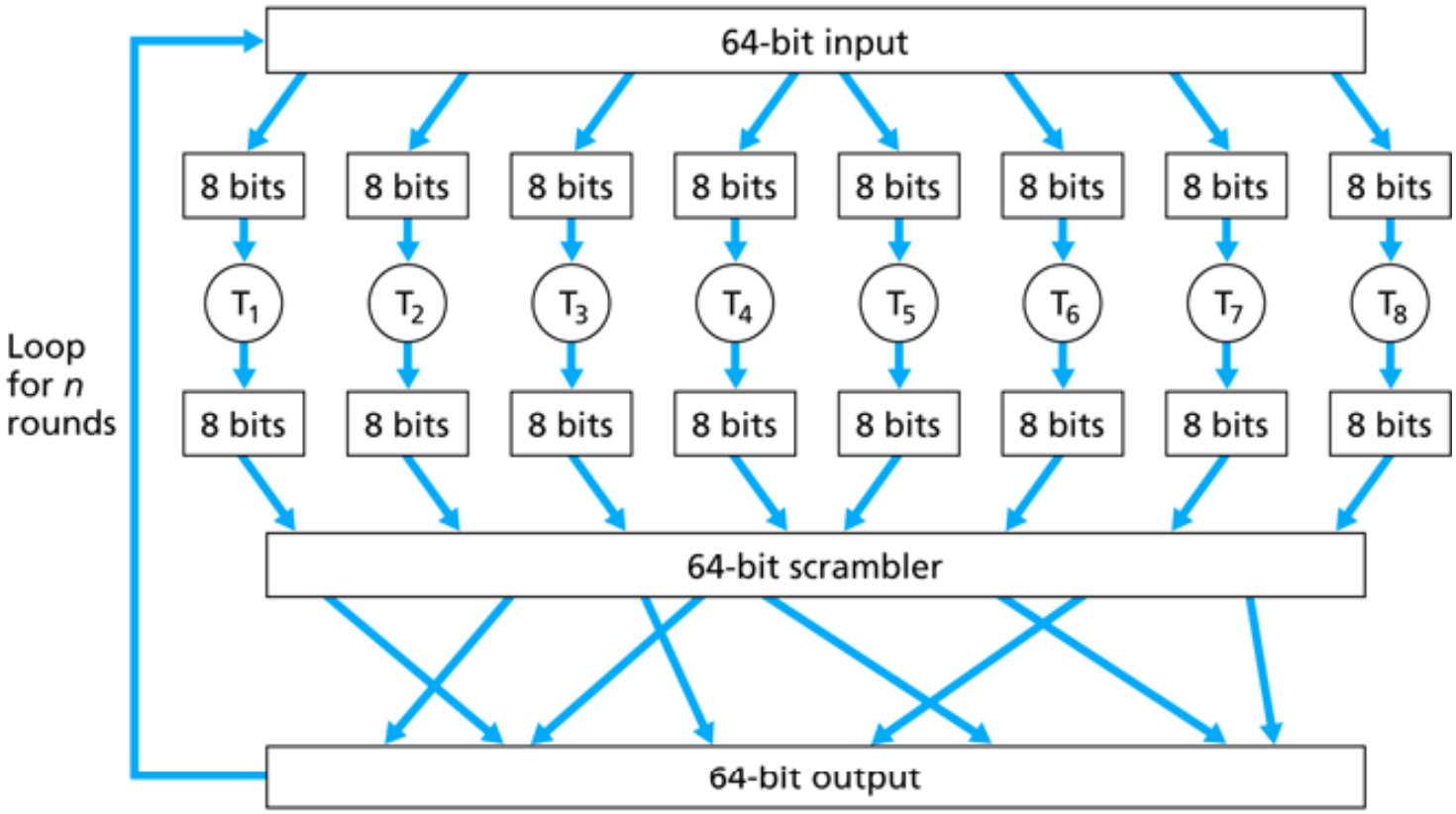
input	output
000	110
001	111
010	101
011	100
100	011
101	010
110	000
111	001

- What is the ciphertext for 010 110 001 111?

Block ciphers

- How many possible mappings are there for k=3?
  - How many 3-bit inputs?
  - How many permutations of the 3-bit inputs?
  - Answer: 40,320
    - Not very many, easy to try all possible values with a computer.
- In general, $2^k!$ mappings
  - For k=4 ➡ 20922789888000 permutations
  - Huge for k=64
- Problem:
  - Table approach requires table with 2^{64} entries, each entry with 64 bits
- Table too big: instead use function that simulates a randomly permuted table

Prototype function



Why rounds in prototpe?

- If only a single round, then one bit of input affects at most 8 bits of output.
- In 2nd round, the 8 affected bits get scattered and inputted into multiple substitution boxes.
- How many rounds?
  - How many times do you need to shuffle cards
  - Becomes less efficient as n increases

Encrypting a large message

- Why not just break message in 64-bit blocks, encrypt each block separately?
  - If same block of plaintext appears twice, will give same ciphertext.

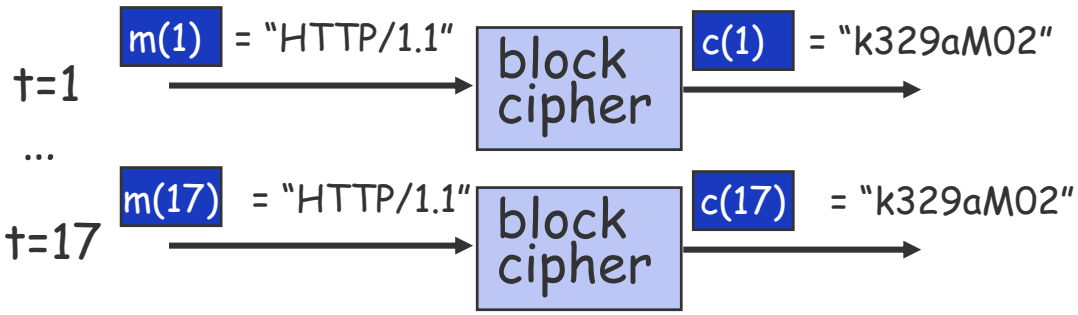
- How about:
  - Generate random 64-bit number $r(i)$ for each plaintext block $m(i)$
  - Calculate $c(i) = K_S(m(i) \oplus r(i))$
  - Transmit $c(i)$, $r(i)$, $i=1,2,\ldots$
  - At receiver: $m(i) = K_S(c(i)) \oplus r(i)$
  - Problem: inefficient, need to send $c(i)$ and $r(i)$

Cipher Block Chaining (CBC)

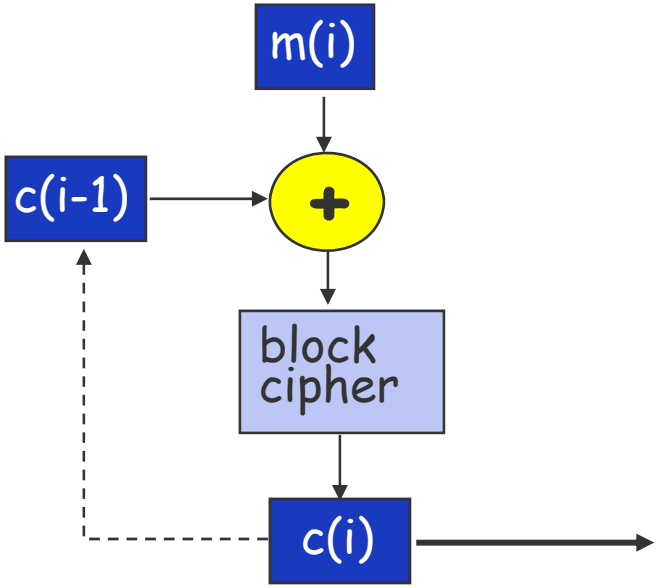
- CBC generates its own random numbers
  - Have encryption of current block depend on result of previous block
  - $c(i) = K_S(m(i) \oplus c(i-1))$
  - $m(i) = K_S(c(i)) \oplus c(i-1)$
- How do we encrypt first block?
  - Initialization vector (IV): random block = $c(0)$
  - IV does not have to be secret
- Change IV for each message (or session)
  - Guarantees that even if the same message is sent repeatedly, the ciphertext will be completely different each time

Cipher Block Chaining

- Cipher block: if input block repeated, will produce same cipher text:



- Cipher block chaining: XOR i-th input block, m(i), with previous block of cipher text, c(i-1)
  - c(0) transmitted to receiver in clear
  - what happens in "HTTP/1.1" scenario from above?



Symmetric Key Cryptography

Symmetric Key Cryptography: DES

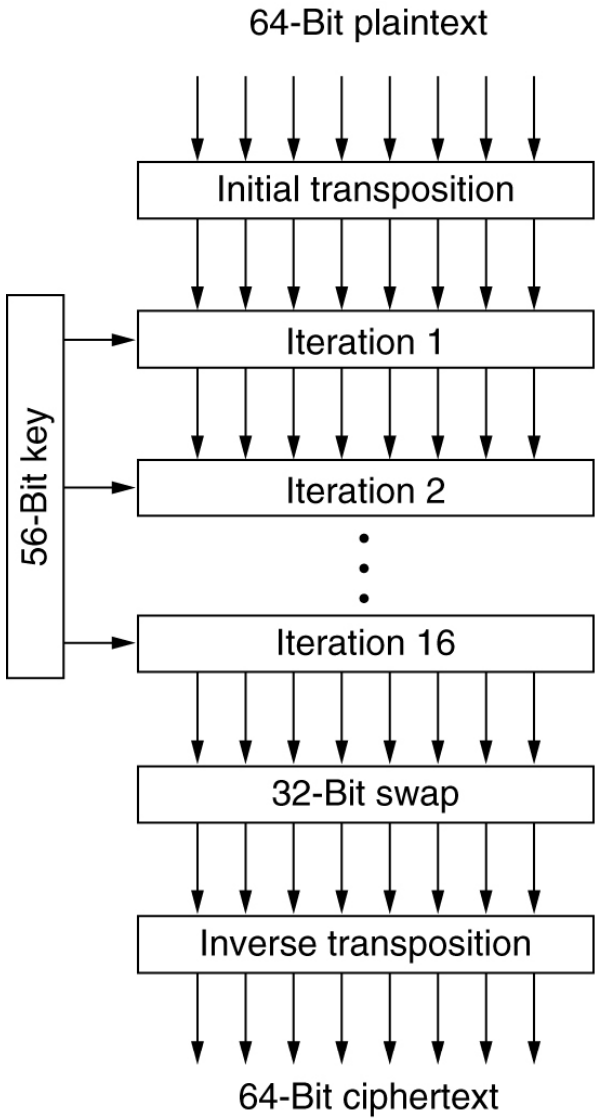
- DES: Data Encryption Standard
  - US encryption standard [NIST 1993]
  - 56-bit symmetric key, 64-bit plaintext input
  - Block cipher with cipher block chaining

- How secure is DES?
  - DES Challenge: 56-bit-key-encrypted phrase decrypted (brute force) in less than a day
  - No known good analytic attack

- Making DES more secure:
  - 3DES: encrypt 3 times with 3 different keys
  - Actually: encrypt, decrypt, encrypt

Symmetric Key Cryptography: DES

- Initial permutation
- 16 identical "rounds" of function application
- Final permutation



AES: Advanced Encryption Standard

- New (Nov. 2001) symmetric-key NIST standard, replacing DES
  - Processes data in 128 bit blocks
  - Key length: 128, 192, or 256 bit

- Brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES

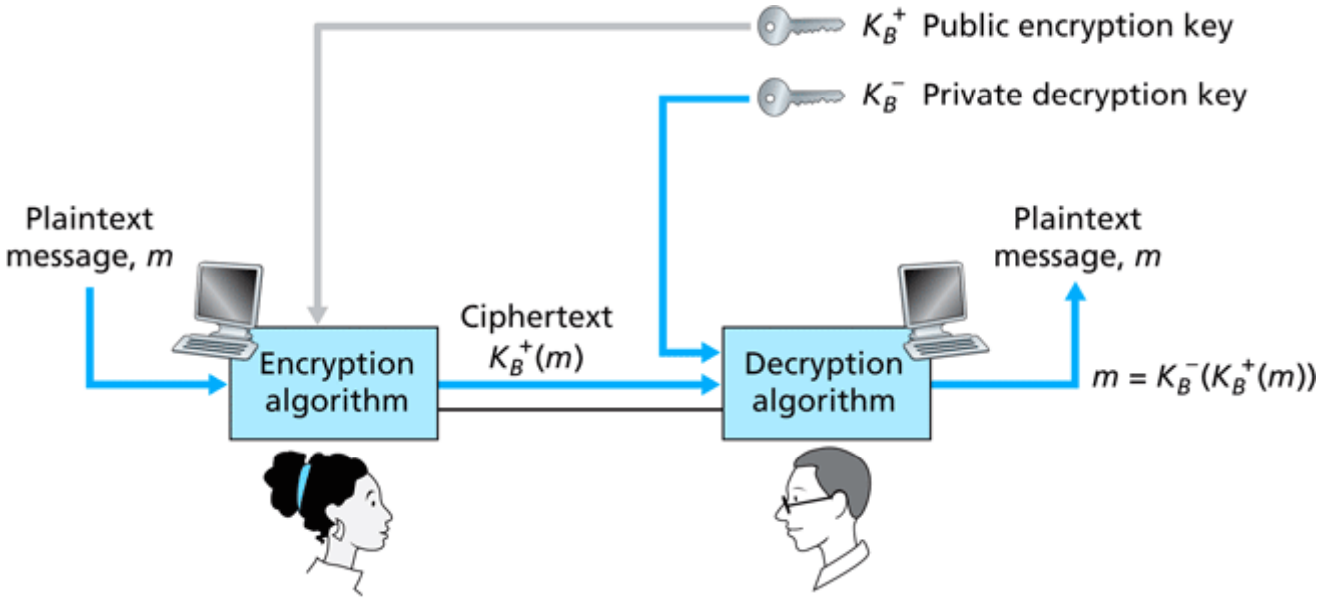
Public Key Cryptography

Public Key Cryptography

- Symmetric key cryptography
  - Requires sender, receiver know shared secret key
  - How to agree on key in first place (particularly if never "met")?

- Public key cryptography
  - Radically different approach [Diffie-Hellman76, RSA78]
  - Sender, receiver do not share secret key
  - Public encryption key known to all
  - Private decryption key known only to receiver

Public Key Cryptography



Public Key Cryptography

- Requirements:

1) Need: $K_B^+(\bullet)$ and $K_B^-(\bullet)$ such that $K_B^-(K_B^+(m)) = \mathrm{m}$

2) Given public key K_B^-, it should be impossible to compute private key K_B^+

- RSA: Rivest, Shamir, Adleman algorithm

Prerequisite: Modular Arithmetic

- $x \bmod n$ = remainder of x when divided by n
- Facts:
  - $[(a \bmod n) + (b \bmod n)] \bmod n = (a+b) \bmod n$
  - $[(a \bmod n) - (b \bmod n)] \bmod n = (a-b) \bmod n$
  - $[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$

- Thus

$$(a \bmod n)^d \bmod n = a^d \bmod n$$

- Example: $x=14$, $n=10$, $d=2$

$$(x \bmod n)^d \bmod n = 14^2 \bmod 10 = 6$$
$$x^d = 14^2 = 196$$
$$196 \bmod 10 = 6$$

RSA: Getting Ready

- A message is a bit pattern
- A bit pattern can be uniquely represented by an integer number
- Thus encrypting a message is equivalent to encrypting a number

- Example
  - m=10010001
    This message is uniquely represented by the decimal number 145
  - To encrypt m, we encrypt the corresponding number, which gives a new number (the ciphertext)

RSA: Creating public/private key pair

1. Choose two large **prime numbers** p, q. (e.g., 1024 bits each)
2. Compute n = pq, z = (p-1)(q-1)
3. Choose e (with e<n) that has no common factors with z. (e, z are "relatively prime").
4. Choose d such that ed-1 is exactly divisible by z. (in other words: ed mod z = 1).
5. Public key is $\underbrace{(n,e)}_{K_B^+}$. Private key is $\underbrace{(n,d)}_{K_B^-}$.

RSA: Encryption and Decryption

Given (n,e) and (n,d) as computed above

1. To encrypt message m (<n), compute

   $c = m^e \bmod n$

2. To decrypt received bit pattern, c, compute

   $m = c^d \bmod n$

Magic happens!

$$m = \underbrace{(m^e \bmod n)}_{c}{}^d \bmod n$$

RSA Example

Bob chooses p=5, q=7 ➡ n=35, z=24

e=5 (so e, z relatively prime)
d=29 (so ed-1 exactly divisible by z)

Encrypting 8-bit messages:

	m	numeric value	m^e	$c = m^e \bmod n$
encrypt:	l	12	248832	17

	c	c^d	$m = c^d \bmod n$
decrypt:	17	481968572106 7509150914118 25223071697	12

Why does RSA work?

- Must show that $c^d \bmod n = m$
  where $c = m^e \bmod n$

- Fact: for any x and y: $x^y \bmod n = x^{(y \bmod z)} \bmod n$
  - where $n = pq$ and $z = (p-1)(q-1)$

- Thus

$$
\begin{aligned}
c^d \bmod n &= (m^e \bmod n)^d \bmod n \\
&= m^{ed} \bmod n \\
&= m^{(ed \bmod z)} \bmod n \\
&= m^1 \bmod n \\
&= m
\end{aligned}
$$

RSA: An important property

The following property will be *very* useful later:

$$\underbrace{K_B^-(K_B^+(m))}_{\text{use public key first, then private key}} = m = \underbrace{K_B^+(K_B^-(m))}_{\text{use private key first, then public key}}$$

Result is the same!

RSA: An important property

Why $K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$?

Follows directly from modular arithmetic:

$$
\begin{aligned}
(m^e \bmod n)^d \bmod n &= m^{ed} \bmod n \\
&= m^{de} \bmod n \\
&= (m^d \bmod n)^e \bmod n
\end{aligned}
$$

Why is RSA Secure?

- Suppose you know Bob's public key (n,e). How hard is it to determine d?
- Essentially need to find factors of n without knowing the factors p and q.
- Fact: factoring a big number is hard.

- Generating RSA keys
  - Have to find big primes p and q
  - Approach: make good guess then apply testing rules

Session keys

- Exponentiation is computationally intensive
- DES is at least 100 times faster than RSA

- Session key K_S
  - Bob and Alice use RSA to exchange a symmetric key K_S
  - Once both have K_S, they use symmetric key cryptography

Message Integrity

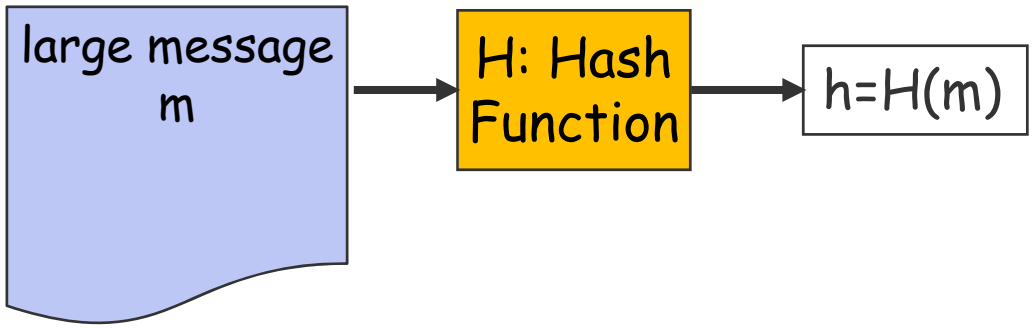
Message Integrity

- Allows communicating parties to verify that received messages are authentic.
  - Content of message has not been altered
  - Source of message is who/what you think it is
  - Message has not been replayed
  - Sequence of messages is maintained
- Let's first talk about message digests

Message Digests

- Function H() that takes as input an arbitrary length message and outputs a fixed-length string: "message signature"
- Note that H() is a many-to-1 function
- H() is often called a "hash function"

- Desirable properties
  - Easy to calculate
  - Irreversibility: Can't determine m from H(m)
  - Collision resistance:
    - Computationally difficult to produce m and m' such that H(m) = H(m')
  - Seemingly random output



Internet checksum: poor message digest

- Internet checksum has some properties of hash functions
  - Produces fixed length digest (16-bit sum) of input
  - Is many-to-one
  - But given message with given hash value, it is easy to find another message with same hash value
  - Example: Simplified checksum: add 4-byte chunks at a time:

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

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

Different messages
but identical checksums!

Hash Function Algorithms

- MD5 hash function widely used (RFC 1321)
  - Computes 128-bit message digest in 4-step process

- SHA-1 is also used
  - US standard [NIST, FIPS PUB 180-1]
  - 160-bit message digest

Message Authentication Code (MAC)

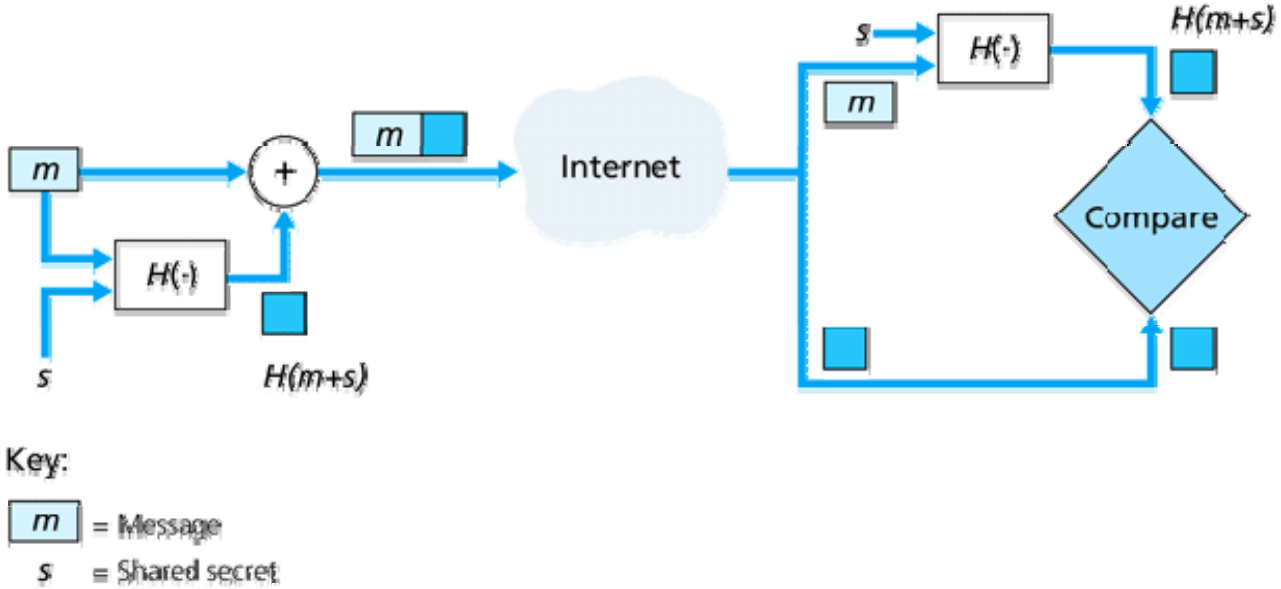
- Process of creating a MAC
  - Alice creates message m and calculates h=H(m)
  - Alice appends h to m, creating an extended message (m, h) and sends to Bob
  - Bob receives (m, h) and calculates H(m). If H(m) = h ➡ everything is fine!

- Problem
  - Trudy creates bogus message m' and pretends to be Alice
  - Trudy calculates h'=H(m') and sends (m', h') to Bob
  - Bob calculates H(m') and verifies to h'

- Need to authenticate the sender!

Message Authentication Code (MAC)

- Alice and Bob need to share additional secret: s
- Process of creating a MAC
  - Alice creates message m, concatenates *s* and calculates h=H(m+*s*)
  - Alice appends h to m, creating an extended message (m, h) and sends to Bob
  - Bob receives (m, h) and calculates H(m+*s*). If H(m) = h ➡ everything is fine!
- Problem
  - How do Alice and Bob agree on s?



Message Authentication Code (MAC)

- HMAC
  - Popular MAC standard
  - Addresses some subtle security flaws

- HMAC in process
  1. Concatenates secret to front of message
  2. Hashes concatenated message
  3. Concatenates the secret to front of digest
  4. Hashes the combination again

Example: OSPF

- Recall that OSPF is an intra-AS routing protocol
- Each router creates map of entire AS (or area) and runs shortest path algorithm over map.
- Router receives link-state advertisements (LSAs) from all other routers in AS.

- Attacks
  - Message insertion
  - Message deletion
  - Message modification

- How do we know if an OSPF message is authentic?

OSPF Authentication

- Within an Autonomous System, routers send OSPF messages to each other.
- OSPF provides authentication choices
  - No authentication
  - Shared password: inserted in clear in 64-bit authentication field in OSPF packet
  - Cryptographic hash
- Cryptographic hash with MD5
  - 64-bit authentication field includes 32-bit sequence number
  - MD5 is run over a concatenation of the OSPF packet and shared secret key
  - MD5 hash then appended to OSPF packet; encapsulated in IP datagram

Message Integrity

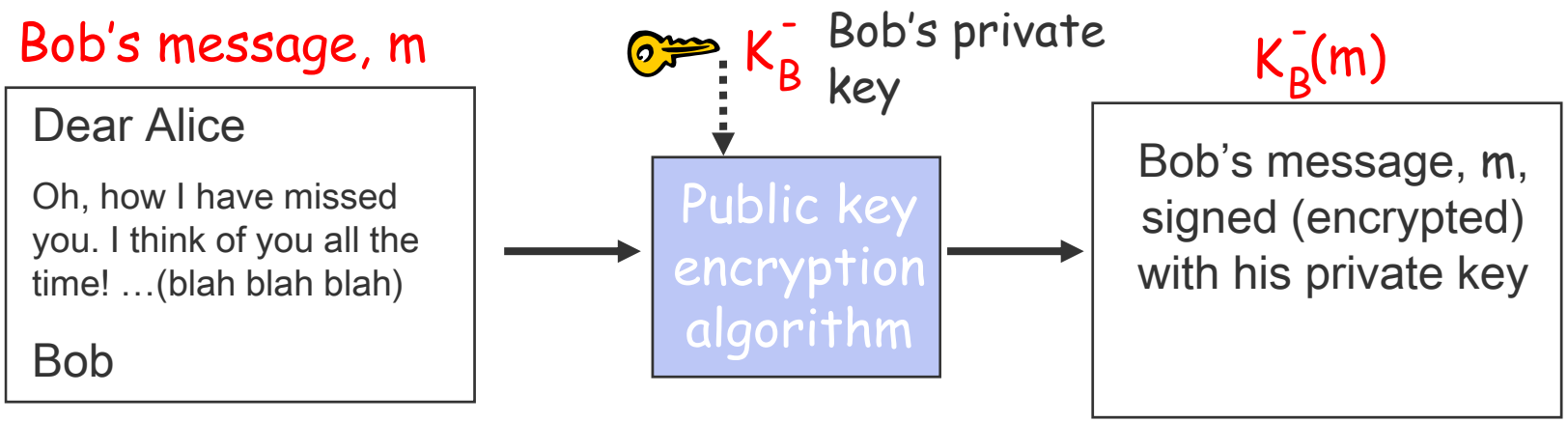
Digital Signatures

Digital Signatures

- Cryptographic technique analogous to hand-written signatures
- Sender (Bob) digitally signs document, establishing he is document owner/creator
- Goal is similar to that of a MAC, except now use public-key cryptography
  - Verifiable and
  - Nonforgeable
  - ➡ Recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document

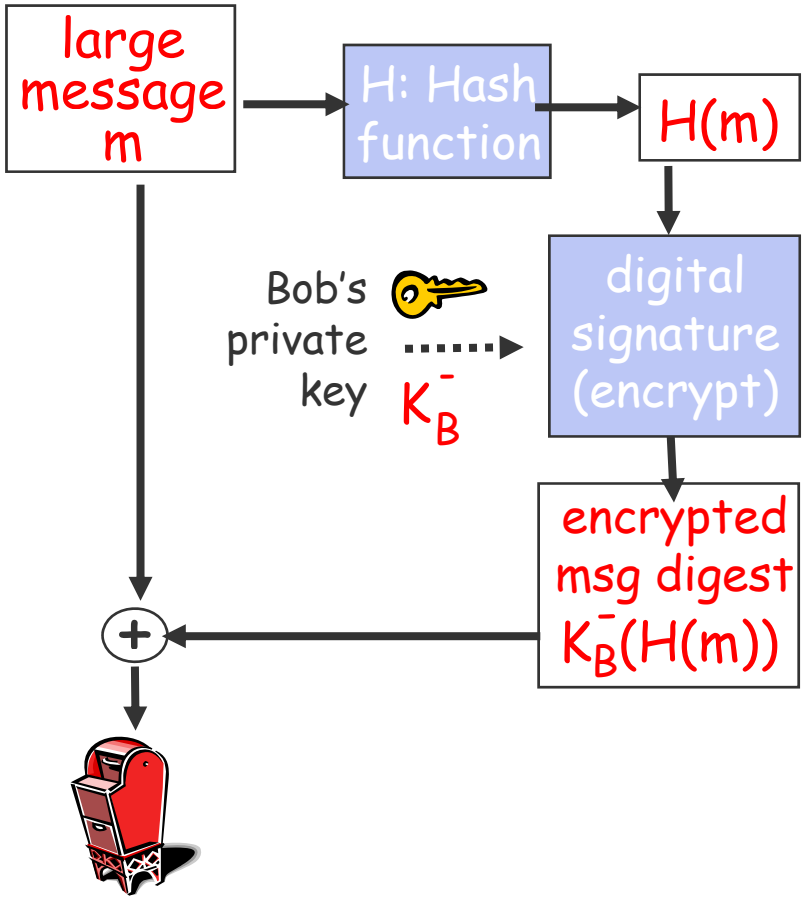
Digital Signatures

- Simple digital signature for message m:
- Bob signs m by encrypting with his private key K_B^-, creating "signed" message, $K_B^-(m)$

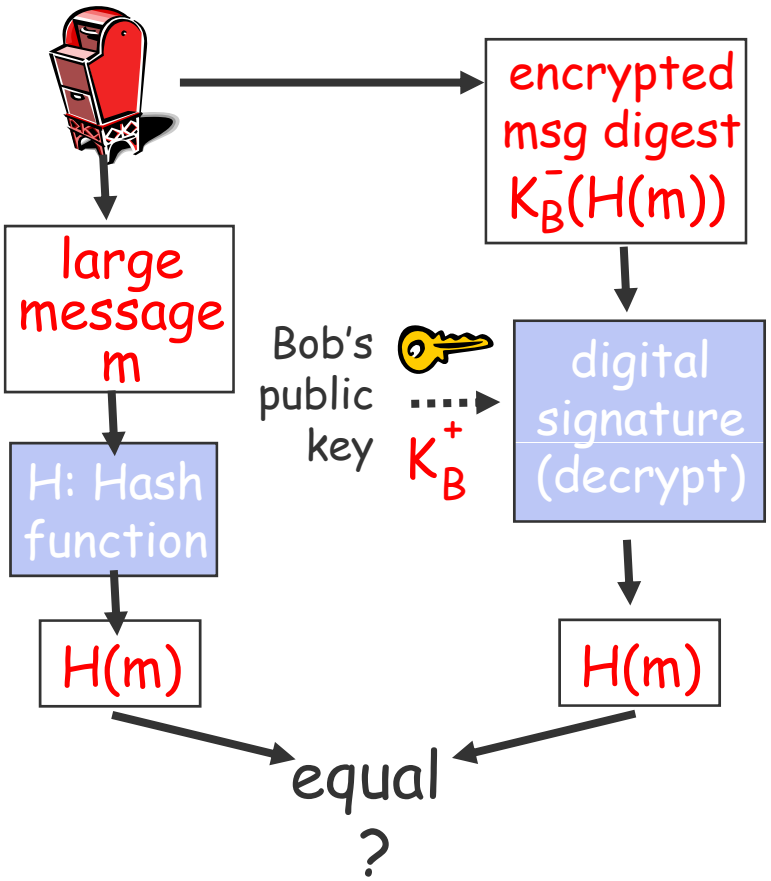


Digital signature = Signed message digest

Bob sends digitally signed message:



Alice verifies signature and integrity of digitally signed message:



Digital signature = Signed message digest

- Suppose Alice receives message m, digital signature $K_{B-}(m)$
- 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

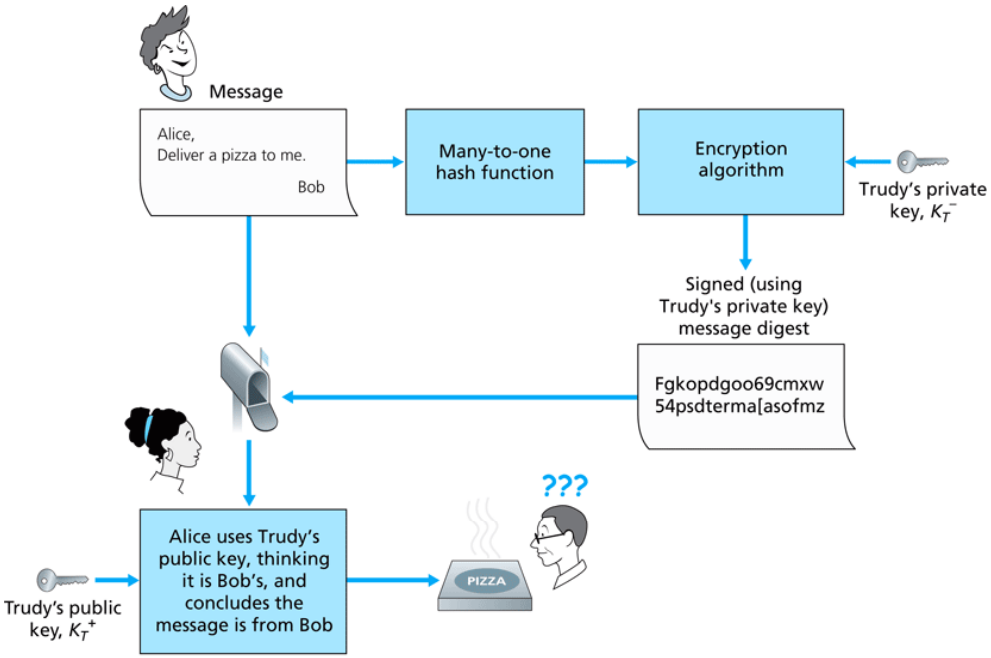
- Alice thus verifies that
  - Bob signed m
  - No one else signed m
  - Bob signed m and not m'
- Non-repudiation
  - Alice can take m, and signature $K_{B-}(m)$ to court and prove that Bob signed m

Message Integrity

Public-key Certification

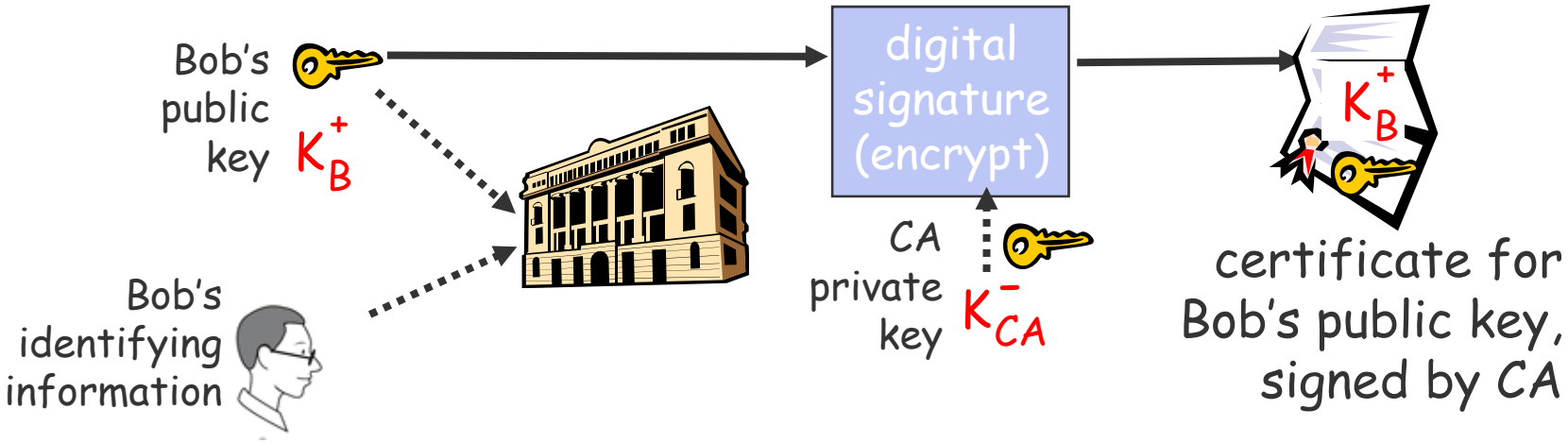
Public-key Certification

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



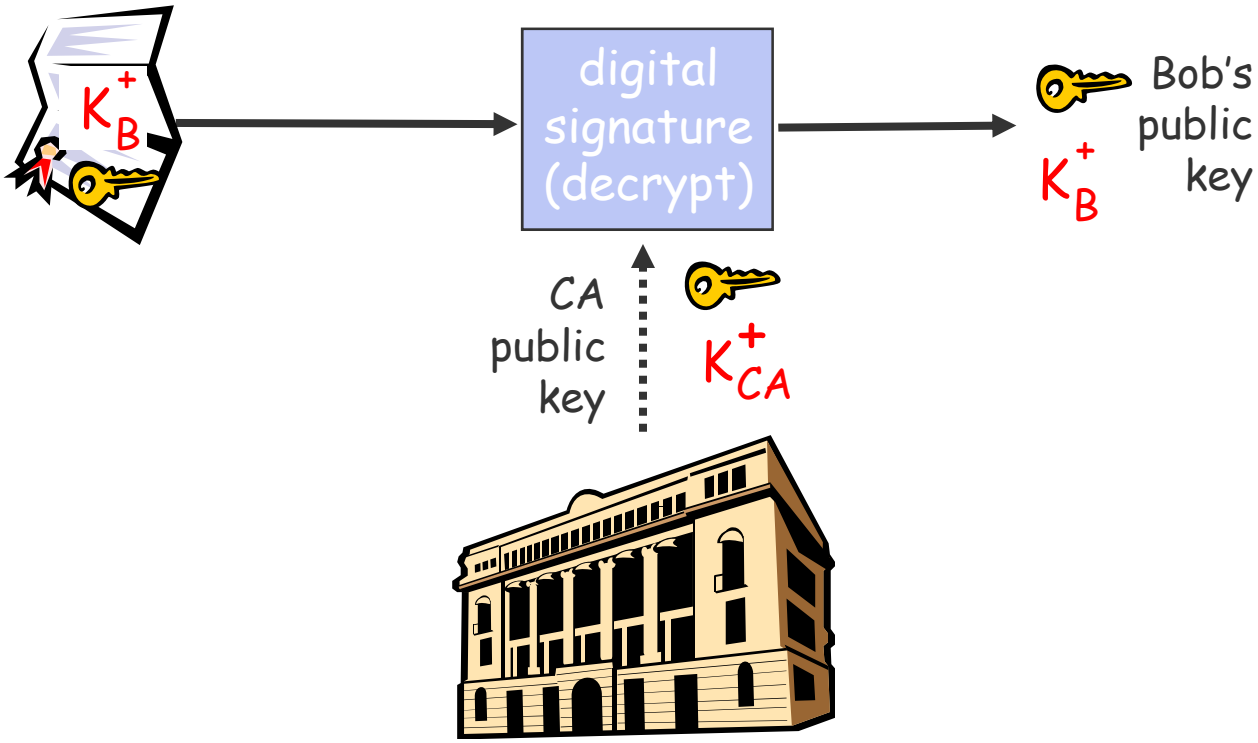
Certification Authorities

- 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



Certificates: Summary

- Primary standard X.509 (RFC 2459)
- Certificate contains:
  - Issuer name
  - Entity name, address, domain name, etc.
  - Entity's public key
  - Digital signature (signed with issuer's private key)
- Public-Key Infrastructure (PKI)
  - Certificates and certification authorities
  - Often considered "heavy"

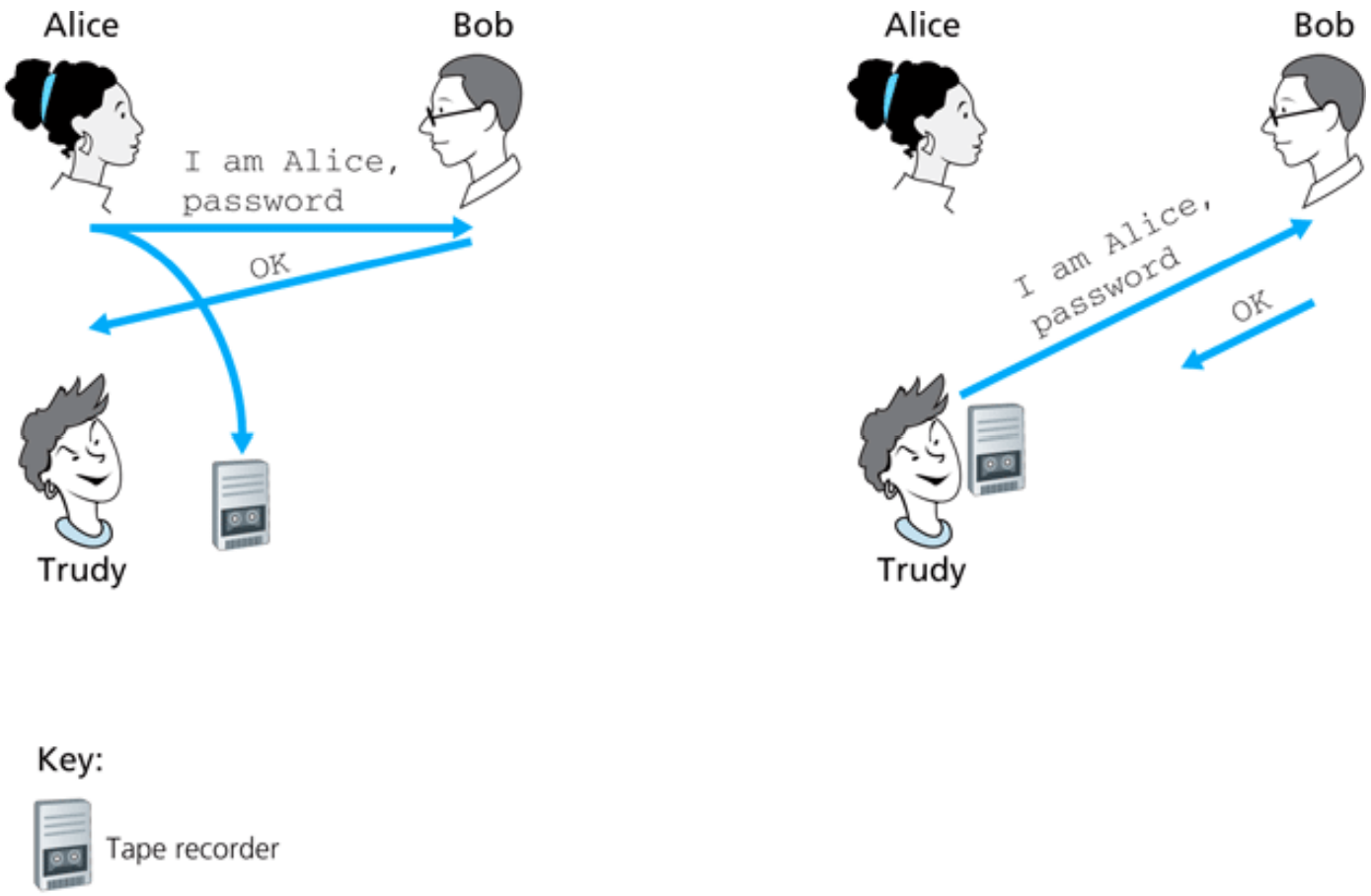
Message Integrity

End-point Authentication

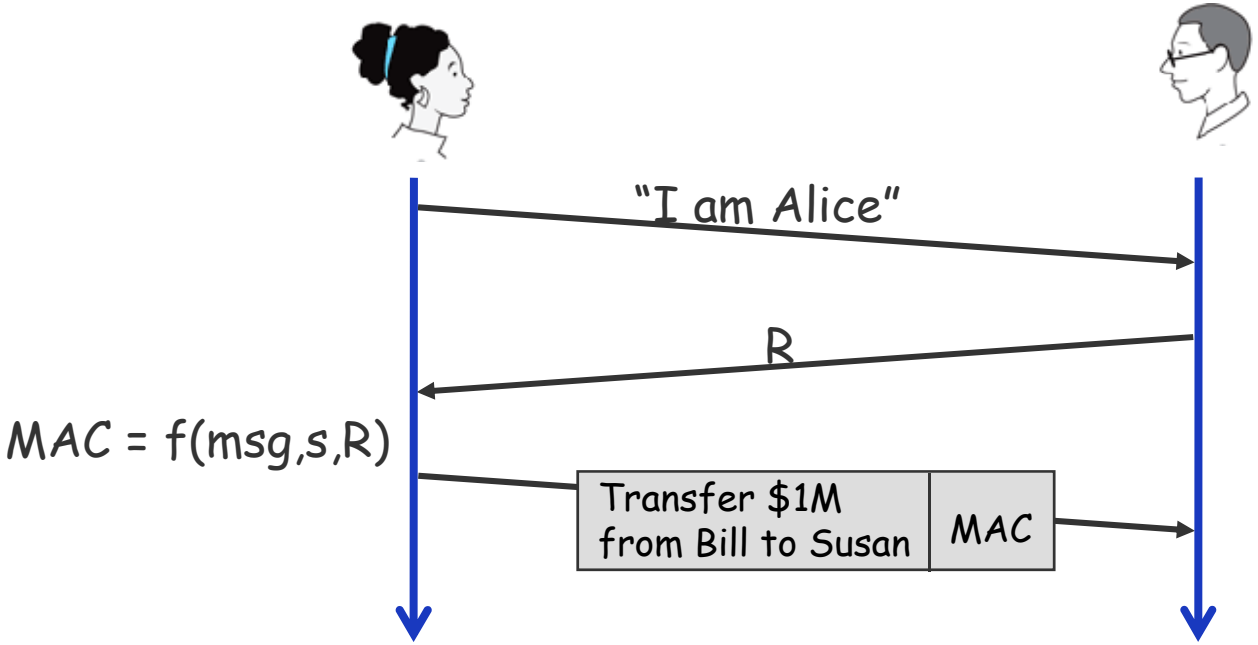
End-point authentication

- Want to be sure of the originator of the message – end-point authentication.
- Assuming Alice and Bob have a shared secret, will MAC provide end-point authentication.
  - We do know that Alice created the message.
  - But did she send it?

Playback/reflection attack



Defending against playback attack: Nonce



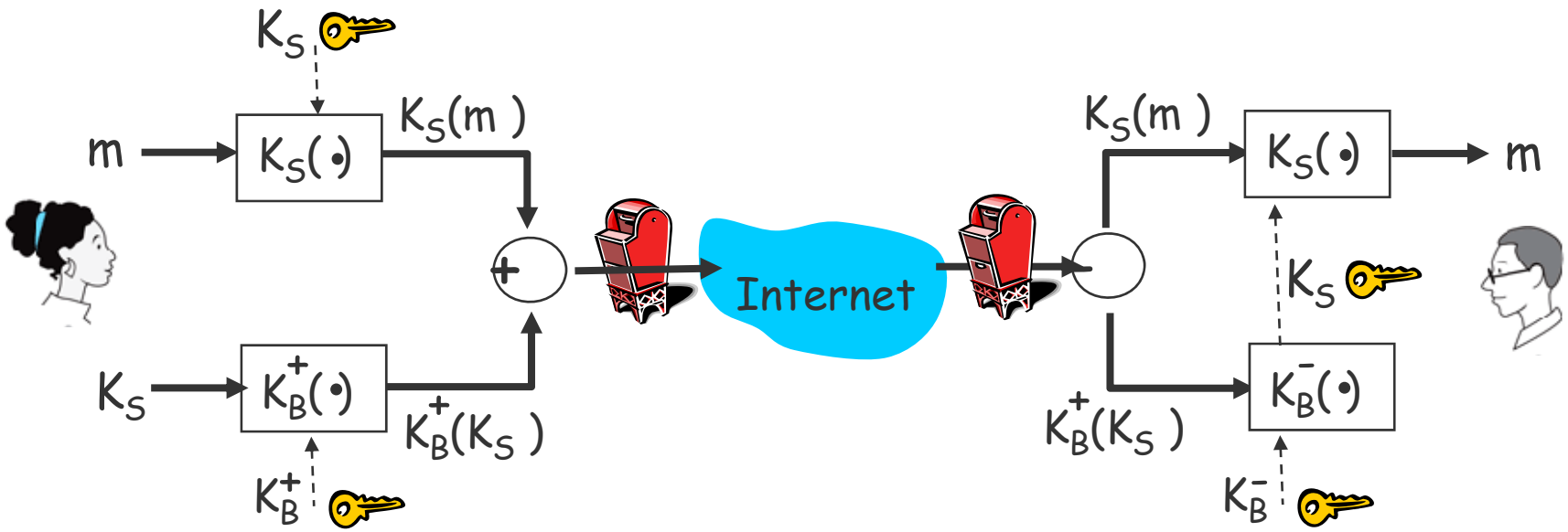
Securing e-mail

Secure e-mail

- What security features should a secure e-mail system provide?
  - Confidentiality
  - Sender authentication
  - Receiver authentication
  - Message integrity

Secure e-mail

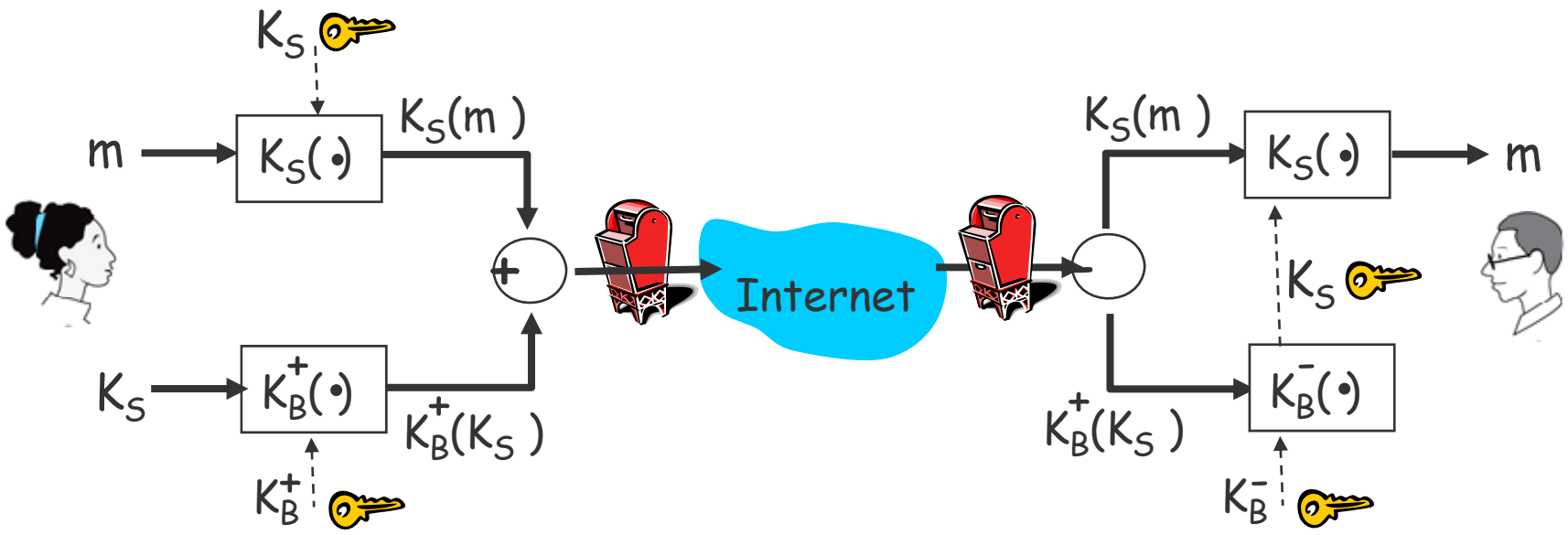
- Alice wants to send confidential e-mail m to Bob.



- 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

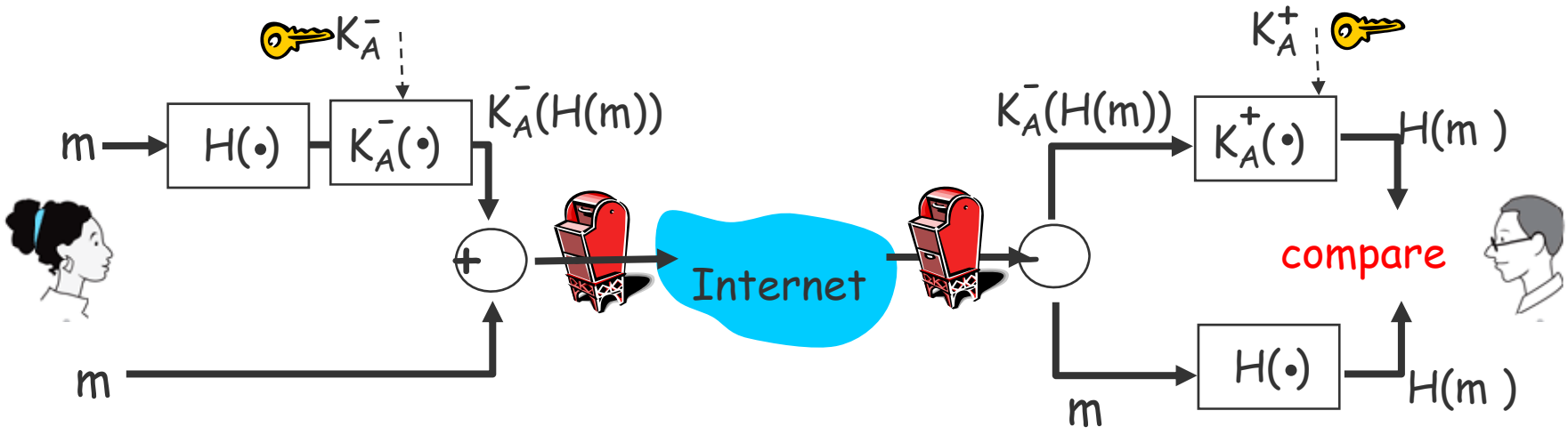
- Alice wants to send confidential e-mail m to Bob.



- 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

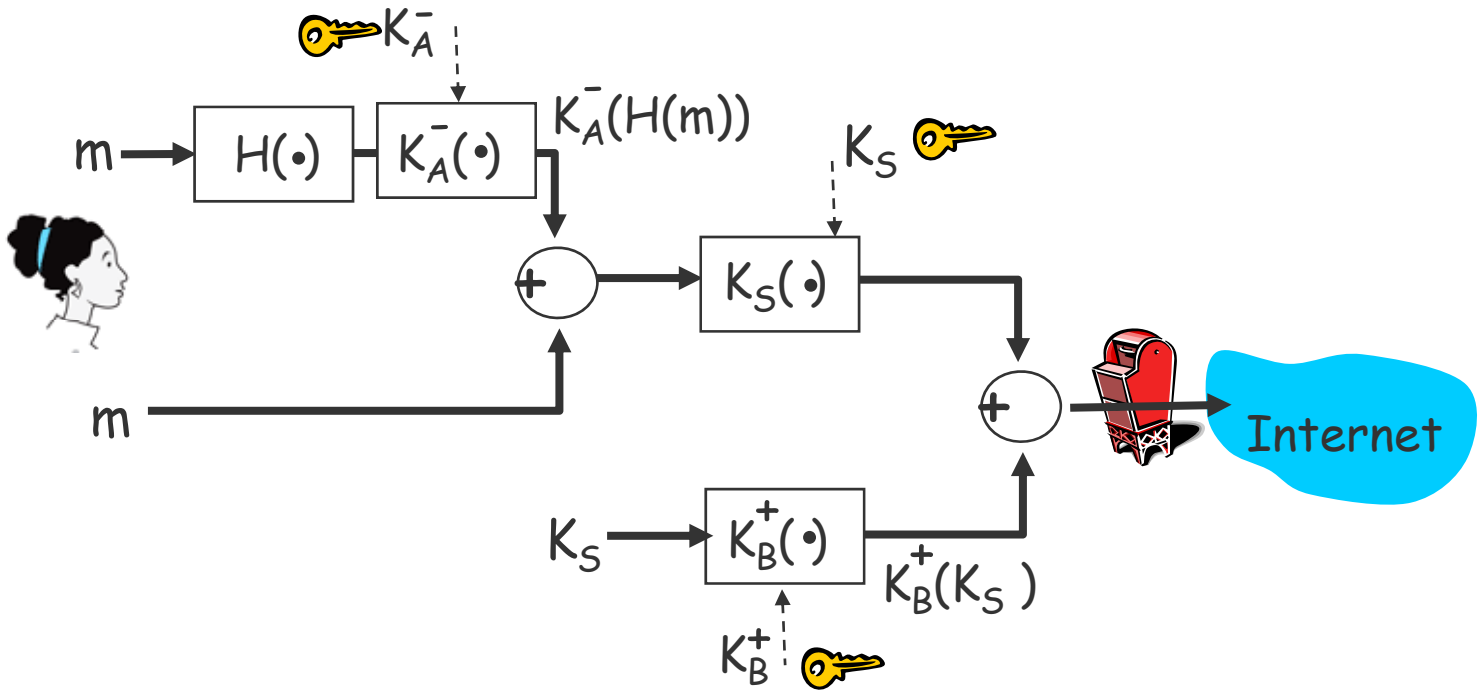
- Alice wants to provide sender authentication message integrity



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

Secure e-mail

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



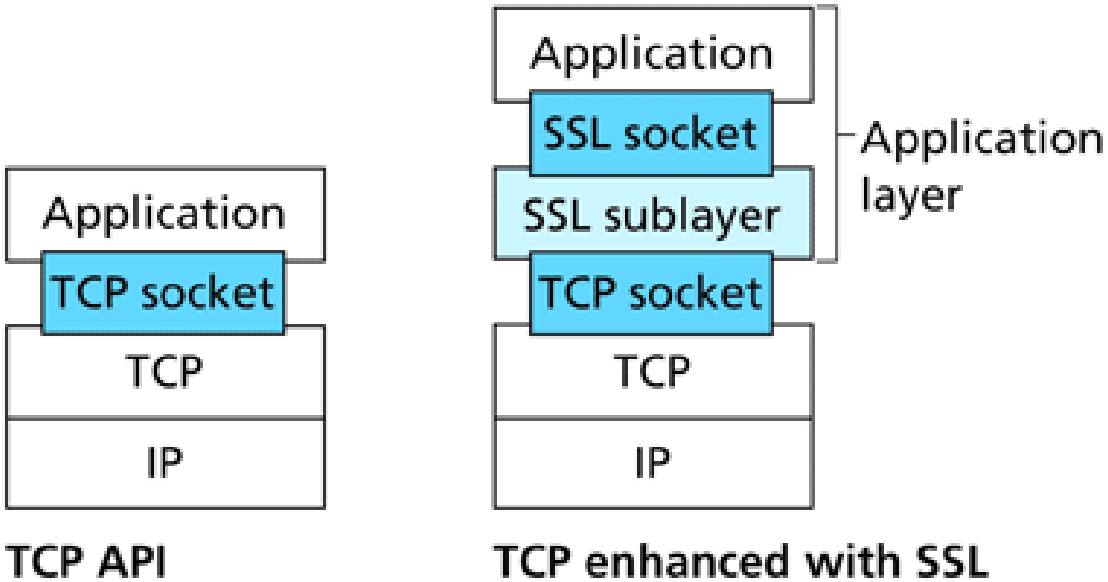
- Alice uses three keys: her private key, Bob's public key, newly created symmetric key

Securing TCP Connections: SSL

Secure Sockets Layer (SSL)

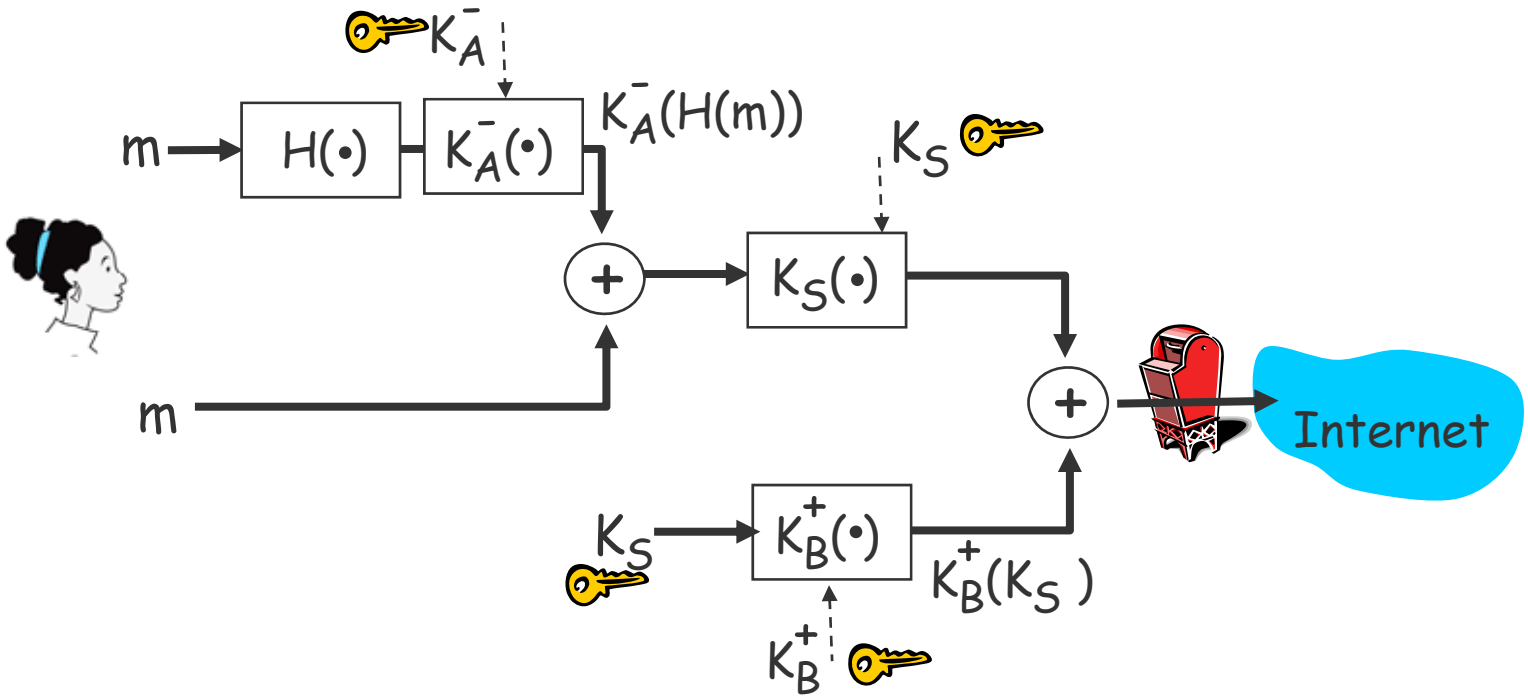
- Widely deployed security protocol
  - Supported by almost all browsers and web servers
  - https
  - Tens of billions $ spent per year over SSL
- Originally designed by Netscape in 1993
- Number of variations:
  - TLS: transport layer security, RFC 2246
- Provides
  - Confidentiality
  - Integrity
  - Authentication
- Original goals:
  - Had Web e-commerce transactions in mind
  - 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



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

Could do something like PGP



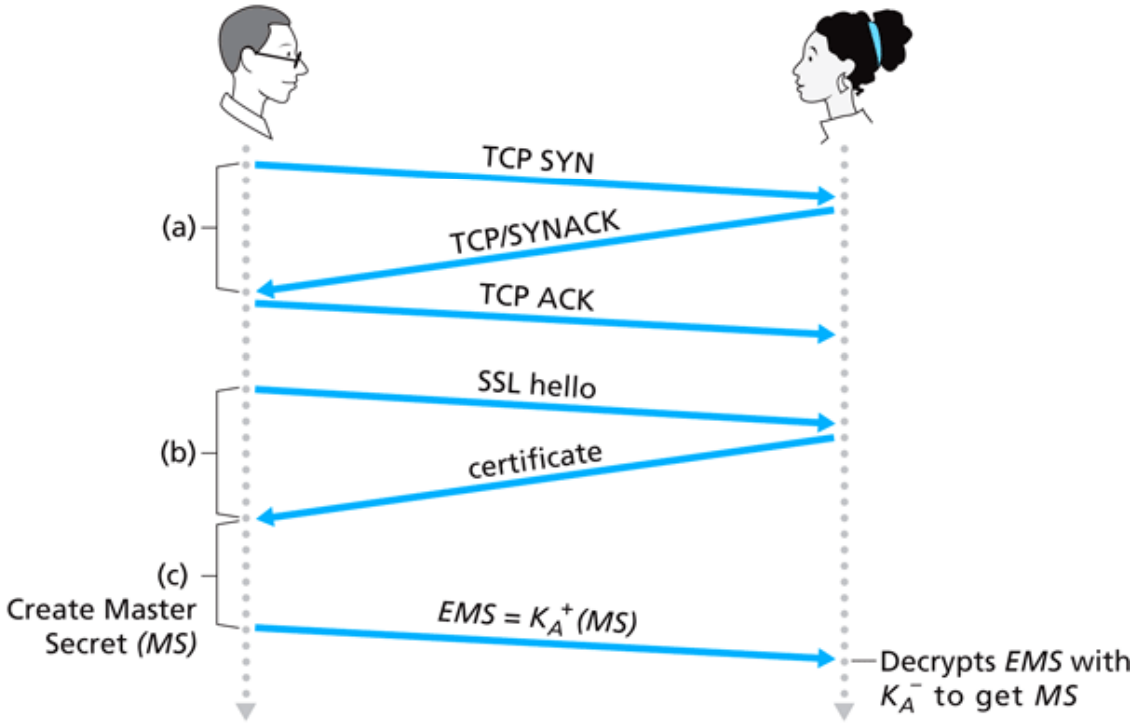
- But want to send byte streams & interactive data
- Want a set of secret keys for the entire connection
- Want certificate exchange part of protocol: handshake phase

Toy SSL: A simple secure channel

- SSL has three (four) phases
  1. Handshake
  2. Key Derivation
  3. Data Transfer
  - Connection termination

Toy SSL: A simple secure channel

1. Handshake
   a) Bob establishes a TCP connection with Alice
   b) Bob verifies that Alice is really Alice
   c) Bob sends Alice a master secret key



MS = Master Secret
EMS = Encrypted Master Secret

Toy SSL: A simple secure channel

2. 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.
  - For example, with instant messaging, how can we do integrity check over all bytes sent before displaying?
- Instead, break TCP 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

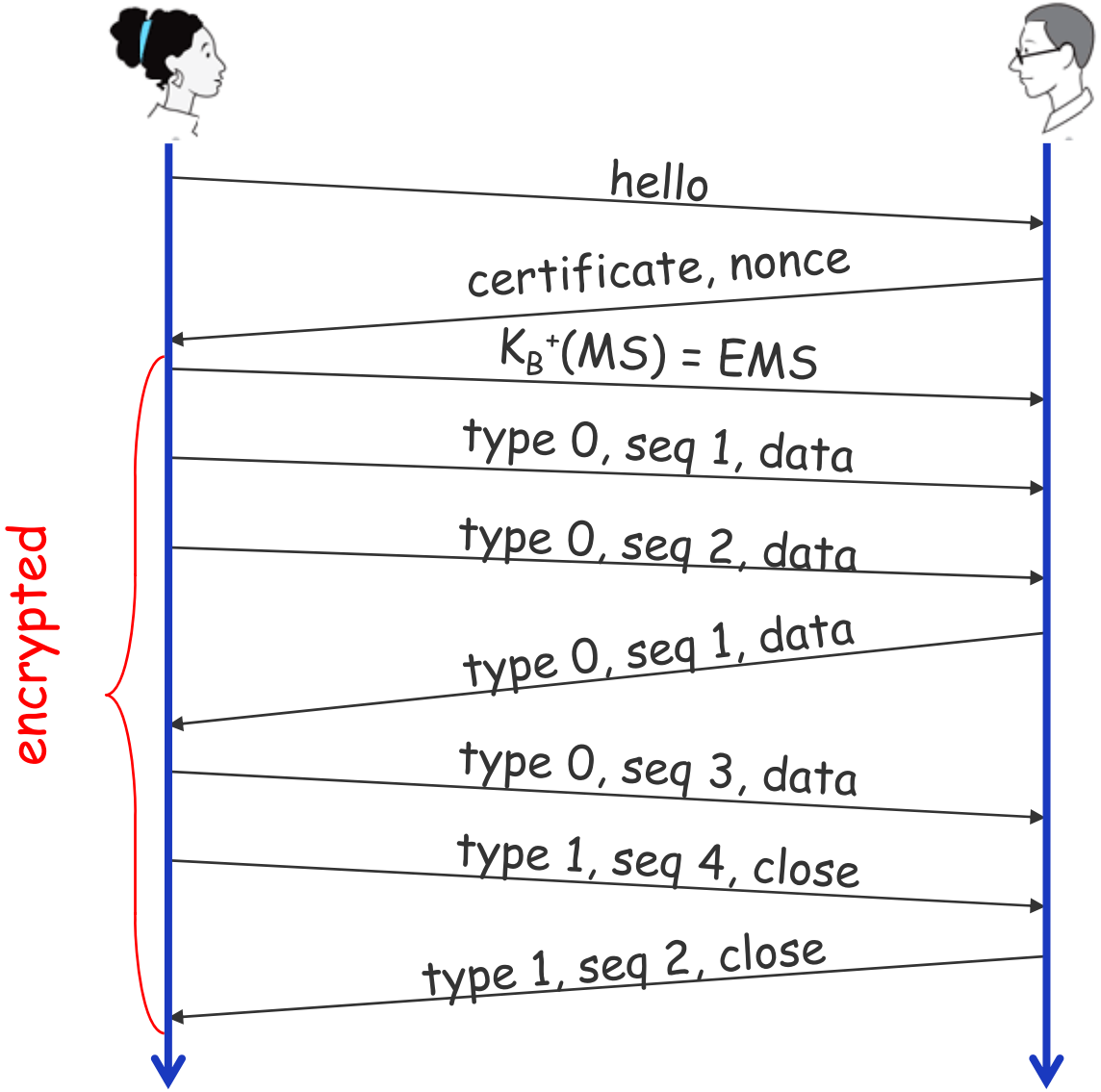
- Attacker can capture and replay record or re-order records
- Solution: put sequence number into MAC:
  - $MAC = MAC(M_x + sequence + data)$
  - Note: no sequence number field
- Attacker could still replay all of the records
  - Use random nonce

Toy: Control information

- 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
- $MAC = MAC(M_x + sequence + type + data)$

length	type	data	MAC

Toy SSL: Summary



Toy SSL isn't complete

- How long are the fields?
- What encryption protocols?
- No negotiation
  - Allow client and server to support different encryption algorithms
  - Allow client and server to choose together specific algorithm before data transfer

- Most common symmetric ciphers in SSL
  - DES: Data Encryption Standard: block
  - 3DES: Triple strength: block
  - RC2: Rivest Cipher 2: block
  - RC4: Rivest Cipher 4: stream

- Public key encryption
  - RSA

SSL Cipher Suite

- Cipher Suite
  - Public-key algorithm
  - Symmetric encryption algorithm
  - MAC algorithm
- SSL supports a variety of cipher suites
- Negotiation: client and server must agree on cipher suite
- Client offers choice; server picks one

Real SSL: Handshake

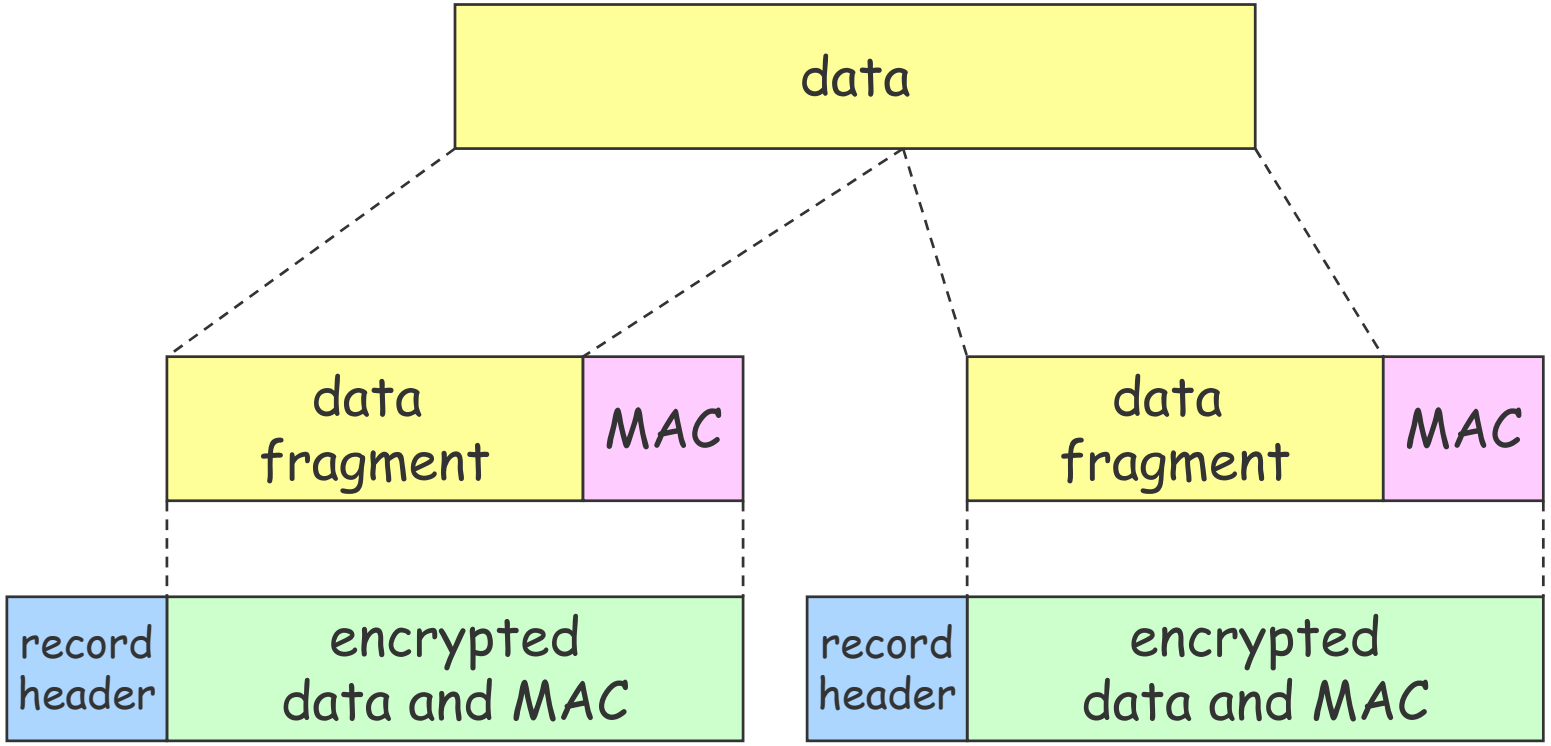
- Purpose
  - Server authentication
  - Negotiation: agree on crypto algorithms
  - Establish keys
  - Client authentication (optional)
- SSL Handshake
  1. Client sends list of algorithms it supports, along with client nonce
  2. Server chooses algorithms from list; sends back: choice + certificate + 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: Handshake

- Last 2 steps protect handshake from tampering
  - Client typically offers range of algorithms, some strong, some weak
  - Man-in-the middle could delete the stronger algorithms from list
  - Last 2 steps prevent this
    - Last two messages are encrypted

- Why the two random nonces?
  - Suppose Trudy sniffs all messages between Alice & Bob.
  - Next day, Trudy sets up TCP connection with Bob, sends the 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

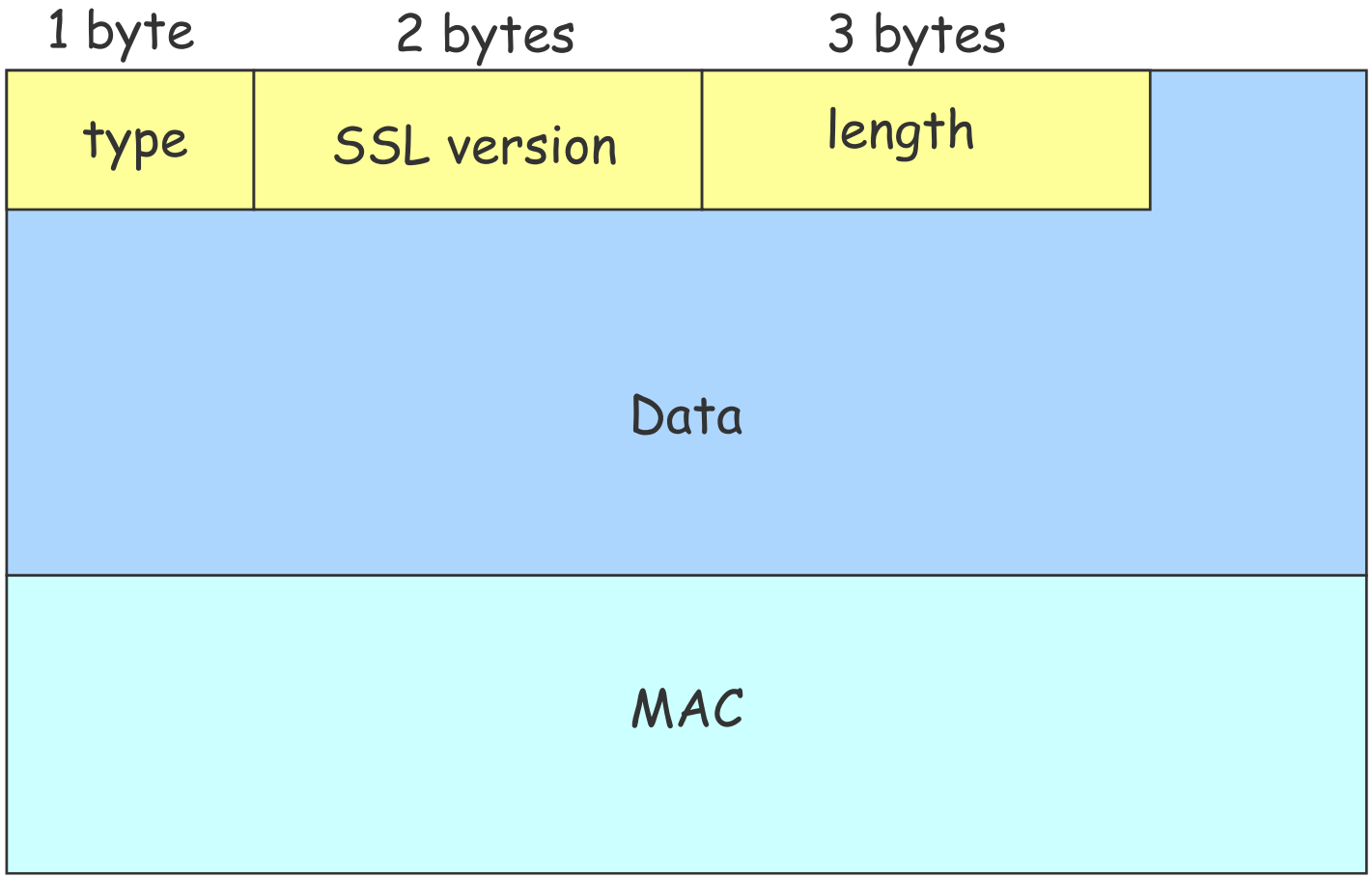


record header: content type; version; length

MAC: includes sequence number, MAC key M_x

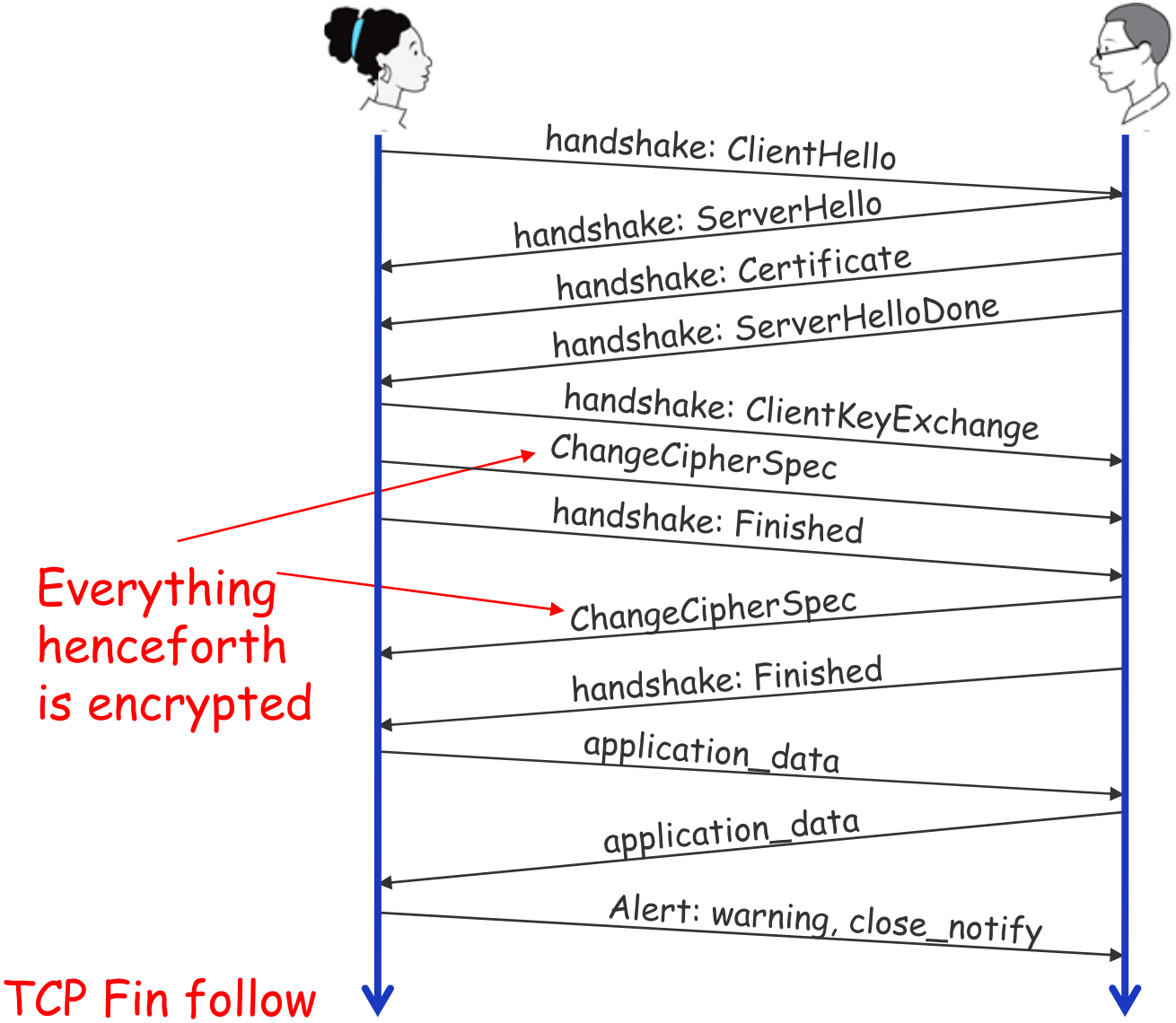
Fragment: each SSL fragment 2^{14} bytes (~16 Kbytes)

SSL Record Format



Data and MAC encrypted (symmetric algo)

Real Connection



Key derivation

- Client nonce, server nonce, and pre-master secret input into pseudo random-number generator.
  - Produces master secret
- Master secret and new nonces inputed 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)

Network Layer Security: IPsec

What is confidentiality at the network-layer?

- Between two network entities:

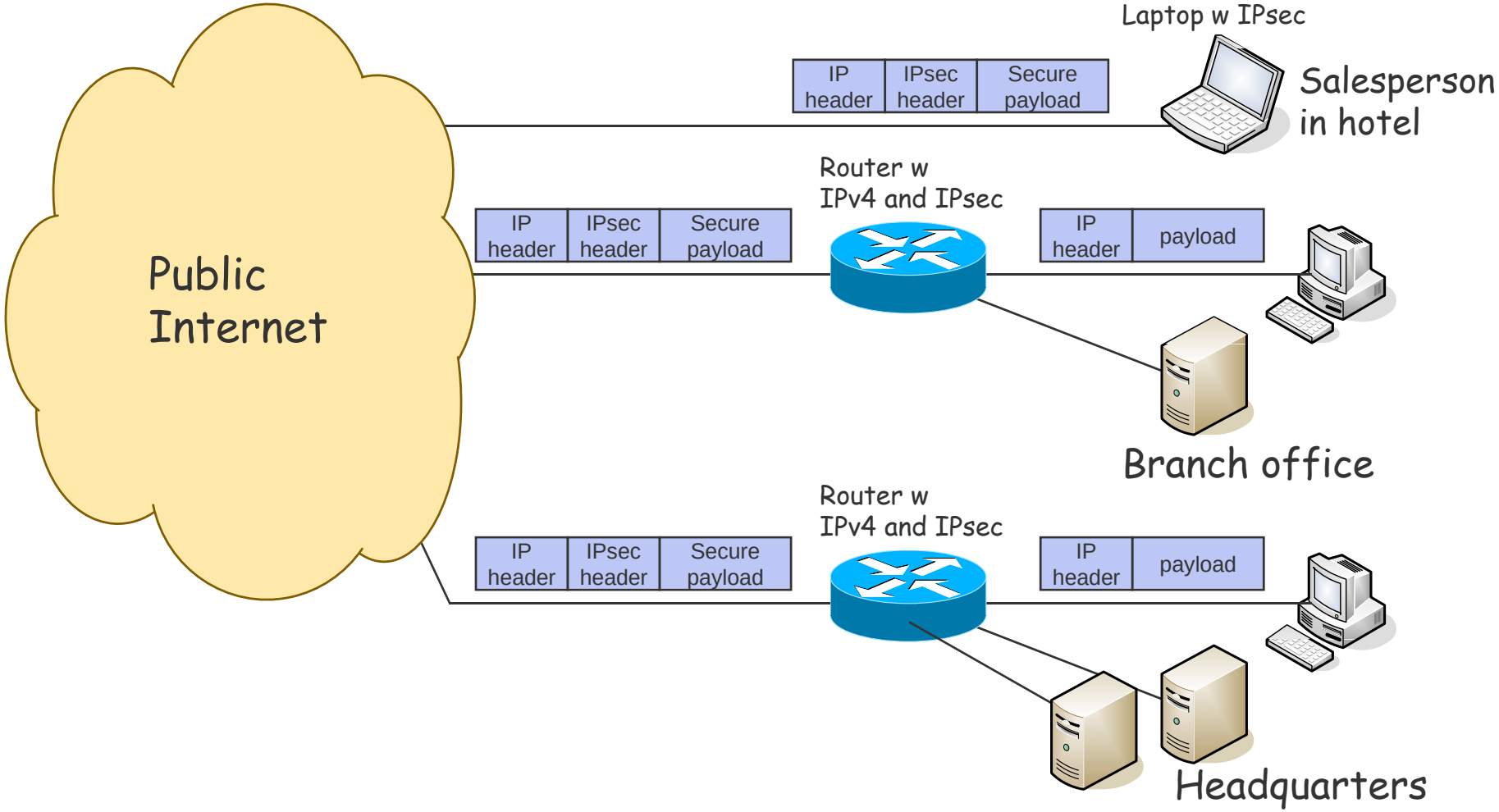
- Sending entity encrypts the payloads of datagrams.
  - Payload could be:
    - TCP segment
    - UDP datagram
    - ICMP message
    - OSPF message, etc.
- All data sent from one entity to the other would be hidden:
  - Web pages, e-mail, P2P file transfers, TCP SYN packets, and so on.

- That is, “blanket coverage”

Virtual Private Networks (VPNs)

- Institutions often want private networks for security.
  - Costly! Separate routers, links, DNS infrastructure.
- With a VPN, institution's inter-office traffic is sent over public Internet instead.
  - But inter-office traffic is encrypted before entering public Internet

Virtual Private Network (VPN)

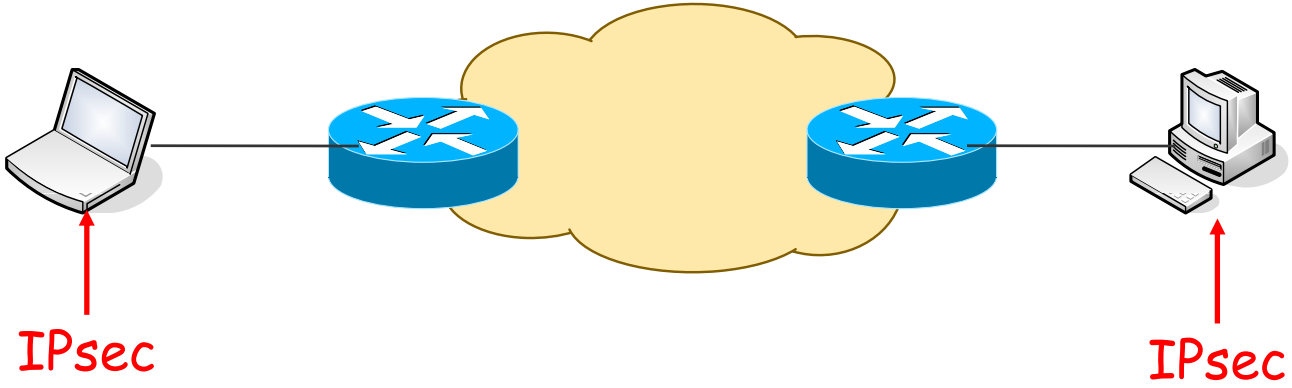


IPsec services

- IPsec provides
  - Data integrity
  - Origin authentication
  - Replay attack prevention
  - Confidentiality

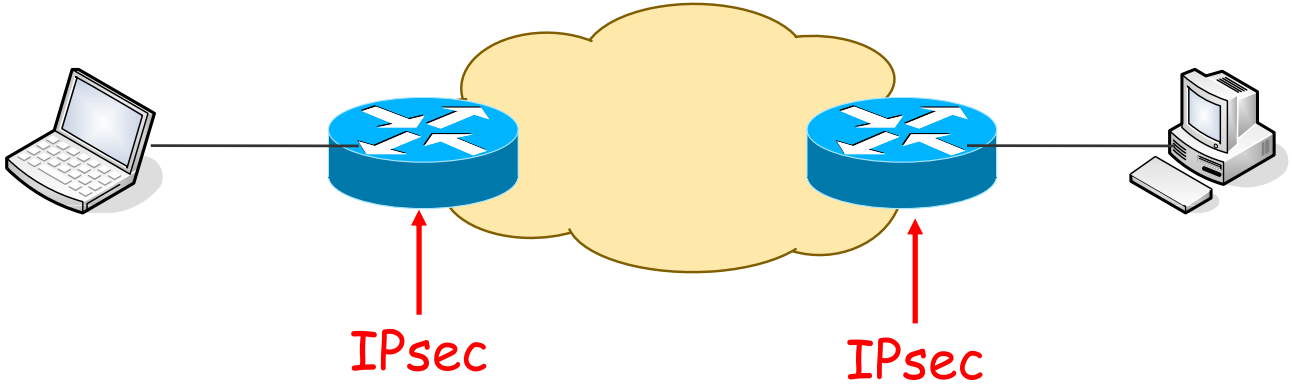
IPsec Transport Mode

- IPsec datagram emitted and received by end-system.
- Protects upper level protocols

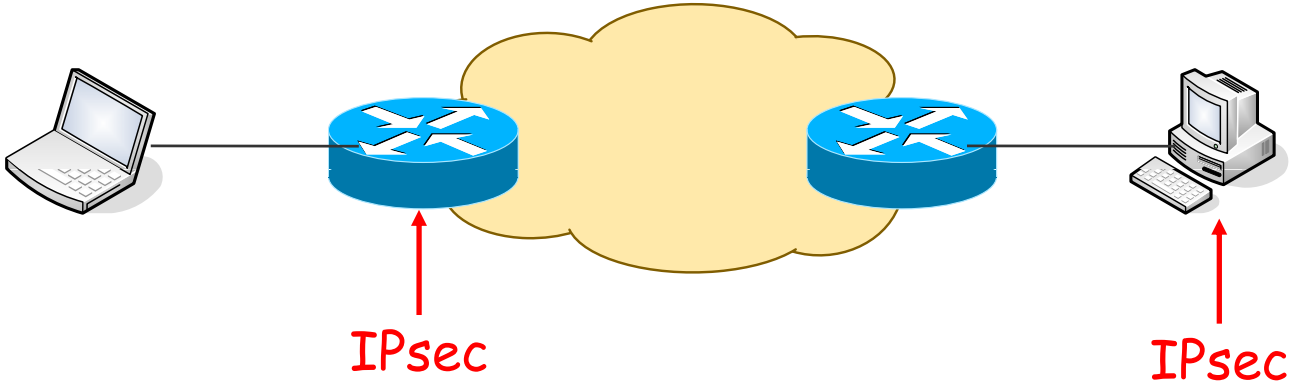


IPsec Tunneling mode

- End routers are IPsec aware. Hosts need not be.



- Also tunneling mode.



IPsec: Service Models

- Two protocols providing different service models:
  - Authentication Header (AH)
  - Encapsulation Security Payload (ESP)

- Authentication Header (AH) protocol
  - Provides source authentication
  - Data integrity

- Encapsulation Security Protocol (ESP)
  - Provides source authentication
  - Data integrity
  - Confidentiality

- ESP is more widely used than AH

IPsec: Service Models

- Four combinations are possible!

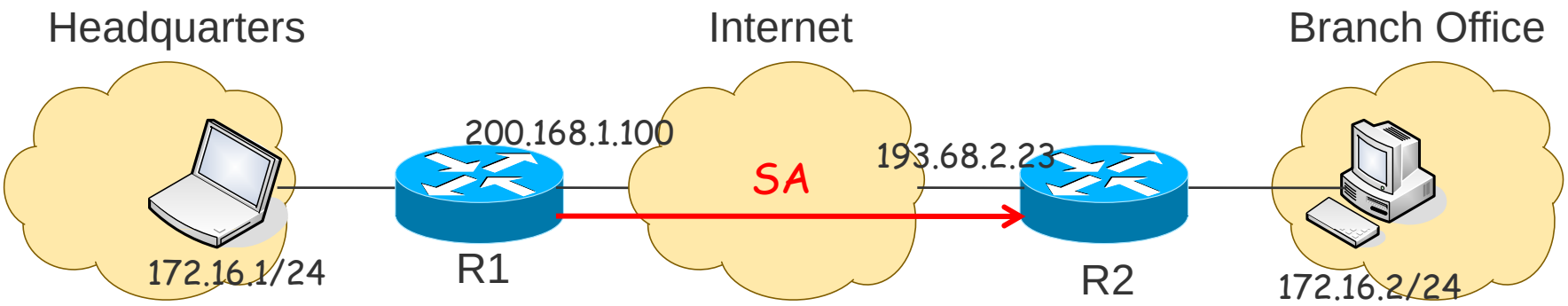
Host mode with AH	Host mode with ESP
Tunnel mode with AH	Tunnel mode with ESP

Most common and most important

Security associations (SAs)

- Before sending data, a virtual connection is established from sending entity to receiving entity.
- Called "security association (SA)"
  - SAs are simplex: for only one direction
- Both entities maintain state information about the SA
  - Recall that TCP endpoints also maintain state information.
  - IP is connectionless, but IPsec is connection-oriented!
- How many SAs in VPN w headquarters, branch office, and n traveling salesperson?

Example SA from R1 to R2



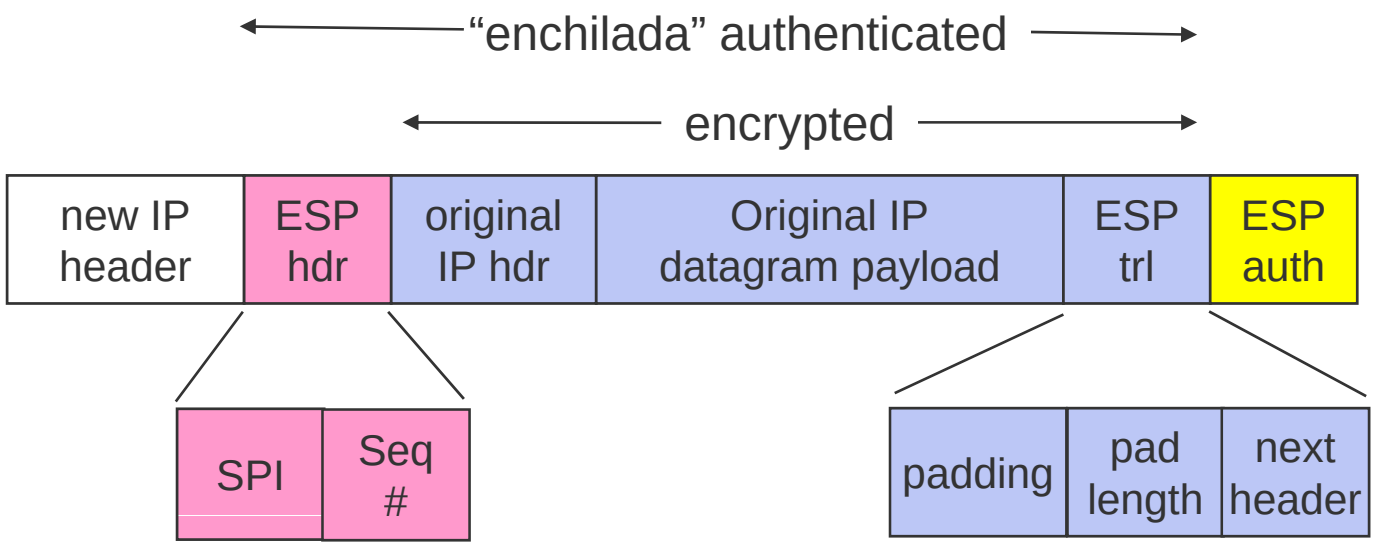
- R1 stores for SA
  - 32-bit identifier for SA: Security Parameter Index (SPI)
  - the origin interface of the SA (200.168.1.100)
  - destination interface of the SA (193.68.2.23)
  - type of encryption to be used (for example, 3DES with CBC)
  - encryption key
  - type of integrity check (for example, HMAC with with MD5)
  - authentication key

Security Association Database (SAD)

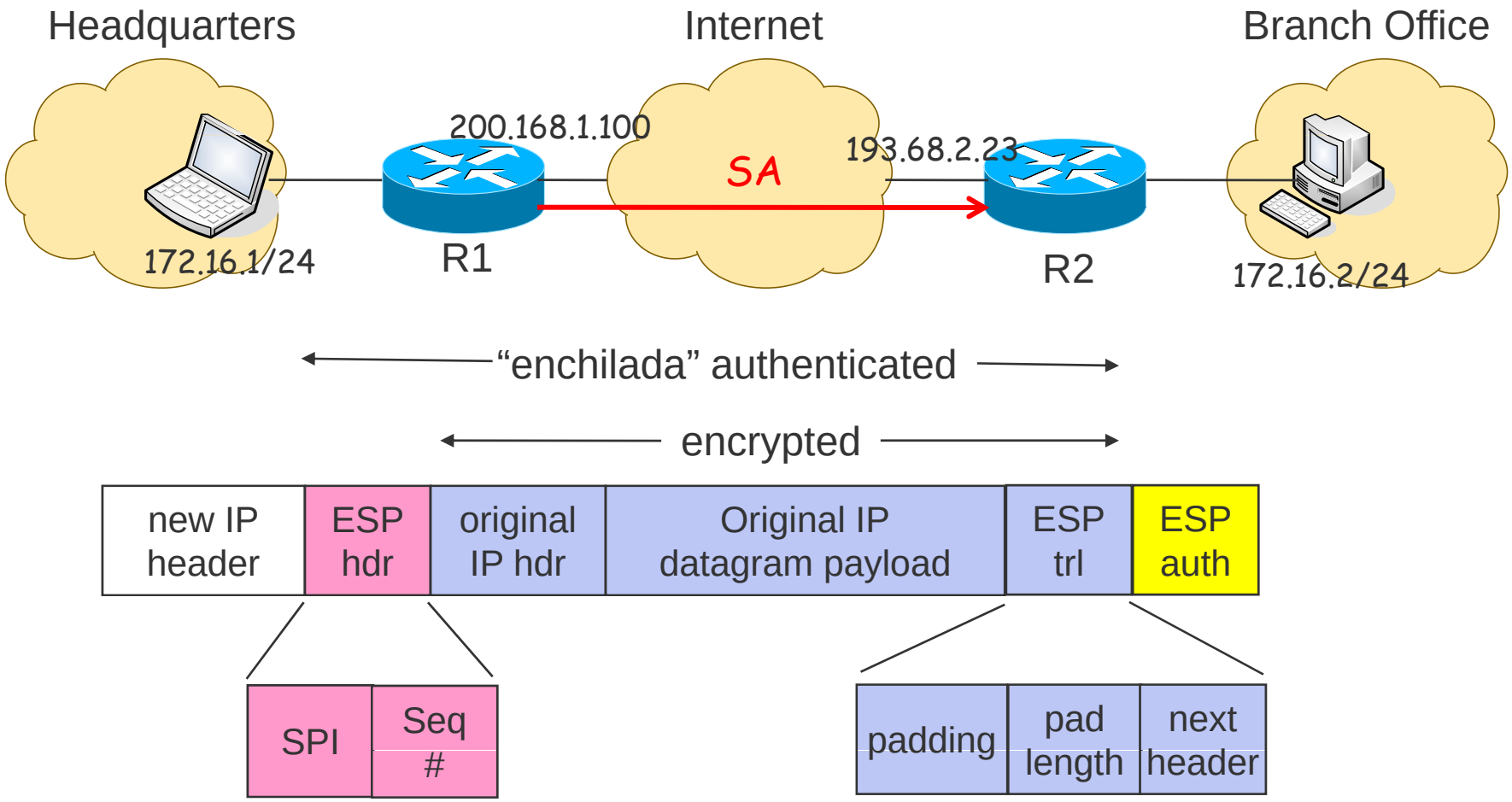
- Endpoint holds state of its SAs in a SAD, where it can locate them during processing.
- With n salespersons, 2 + 2n SAs in R1's SAD
- When sending IPsec datagram, R1 accesses SAD to determine how to process datagram.
- When IPsec datagram arrives to R2, R2 examines SPI in IPsec datagram, indexes SAD with SPI, and processes datagram accordingly.

IPsec datagram

- Focus for now on tunnel mode with ESP



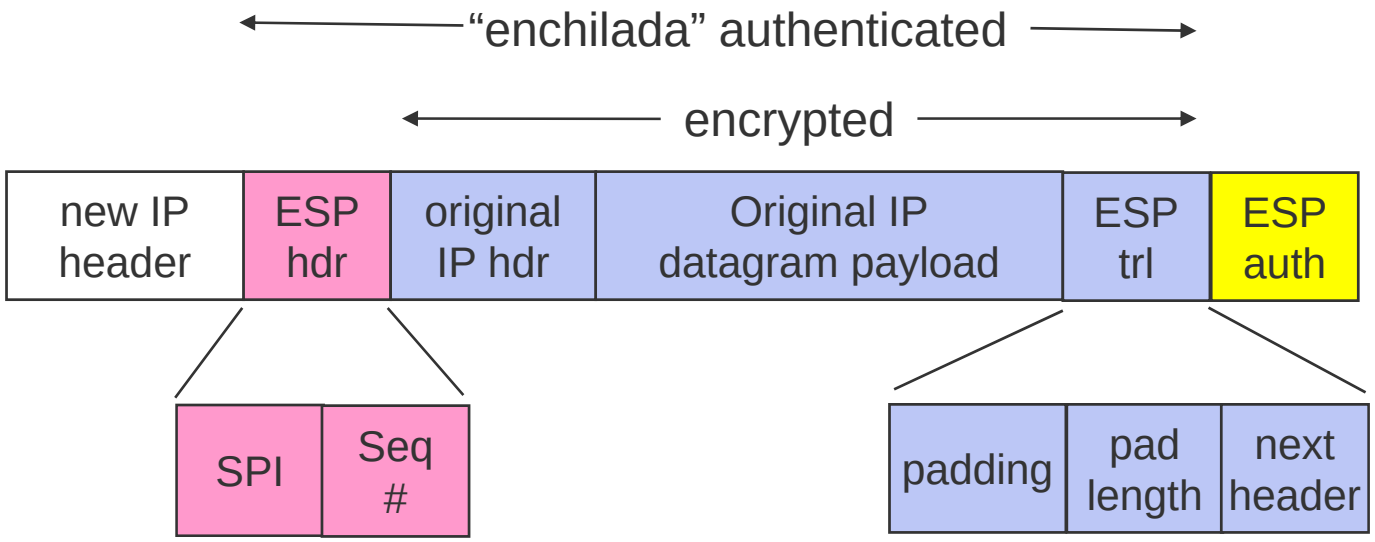
What happens?



R1 converts original datagram into IPsec datagram

- Appends to back of original datagram (which includes original header fields!) an "ESP trailer" field.
- Encrypts result using algorithm & key specified by SA.
- Appends to front of this encrypted quantity the "ESP header, creating "enchilada".
- Creates authentication MAC over the whole enchilada, using algorithm and key specified in SA;
- Appends MAC to back of enchilada, forming payload;
- Creates brand new IP header, with all the classic IPv4 header fields, which it appends before payload.

Inside the enchilada:



- ESP trailer: Padding for block ciphers
- ESP header:
  - SPI, so receiving entity knows what to do
  - Sequence number, to thwart replay attacks
- MAC in ESP auth field is created with shared secret key

IPsec sequence numbers

- For new SA, sender initializes seq. # to 0
- Each time datagram is sent on SA:
  - Sender increments seq # counter
  - Places value in seq # field
- Goal:
  - Prevent attacker from sniffing and replaying a packet
    - Receipt of duplicate, authenticated IP packets may disrupt service
- Method:
  - Destination checks for duplicates
  - But doesn't keep track of ALL received packets; instead uses a window

Security Policy Database (SPD)

- Policy: For a given datagram, sending entity needs to know if it should use IPsec.
- Needs also to know which SA to use
  - May use: source and destination IP address; protocol number.
- Info in SPD indicates “what” to do with arriving datagram;
- Info in the SAD indicates “how” to do it.

Summary: IPsec services

- Suppose Trudy sits somewhere between R1 and R2. She doesn't know the keys.
  - Will Trudy be able to see contents of original datagram? How about source, dest IP address, transport protocol, application port?
  - Flip bits without detection?
  - Masquerade as R1 using R1's IP address?
  - Replay a datagram?

Internet Key Exchange

- In previous examples, we manually established IPsec SAs in IPsec endpoints:
- Example SA
  - SPI: 12345
  - Source IP: 200.168.1.100
  - Dest IP: 193.68.2.23
  - Protocol: ESP
  - Encryption algorithm: 3DES-cbc
  - HMAC algorithm: MD5
  - Encryption key: 0x7aeaca…
  - HMAC key:0xc0291f…
- Such manually keying is impractical for large VPN with, say, hundreds of sales people.
- Instead use IPsec IKE (Internet Key Exchange)

IKE: PSK and PKI

- Authentication (proof who you are) with either
  - pre-shared secret (PSK) or
  - with PKI (pubic/private keys and certificates).
- With PSK, both sides start with secret:
  - then run IKE to authenticate each other and to generate IPsec SAs (one in each direction), including encryption and authentication keys
- With PKI, both sides start with public/private key pair and certificate.
  - run IKE to authenticate each other and obtain IPsec SAs (one in each direction).
  - Similar with handshake in SSL.

IKE Phases

- IKE has two phases
  - Phase 1: Establish bi-directional IKE SA
    - Note: IKE SA different from IPsec SA
    - Also called ISAKMP security association
  - Phase 2: ISAKMP is used to securely negotiate the IPsec pair of SAs
- Phase 1 has two modes: aggressive mode and main mode
  - Aggressive mode uses fewer messages
  - Main mode provides identity protection and is more flexible

Summary of IPsec

- IKE message exchange for algorithms, secret keys, SPI numbers
- Either the AH or the ESP protocol (or both)
- The AH protocol provides integrity and source authentication
- The ESP protocol (with AH) additionally provides encryption
- IPsec peers can be two end systems, two routers/firewalls, or a router/firewall and an end system

Summary

- Basic techniques……
  - cryptography (symmetric and public)
  - message integrity
  - end-point authentication
- …. used in many different security scenarios
  - secure email
  - secure transport (SSL)
  - IP sec
  - 802.11
- Operational Security: firewalls and IDS