Internet Technology

14. Network Security

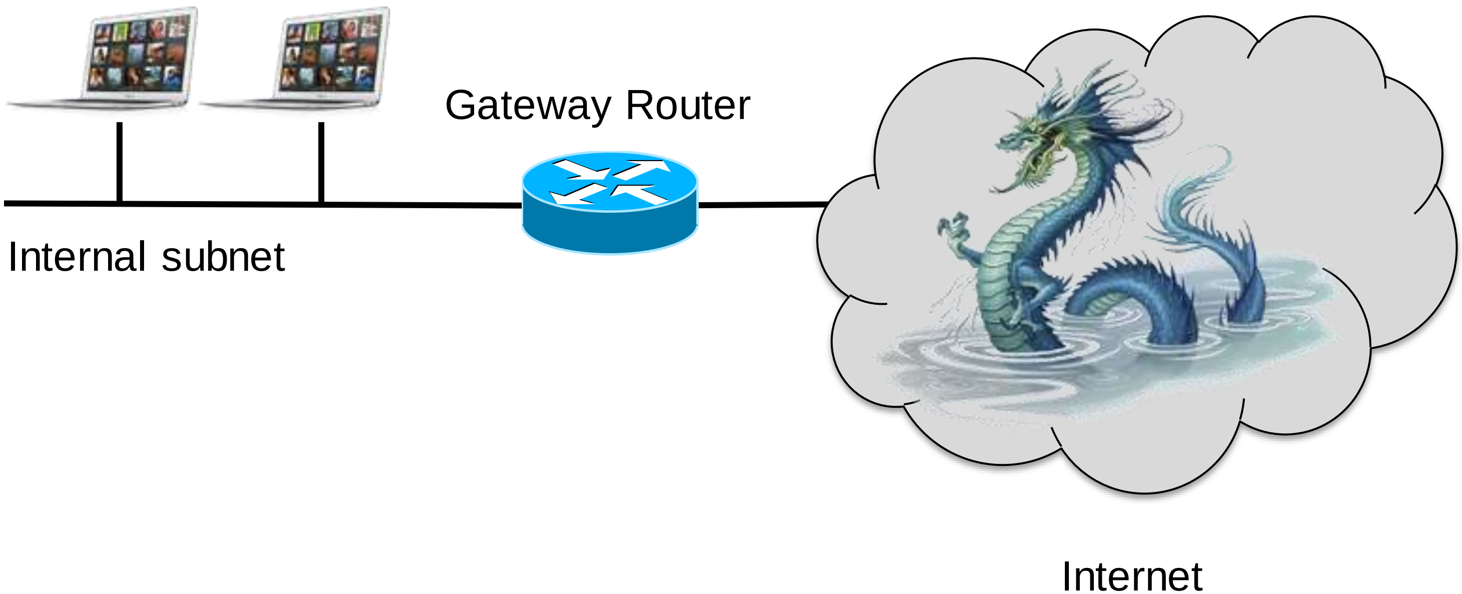
Paul Krzyzanowski

Rutgers University

Spring 2016

Network Security Goals

- Confidentiality: sensitive data & systems not accessible
- Integrity: data not modified during transmission
- Availability: systems should remain accessible



Firewall

- Separate your local network from the Internet
  - Protect the border between trusted internal networks and the untrusted Internet

- Approaches
  - Packet filters
  - Application proxies
  - Intrusion detection / intrusion protection systems

Screening router

- Border router (gateway router)
  - Router between the internal network(s) and external network(s)
  - Any traffic between internal & external networks passes through the border router

Instead of just routing the packet, decide *whether* to route it

- Screening router = Packet filter
  Allow or deny packets based on
  - Incoming interface, outgoing interface
  - Source IP address, destination IP address
  - Source TCP/UDP port, destination TCP/UDP port, ICMP command
  - Protocol (e.g., TCP, UDP, ICMP, IGMP, RSVP, etc.)

Filter chaining

- An IP packet entering a router is matched against a set of rules: access control list (ACL) or chain
- Each rule contains criteria and an action
  - Criteria: packet screening rule
  - Actions
    - *Accept* – and stop processing additional rules
    - *Drop* – discard the packet and stop processing additional rules
    - *Reject* – and send an error to the sender (ICMP Destination Unreachable)
  - Also
    - *Route* – rereoute packets
    - *Nat* – perform network address translation
    - *Log* – record the activity

Filter structure is vendor specific

Examples

- Windows
  - *Allow*, *Block*
  - Options such as
    - Discard all traffic except packets allowed by filters *(default deny)*
    - Pass through all traffic except packets prohibited by filters *(default allow)*
- OpenBSD
  - *Pass* (allow), *Block*
- Linux nftables
  - Chain types: *filter, route, nat*
  - Chain control
    - *Return* – stop traversing a chain
    - *Jump* – jump to another chain (*goto* = same but no return)

Network Ingress Filtering (incoming packets)

Basic firewalling principle

All traffic must flow through a firewall and be inspected

- Determine which services you want to expose to the Internet
  - e.g., HTTP & HTTPS: TCP ports 80 and 443
- Create a list of services and allow only those inbound ports and protocols to the machines hosting the services.
- Default Deny model - by default, "deny all”
  - Anything not specifically permitted is dropped
  - May want to log denies to identify who is attempting access

Network Ingress Filtering

- Disallow IP source address spoofing
  - Restrict forged traffic (RFC 2827)
- At the ISP
  - Filter upstream traffic - prohibit an attacker from sending traffic from forged IP addresses
  - Attacker must use a valid, reachable source address
- Disallow incoming/outgoing traffic from private, non-routable IP addresses
  - Helps with DDoS attacks such as SYN flooding from lots of invalid addresses

```
access-list 199 deny ip 192.168.0.0 0.0.255.255 any log
access-list 199 deny ip 224.0.0.0 0.0.0.255 any log
                    ....
access-list 199 permit ip any any
```

Network Egress Filtering (outbound)

- Usually we don't worry about outbound traffic.
  - *Communication from a higher security network (internal) to a lower security network (Internet) is usually fine*

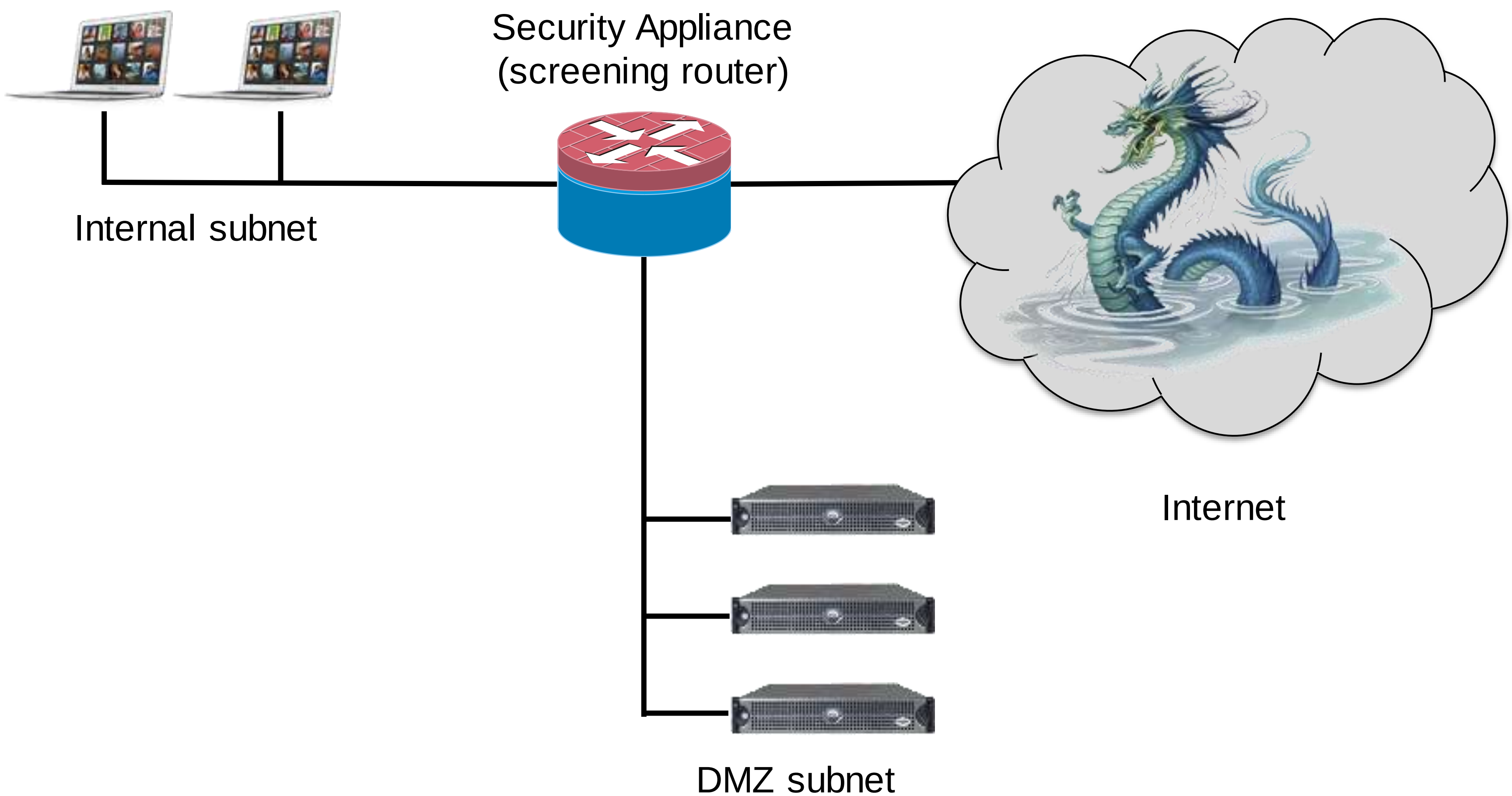
- Why might we want to restrict it?
  - Consider: if a web server is compromised & all outbound traffic is allowed, it can connect to an external server and download more malicious code
    ... or launch a DoS attack on the internal network

  - Also, log which servers are trying to access external addresses

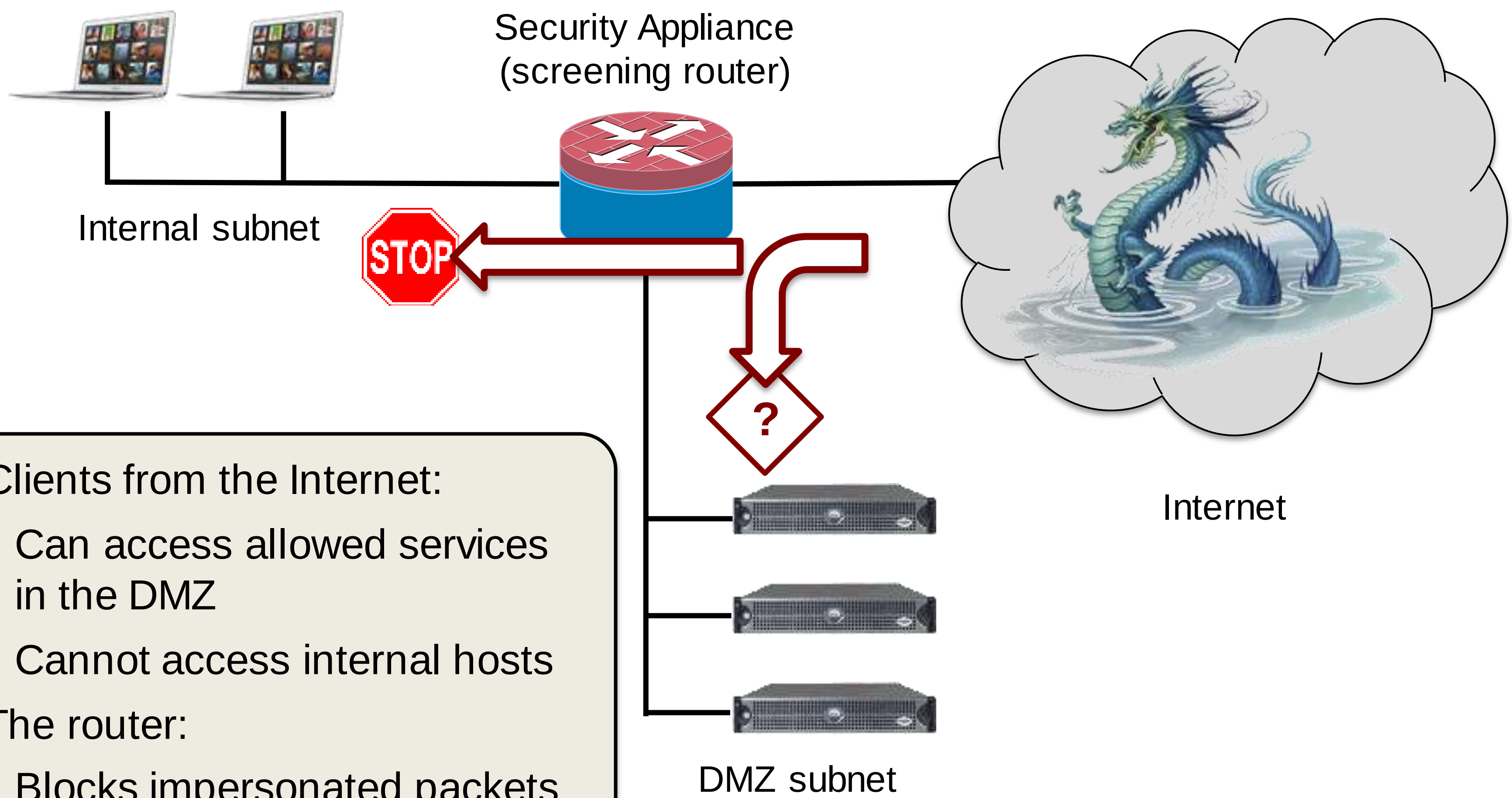
Stateful Filters

- Retain state information about a stream of related packets
- Examples
  - TCP connection tracking
    - Disallow TCP data packets unless a connection is set up
  - ICMP echo-reply
    - Allow ICMP echo-reply only if a corresponding echo request was sent.
  - Related traffic
    - Identify & allow traffic that is related to a connection
    - Example: related ports in FTP

Network Design: DMZ



Network Design: DMZ



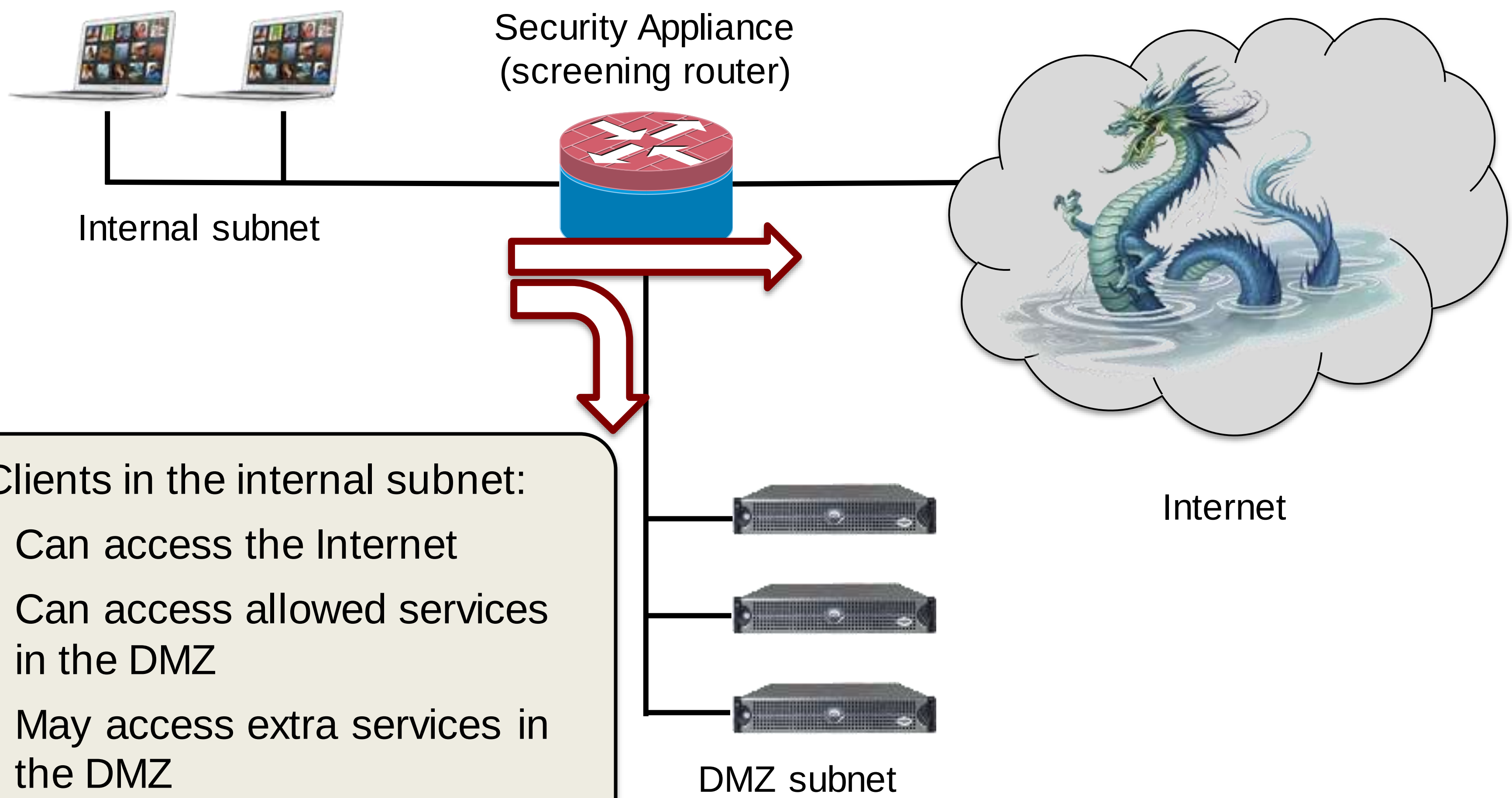
Clients from the Internet:

- Can access allowed services in the DMZ
- Cannot access internal hosts

The router:

- Blocks impersonated packets

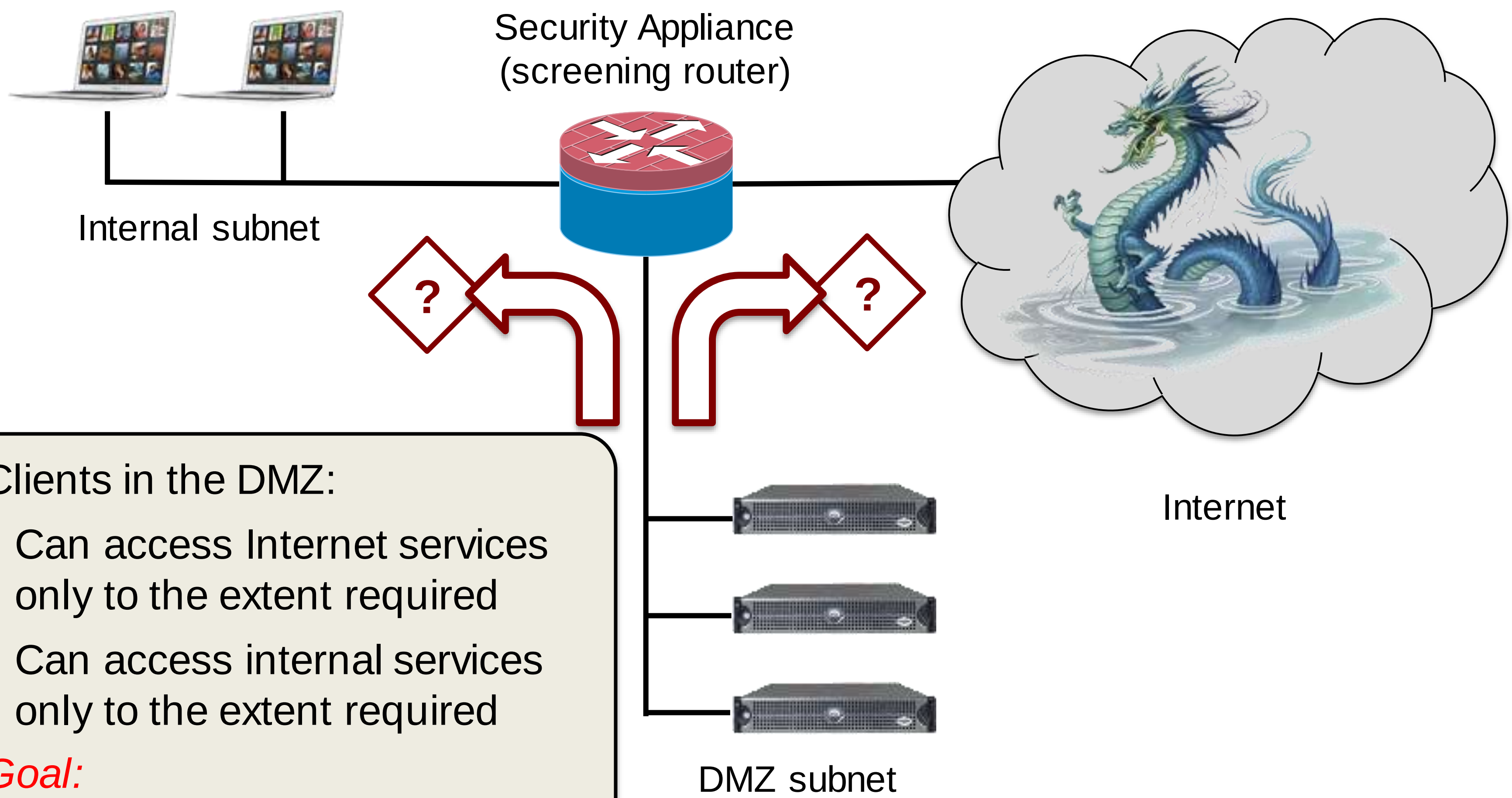
Network Design: DMZ



Clients in the internal subnet:

- Can access the Internet
- Can access allowed services in the DMZ
- May access extra services in the DMZ

Network Design: DMZ



Clients in the DMZ:

- Can access Internet services only to the extent required
- Can access internal services only to the extent required

Goal:
Limit possible damage if DMZ machines are compromised

Network Design: NAT

- NAT is an implicit firewall (sort of)
  - Arbitrary hosts and services on them (ports) cannot be accessed unless they are specifically mapped to a specific host/port by the administrator

Application-Layer Filtering

- Deep packet inspection
  - Look beyond layer 3 & 4 headers
  - Need to know something about application protocols & formats

- Example
  - URL filtering
    - Normal source/destination host/port filtering +
      URL pattern/keywords, rewrite/truncate rules, protocol content filters
    - Detect ActiveX and Java applets; configure specific applets as trusted
    - Filter others from the HTML code

IDS/IPS

- Intrusion Detection/Prevention Systems
  - Identify threats and attacks

- Types of IDS
  - Protocol-based
  - Signature-based
  - Anomaly-based

Protocol-Based IDS

- Reject packets that do not follow a prescribed protocol
- Permit return traffic as a function of incoming traffic
- Define traffic of interest (filter), filter on traffic-specific protocol/patterns
- Examples
  - DNS inspection: prevent spoofing DNS replies: make sure they match IDs of DNS requests
  - SMTP inspection: restrict SMTP command set (and command count, arguments, addresses)
  - FTP inspection: restrict FTP command set (and file sizes and file names)

Signature-based IDS

- Don't search for protocol violations but for exploits in programming

- Match patterns of known “bad” behavior
  - Viruses
  - Malformed URLs
  - Buffer overflow code

Anomaly-based IDS

- Search for statistical deviations from normal behavior
  - Measure baseline behavior first
  - Use heuristics, not bit patterns

- Examples:
  - Port scanning
  - Imbalance in protocol distribution
  - Imbalance in service access

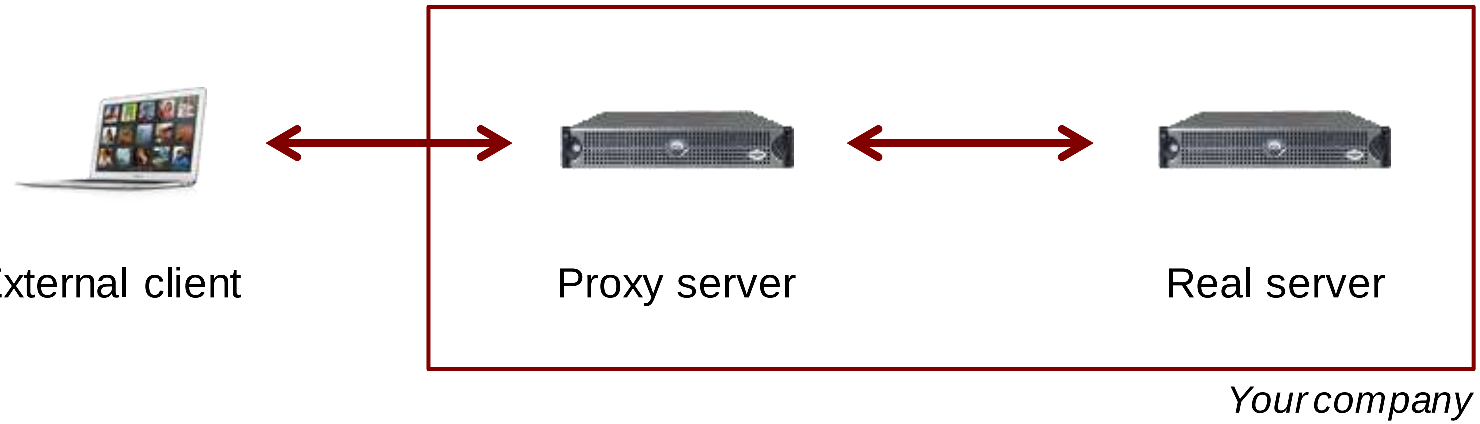
Other intrusion prevention approaches

- Port reassignment
  - Avoid well-known ports if only trusted users will access the services
  - E.g.,
    - Run *sshd* on port 2122 instead of 22
    - Run *httpd* on port 8180 instead of 80
  - The vast majority of attacks are casual

- fail2ban: host-based intrusion prevention framework
  - Scan log files for suspicious activity
  - Block IP addresses that are causing this activity for a period of time

Application proxies

- Proxy servers
  - Intermediaries between clients and servers
  - Stateful inspection and protocol validation
  - Incoming traffic *must* go through the application proxy



Cryptography: Basic Concepts

Terms

Plaintext (cleartext) message P

Encryption *E*(P)

Produces Ciphertext, C = *E*(P)

Decryption, P = *D*(C)

Cipher = cryptographic algorithm

Symmetric-key algorithm

- Same secret key, *K*, for encryption & decryption

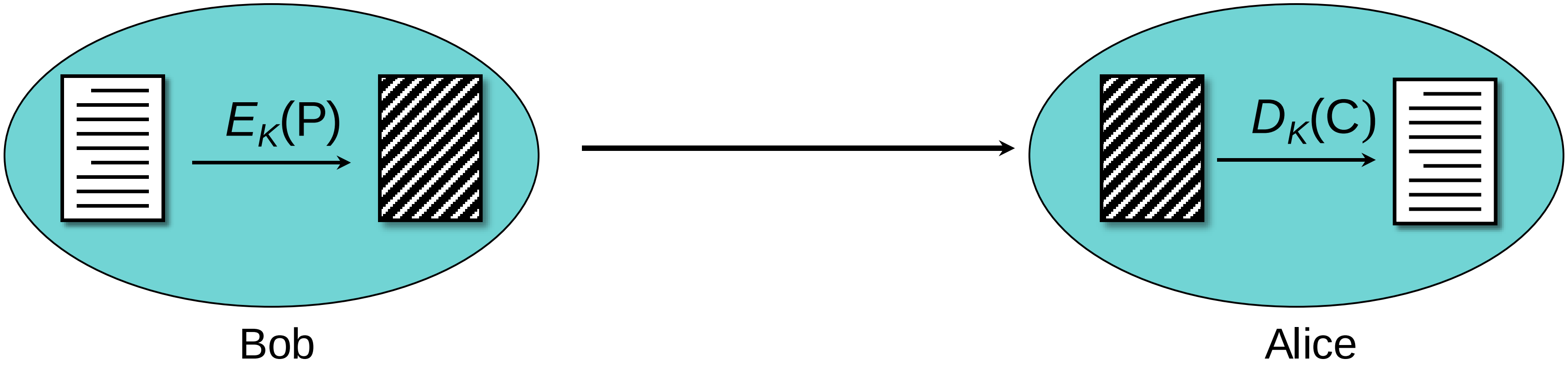
$$C = E_K(P) \qquad\qquad P = D_K(C)$$

- Examples: AES, 3DES, IDEA, RC5

- Key length
  - Determines number of possible keys
    - DES: 56-bit key: $2^{56} = 7.2 \times 10^{16}$ keys
    - AES-256: 256-bit key: $2^{256} = 1.1 \times 10^{77}$ keys
  - *Brute force attack*: try all keys
    - Each extra bit in the key doubles # possible keys

Communicating with symmetric cryptography

- Both parties must agree on a secret key, *K*
- Message is encrypted, sent, decrypted at other side



- Key distribution must be secret
  - otherwise messages can be decrypted
  - users can be impersonated

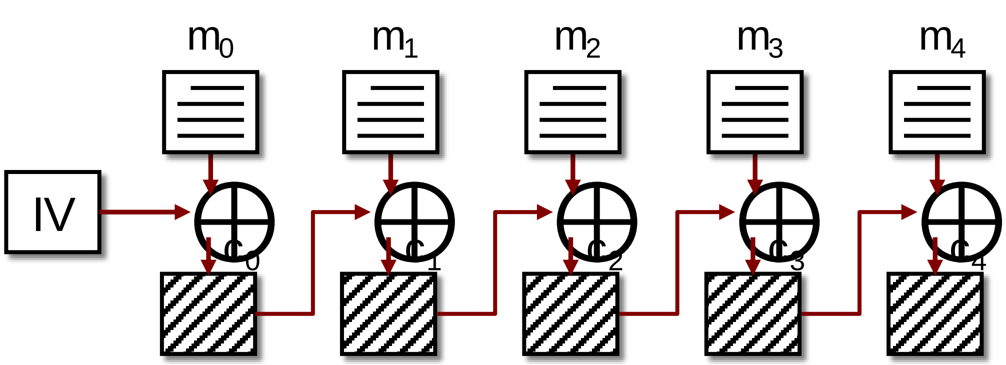
Cipher Block Chaining

- Streams of data are broken into *k*-byte blocks
  - Each block encrypted separately
- Problems
  1. Same plaintext results in identical encrypted blocks
  2. Attacker can add/delete/replace blocks

Cipher Block Chaining

- Streams of data are broken into *k*-byte blocks
  - Each block encrypted separately
- Problems
  1. Same plaintext results in identical encrypted blocks
  2. Attacker can add/delete/replace blocks

- Solution: **Cipher Block Chaining** (CBC)
  - Random initialization vector = bunch of *k* random bits
  - Exclusive-or with first plaintext block – then encrypt the block
  - Take exclusive-or of the result with the next plaintext block



$c_i = E_K(m) \oplus c_{i-1}$

Key distribution

Secure key distribution is the biggest problem with symmetric cryptography

Diffie-Hellman Key Exchange

Key distribution algorithm

- First algorithm to use public/private "keys"
- *Not* public key encryption
- Based on difficulty of computing discrete logarithms in a finite field compared with ease of calculating exponentiation

Allows us to negotiate a secret **common key** without fear of eavesdroppers

Diffie-Hellman Key Exchange

- All arithmetic performed in a field of integers modulo some large number
- Both parties agree on
  - a **large prime number *p***
  - and a number $\alpha < p$
- Each party generates a public/private key pair

Private key for user *i*: X_i

Public key for user *i*: $Y_i = \alpha^{X_i} \bmod p$

Diffie-Hellman exponential key exchange

- Alice has secret key X_A
- Alice has public key Y_A
- Alice computes

$$K = Y_B^{X_A} \bmod p$$

- Bob has secret key X_B
- Bob has public key Y_B

K = (Bob's public key)$^{\text{(Alice's private key)}}$ mod p

Diffie-Hellman exponential key exchange

- Alice has secret key X_A
- Alice has public key Y_A
- Alice computes

$$K = Y_B^{X_A} \bmod p$$

- Bob has secret key X_B
- Bob has public key Y_B
- Bob computes

$$K = Y_A^{X_B} \bmod p$$

K' = (Alice's public key) $^{\text{(Bob's private key)}}$ mod p

Diffie-Hellman exponential key exchange

- Alice has secret key X_A
- Alice has public key Y_A
- Alice computes

$$K = Y_B^{X_A} \bmod p$$

- expanding:

$$\begin{aligned} K &= Y_B^{X_A} \bmod p \\ &= (\alpha^{X_B} \bmod p)^{X_A} \bmod p \\ &= \alpha^{X_B X_A} \bmod p \end{aligned}$$

- Bob has secret key X_B
- Bob has public key Y_B
- Bob computes

$$K = Y_A^{X_B} \bmod p$$

- expanding:

$$\begin{aligned} K &= Y_B^{X_B} \bmod p \\ &= (\alpha^{X_A} \bmod p)^{X_B} \bmod p \\ &= \alpha^{X_A X_B} \bmod p \end{aligned}$$

K = K'

K is a *common key*, known *only* to Bob and Alice

Public-key algorithm

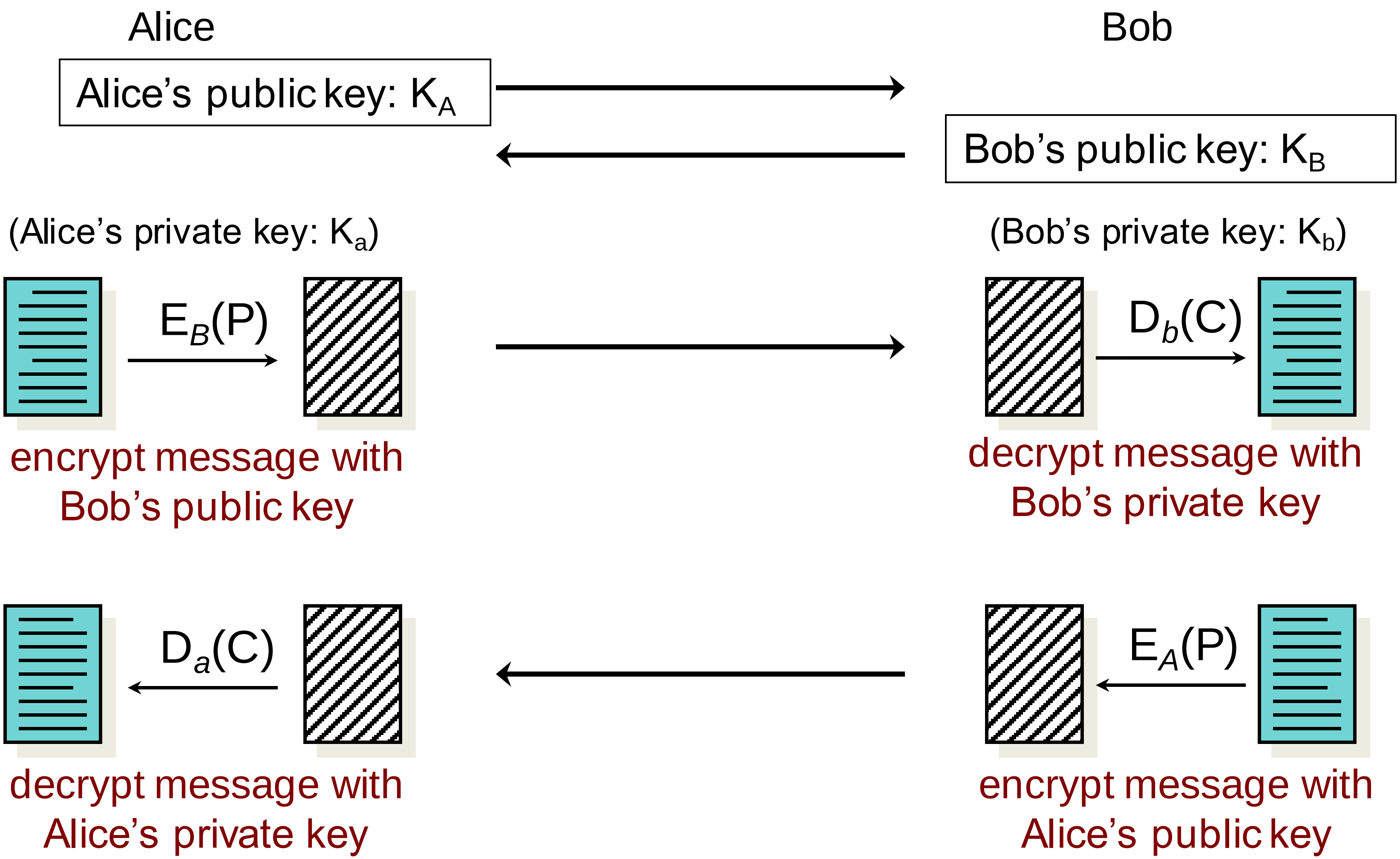
- Two related keys.

$$C = E_{K1}(P) \qquad P = D_{K2}(C)$$
$$C' = E_{K2}(P) \qquad P = D_{K1}(C')$$

K_1 is a public key
K_2 is a private key

- Examples:
  - RSA, Elliptic curve algorithms
    DSS (digital signature standard),
    Diffie-Hellman (key exchange only!)
- Key length
  - Unlike symmetric cryptography, not every number is a valid key
  - 3072-bit RSA = 256-bit elliptic curve = 128-bit symmetric cipher
  - 15360-bit RSA = 521-bit elliptic curve = 256-bit symmetric cipher

Communication with public key algorithms



Hybrid Cryptosystems

- Session key: randomly-generated key for one communication session

- Use a public key algorithm to send the session key

- Use a symmetric algorithm to encrypt data with the session key

Public key algorithms are almost never used to encrypt messages

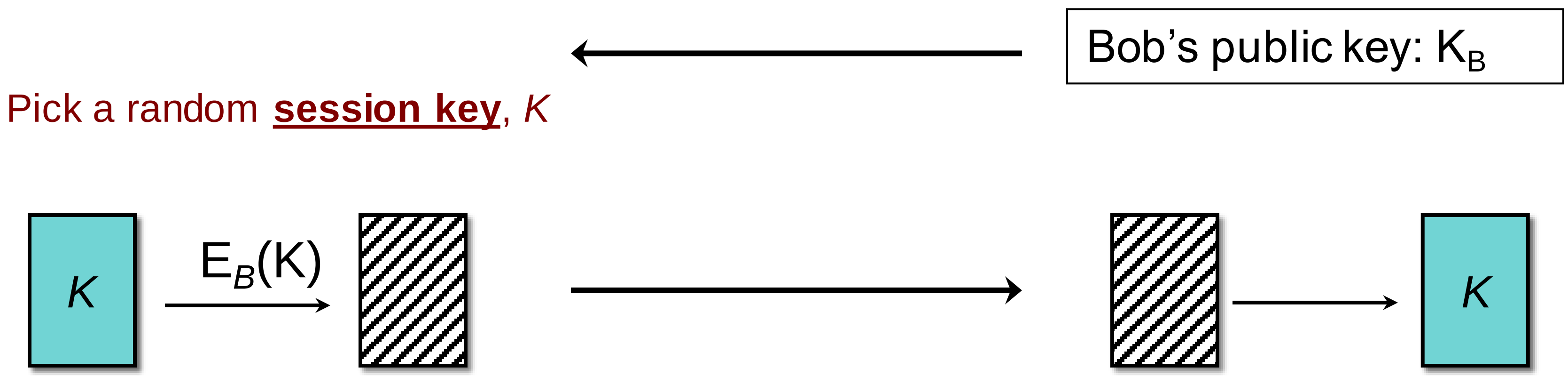
– MUCH slower; vulnerable to *chosen-plaintext attacks*

– RSA-2048 approximately 55x slower to encrypt and 2,000x slower to decrypt than AES-256

Communication with a hybrid cryptosystem

Alice

Bob



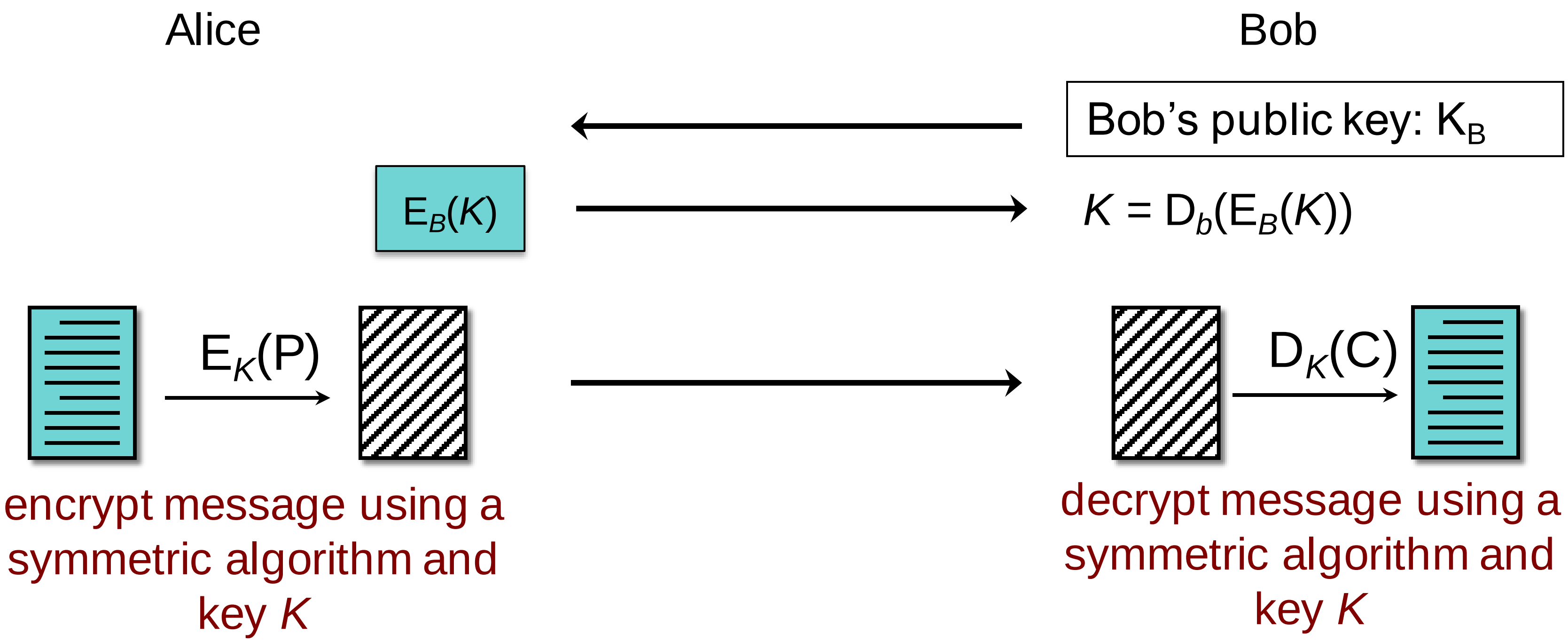
encrypt session key with
Bob’s public key

$K = D_b(E_B(K))$

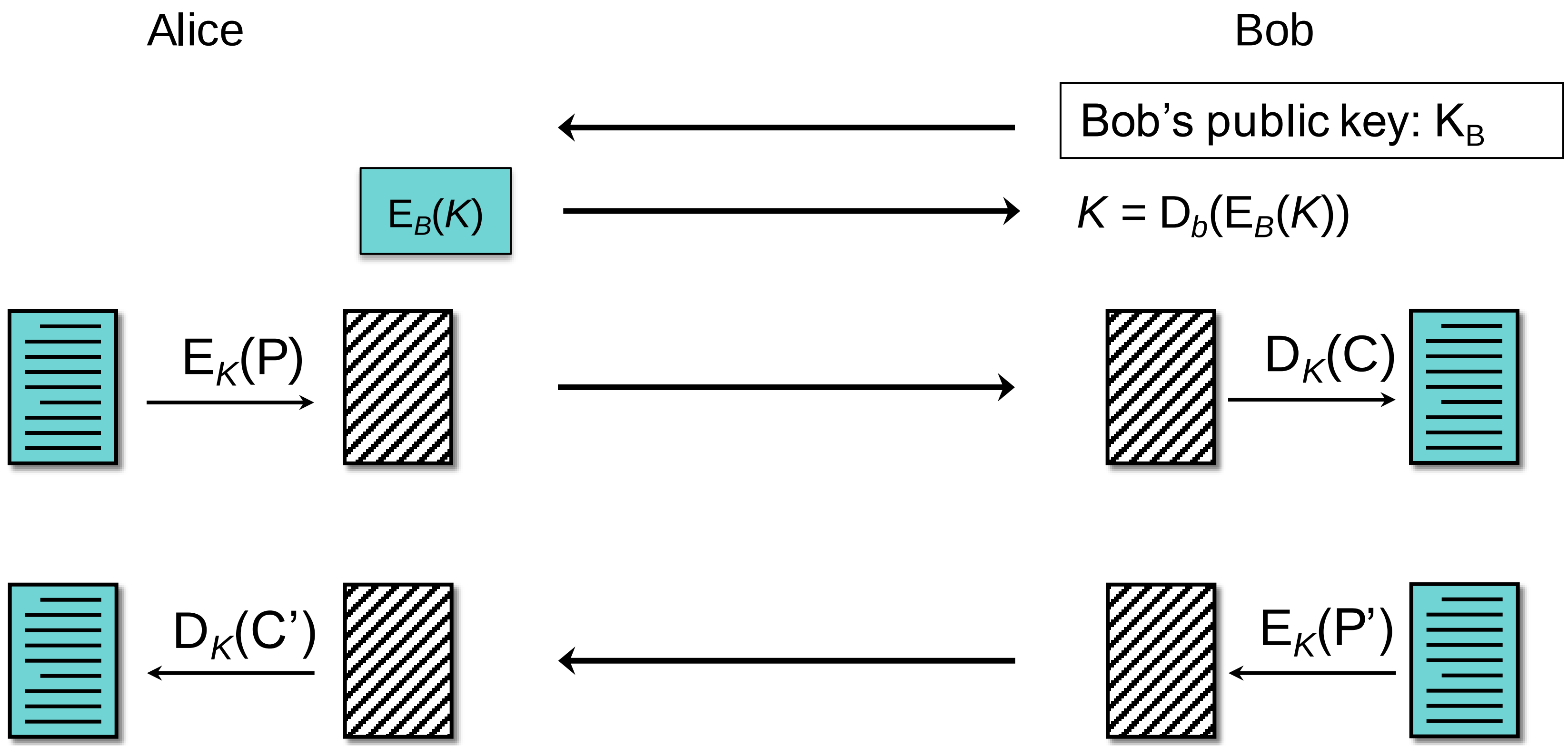
Bob decrypts *K* with
his private key

Now Bob knows the secret session key, K

Communication with a hybrid cryptosystem



Communication with a hybrid cryptosystem



decrypt message using a symmetric algorithm and key K

encrypt message using a symmetric algorithm and key K

Hash functions

- **Cryptographic hash function** (also known as a digest)
  - Input: arbitrary data
  - Output: fixed-length bit string
- Properties
  - **One-way function**
    - Given *H=hash(M)*, it should be difficult to compute *M*, given *H*
  - **Collision resistant**
    - Given *H=hash(M)*, it should be difficult to find *M’,* such that *H=hash(M’)*
    - For a hash of length L, a perfect hash would take $2^{(L/2)}$ attempts
  - **Efficient**
    - Computing a hash function should be computationally efficient
- Common hash functions: SHA-2, SHA-3 (256 & 512 bit), MD5

Message Authentication

- **Message Authentication Code (MAC)**
  - Hash encrypted with a symmetric key:
    An intruder will not be able to replace the hash value

- **Digital Signature**
  - Hash function encrypted with the owner's private key
    - Alice encrypts the hash with her private key
    - Bob validates it by decrypting it with her public key & comparing with *hash(M)*
  - Provides non-repudiation

Authentication

Key concept: prove that you can encrypt data that is presented to you

- Pre-shared keys
- Challenge Handshake Authentication Protocol (CHAP)
  - f(shared key, challenge #)
- Diffie-Hellman
  - Key exchange protocol: precursor to public key cryptography
  - Using Bob's public "key" and her private "key", Alice can compute a common key
  - Using Alice's public "key" and his private "key", Bob can compute the same common key
  - Prove that you can encrypt or decrypt data using the common key
- Public-key
  - Prove that you can encrypt or decrypt data using your private key

Public Key Authentication

Public key authentication

Demonstrate we can encrypt or decrypt a *nonce*

A random bunch of bits

- Alice wants to authenticate herself to Bob:
- Bob: generates nonce, *S*
  - Sends it to Alice
- Alice: encrypts *S* with her private key (signs it)
  - Sends result to Bob

Public key authentication

Bob:

1. Look up "alice" in a database of public keys
2. Decrypt the message from Alice using Alice's public key
3. If the result is *S*, then Bob is convinced he's talking with Alice

For mutual authentication, Alice has to present Bob with a nonce that Bob will encrypt with his private key and return

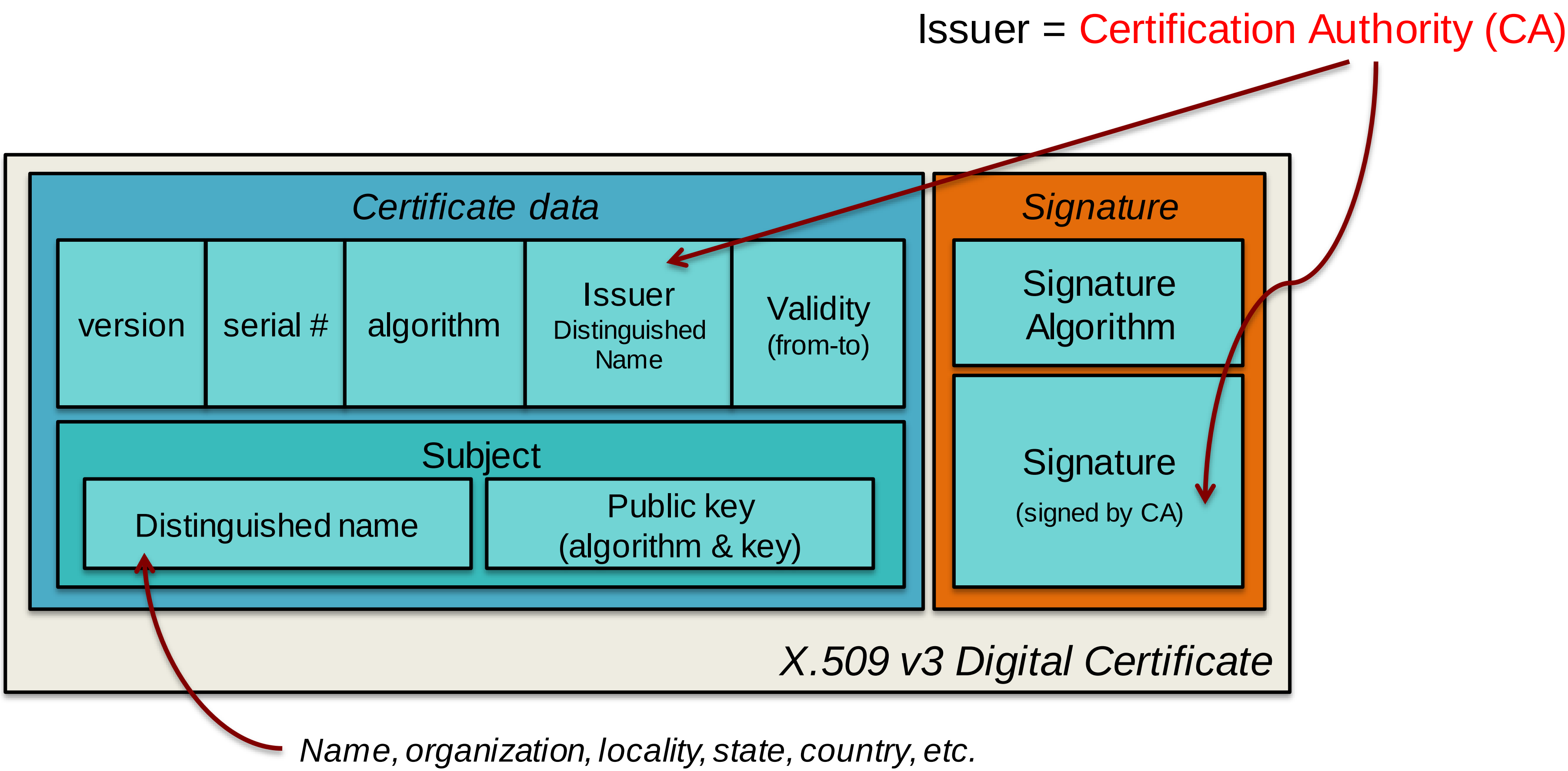
Public key authentication

- Identity is based on the key
  - *How do you know it really is Alice's public key?*
- One option:
  get keys from a trusted source
- Problem: requires always going to the source
  - cannot pass keys around
- Another option: *sign the public key*
  - Contents cannot be modified
  - **digital certificate**

X.509 Certificates

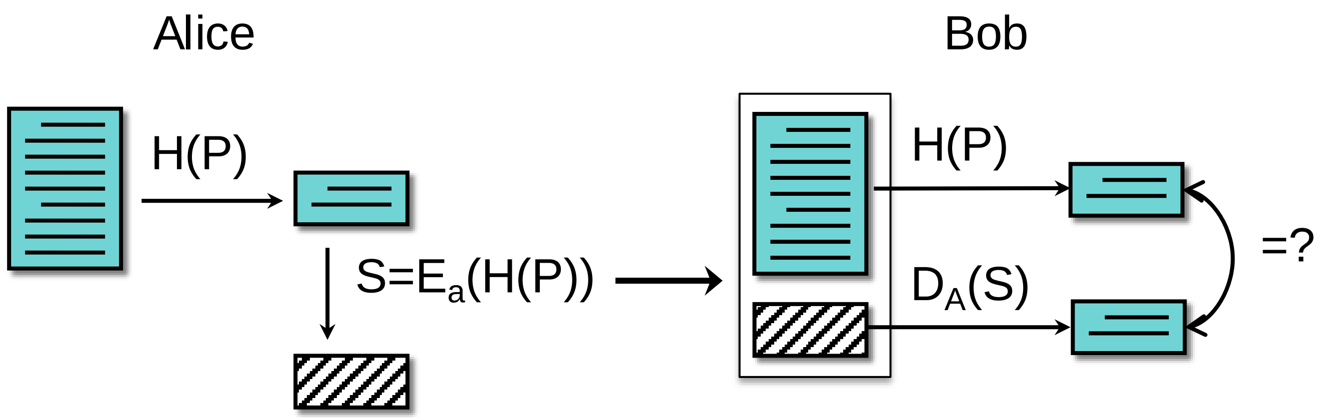
ISO introduced a set of authentication protocols

X.509: Structure for public key certificates:



Reminder: What's a digital signature?

Hash of a message encrypted with the signer's private key



X.509 certificates

When you get a certificate

- Verify its signature:
  - hash contents of certificate data
  - Decrypt CA's signature with CA's public key

Obtain CA's public key (certificate) from trusted source

Certificates prevent someone from using a phony public key to masquerade as another person

…*if you trust the CA*

Built-in trusted root certificates in iOS 9

- A-Trust-nQual-01
- A-Trust-Qual-01
- A-Trust-Qual-02
- AAA Certificate Services
- Actalis Authentication Root CA
- AddTrust Class 1 CA Root
- AddTrust External CA Root
- AddTrust Public CA Root
- AddTrust Qualified CA Root
- Admin-Root-CA
- AdminCA-CD-T01
- AffirmTrust Commercial
- AffirmTrust Networking
- AffirmTrust Premium ECC
- AffirmTrust Premium
- ANF Global Root CA
- Apple Root CA - G2
- Apple Root CA - G3
- Apple Root CA
- Apple Root Certificate Authority
- Application CA G2
- ApplicationCA
- ApplicationCA2 Root
- Autoridad de Certificacion Firmaprofesional CIF A62634068
- Autoridad de Certificacion Raiz del Estado Venezolano
- Baltimore CyberTrust Root
- Belgium Root CA2
- Buypass Class 2 CA 1
- Buypass Class 2 Root CA
- Buypass Class 3 CA 1
- Buypass Class 3 Root CA
- CA Disig Root R1
- CA Disig Root R2
- CA Disig
- Certigna
- Certinomis - Autorité Racine
- Certinomis - Root CA
- certSIGN ROOT CA
- Certum CA
- Certum Trusted Network CA 2
- Certum Trusted Network CA
- Chambers of Commerce Root - 2008
- Chambers of Commerce Root
- Cisco Root CA 2048
- Class 2 Primary CA
- Common Policy
- COMODO Certification Authority
- ComSign CA
- ComSign Global Root CA
- ComSign Secured CA
- D-TRUST Root Class 3 CA 2 2009
- D-TRUST Root Class 3 CA 2 EV 2009
- Deutsche Telekom Root CA 2
- DigiCert Assured ID Root CA
- DigiCert Assured ID Root G2
- DigiCert Assured ID Root G3
- DigiCert Global Root CA
- DigiCert Global Root G2
- DigiCert Global Root G3
- DigiCert High Assurance EV Root CA
- DigiCert Trusted Root G4
- DoD Root CA 2
- DST ACES CA X6
- DST Root CA X3
- DST Root CA X4
- E-Tugra Certification Authority
- EBG Elektronik Sertifika Hizmet Sağlayıcısı
- Echoworx Root CA2
- EE Certification Centre Root CA
- Entrust Root Certification Authority - EC1
- Entrust Root Certification Authority - G2
- Entrust Root Certification Authority
- Entrust.net Certification Authority (2048)
- Entrust.net Certification Authority (2048)
- ePKI Root Certification Authority
- Federal Common Policy CA
- GeoTrust Global CA
- GeoTrust Primary Certification Authority - G2
- GeoTrust Primary Certification Authority - G3
- GeoTrust Primary Certification Authority
- Global Chambersign Root - 2008
- Global Chambersign Root
- GlobalSign Root CA

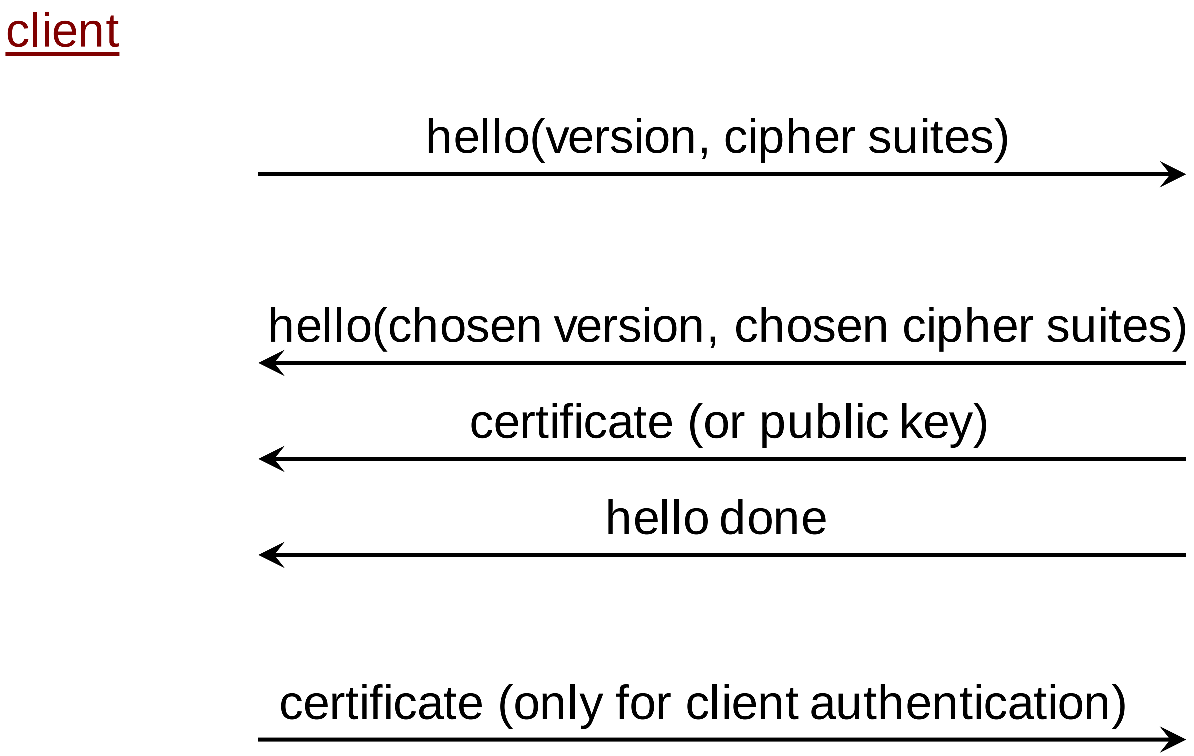
Partial list from https://support.apple.com/en-us/HT205205

SSL/TLS

Transport Layer Security (TLS)

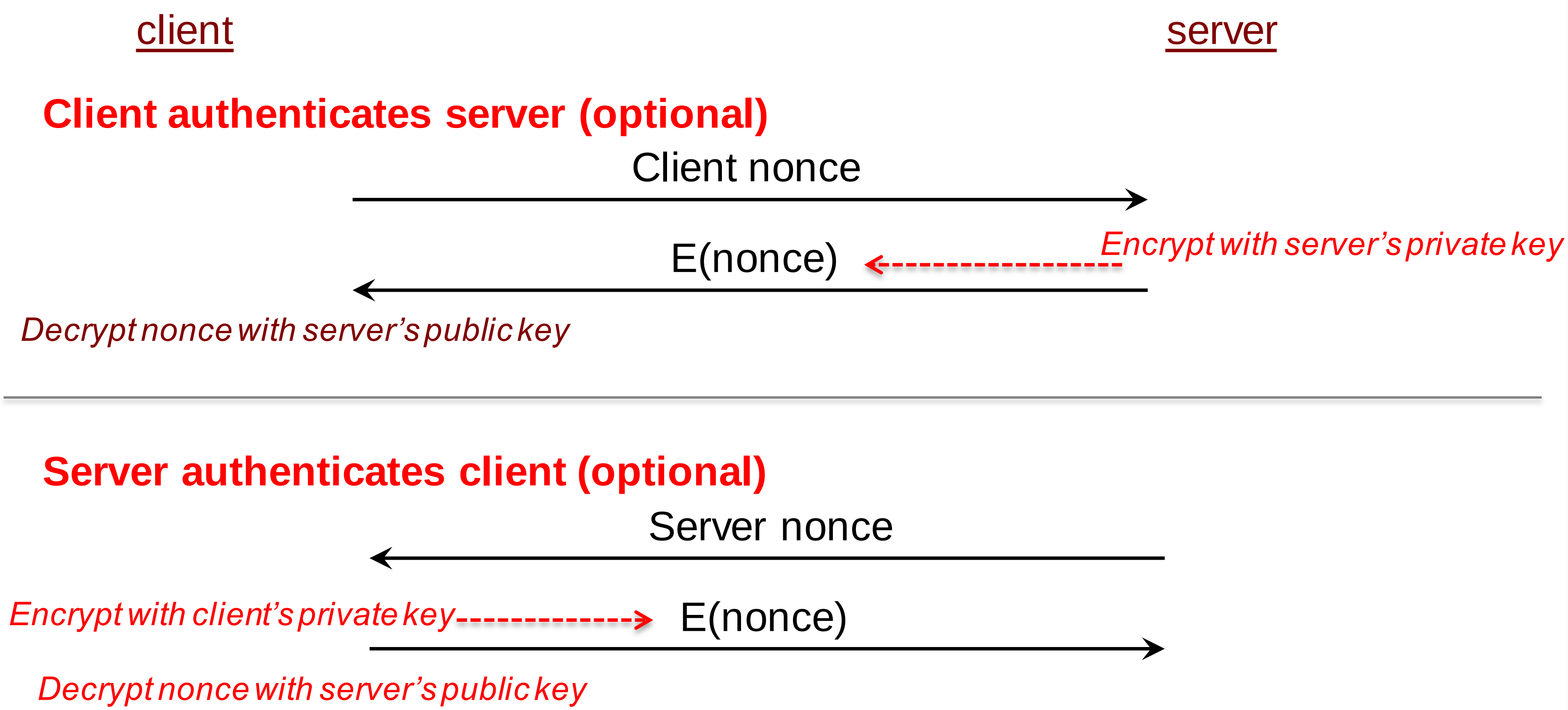
- *aka* Secure Socket Layer (SSL), which is an older protocol
- Sits on top of TCP/IP
- Goal: provide an encrypted and possibly authenticated communication channel
  - Provides authentication via RSA and X.509 certificates
  - Encryption of communication session via a symmetric cipher
- Hybrid cryptosystem: (usually, but also supports Diffie-Hellman)
  - Public key for authentication
  - Symmetric for data communication
- Enables TCP services to engage in secure, authenticated transfers
  - http, telnet, ntp, ftp, smtp, …

Transport Layer Security (TLS)



1. Establish protocol, version, cipher suite
   Get server certificate (or public key)
   [details depend on chosen cipher]

Transport Layer Security (TLS)



2. Authenticate: unidirectional or mutual (optional)

Transport Layer Security (TLS)

client

server

Pick a session key

Encrypt with server's public key - - - - - - → E(session key) →

Decrypt with server's private key

3. Establish a session key for symmetric cryptography

Transport Layer Security (TLS)

client

server

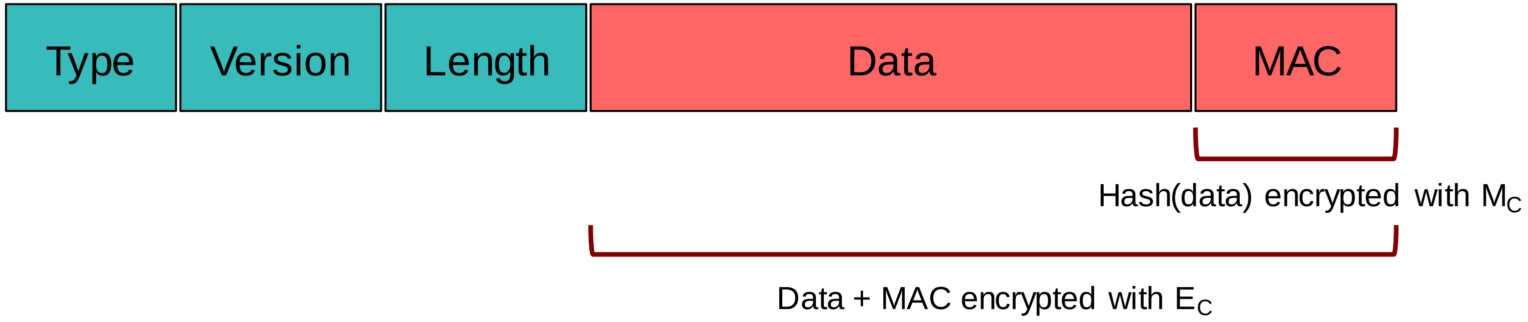
E_S(data)

Encrypt & decrypt with session key and symmetric algorithm (e.g., RC4 or AES)

4. Exchange data (symmetric encryption)

SSL Keys

- SSL really uses four session keys
  - E_C – encryption key for messages from Client to Server
  - M_C – MAC encryption key for messages from Client to Server
  - E_S – encryption key for messages from Server to Client
  - M_S – MAC encryption key for messages from Server to Client
- They are all derived from the random key selected by the client



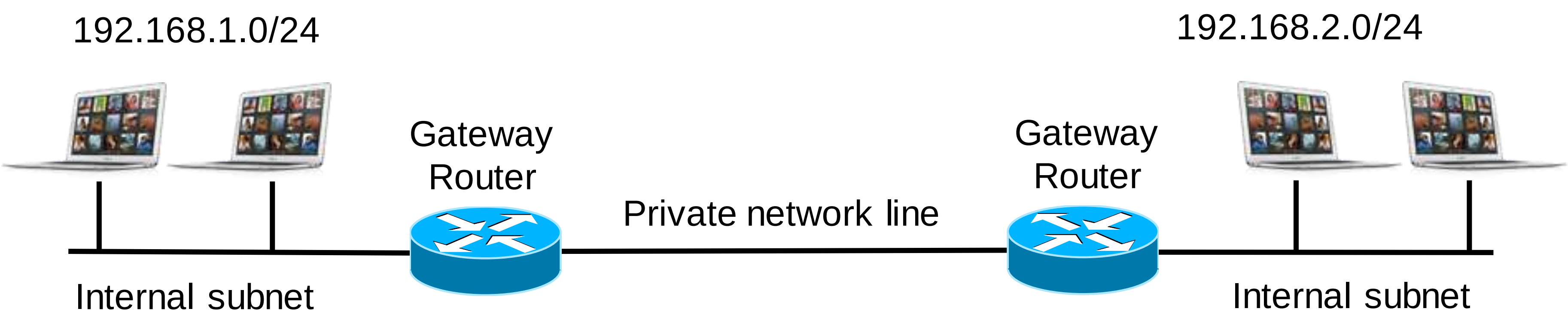
Cryptographic toolbox

- Symmetric encryption
- Public key encryption
- One-way hash functions
- Random number generators

Virtual Private Networks

Private networks

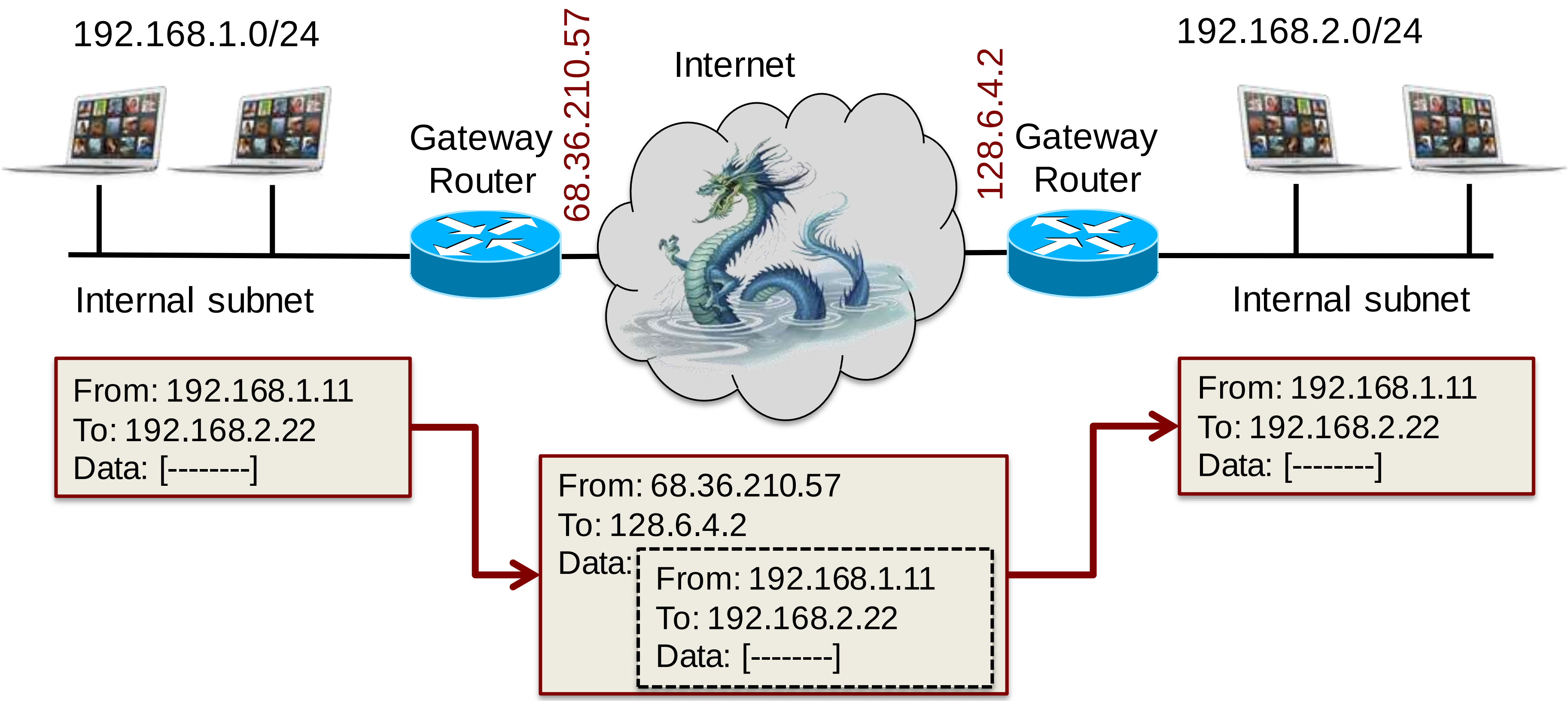
Connect multiple geographically-separated private subnetworks together



What's a tunnel?

Packet encapsulation

– Treat an entire IP datagram as payload on the public network



Tunnel mode vs. transport mode

- Tunnel mode
  - Communication between gateways
  - Or a host-to-gateway
  - Datagram is encapsulated

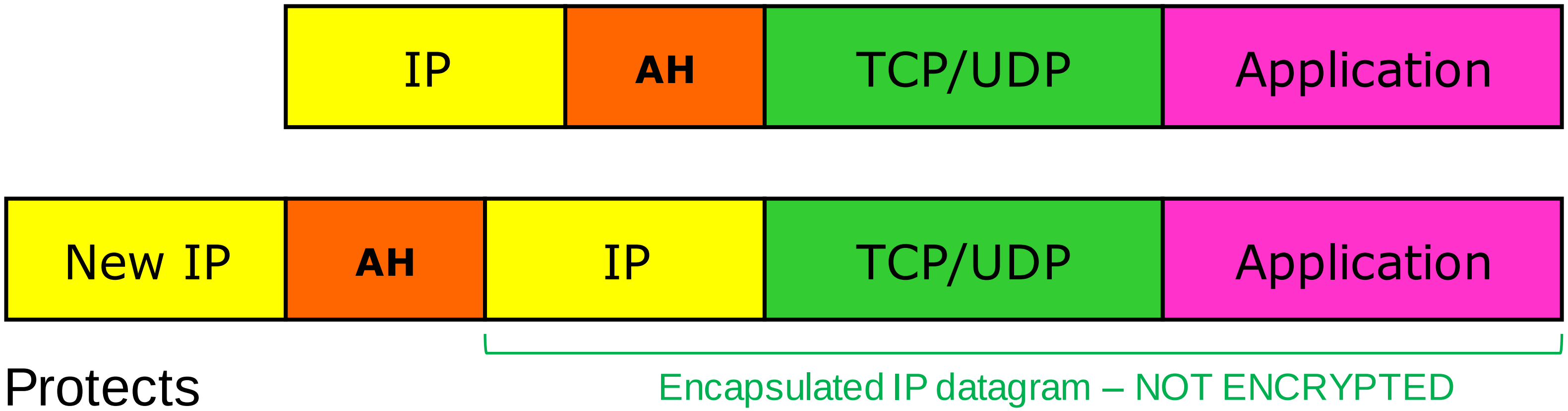
- Transport mode
  - Communication between hosts
  - IP header is not modified – routes to destination host

IPsec

- IPsec = Internet Protocol Security
- End-to-end VPN at the IP layer
- Two protocols:
  - IPsec Authentication Header Protocol (AH)
  - IPsec Encapsulating Security Payload (ESP)

IPsec Authentication Header (AH)

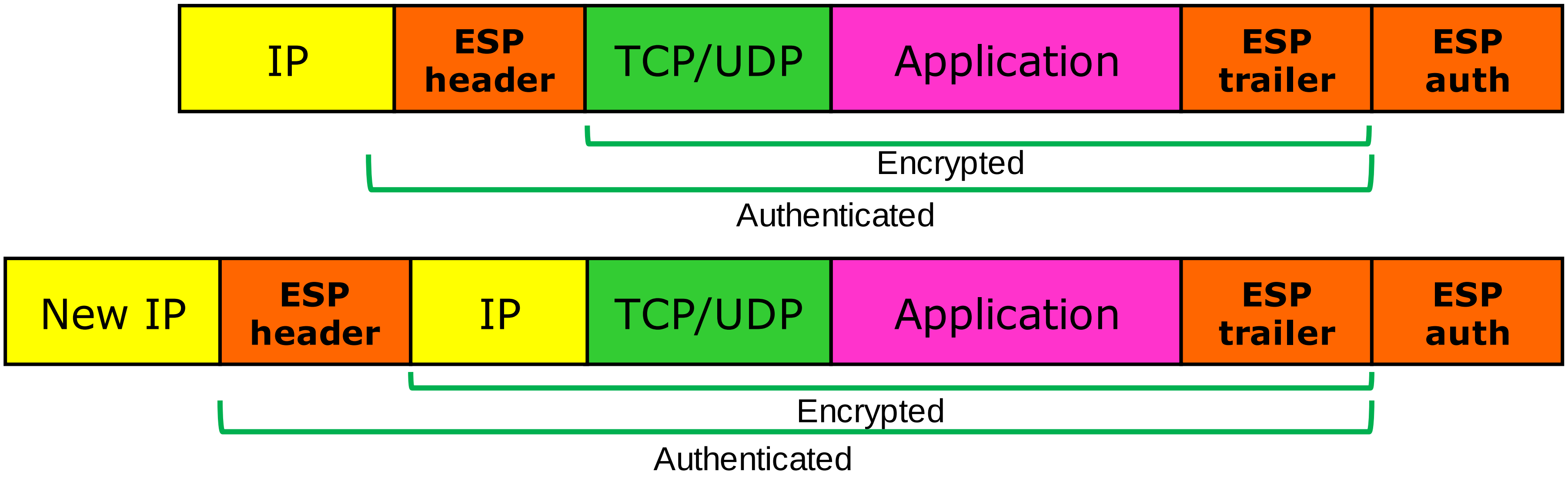
- Ensures the integrity & authenticity of IP packets
  - Digital signature for the contents of the entire IP packet
  - Over unchangeable IP datagram fields (e.g., not TTL or fragmentation)



- Protects
  - Tampering
  - Forging addresses
  - Replay attacks (signed sequence number in AH)
- Directly on top of IP (protocol 51) - not UDP or TCP

IPsec Encapsulating Security Payload (ESP)

- Encrypts entire payload
  - Optional authentication of payload + IP header (everything AH does)



- Directly on top of IP (protocol 51) - not UDP or TCP

The end