Network Security Protocols

Note: The slides are adapted from the materials from Prof. Richard Han at CU Boulder and Profs. Jennifer Rexford and Mike Freedman at Princeton University, and the networking book (Computer Networking: A Top Down Approach) from Kurose and Ross.

Introduction to Cryptography

What is Cryptography?

- Comes from Greek word meaning "secret"
  - Primitives also can provide integrity, authentication
- Cryptographers invent secret codes to attempt to hide messages from unauthorized observers

plaintext $\xrightarrow{\textbf{encryption}}$ **ciphertext** $\xrightarrow{\textbf{decryption}}$ **plaintext**

- Modern encryption:
  - *Algorithm* public, *key* secret and provides security
  - May be symmetric (secret) or asymmetric (public)

Cryptographic Algorithms: Goal

- Given key, relatively easy to compute
- Without key, hard to compute (invert)
- “Level” of security often based on “length” of key

Three Types of Functions

- Cryptographic hash Functions
  - Zero keys
- Secret-key functions
  - One key
- Public-key functions
  - Two keys

Cryptographic hash functions

Cryptography Hash Functions

- Take message, *m*, of arbitrary length and produces a smaller (short) number, *h(m)*
- Properties
  - Easy to compute *h(m)*
  - Pre-image resistance: Hard to find an *m*, given *h(m)*
    - *"One-way function"*
  - Second pre-image resistance: Hard to find two values that hash to the same *h(m)*
    - *E.g.* discover collision: *h(m) == h(m') for m != m'*
  - Often assumed: output of hash fn's "looks" random

Example use #1: Passwords

- Password hashing
  - Can't store passwords in a file that could be read
    - Concerned with insider attacks!
  - Must compare typed passwords to stored passwords
    - Does hash (typed) == hash (password) ?
  - Actually, a "salt" is often used: hash (input || salt)
    - Avoids precomputation of all possible hashes in "rainbow tables" (available for download from file-sharing systems)

Example use #2: Self-certifying naming

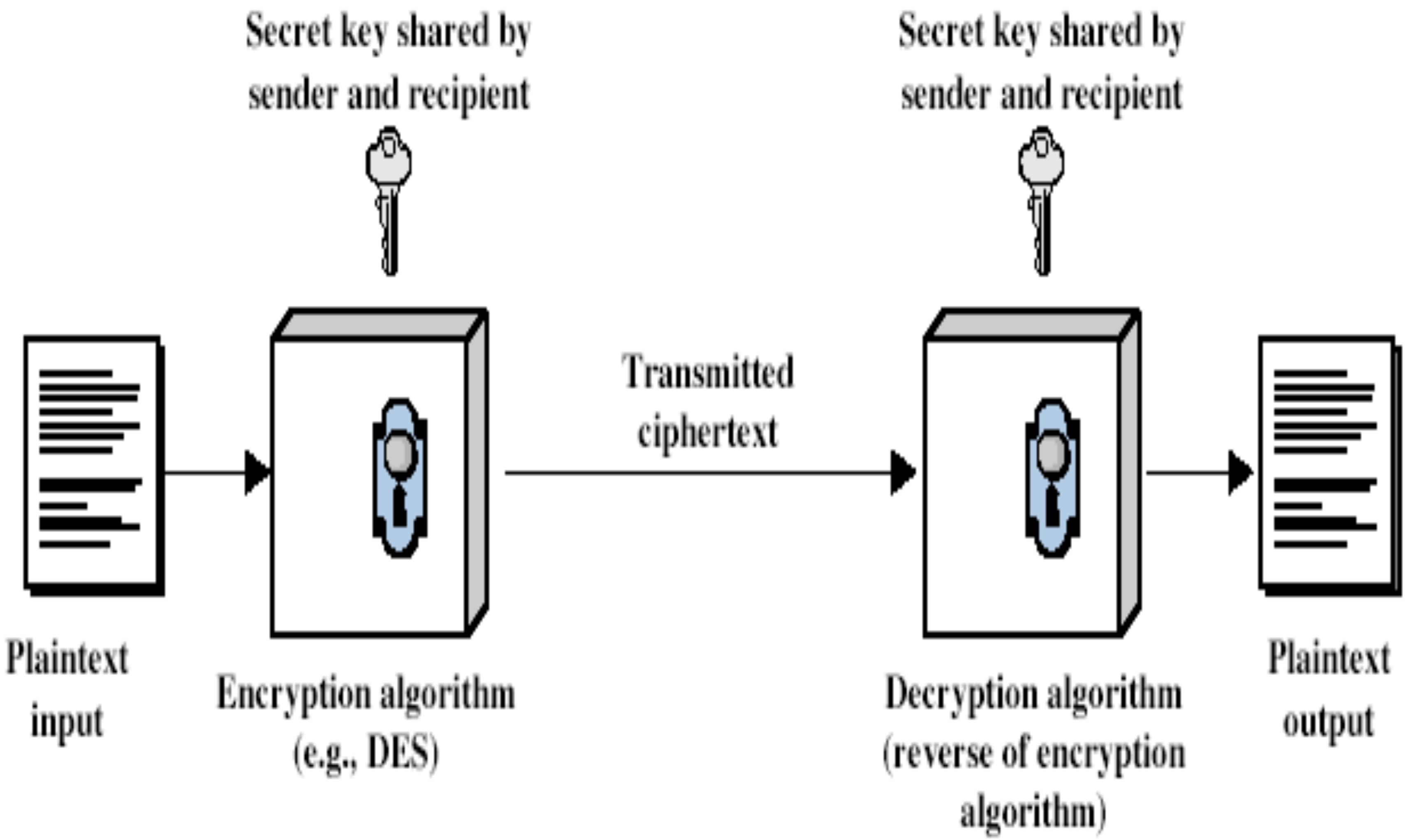
- File-sharing software (LimeWire, BitTorrent)
  - File named by F_{name} = hash (data)
  - Participants verify that hash (downloaded) == F_{name}
    - If check fails, reject data
- Recursively applied…
  - BitTorrent file has many chunks
  - Control file downloaded from tracker includes:
    - \forall chunks, $F_{chunk\ name}$ = hash (chunk)
  - BitTorrent client verifies each individual chunk

Symmetric (Secret) Key Cryptography

Symmetric Encryption

- Also: "conventional / private-key / single-key"
  - Sender and recipient share a common key
  - All classical encryption algorithms are private-key
  - Dual use: confidentiality or authentication/integrity
    - Encryption vs. msg authentication code (MAC)
- Was only type of encryption prior to invention of public-key in 1970's
  - Most widely used
  - More computationally efficient than "public key"

Symmetric Cipher Model



Use and Requirements

- Two requirements
  - Strong encryption algorithm
  - Secret key known only to sender / receiver
- Goal: Given key, generate 1-to-1 mapping to ciphertext that looks random if key unknown
  - Assume *algorithm* is known (no security by obscurity)
  - Implies secure channel to distribute key

Confidentiality (Encryption)

Sender:

- Compute C = $AES_K(M)$
- Send C

Receiver:

- Recover M = $AES'_K(C)$

Auth/Integrity (MAC)

Sender:

- Compute H = AES_K(SHA1 (M))
- Send <M, H>

Receiver:

- Computer H' = AES_K(SHA1 (M))
- Check H' == H

Public-Key Cryptography

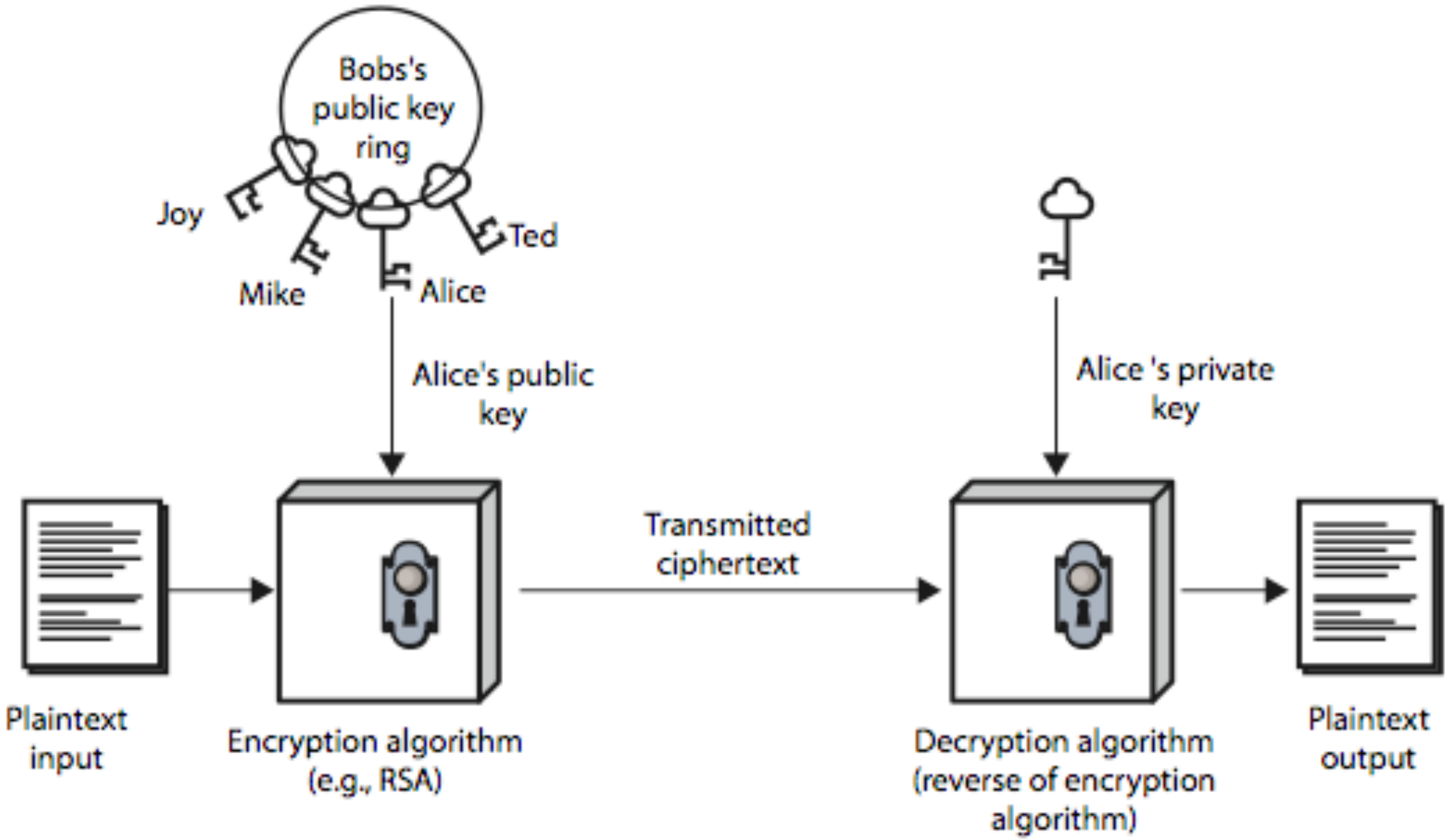
Why Public-Key Cryptography?

- Developed to address two key issues:
  - Key distribution: Secure communication w/o having to trust a key distribution center with your key
  - Digital signatures: Verify msg comes intact from claimed sender (w/o prior establishment)
- Public invention due to Whitfield Diffie & Martin Hellman in 1976
  - Known earlier in classified community

Public-Key Cryptography

- **Public-key/asymmetric** crypto involves use of two keys
  - **Public-key:** Known by anybody, and can be used to encrypt messages and verify signatures
  - **Private-key:** Known only to recipient, used to decrypt messages and sign (create) signatures
- **Asymmetric** because
  - Can encrypt messages or verify signatures w/o ability to decrypt messages or create signatures

Public-Key Cryptography



Security of Public Key Schemes

- Like private key schemes, brute force search possible
  - But keys used are too large (e.g., >= 1024bits)
- Security relies on a difference in computational difficulty b/w easy and hard problems
  - RSA: exponentiation in composite group vs. factoring
  - ElGamal/DH: exponentiation vs. discrete logarithm in prime group
  - Hard problem is known, but computationally expensive
- Requires use of very large numbers
  - Hence is slow compared to private key schemes
  - RSA-1024: 80 us / encryption; 1460 us / decryption [cryptopp.com]
  - AES-128: 109 MB / sec = 1.2us / 1024 bits

(Simple) RSA Algorithm

- **Security** due to cost of factoring large numbers
  - Factorization takes $O(e^{\log n \log \log n})$ operations (hard)
  - Exponentiation takes $O((\log n)^3)$ operations (easy)
- To encrypt a message M the sender:
  - Obtain public key `{e,n}`; compute $C = M^e \bmod n$
- To decrypt the ciphertext C the owner:
  - Use private key `{d,n}`; computes $M = C^d \bmod n$
- Note that msg M must be smaller than the modulus n

(Simple) RSA Algorithm

- To encrypt a message M the sender:
  - Obtain public key `{e,n};` compute $C = M^e \bmod n$
- To decrypt the ciphertext C the owner:
  - Use private key `{d,n};` computes $M = C^d \bmod n$
- Based on the difficulty of factoring the product of two prime numbers
  - Choose 2 large prime numbers p and q
  - n = p * q should be about 1024 bits long
  - z = (p-1)*(q-1)
  - Choose e<n with no common factors with z
  - Find d such at (e*d) mod z = 1
- Public key is (n,e), private key is (n,d)
- Message is encrypted to $c = m^e \bmod n$
- Ciphertext c is decrypted to $m = c^d \bmod n$

RSA Example

A host chooses *p=5, q=7*. Then *n=35, z=24*.

e=5 (so *e, z* relatively prime).

d=29 (so *ed-1* exactly divisible by z).

	letter	m	m^e	$c = m^e \bmod n$
encrypt:	"L"	12	1524832	17

	c	c^d	$m = c^d \bmod n$	letter
decrypt:	17	481968572106750915091411825223072000	12	"L"

- Public-key cryptography is slow because of the *exponentiation* (on both sides) – don't use for time-sensitive data
  - 2 orders of magnitude slower than symmetric key crypto

Security

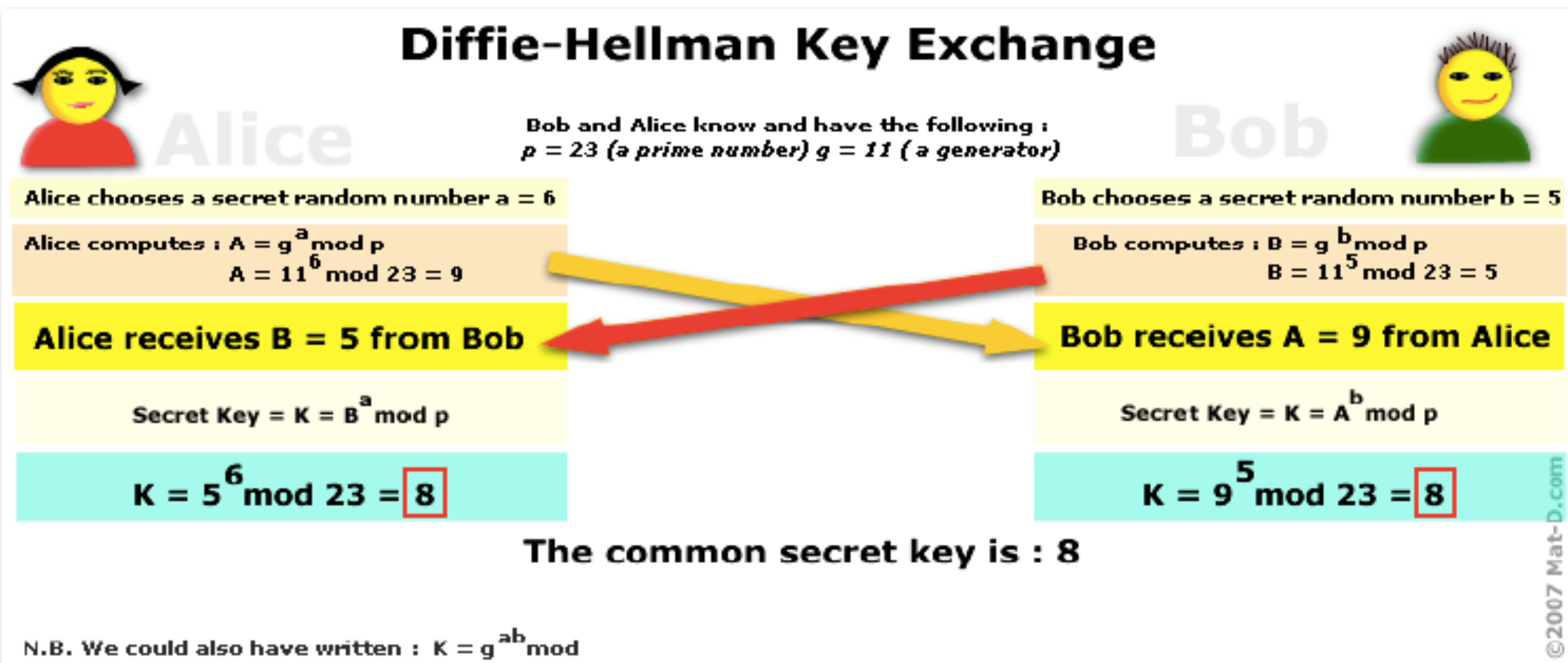
- Provides security because:
  - There are no known algorithms for quickly factoring n=p*q, the product of two large prime numbers
  - If we could factor n into p and q, then it would be easy to break the algorithm
- A 512 bit number (155 decimals) was factored into two primes in 1999 using one Cray and 300 workstations
  - 1024 bit keys still safe? 2048 and 4096 bits Yes!

Properties

- Incredibly useful property of public-key cryptography:
  - $m = c^d \bmod n = (m^e \bmod n)^d$
    $= (m^e)^d \bmod n = (m^d)^e \bmod n$
  - Thus, can swap the order in which the keys are used
  - Example: can use private key for encryption and a public key for decryption

Diffie-Hellman Symmetric Key Exchange

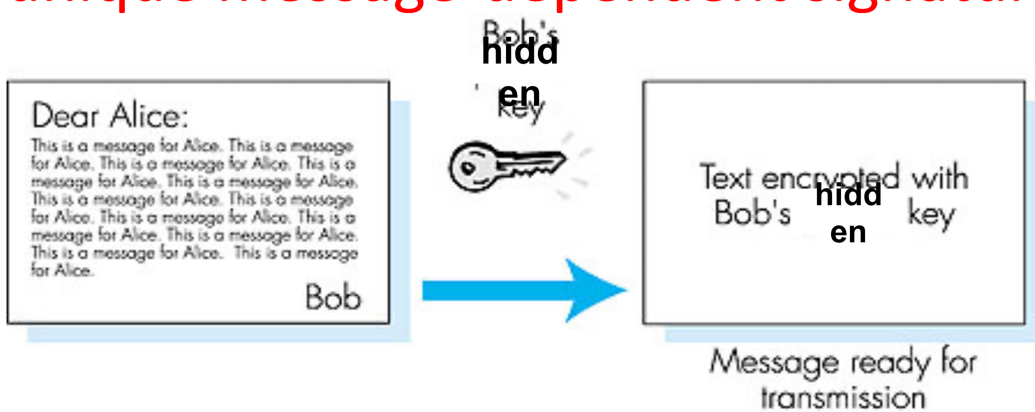
- Establish a shared key between two parties (without prior knowledge of each other) under insecure communication channel (e.g. Internet)
  - Alice and Bob, prime ***p***-ordered group ***G***, a generator ***g***
  - Alice chooses a random number ***a*** and sends g^a to Bob
  - Bob chooses a random number ***b*** and sends g^b to Alice
  - Alice computes $K_{Alice} = (g^b)^a = g^{ab}$, Bob computes $K_{Bob} = (g^a)^b = g^{ab}$
  - Note that $g^a * g^b = g^{a+b} \neq g^{ab}$



Authentication

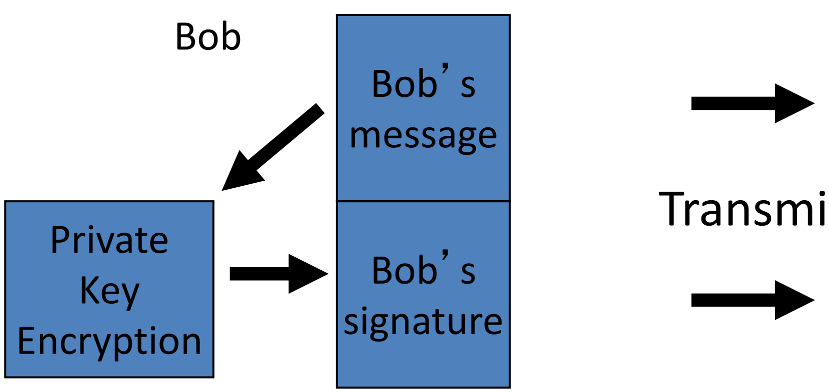
Authentication by Uniquely Encrypting Messages

- Similar conceptually to handwritten signatures
- Idea is to encrypt a message m using your hidden key (symmetric secret key, or in the case of asymmetric key cryptography the private key
  - Encryption of a message m by your hidden key K generates a unique ciphertext c=E(m,K).
  - Only your key could have generated this ciphertext c, so c is your unique message-dependent signature



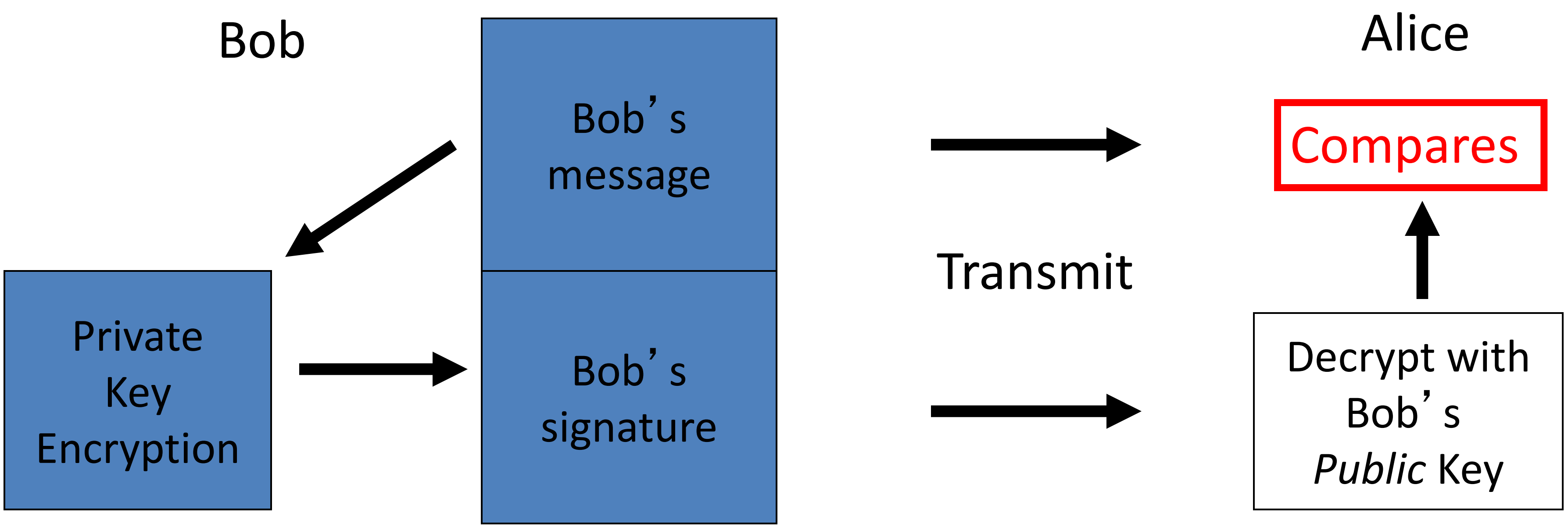
Authentication via Digital Signatures (1)

- Apply public key cryptography to sign messages
- Uses a property of public-key cryptography (e.g. RSA)
  - $m = c^d \bmod n = (m^e \bmod n)^d = (m^e)^d \bmod n = (m^d)^e \bmod n$
  - Thus, can swap the order: use private key for encryption and a public key for decryption
- Method 1
  - Bob encrypts entire message with Bob's private key. This is Bob's signature. Bob sends both the message and the digital signature



Authentication via Digital Signatures (2)

- Method 1 (cont.)
  - Alice decrypts Bob's message using Bob's public key
  - If decrypted message matches the message, Alice knows that the signed message could only have come from Bob (assuming only Bob knows his private key)

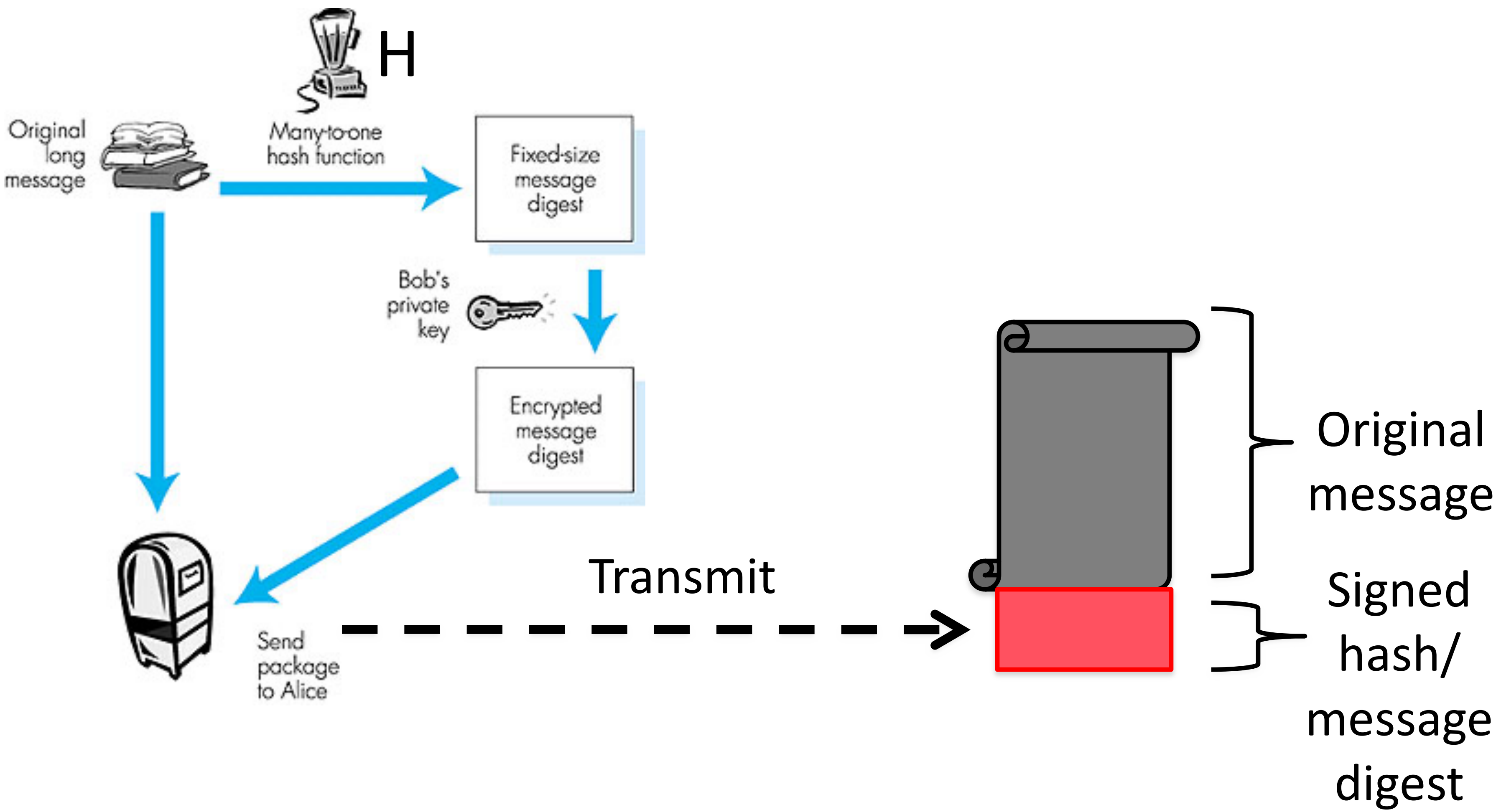


Authentication via Digital Signatures (3)

- Do you see an inefficiency with this technique (Method 1)?
  - Signing the full document/message is computationally expensive and doubles the overhead/bandwidth
- Method 2: Instead, compute a hash on the document
  - The hash is much smaller than the document, resembles a CRC. Also called a message digest
  - Hash function H generates the hash
  - Use private key to encrypt **only** the message digest

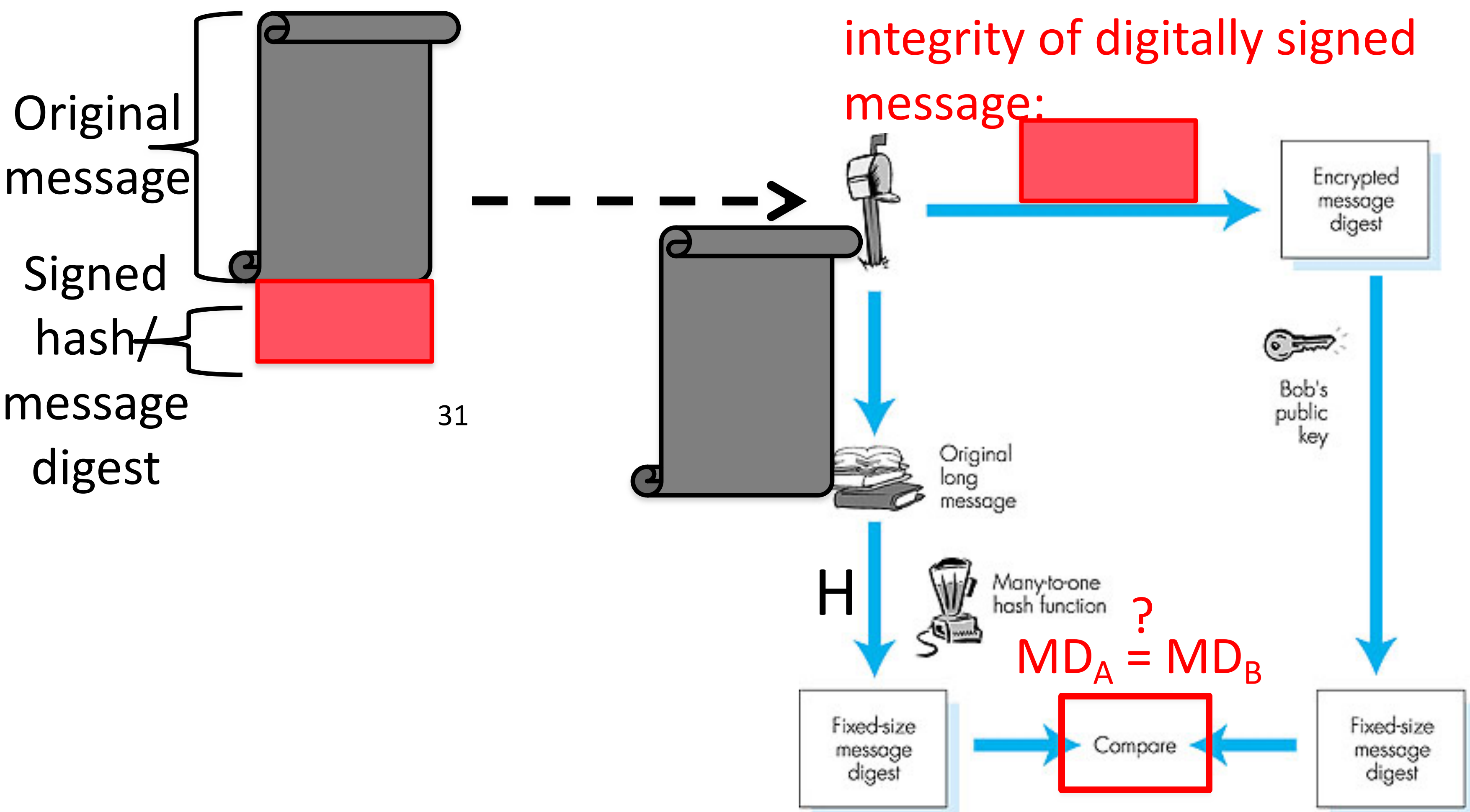
Digital signature = Signed Message Digest

Bob sends digitally signed message:



Digital signature = Signed Message Digest

Alice verifies signature and integrity of digitally signed message:



Email Security:
Pretty Good Privacy (PGP)

E-Mail Security

- Security goals
  - Confidentiality: only intended recipient sees data
  - Integrity: data cannot be modified en route
  - Authenticity: sender and recipient are who they say
- Security non-goals
  - Timely or successful message delivery
  - Avoiding duplicate (replayed) message
  - (Since e-mail doesn't provide this anyway!)

Sender and Receiver Keys

- If the sender knows the receiver's public key
  - Confidentiality
  - Receiver authentication

- If the receiver knows the sender's public key
  - Sender authentication
  - Sender non-repudiation

Sending an E-Mail Securely

- Sender digitally signs the message
  - Using the sender’s private key
- Sender encrypts the data
  - Using a one-time session key
  - Sending the session key, encrypted with the receiver’s public key
- Sender converts to an ASCII format
  - Converting the message to base64 encoding
  - (Email messages must be sent in ASCII)

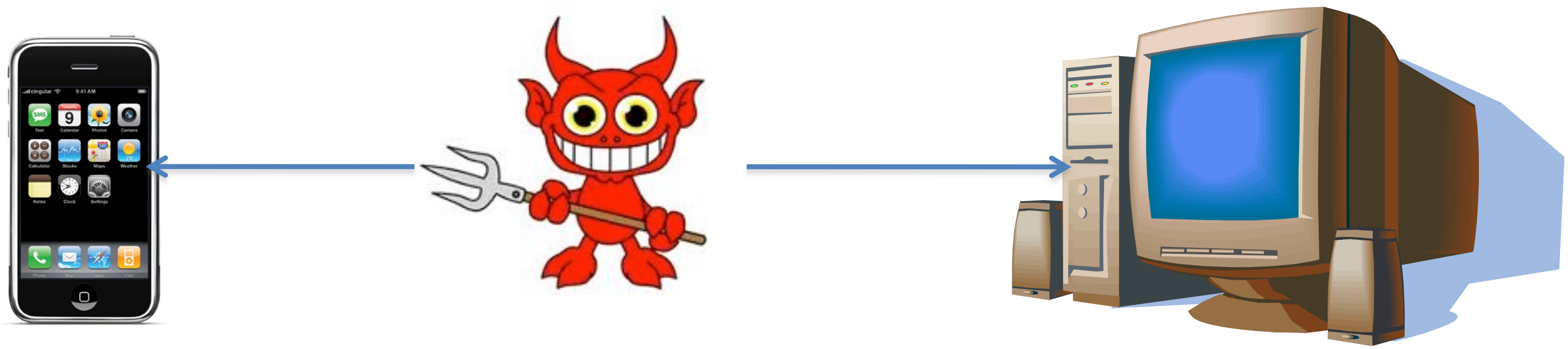
Public Key Certificate

- Binding between identity and a public key
  - "Identity" is, for example, an e-mail address
  - "Binding" ensured using a digital signature
- Contents of a certificate
  - Identity of the entity being certified
  - Public key of the entity being certified
  - Identity of the signer
  - Digital signature
  - Digital signature algorithm id

HTTP Security

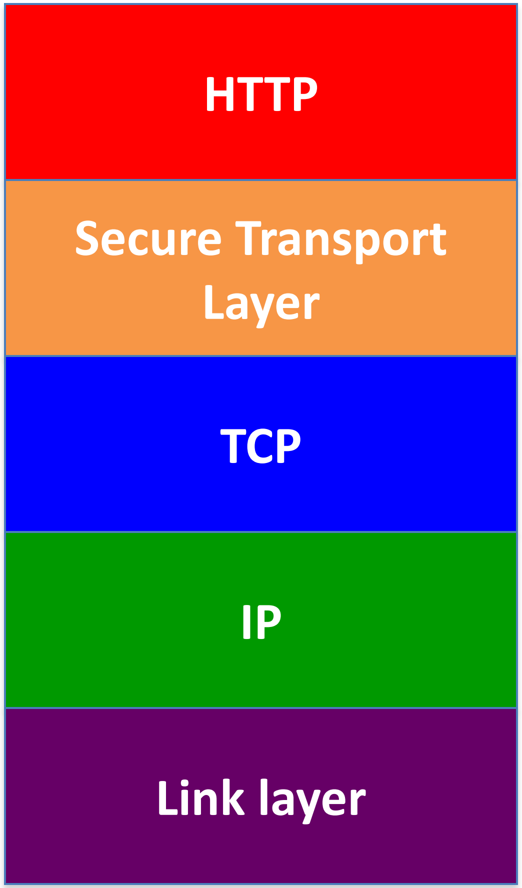
HTTP Threat Model

- Eavesdropper
  - Listening on conversation (confidentiality)
- Man-in-the-middle
  - Modifying content (integrity)
- Impersonation
  - Bogus website (authentication, confidentiality)



HTTP-S: Securing HTTP

- HTTP sits on top of secure channel (SSL/TLS)
  - https:// vs. http://
  - TCP port 443 vs. 80
- All (HTTP) bytes encrypted and authenticated
  - No change to HTTP itself!
- Where to get the key???



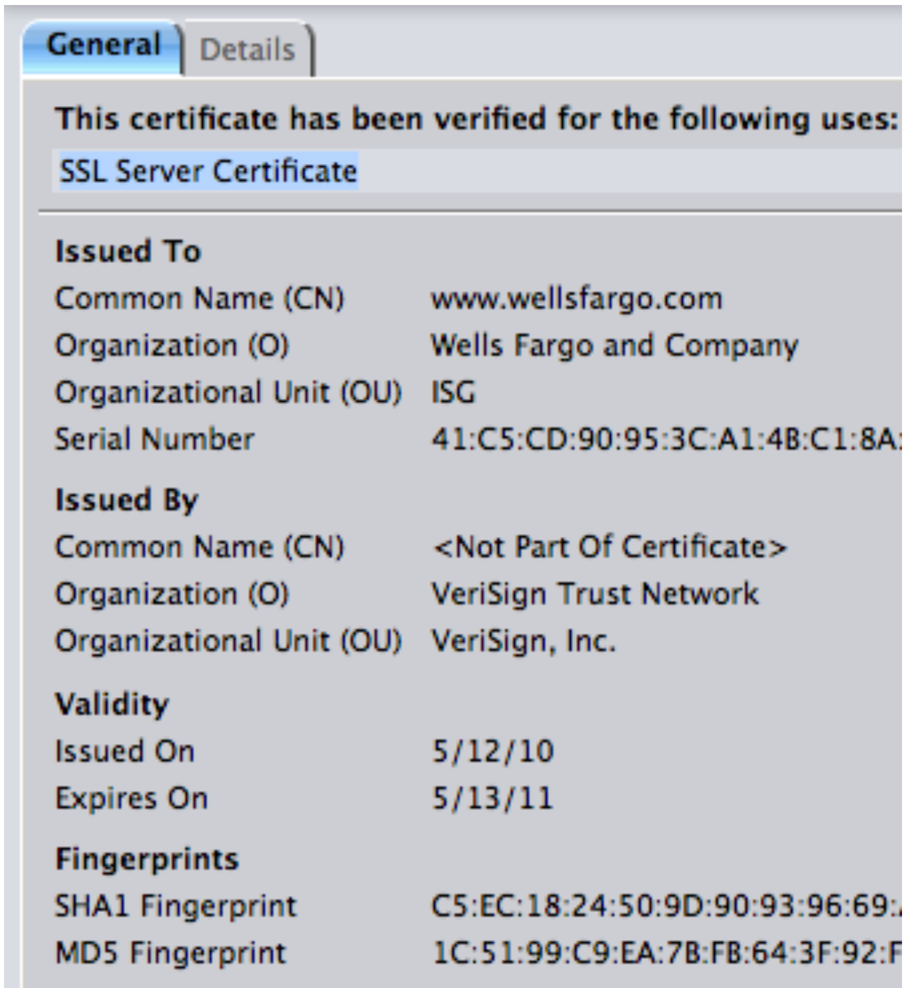
Learning a Valid Public Key

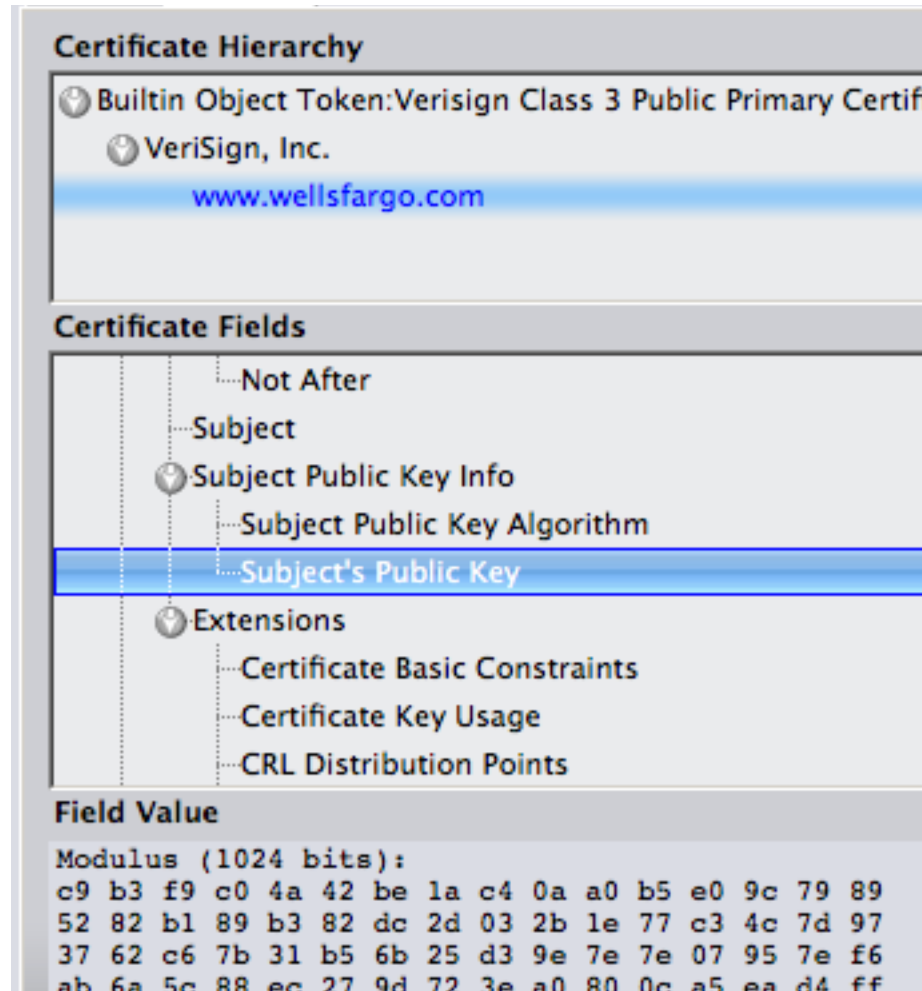
- What is that lock?
  - Securely binds domain name to public key (PK)
    - If PK is authenticated, then any message signed by that PK cannot be forged by non-authorized party
  - Believable only if you trust the attesting body
    - Bootstrapping problem: Who to trust, and how to tell if this message is actually from them?

Hierarchical Public Key Infrastructure

- Public key certificate
  - Binding between identity and a public key
  - "Identity" is, for example, a domain name
  - Digital signature to ensure integrity
- Certificate authority
  - Issues public key certificates and verifies identities
  - Trusted parties (e.g., VeriSign, GoDaddy, Comodo)
  - Preconfigured certificates in Web browsers

Public Key Certificate





Transport Layer Security (TLS)

Based on the earlier Secure Socket Layer (SSL) originally developed by Netscape

TLS Handshake Protocol

- Send new random value, list of supported ciphers

- Send new random value, digital certificate with PK

- Send pre-secret, encrypted under PK

- Create shared secret key from pre-secret and random
- Switch to new symmetric-key cipher using shared key

- Create shared secret key from pre-secret and random
- Switch to new symmetric-key cipher using shared key

Comments on HTTPS

- HTTPS authenticates server, not content
  - If CDN (Akamai) serves content over HTTPS, customer must trust Akamai not to change content
- Symmetric-key crypto after public-key ops
  - Handshake protocol using public key crypto
  - Symmetric-key crypto much faster (100-1000x)
- HTTPS on top of TCP, so reliable byte stream
  - Can leverage fact that transmission is reliable to ensure: each data segment received exactly once
  - Adversary can’t successfully drop or replay packets

IP Security

IP Security

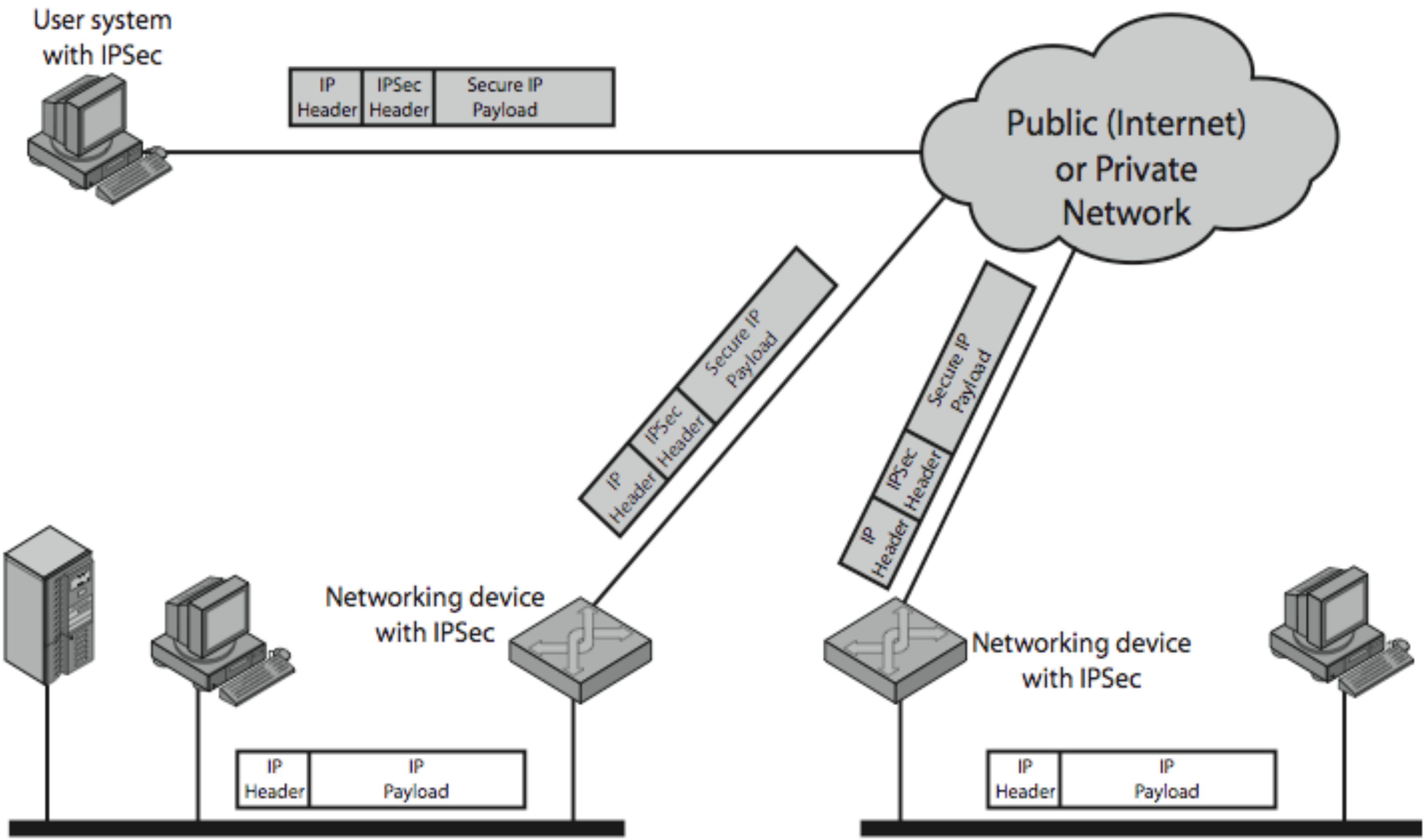
- There are range of app-specific security mechanisms
  - eg. TLS/HTTPS, S/MIME, PGP, Kerberos, …
- But security concerns that cut across protocol layers
- Implement by the network for all applications?

Enter IPSec!

IPSec

- General IP Security framework
- Allows one to provide
  - Access control, integrity, authentication, originality, and confidentiality
- Applicable to different settings
  - Narrow streams: Specific TCP connections
  - Wide streams: All packets between two gateways

IPSec Uses


User system
with IPSec
IP Header
IPSec Header
Secure IP Payload
Public (Internet) or Private Network
IP Header
IPSec Header
Secure IP Payload
IP Header
IPSec Header
Secure IP Payload
Networking device
with IPSec
Networking device
with IPSec
IP Header
IP Payload
IP Header
IP Payload

Benefits of IPSec

- If in a firewall/router:
  - Strong security to all traffic crossing perimeter
  - Resistant to bypass
- Below transport layer
  - Transparent to applications
  - Can be transparent to end users
- Can provide security for individual users

IP Security Architecture

- Specification quite complex
  - Mandatory in IPv6, optional in IPv4
- Two security header extensions:
  - Authentication Header (AH)
    - Connectionless integrity, origin authentication
      - MAC over most header fields and packet body
    - Anti-replay protection
  - Encapsulating Security Payload (ESP)
    - These properties, plus confidentiality

Encapsulating Security Payload (ESP)

- Transport mode: Data encrypted, but not header
  - After all, network headers needed for routing!
  - Can still do traffic analysis, but is efficient
  - Good for host-to-host traffic
- Tunnel mode: Encrypts entire IP packet
  - Add new header for next hop
  - Good for VPNs, gateway-to-gateway security

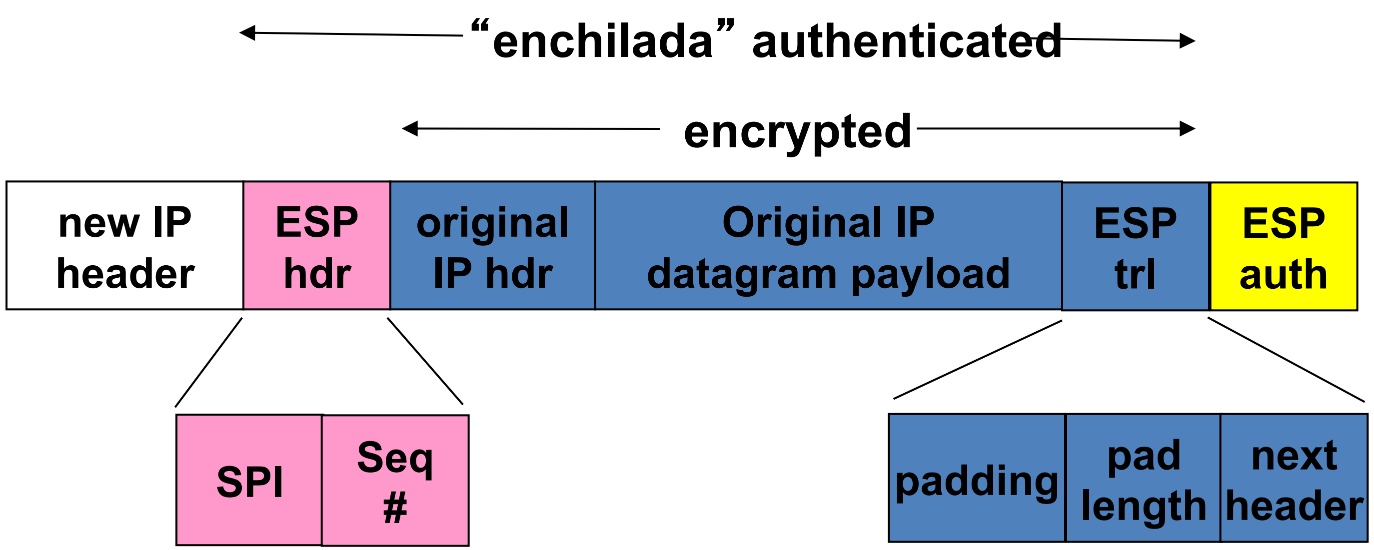
Four combinations are possible!

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

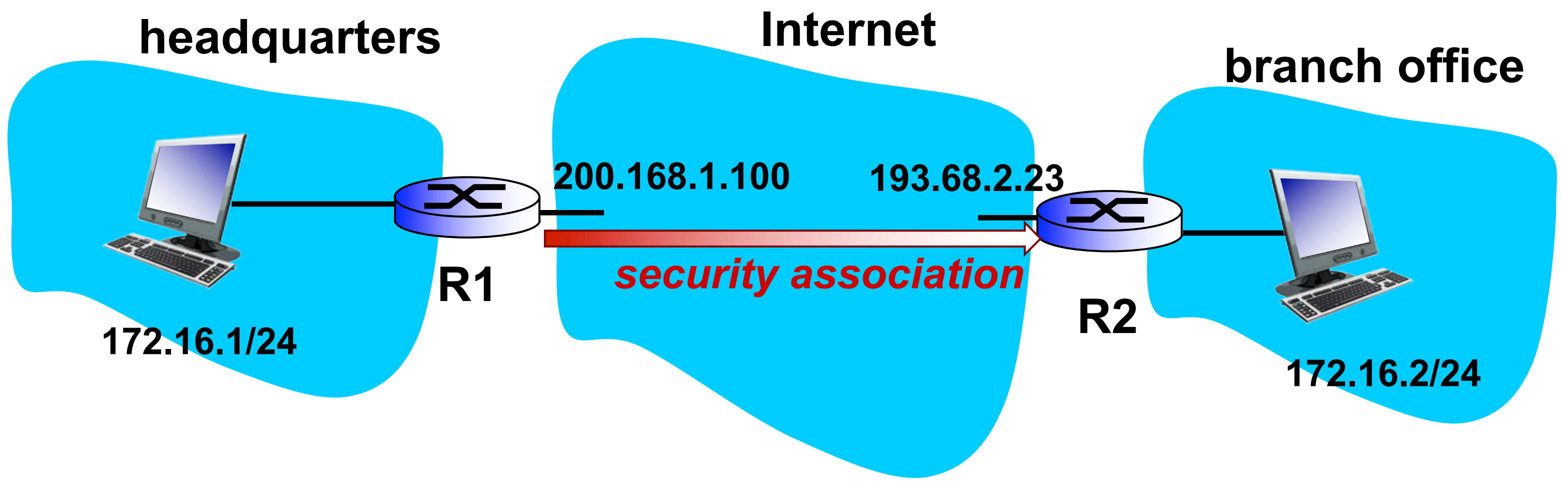
most common and most important

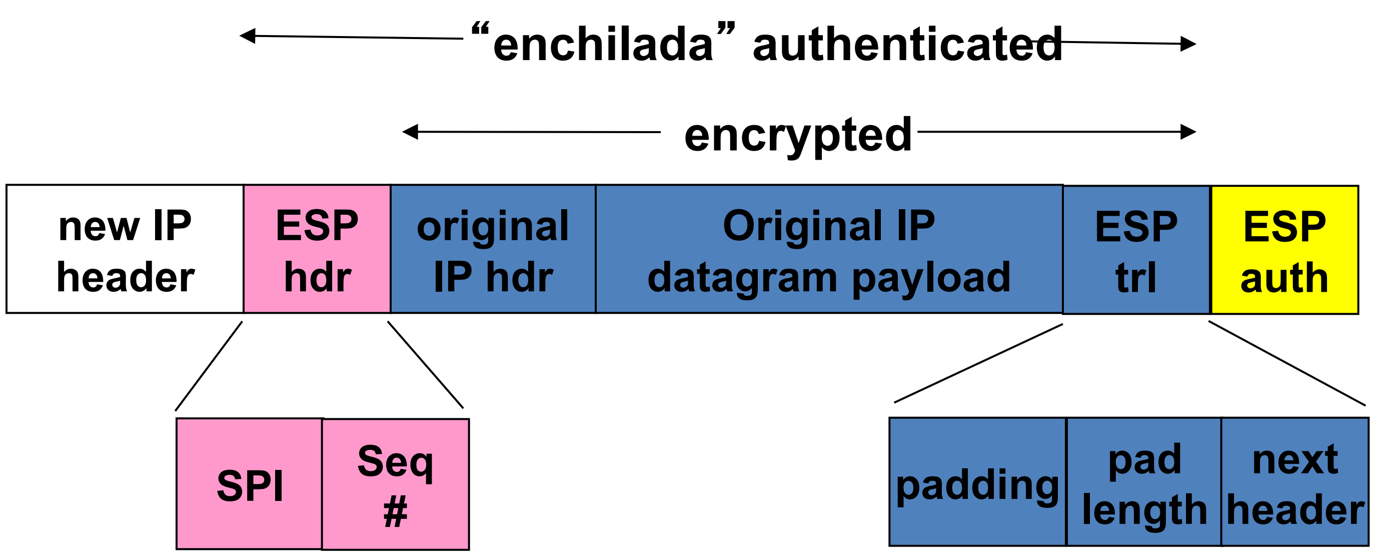
IPsec datagram

focus for now on tunnel mode with ESP



What happens?





Conclusions

- Security at many layers
  - Application, transport, and network layers
  - Customized to the properties and requirements
- Exchanging keys
  - Public key certificates
  - Certificate authorities vs. Web of trust