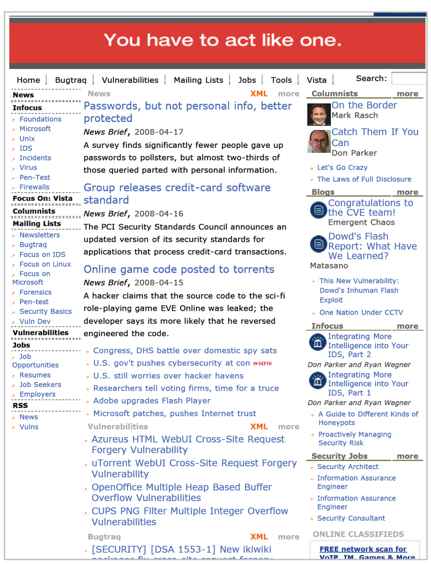
Security

- Overview
  - Security Goals
  - The Attack Space
- Security Mechanisms
  - Introduction to Cryptography
  - Authentication
  - Authorization
  - Confidentiality
- Case Studies

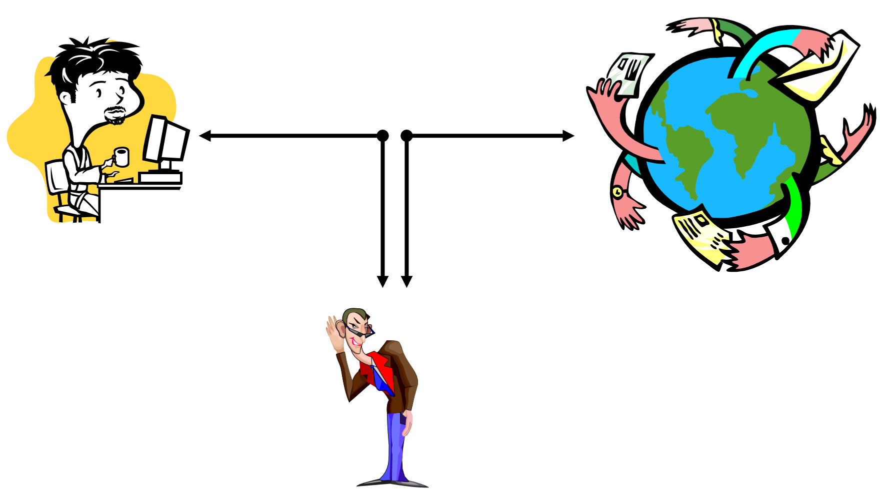
Security Today…



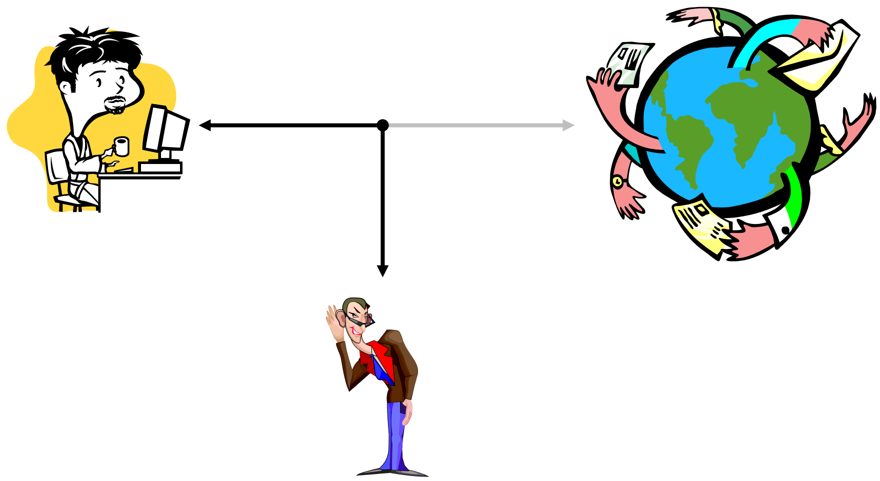
Typical Attacks: Penetration Attempts

- Two basic forms:
  - completely bypass authentication mechanism
  - obtain information or alter the system so as to enter system as authorized user
- Attempts:
  - **Wire tapping** (active *vs.* passive)
  - **Trial and error**
  - **Browsing**
    - Search storage (in particular previously allocated, but now available) for unauthorized information.
  - **Trap doors**
    - Unspecified and undocumented features of the system that may be exploited to perform unauthorized actions.
  - **Trojan horse**
  - **Searching of waste**

Typical Attacks: Man-In-The-Middle

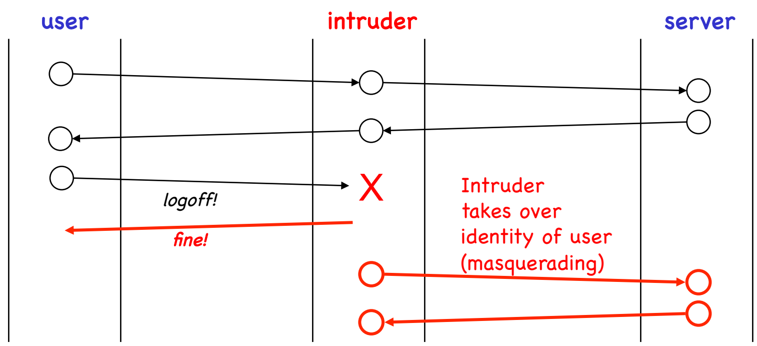


Typical Attacks: Masquerading



Man-In-The-Middle: Example

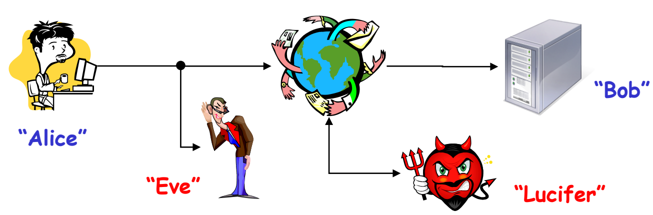
- **Passive tapping**
  - Listen to communication without altering contents.
- **Active wire tapping**
  - Modify data being transmitted
  - Example:



Security Threats

- **Information Disclosure:**
  - unauthorized dissemination of information
  - result of theft or illegal action of who has access to information
- **Information Destruction:**
  - loss of internal data structures
  - loss of stored information
  - information may be destroyed without being disclosed
- **Unauthorized Use of Service:**
  - bypass system accounting policies
  - unauthorized use of some proprietary services
- **Denial of Service:**
  - prevent an authorized user from utilizing the system's services in a timely manner

Security Goals



- **Authentication** of Alice (the client)
- **Authorization** of request from Alice
- **Confidentiality** (e.g. protect the content of request)
- **Accountability** (non-repudiation)
- **Availability**

Security: Systems Overview

Functionality	Authentication	Authorization	Confidentiality
Primitives	`sign()` `verify()`	Access control lists Capabilities "magic cookies"	`encrypt()` `decrypt()`
Cryptography	cyphers and hashes		

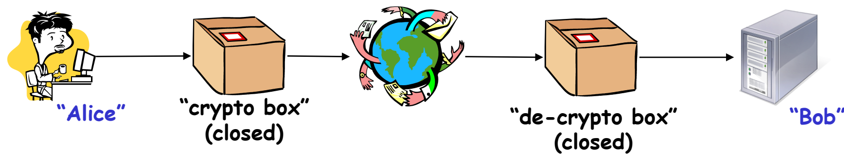
Cryptography

Functionality	Authentication	Authorization	Confidentiality
Primitives	`sign()` `verify()`	Access control lists Capabilities "magic cookies"	`encrypt()` `decrypt()`
Cryptography	cyphers and hashes		

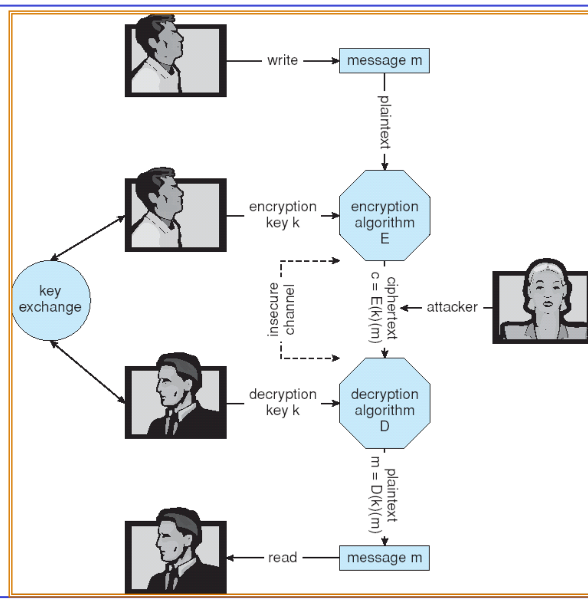
Cryptography:

- Closed-Design vs. Open-Design Cryptography
- Symmetric Encryption
- Asymmetric ("Public-Key") Encryption

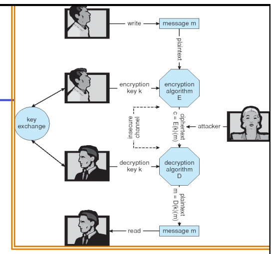
Closed-Design Cryptography



Open-Design Cryptography



Encryption



- Encryption algorithm consists of
  - Set of K **keys**
  - Set of M **Messages**
  - Set of C **ciphertexts** (encrypted messages)
  - A function $E : K \rightarrow (M \rightarrow C)$. That is, for each $k \in K$, $E(k)$ is a function for generating ciphertexts from messages.
    - Both E and $E(k)$ for any k should be efficiently computable functions.
  - A function $D : K \rightarrow (C \rightarrow M)$. That is, for each $k \in K$, $D(k)$ is a function for generating messages from ciphertexts.
    - Both D and $D(k)$ for any k should be efficiently computable functions.

- An encryption algorithm must provide this essential property:

> Given a ciphertext $c \in C$, a computer can compute m such that $E(k)(m) = c$ only if it possesses $D(k)$.

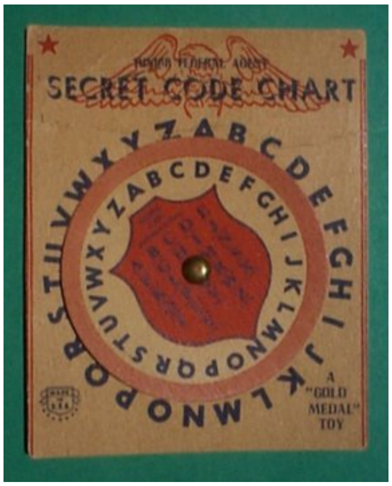
  - Thus, a computer holding $D(k)$ can decrypt ciphertexts to the plaintexts used to produce them, but a computer not holding $D(k)$ cannot decrypt ciphertexts.
  - Since ciphertexts are generally exposed (for example, sent on the network), it is important that it be infeasible to derive $D(k)$ from the ciphertexts

Symmetric Encryption

- Same key used to encrypt and decrypt
  - $E(k)$ can be derived from $D(k)$, and vice versa

- Examples:
  - Data Encryption Standard (**DES**)
  - **Triple-DES**
  - Advanced Encryption Standard (**AES**)
  - **Twofish**

Symmetric Encryption: Caesar Cipher



MERRY CHRISTMAS

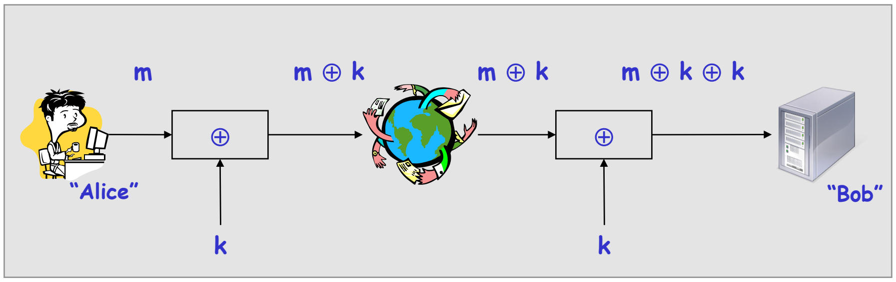
PHUUB FKULVWPDV

Symmetric Encryption: Jefferson's Wheel Cipher

Monticello Web Site: www.monticello.org/reports/interests/wheel_cipher.html

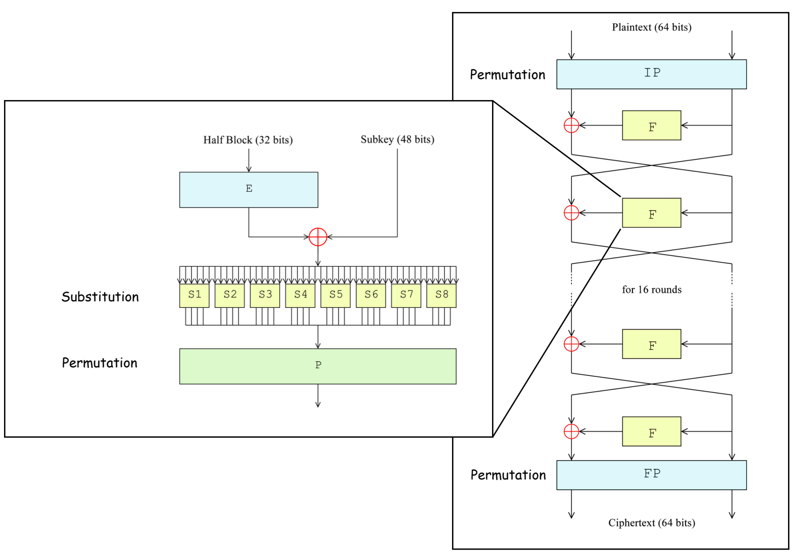
- Sender:
  - assemble wheels in some (secret) order.
  - Align message on one line.
  - Choose any of the other lines as ciphertext.
- Receive:
  - Assemble wheels in same secret order.
  - Align cipertext on one line.
  - Look for meaningful message on other lines.

Symmetric Encryption: XOR

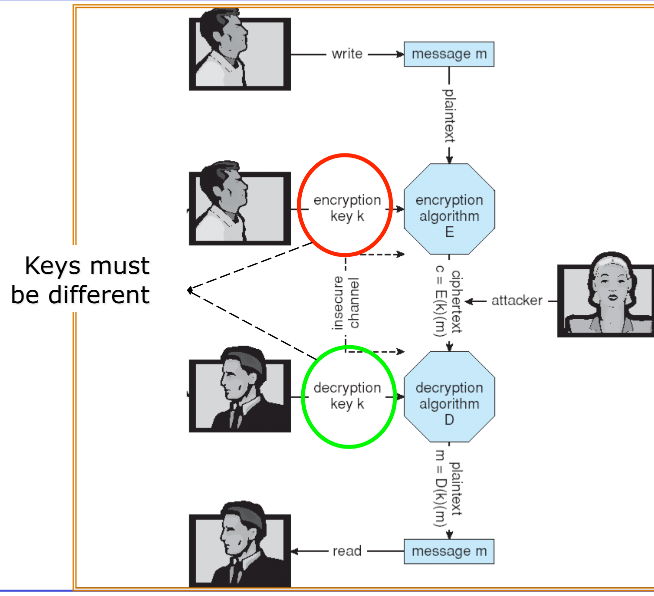


⊕	0	1
0	0	1
1	1	0

Symmetric Encryption: DES (Data Encryption Standard)



Asymmetric Encryption



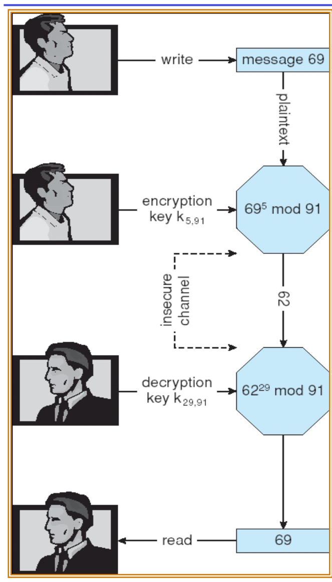
Asymmetric Encryption (cont.)

- Public-key encryption based on each user having two keys:
  - **public key** – published key used to encrypt data
  - **private key** – key known only to individual user used to decrypt data
- Must be an encryption scheme that can be made public without leaking the decryption scheme
  - Most common is **RSA block cipher**
  - Efficient algorithms exist for testing whether or not a number is prime
  - No efficient algorithm is known for finding the prime factors of a number

RSA (cont)

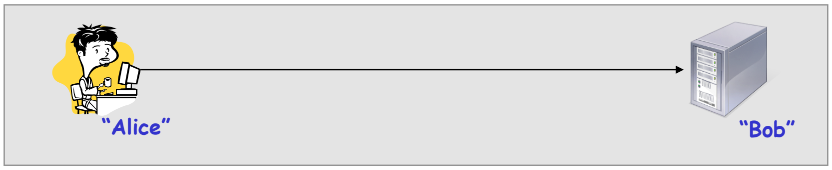
- If it is computationally infeasible to derive $D(k_d , N)$ from $E(k_e , N)$, $E(k_e , N)$ need not be kept secret and can be widely disseminated
  - $E(k_e , N)$ is the **public key**
  - $D(k_d , N)$ is the **private key**
  - N is the product of two large, randomly chosen prime numbers p and q (for example, p and q are 512 bits each)
  - Encryption algorithm is $E(k_e , N)(m) = m^{k_e} \bmod N$, where k_e satisfies $k_e k_d \bmod (p-1)(q-1) = 1$
  - The decryption algorithm is then $D(k_d , N)(c) = c^{k_d} \bmod N$

RSA: Example



- Make $p = 7$ and $q = 13$
- We then calculate
  $N = 7*13 = 91$ and $(p-1)(q-1) = 72$
- We next select k_e relatively prime to 72 and< 72, yielding 5
- Finally, we calculate k_d such that $k_e k_d \bmod 72 = 1$, yielding 29
- We how have our keys
  - Public key, $(k_e, N) = (5, 91)$
  - Private key, $(k_d, N) = (29, 91)$
- Encrypting the message 69 with the public key results in the ciphertext 62
  - $69^5 \bmod 91 = 62$
- Ciphertext can be decoded with the private key
  - $62^{29} \bmod 91 = 69$
- Public key can be distributed in clear text to anyone who wants to communicate with holder of public key

RSA in Practice…



$\{m\}^{kBpub}$: A encrypts message with B's public key.

$\{m\}^{kApriv}$: A signs a message with A's private key.

Symmetric vs. Asymmetric Encryption

- Symmetric cryptography based on simple transformations
- Asymmetric based on time consuming mathematical functions
  - Asymmetric much more compute intensive
  - Typically not used for bulk data encryption
  - Used, instead, for short plaintexts, for example symmetric keys.

Key Exchange: Diffie Hellman

Step 1 Alice and Bob agree on a large prime m and "primitive root" g mod m.
Note: m and g need not be secret.

Step 2 Alice and Bob privately pick random integer x and y, respectively.

Step 3 Alice and Bob exchange $X = g^x$ mod m and $Y = g^y$ mod m, respectively.

Step 4 Alice and Bob privately compute $k = Y^x$ mod m and $k' = X^y$ mod m, respectively.

$k = k'$ mod m, since

$$k' = X^y = (g^x)^y = g^{xy} = (g^y)^x = Y^x = k \bmod m$$

Scheme can be broken if Eve succeeds to solve the equation

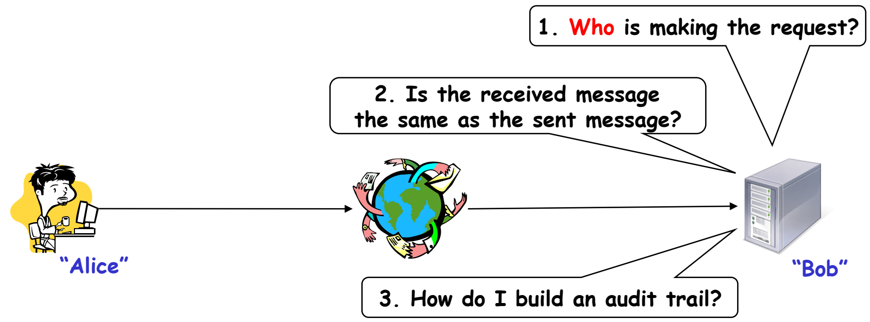
$$g^x = X \bmod m$$

for x, the "discrete logarithm base g of X modulo m".

Authentication

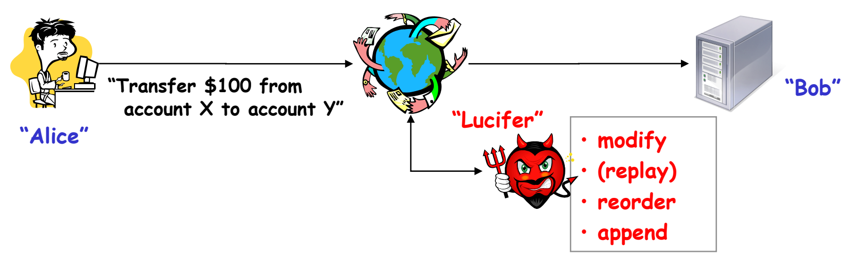
Functionality	Authentication	Authorization	Confidentiality
Primitives	`sign()` `verify()`	Access control lists Capabilities "magic cookies"	`encrypt()` `decrypt()`
Cryptography	cyphers and hashes		

Authentication



1. Authentication
2. Message Integrity
3. Accountability / Non-Repudiation

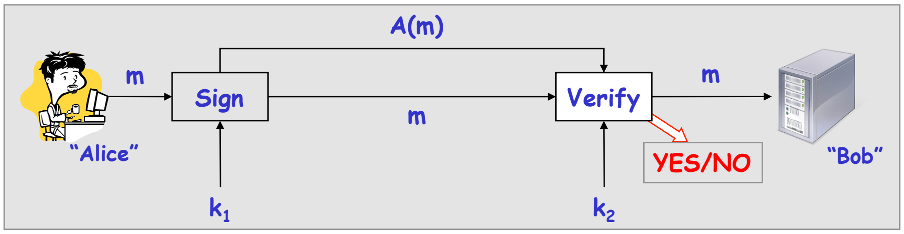
Message Integrity



- Message Integrity can be guaranteed through **Error-Detection Code**. (e.g. cryptographic hash)

Message Integrity ≠ Authenticity ≠ Confidentiality

Authentication: Model



- Symmetric Encryption ($k_1 = k_2$):
  - A(m) is **"message authenticator"**
- Asymmetric Encryption ($k_1 \mathrel{!=} k_2$):
  - A(m) is **"signature"**
  - Example: **A(m) = {Hash(m)}^{kApriv}**
  - **Cryptographically secure hash**:
    - Prob(Hash(m) = Hash(m')) is very low ("low collision prob.")
    - SHA1, SHA256, etc.

Authentication: `Sign()` and `Verify()`

- Algorithm components
  - A set *K* of **keys**
  - A set *M* of **messages**
  - A set *A* of **authenticators**
  - A function ***S*** **:** ***K*** **→ (*M*→ *A*)**
    - That is, for each $k \in K$, *S*(*k*) is a function for generating authenticators from messages
    - Both *S* and *S*(*k*) for any *k* should be efficiently computable functions
  - A function ***V*** **:** ***K*** **→ (*M* × *A*→ {true, false})**. That is, for each $k \in K$, *V*(*k*) is a function for verifying authenticators on messages
    - Both *S* and *V*(*k*) for any *k* should be efficiently computable functions

RSA in Practice…

$\{m\}^{kBpub}$: A **encrypts** message with B's **public key**.

$\{\{m\}^{kBpub}\}^{kBpriv}$: B **decrypts** message with B's **private key**.

$\{m\}^{kApriv}$: A **signs** a message with A's **private key**.

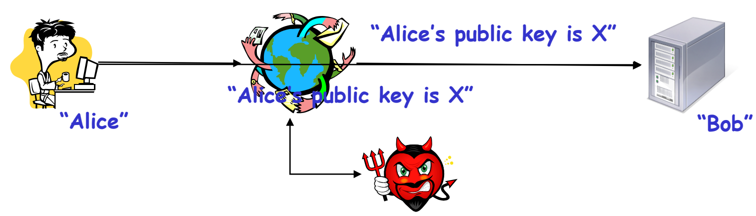
$\{\{m\}^{kApriv}\}^{kApub}$: B **verifies** a message with A's **public key**.

Authentication (Cont.)

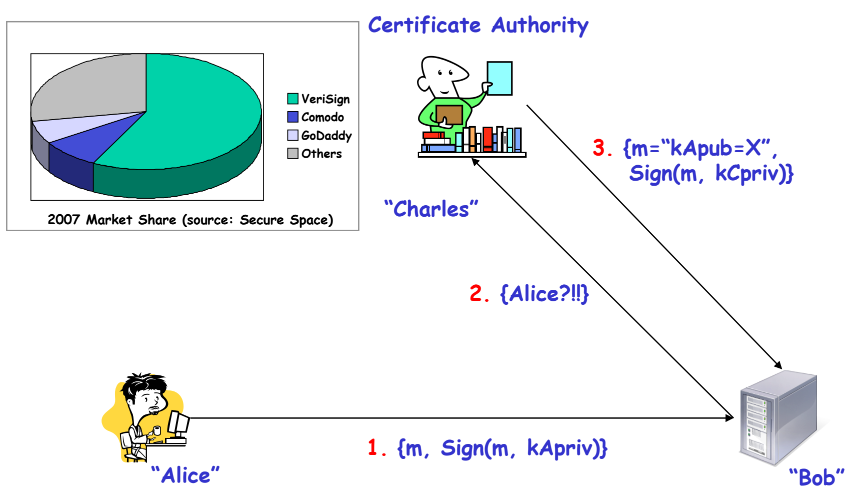
- For a message m, a computer can generate an authenticator $a \in A$ such that $V(k)(m, a)$ = `true` only if it possesses $S(k)$.
- Thus, computer holding $S(k)$ can generate authenticators on messages so that any other computer possessing $V(k)$ can verify them
- Computer not holding $S(k)$ cannot generate authenticators on messages that can be verified using $V(k)$.
- Since authenticators are generally exposed (for example, they are sent on the network with the messages themselves), it must not be feasible to derive $S(k)$ from the authenticators.

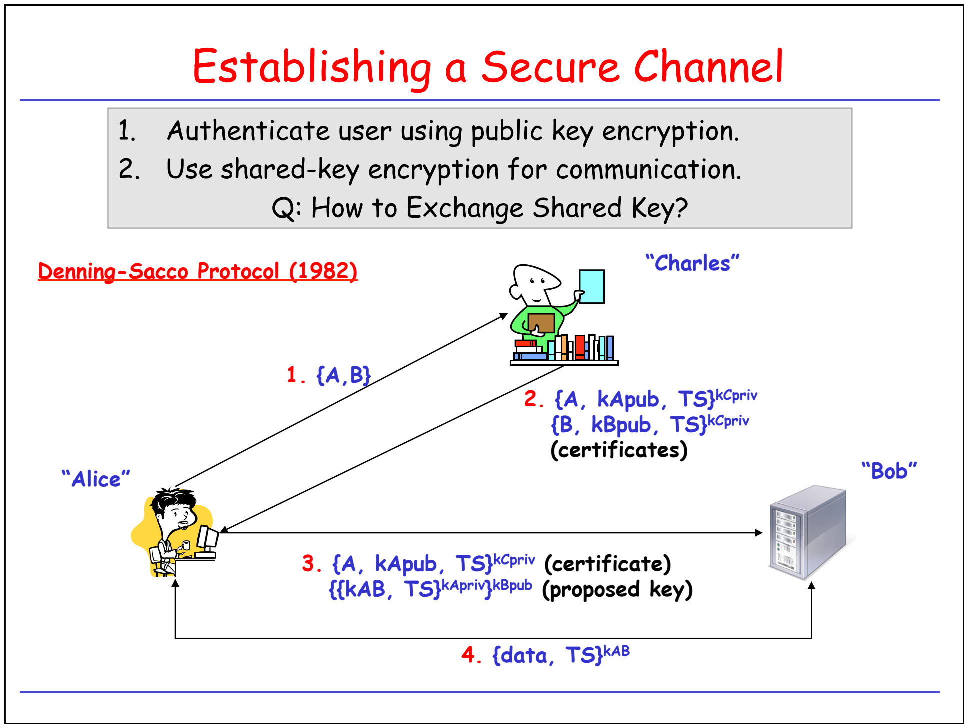
Key Distribution Problem

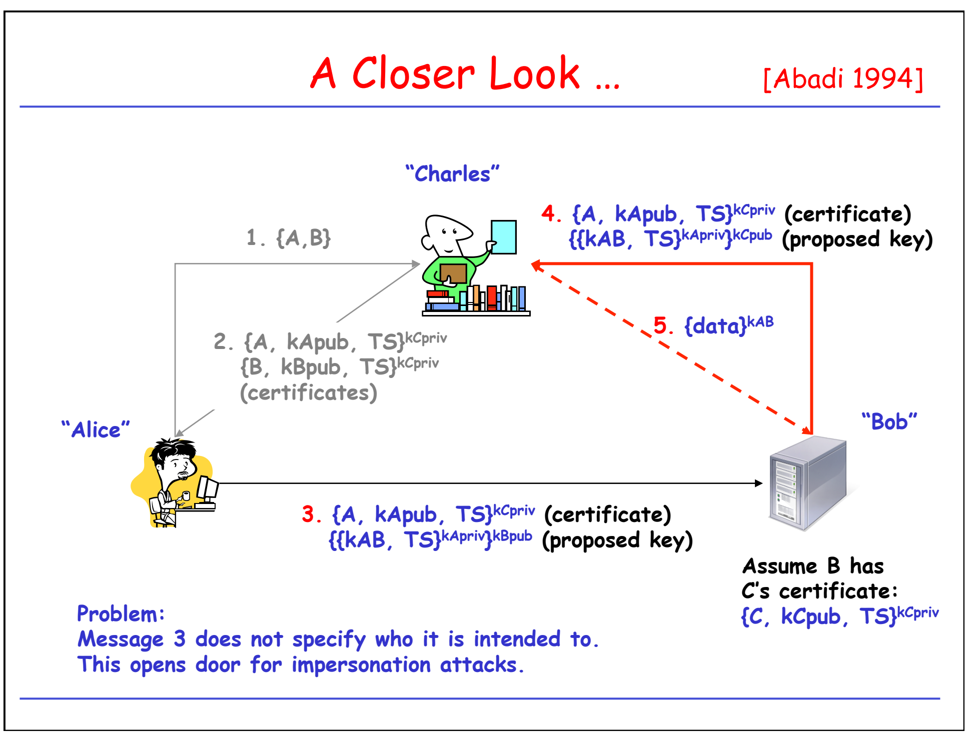
- Q: How does Bob learn **Alice's key**?
  - Q.1: Alice's **public** key?
  - Q.2: Alice's **shared** key?



Key Distribution: Certificates




Establishing a Secure Channel
1. Authenticate user using public key encryption.
2. Use shared-key encryption for communication.
Q: How to Exchange Shared Key?
Denning-Sacco Protocol (1982)
"Charles"
1. {A,B}
2. $\{A, kApub, TS\}^{kCpriv}$
$\{B, kBpub, TS\}^{kCpriv}$
(certificates)
"Alice"
"Bob"
3. $\{A, kApub, TS\}^{kCpriv}$ (certificate)
$\{\{kAB, TS\}^{kApriv}\}^{kBpub}$ (proposed key)
4. $\{data, TS\}^{kAB}$


A Closer Look …
[Abadi 1994]
"Charles"
1. {A,B}
4. $\{A, kApub, TS\}^{kCpriv}$ (certificate)
$\{\{kAB, TS\}^{kApriv}\}^{kCpub}$ (proposed key)
5. $\{data\}^{kAB}$
2. $\{A, kApub, TS\}^{kCpriv}$
$\{B, kBpub, TS\}^{kCpriv}$
(certificates)
"Alice"
"Bob"
3. $\{A, kApub, TS\}^{kCpriv}$ (certificate)
$\{\{kAB, TS\}^{kApriv}\}^{kBpub}$ (proposed key)
Assume B has
C's certificate:
$\{C, kCpub, TS\}^{kCpriv}$
Problem:
Message 3 does not specify who it is intended to.
This opens door for impersonation attacks.

SSL

- Applications: HTTP, IMAP, FTP, etc…
- Client and server negotiate symmetric key that they will use for the length of the data session.
- Two phases in SSL:
  - Phase 1: Connection Establishment
  - Phase 2: Data Transfer

SSL: Connection Establishment

- **Step 1**: Client sends **request** to server, containing
  - SSL version; connection preferences; nonce (i.e. some random number)
- **Step 2**: Server chooses among preferences, and sends **reply**, containing
  - Chosen preferences; nonce; public-key certificate
  - Public-key certificate is a public key that has been digitally signed by a trusted authority.
- **Step 3**: Client can use certification authority's public key to check authenticity of server's public key.
- **Step 4**: Server can request public key of client and verify it similarly (optional)
- **Step 5**: Client chooses random number (**premaster secret**), encrypts it with server's public key, and sends it to server.
- **Step 6**: Both parties compute **session key** (used during data transfer) based on premaster secret and the two nonces.
  - Note: At no point is the session key transferred between client and server.

SSL: Data Transfer

- Messages are fragmented into 16kB portions.
- Each portion is optionally compressed.
- A **Message Authentication Code** (MAC) is appended
  - MAC is a hash derived from plaintext, two nonces, and pre-master secret
- Plaintext and MAC are encrypted using the symmetric key constructed during connection establishment.