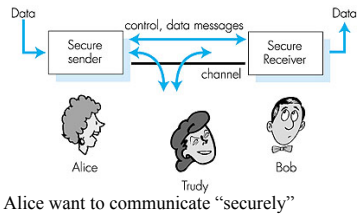
Security

Outline

- Encryption Algorithms
- Authentication Protocols
- Message Integrity Protocols
- Key Distribution
- Firewalls

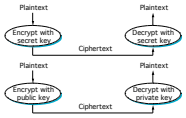
Friends and Enemies

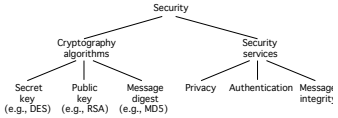


- Bob, Alice want to communicate "securely"
- Trudy, the "intruder" may intercept, delete, add messages

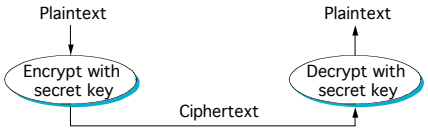
Overview

- Cryptography functions
  - Secret key (e.g., DES)
  - Public key (e.g., RSA)
  - Message digest (e.g., MD5)
- Security services
  - Privacy: preventing unauthorized release of information
  - Authentication: verifying identity of the remote participant
  - Integrity: making sure message has not been altered





Secret Key (DES)



Symmetric Key

Substitution cipher: substituting one thing for another

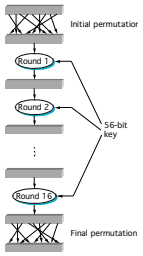
- monoalphabetic cipher: substitute one letter for another

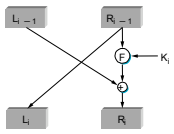
```
plaintext:  abcdefghijklmnopqrstuvwxyz
ciphertext: mnbvcxzasdfghjklpoiuytrewq
```

E.g.:

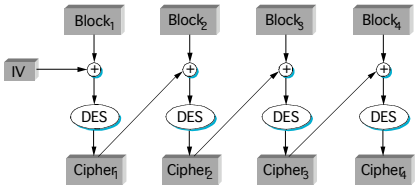
```
Plaintext: bob. i love you. alice
ciphertext: nkn. s gktc wky. mgsbc
```


- 64-bit key (56-bits + 8-bit parity)
- 16 rounds
- Each Round

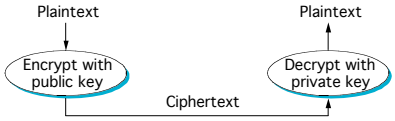




- Repeat for larger messages



Public Key (RSA)



- Encryption & Decryption

$c = m^e \bmod n$

$m = c^d \bmod n$

RSA (cont)

- Choose two large prime numbers p and q (each 256 bits)
- Multiply p and q together to get n
- Choose the encryption key e, such that e and $(p - 1) \times (q - 1)$ are relatively prime.
- Two numbers are relatively prime if they have no common factor greater than one
- Compute decryption key d such that

  $d*e = 1 \bmod ((p - 1) \times (q - 1))$
- Construct public key as (e, n)
- Construct private key as (d, n)
- Discard (do not disclose) original primes p and q

RSA Example

Bob 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

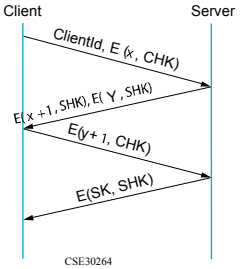
	c	c^d	$m = c^d \bmod n$	letter
decrypt:	17	481968572106750915091411825223072000	12	l

Message Digest

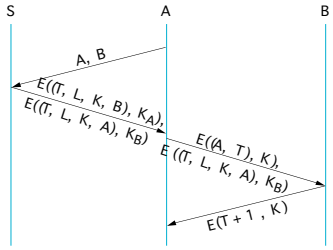
- Cryptographic checksum
  - just as a regular checksum protects the receiver from accidental changes to the message, a cryptographic checksum protects the receiver from malicious changes to the message.
- One-way function
  - given a cryptographic checksum for a message, it is virtually impossible to figure out what message produced that checksum; it is not computationally feasible to find two messages that hash to the same cryptographic checksum.
- Relevance
  - if you are given a checksum for a message and you are able to compute exactly the same checksum for that message, then it is highly likely this message produced the checksum you were given.

Authentication Protocols

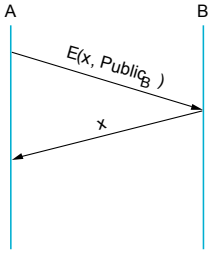
- Three-way handshake



- Trusted third party (Kerberos)



- Public key authentication



Message Integrity Protocols

- Digital signature using RSA
  - special case of a message integrity where the code can only have been generated by one participant
  - compute signature with private key and verify with public key
- Keyed MD5
  - sender: m + MD5(m + k) + E(E(k,*rcv_public*), *snd_private*)
  - receiver
    - recovers random key using the sender's public key
    - applies MD5 to the concatenation of this random key message
- MD5 with RSA signature
  - sender: m + E(MD5(m), *private*)
  - receiver
    - decrypts signature with sender's public key
    - compares result with MD5 checksum sent with message

Public Key Distribution

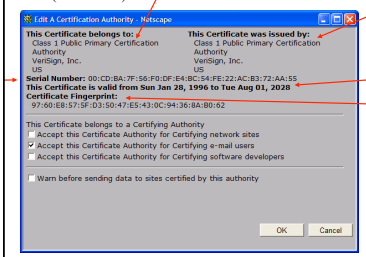
- Certificate
  - special type of digitally signed document:
    "I certify that the public key in this document belongs to the entity named in this document, signed X."
  - the name of the entity being certified
  - the public key of the entity
  - the name of the certification authority
  - a digital signature
- Certification Authority (CA)
  - administrative entity that issues certificates
  - useful only to someone that already holds the CA's public key.

Key Distribution (cont)

- Chain of Trust
  - if *X* certifies that a certain public key belongs to *Y*, and *Y* certifies that another public key belongs to *Z*, then there exists a chain of certificates from *X* to *Z*
  - someone that wants to verify *Z*'s public key has to know *X*'s public key and follow the chain
- Certificate Revocation List

Certificate

- Serial number (unique to issuer)
- info about certificate owner, including algorithm and key value itself (not shown)



- info about certificate issuer
- valid dates
- digital signature by issuer

Pretty Good Privacy (PGP)

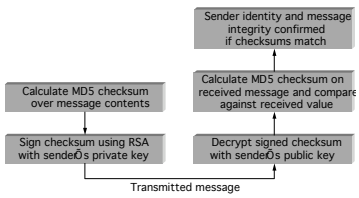
- PGP designed by Phillip Zimmerman for electronic mail
- Uses three known techniques:
  - IDEA for encrypting email message
    - International Data Exchange Algorithm
    - block cipher with 64-bit blocks
    - similar in concept but different in details from DES
    - uses 128-bit keys
    - patented, but free for non-commercial use

Pretty Good Privacy (PGP)

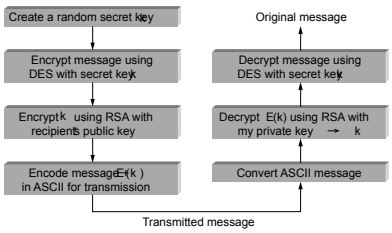
  - RSA public key encryption
    - permits keys up to 2,047 bits in length
  - Digital signatures use MD5 or SHA-1
- PGP generates a random 128-bit symmetric key, used by IDEA for each email message
- PGP generates its own public/private key pairs
- Keys are stored locally using a hashed pass phrase
- PGP does not use conventional certificates (too expensive)
- Instead,
  - users generate and distribute their own public keys
  - sign each other's public keys
  - save trusted public keys on *public-key ring*
  - users build a web of trust
  - users determine how much to trust

Pretty Good Privacy (PGP)

- Encryption and authentication for email.



PGP (cont)



GPG: Gnu Privacy Guard

- GPG is primarily meant for securing email:
  - can also use it to encrypt/decrypt/sign any message...
  - free alternative to PGP
  - http://www.gpg.org
- Allows you to:
  - create public/private key pairs
  - encrypt/decrypt messages
  - sign/check messages

GPG

- Create your own public/private key pair:
  gpg --gen-keys
  - you will be asked several questions: which kind of key you want, which size, how long it will stay valid, username, email address, etc. (keep default values)
  - then it will ask you for a passphrase:
    - type any passphrase (e.g., "I like rabbits very much")
    - your private key will be encrypted with the passphrase before storing on disk
    - GPG will ask you for your passphrase each time it needs to access your private key

Distributing Your Public Key

- You can give it to anyone:
  - people who may want to communicate with you using GPG
  - never hand out your private key
- What's my public key?
  gpg --export --armor <Your_Name>
- You can add somebody else's public key to your keyring:
  gpg --import <filename>
- You can list the keys in your keyring:
  gpg --list-keys

Encryption/Decryption with GPG

- Encryption:
  - must specify the destination of your message, message will be encrypted with the recipient's public key (key must be in key ring)

  gpg --armor --encrypt -r <recipient> < <file_to_encrypt>
- Decryption:
  - GPG will use your private key to decrypt a received file, you will be asked for your passphrase

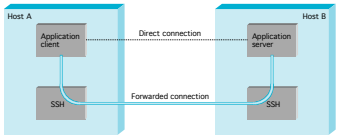
  gpg --decrypt < <file_to_decrypt>

Signatures with GPG

- Sign a message (you will be asked for your passphrase):
  gpg --output <output_file> --clearsign <document_to_sign>
  - message and signature will be written to output_file
- Check a signature:
  gpg --verify <document_to_check>
- You can encrypt and sign a document:
  gpg --armor --sign --encrypt -r cpoellab@cse.nd.edu < message
- To decrypt and check:
  gpg --decrypt <encrypted_and_signed_message>

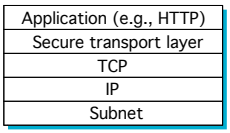
Secure Shell (SSH)

- Remote login service (replaces telnet and rlogin).
- Provides authentication, integrity, and confidentiality.
- SSH version 2: SSH-TRANS, SSH-AUTH, SSH-CONN.

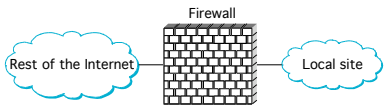


Transport Layer Security (TLS)

- Secure Socket Layer (SSL).
- Secure HTTP (HTTPS).
- Handshake protocol and record protocol.



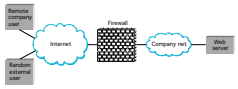
Firewalls



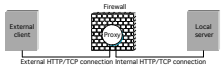
- Filter-Based Solution
  - example
    (192.12.13.14, 1234, 128.7.6.5, 80)
    (*,*, 128.7.6.5, 80)
  - default: forward or not forward?
  - how dynamic?

Proxy-Based Firewalls

- Problem: complex policy
- Example: web server



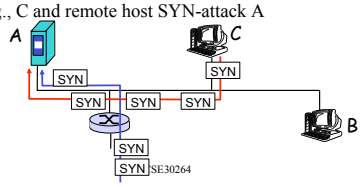
- Solution: proxy



- Design: transparent vs. classical
- Limitations: attacks from within

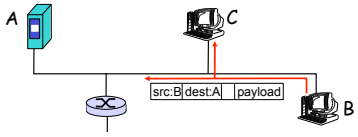
Denial of Service

- flood of maliciously generated packets "swamp" receiver
- Distributed DOS (DDOS): multiple coordinated sources swamp receiver
- e.g., C and remote host SYN-attack A



Packet Sniffing

- broadcast media
- promiscuous NIC reads all packets passing by
- can read all unencrypted data (e.g. passwords)
- e.g.: C sniffs B's packets



IP Spoofing

- can generate "raw" IP packets directly from application, putting any value into IP source address field
- receiver can't tell if source is spoofed
- e.g.: C pretends to be B