

Network Security: Theory & Practice

- **Two different “kinds” of security.**
 - Theory: Are you using cryptography to make security guarantees?
 - Engineering: Is your software designed and implemented in way that prevents abuse?

What really works?

- **Avoid exposure: unplug that Internet connection.**
- **Audit: balance your check-book.**
- **Assume you will lose: be able to tolerate losses (insurance).**
- **Legal or social mechanisms**

A “simple” example

- On-line ordering system
- Me, Internet, Amazon
- I type my credit card number
- Goal: I get the book I wanted, Amazon gets \$20, no one else gets anything.

Potential problems

- What if a bad person is tapping an Internet link?
- What if a bad person has broken into a router?
- How do I know I'm talking to Amazon?
- How does Amazon know it's my credit card?
- What if a bad employee copies my card number?
- What if someone steals the book from my mailbox?
- What if someone has broken into my computer?
- Where did my browser come from? Or my OS?
- What if my display or keyboard radiates?

But this is a networking course

- **We'll discuss only two aspects.**
 - Cryptography, since it's well understood.
 - System structure, since that can limit damage.
- **Today: Cryptographic primitives**
 - Privacy / confidentiality
 - Integrity / authenticity

What crypto primitives are available?

- There are many, we will talk about five.
- Our goal is to understand protocol designs.
 - not to learn to build crypto primitives
 - not to understand why they're secure.
- Even using crypto primitives correctly is very hard.
- So we just want to know what properties we can assume.

Keeping communications secret

- Encryption guarantees secrecy
- Symmetric encryption
 - Encryption algorithm consists of two functions E and D
 - To communicate secretly, parties share secret key K
 - Given message M , $E(K, M) \rightarrow C$, $D(K, C) \rightarrow M$
 - M is **plaintext**, C is **ciphertext**
 - Attacker cannot derive M from C without K

Symmetric key encryption

- **One time pad algorithm**
 - Share a completely random string P
 - Encrypt M by XORing with P : $C \leftarrow M \oplus P$
 - Decrypt by XORing again: $M \leftarrow C \oplus P$
- **Stream ciphers – pseudo-random pad**
 - Generate pseudo-random stream of bits from short key
 - Encrypt/decrypt by XORing as with one-time pad
- **Most common algorithm type: Block cipher**
 - Operates on fixed-size blocks (e.g., 64 or 128 bits)
 - Maps plaintext blocks to same size ciphertext blocks

Example stream cipher (RC4)

- **Initialization:**

- $S[0 \dots 255] \leftarrow$ permutation $\langle 0, \dots, 255, \rangle$
(based on key—specifics omitted)
- $i \leftarrow 0; j \leftarrow 0$

- **Generating pseudo-random bytes:**

```
 $i \leftarrow (i + 1) \bmod 256;$   
 $j \leftarrow (j + S[i]) \bmod 256;$   
swap  $S[i] \leftrightarrow S[j];$   
 $t \leftarrow (S[i] + S[j]) \bmod 256;$   
return  $S[t];$ 
```

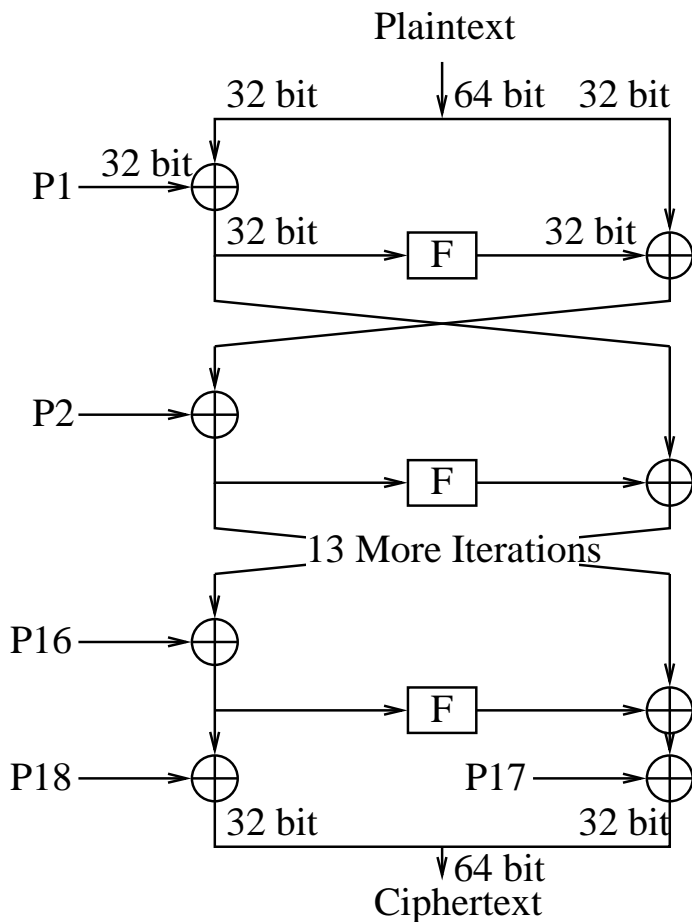
Example use of stream cipher

- **Pre-arrange to share secret s with web vendor**
- **Exchange payment information as follows**
 - Send: $\text{Encrypt}(s, \text{"Visa card \#3273..."})$
 - Receive: $\text{Encrypt}(s, \text{"Order confirmed, have a nice day"})$
- **Now an eavesdropper can't figure out your Visa #**

Wrong!

- **Let's say an attacker has the following:**
 - $c_1 = \text{Encrypt}(s, \text{"Visa card \#3273..."})$
 - $c_2 = \text{Encrypt}(s, \text{"Order confirmed, have a nice day"})$
- **Now compute:**
 - $m \leftarrow c_1 \oplus c_2 \oplus \text{"Order confirmed, have a nice day"}$
- **Lesson: Never re-use keys with a stream cipher**
 - Similar lesson applies to one-time pads
(That's why they're called **one-time** pads.)

Example block cipher (blowfish)



- Derive F and 18 subkeys from Key— $P_1 \dots P_{18}$
- Divide plaintext block into two halves, L_0 and R_0
- $R_i = L_{i-1} \oplus P_i$
 $L_i = R_{i-1} \oplus F(R_i)$
- $R_{17} = L_{16} \oplus P_{17}$
 $L_{17} = R_{16} \oplus P_{18}$
- Output $L_{17}R_{17}$.

(Note: This is just to give an idea; it's not a complete description)

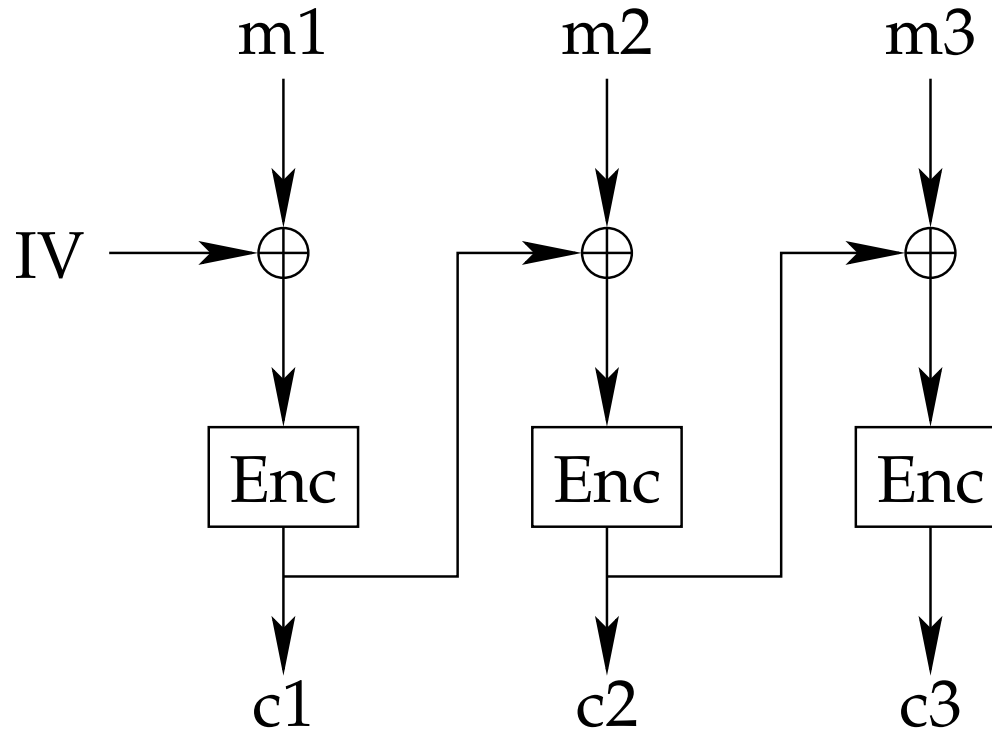
Using a block cipher

- **In practice, message may be more than one block**
- **Encrypt with ECB (electronic code book) mode:**
 - Split plaintext into blocks, and encrypt separately
 - Attacker can't decrypt any of the blocks
 - Message is secure
- **Distributing a secret key can be expensive**
 - Want to maximize the use of a shared secret key
 - Can encrypt multiple messages, since each is secure
 - This is okay, since it's not a stream cipher

Wrong!

- **Attacker will learn of repeated plaintext blocks**
 - If transmitting sparse file, will know where non-zero regions lie
- **Example: Intercepting military instructions**
 - Most days, send encryption of “nothing to report.”
 - On eve of battle, send “attack at dawn.”
 - Attacker will know when battle plans are being made
- **Solution: Cipher-block chaining**
 - Ensures repeated blocks are not encrypted the same

Cipher-block chaining



Given a shared key, can you transmit files securely over the Internet if you encrypt them in CBC mode?

Problem: Integrity

- **Attacker can tamper with messages**
 - *E.g.*, corrupt a block to flip a bit in next
- **Encryption does not guarantee integrity**
 - A system that uses encryption alone (no integrity check) is often incorrectly designed.
 - Exception: Cryptographic storage (to protect disk if stolen)

Message authentication codes

- **Message authentication codes (MACs)**
 - Sender & receiver share secret key K
 - On message m , $\text{MAC}(K, m) \rightarrow v$
 - Attacker cannot produce valid $\langle m, v \rangle$ without K
- **To send message securely, append MAC**
 - Send $\{m, \text{MAC}(K, m)\}$, where m could be ciphertext, $E(K', M)$
 - Receiver of $\{m, v\}$ checks $v \stackrel{?}{=} \text{MAC}(K, m)$
- **Careful of Replay – don't believe previous $\{m, v\}$**

Cryptographic hashes

- **Hash arbitrary-length input to fixed-size output**
 - Typical output size 128 or 160 bits
 - Cheap to compute on large input (faster than network)
- **Collision-resistant: Computationally infeasible to find $x \neq y, H(x) = H(y)$**
 - Many such collisions exist
 - No one has been able to find one, even after analyzing the algorithm
- **SHA-1, MD5 in common use, but are now suspect (use SHA-256).**

Applications of cryptographic hashes

- **Small hash uniquely specifies large data**
 - Hash a file, remember the hash value
 - Recompute hash later, if same value no tampering
 - Hashes often published for software distribution
- **$\text{HMAC}(K, m) = H(K \oplus \text{opad} || H(K \oplus \text{ipad} || m))$**
 - H is a cryptographic hash like SHA-1
 - ipad is 0x36 repeated 64 times, opad 0x5c repeated 64 times

Order of Encryption and MACs

- **Should you Encrypt then MAC, or vice versa?**
- **MACing encrypted data is always secure**
- **Encrypting Data+MAC may not be secure!**
 - Consider the following secure, but stupid encryption alg
 - Transform $m \rightarrow m'$ by mapping each bit to two bits:
Map $0 \rightarrow 00$ (always), $1 \rightarrow \{10, 01\}$ (randomly pick one)
 - Now encrypt m' with a stream cipher to produce c
 - Attacker flips two bits of c —if msg rejected, was 0 bit in m

Public key encryption

- **Three algorithms:**
 - *Generate* – $G(1^k) \rightarrow K, K^{-1}$
 - *Encrypt* – $E(K, m) \rightarrow \{m\}_K$
 - *Decrypt* – $D(K^{-1}, \{m\}_K) \rightarrow m$
- **Provides secrecy, like conventional encryption**
 - Can't derive m from $\{m\}_K$ without knowing K^{-1}
- **Encryption key K can be made public**
 - Can't derive K^{-1} from K
 - Everyone can use the same public key to encrypt messages for one recipient.

Digital signatures

- **Three algorithms:**

- *Generate* – $G(1^k) \rightarrow K, K^{-1}$
- *Sign* – $S(K^{-1}, m) \rightarrow \{m\}_{K^{-1}}$
- *Verify* – $V(K, \{m\}_{K^{-1}}, m) \rightarrow \{\text{true}, \text{false}\}$

- **Provides integrity, like a MAC**

- Cannot produce valid $\langle m, \{m\}_{K^{-1}} \rangle$ pair without K^{-1}

- **Many keys support both signing & encryption**

- But Encrypt/Decrypt and Sign/Verify different algorithms!
- Common error: Sign by “encrypting” with private key

Cost of cryptographic operations

Operation	msec
Encrypt	0.18
Decrypt	6.60
Sign	6.71
Verify	0.03

[1,280-bit Rabin-Williams keys on 3 GHz Pentium IV]

- **Cost of public key algorithms significant**
 - Encryption only on small messages (< size of key)
 - Signature cost relatively insensitive to message size
- **In contrast, symmetric algorithms much cheaper**
 - Symmetric can encrypt+MAC faster than 100Mbit/sec LAN

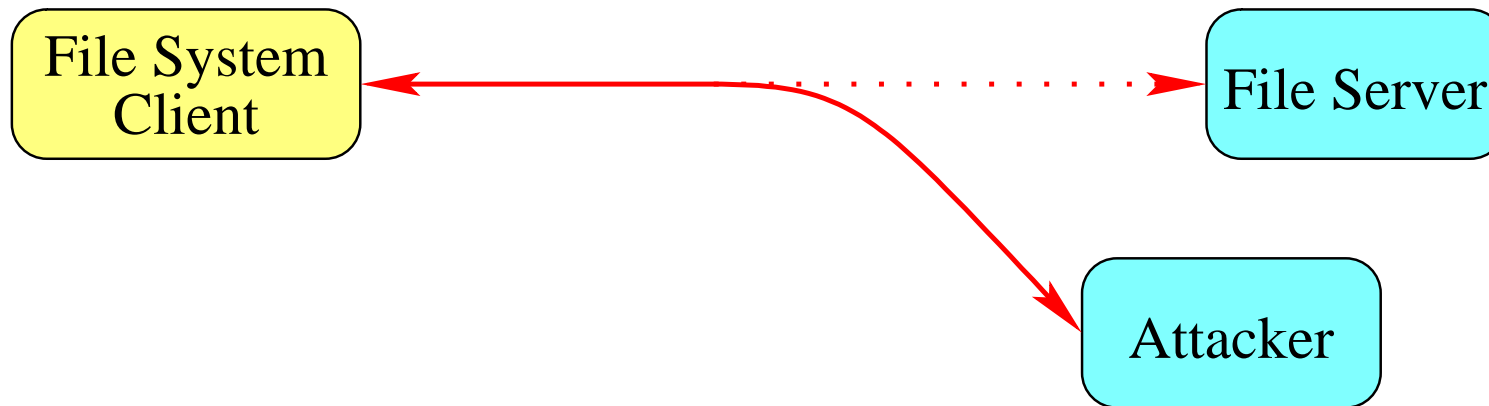
Hybrid schemes

- **Use public key to encrypt symmetric key**
 - Send message symmetrically encrypted: $\{\text{msg}\}_{K_S}, \{K_S\}_{K_P}$
- **Use PK to negotiate secret session key**
 - *E.g.*, Client sends server $\{K_1, K_2, K_3, K_4\}_{K_P}$
 - Client sends server: $\{\{m_1\}_{K_1}, \text{MAC}(K_2, \{m_1\}_{K_1})\}$
 - Server sends client: $\{\{m_2\}_{K_3}, \text{MAC}(K_4, \{m_2\}_{K_3})\}$
- **Often want mutual authentication (client & server)**
 - Or more complex, user(s), client, & server

Server authentication

- **An approach: Use public key cryptography**
 - Give client public key of server
 - Lets client authenticate secure channel to server
- **Problem: Key management problem**
 - How to get server's public key?
 - How to know the key is really server's?

The danger: Attackers impersonating servers



- **File system example:**

- Attacker pretends to be server, gives its own public key
- Attacker substitutes modified data for file
- User writes sensitive file to fake server

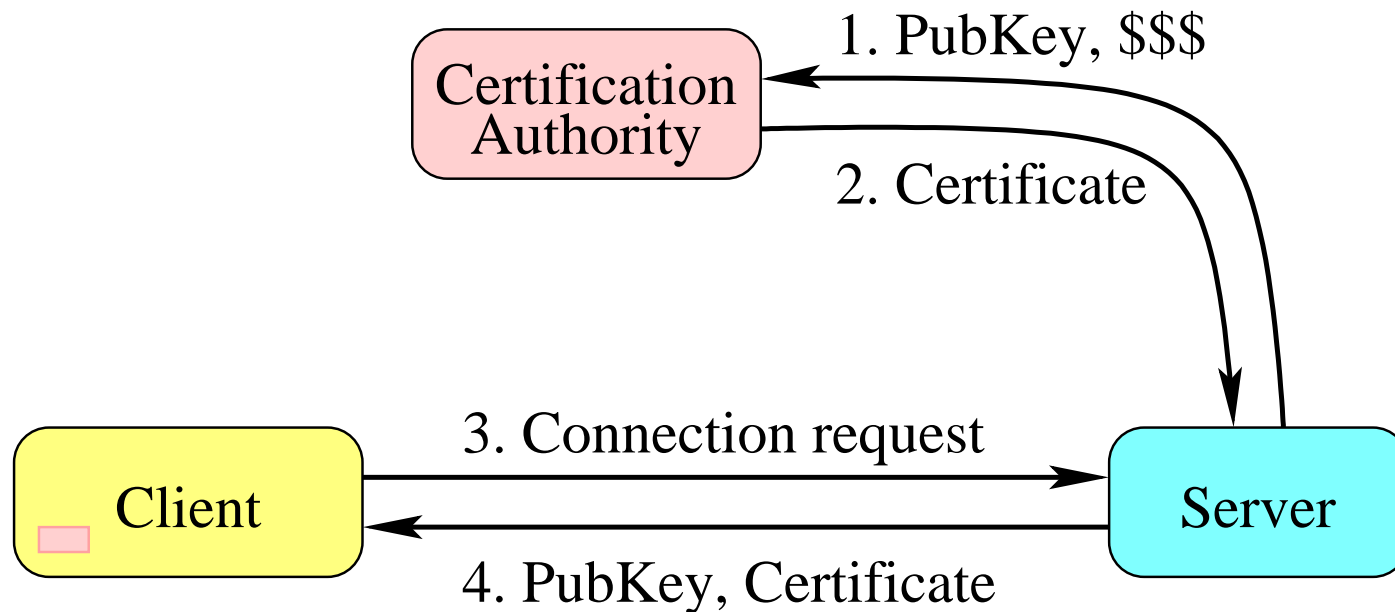
Man in the middle attacks

- **Attacker might not look like server**
 - User would notice if file system didn't contain right files
- **Man in the middle attack foils user:**
 - Attacker emulates server when talking to client
 - Attacker emulates client when talking to server
 - Attacker passes most messages through unmodified
 - Attacker substitutes own public key for client's & server's
 - Attacker records secret data, or tampers to cause damage

Key management

- **Put public keys in the phone book**
 - How do you know you have the real phone book?
 - How is a program supposed to use phone book
www.phonebook.com? (are you talking to real web server)
- **Exchange keys with people in person**
- **“Web of trust” – get keys from friends you trust**

Certification authorities



- **Everybody trusts some certification authority**
- **Everybody knows authority's public key**
 - *E.g., built into web browser*

Coming up

- **Today: HW3 out**
- **This week: TCP checkpoint**
- **Tue: SSL**
- **11 days: TCP Due**