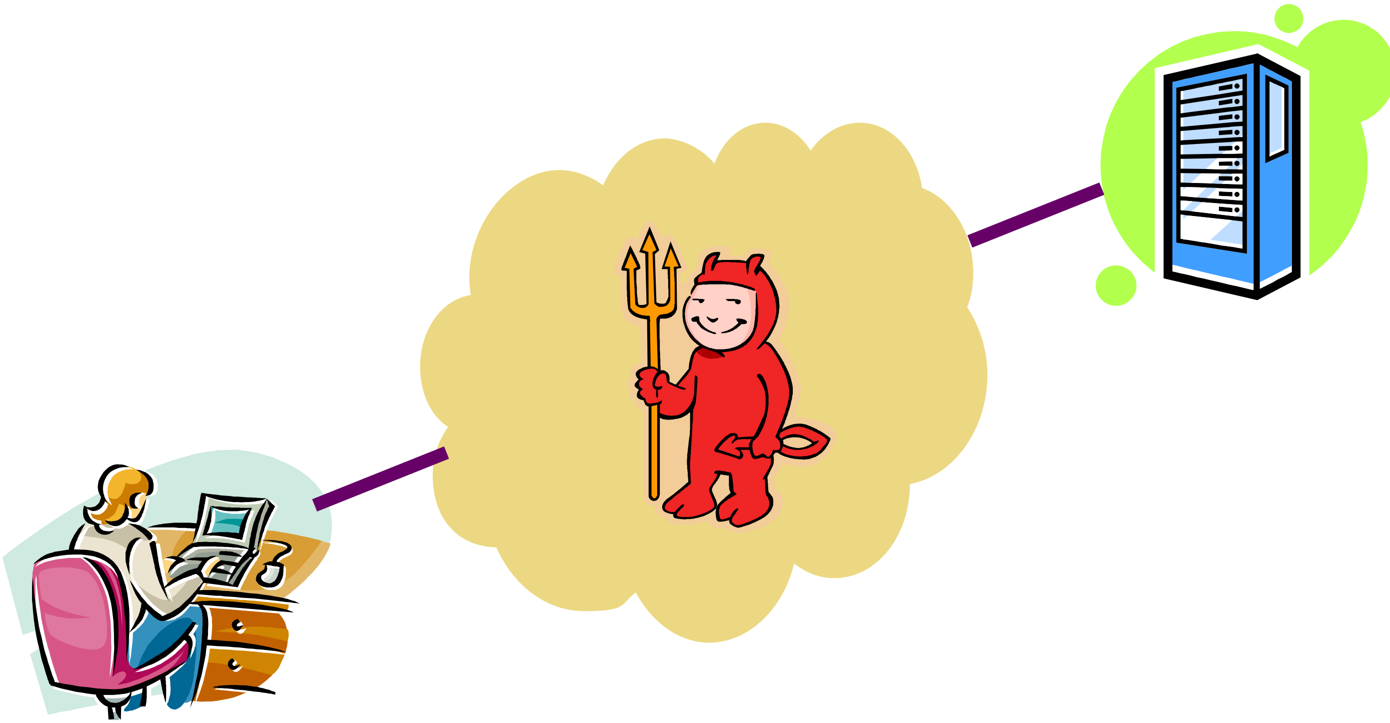
How to use Cryptography

CS155

How *Not* to use Cryptography

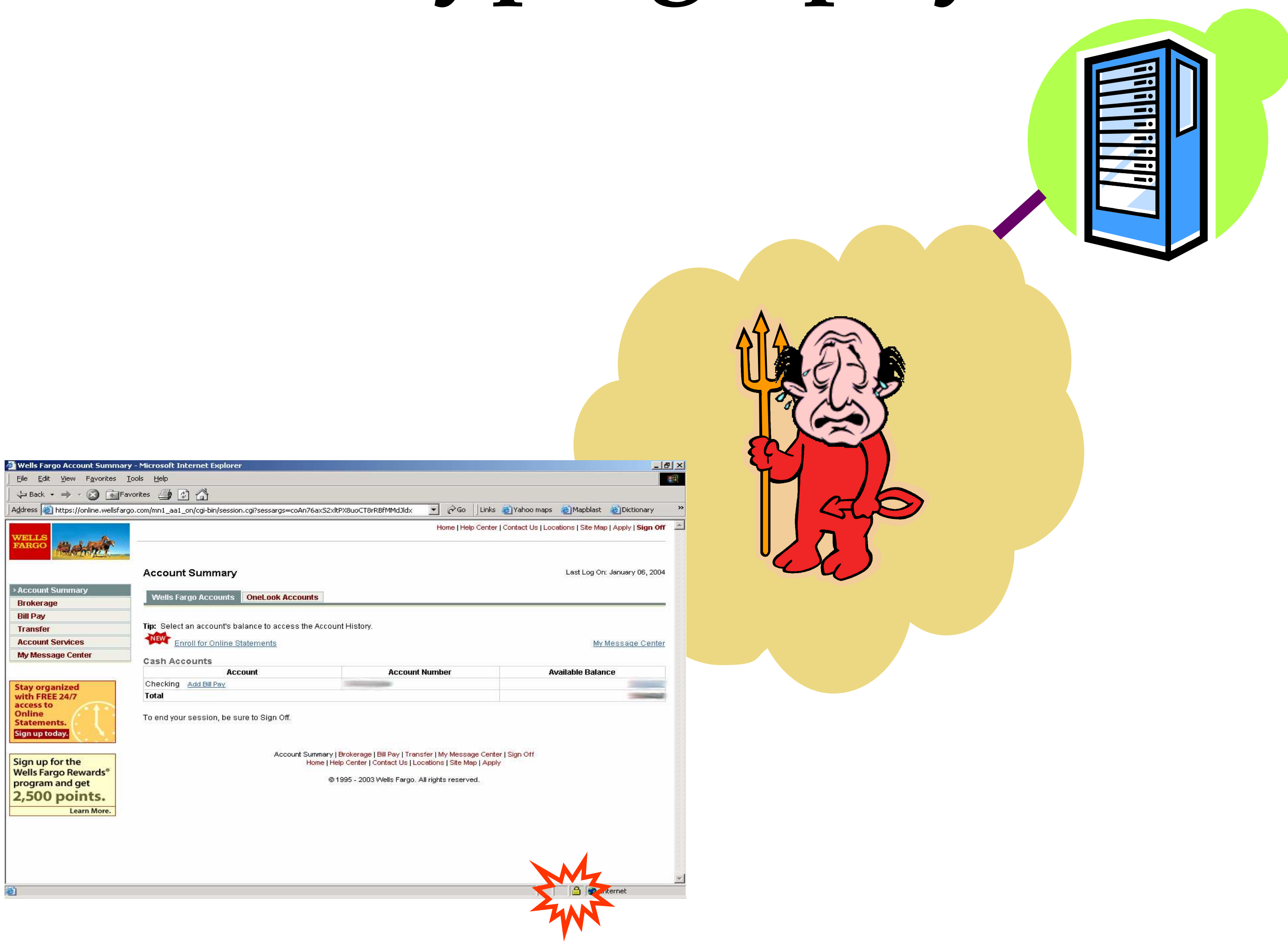
CS155

Motivation: communication security



- **To a first approximation, attackers control network**
  - We will talk about *how* they do this in two weeks
  - But imagine attackers can intercept you packets, tamper with or suppress them, and inject arbitrary packets

Cryptography



- **Still possible to communicate securely**
  - Cryptography is a tool that can often help

[Symmetric] Encryption

- **Encryption keeps communications secret**
- **An encryption algorithm has two functions:** E **and** D
  - To communicate secretly, parties share secret key K
- **Given a message** M**, and a key** K**:**
  - M is known as the *plaintext*
  - $E(K, M) \rightarrow C$ (C known as the *ciphertext*)
  - $D(K, C) \rightarrow M$
  - Attacker cannot efficiently derive M from C without K
- **Note** E **and** D **take same argument** K
  - Thus, also sometimes called *symmetric* encryption

One-time pad

- **Share a completely random key** K
- **Encrypt** M **by XORing with** K**:**

$$E(K, M) = M \oplus K$$

- **Decrypt by XORing again:**

$$D(K, C) = C \oplus K$$

- **Advantage: Information-theoretically secure**
  - Given C but not K, any M of same length equally likely
- **Disadvantage:** K **must be as long as** M
  - Makes distributing K for each message difficult

Idea: Computational security

- **Distribute small K securely (e.g., 128 bits)**
- **Use K to encrypt far larger M (e.g., 1 MByte file)**
- **Given $C = E(K, M)$, may be only one possible M**
  - If M has redundancy
- **But believed computationally intractable to find**
  - E.g., could try every possible K, but 2^{128} keys a lot of work!

Types of encryption

- **Stream ciphers – pseudo-random pad**
  - Generate pseudo-random stream of bits from short key
  - Encrypt/decrypt by XORing as with one-time pad
  - But **NOT** one-time PAD! (People who claim so are frauds!)
- **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
  - Today should use AES; other algorithms: DES, Blowfish, …

Example stream cipher (RC4)

- **Initialization:**
  - $S[0 \ldots 255] \leftarrow$ permutation $\langle 0, \ldots 255, \rangle$ (based on key—specifics omitted)
  - $i \leftarrow 0; j \leftarrow 0$
- **Generating pseudo-random bytes:**

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

RC4 security

- **Goal: be indistinguishable from random sequence**
  - given part of the output stream, it should be intractable to distinguish it from a truly random string
- **Problems**
  - Second byte of RC4 is 0 with twice expected probability [MS01]
  - Bad to use many related keys (see WEP 802.11b) [FMS01]
  - Recommendation: Discard the first 256 bytes of RC4 output [RSA, MS]

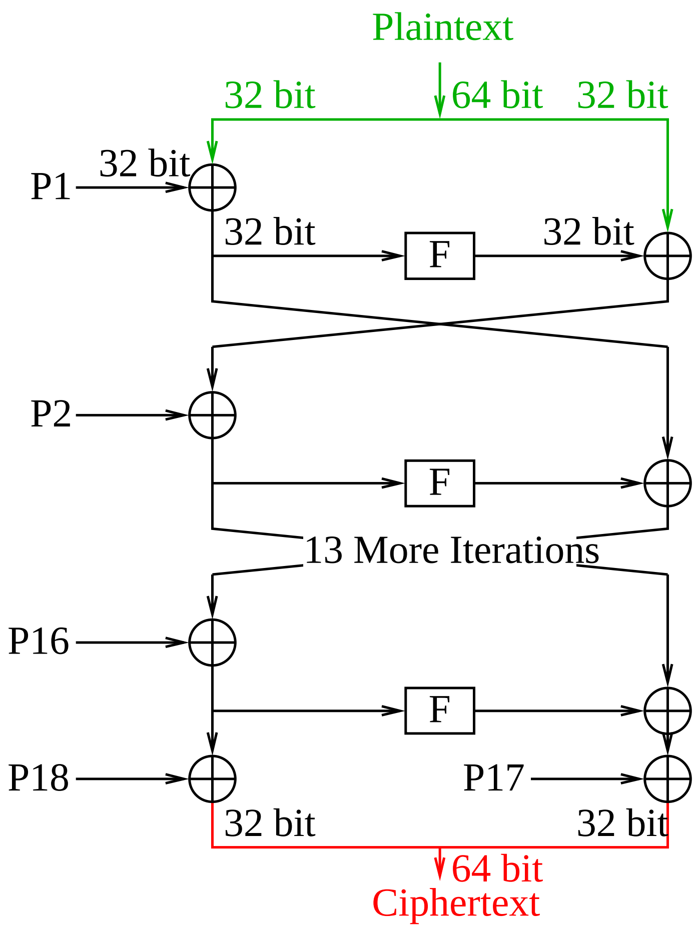
Example use of stream cipher

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

Wrong!

- **Let's say an attacker has the following:**
  - $c_1 = \text{Encrypt}(s, \text{“Visa card \#3273\ldots”})$
  - $c_2 = \text{Encrypt}(s, \text{“Order confirmed, have a nice day”})$
- **Now compute:**
  - $m \leftarrow c_1 \oplus c_2 \oplus$ “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)



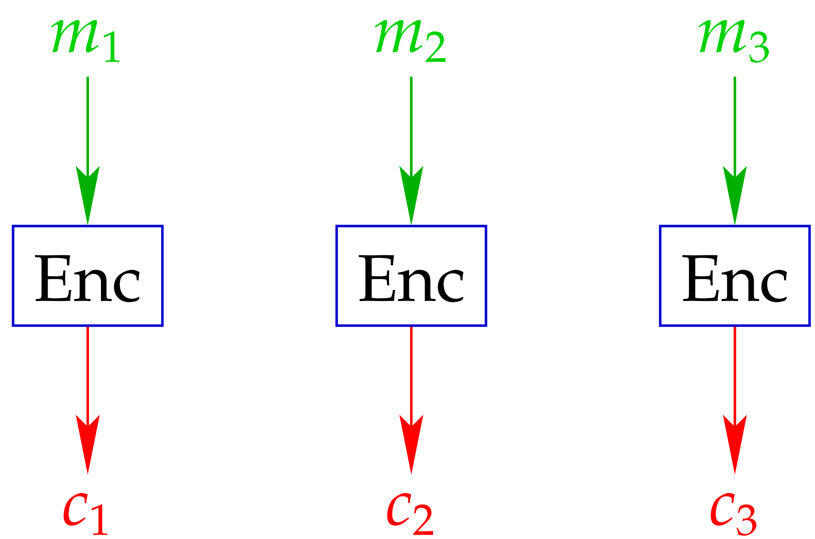
"Feistel network"

- **Derive F and 18 subkeys from Key—$P_1 \ldots P_{18}$**
- **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

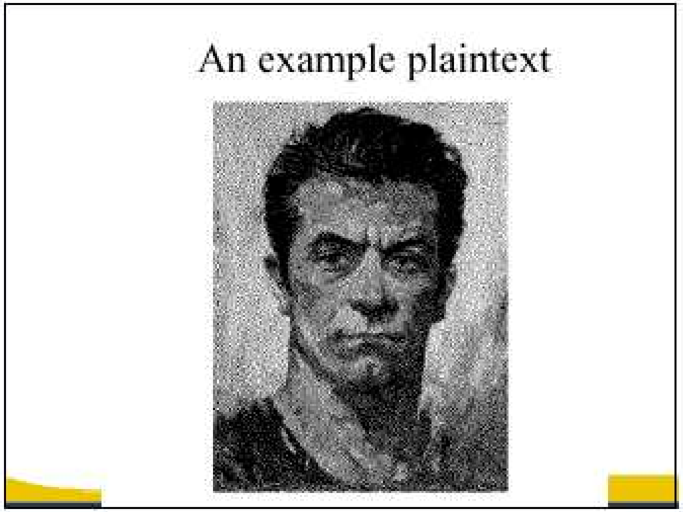


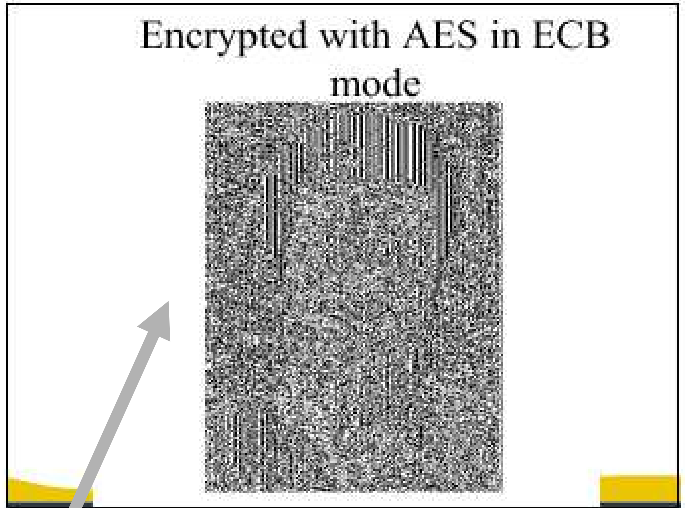
  - Attacker can't decrypt any of the blocks; message secure
- **Note: can re-use keys, unlike stream cipher**
  - Every block encrypted with cipher will be secure

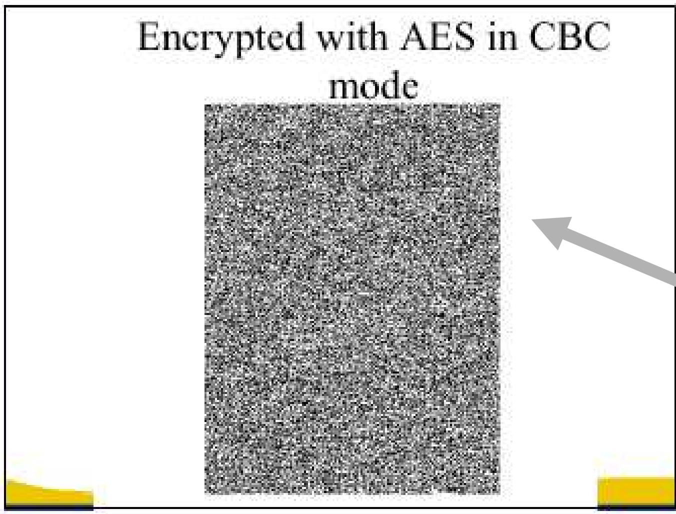
Wrong!

- **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

Another example [Preneel]





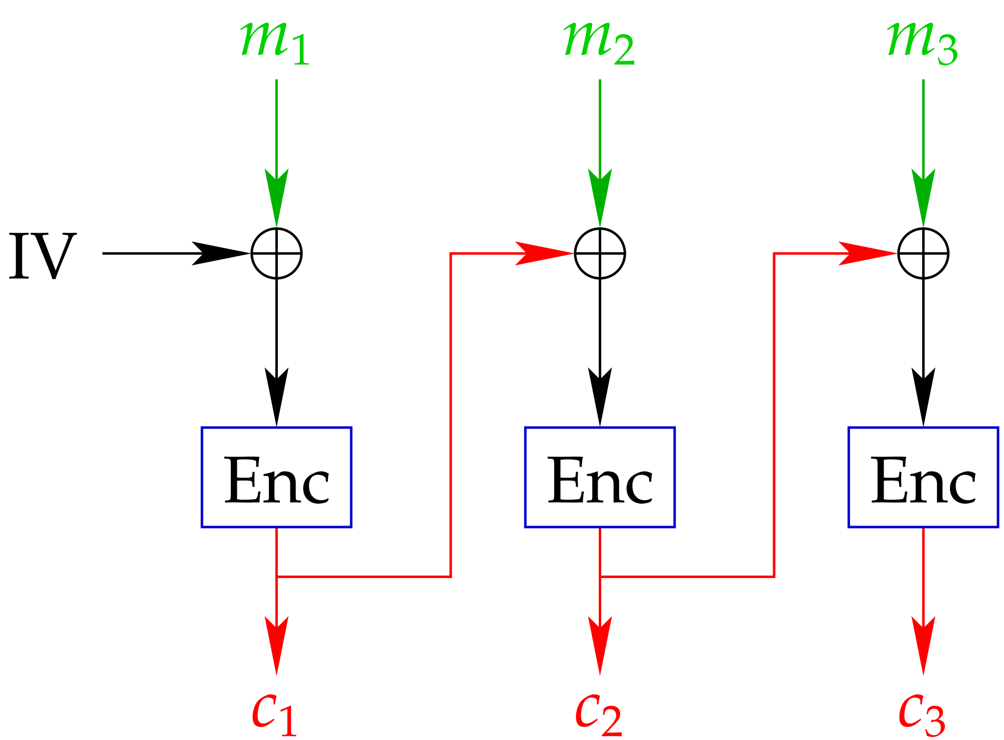


Similar plaintext blocks produce similar ciphertext (see outline of head)

What we want: No apparent pattern

Cipher-block chaining (CBC)

- $c_1 = E(K, m_i \oplus IV)$, $c_i = E(K, m_i \oplus c_{i-1})$
- **Ensures repeated blocks are not encrypted the same**



Encryption modes

- **CBC, ECB are encryption modes, but there are others**
- **Cipher Feedback (CFB) mode:** $c_i = m_i \oplus E(K, c_{i-1})$
  - Useful for messages that are not multiple of block size
- **Output Feedback (OFB) mode:**
  - Repeatedly encrypt IV & use result like stream cipher
- **Counter (CTR) mode:** $c_i = m_i \oplus E(K, i)$
  - Useful if you want to encrypt in parallel
- **Q: Given a shared key, can you transmit files securely over net by just encrypting them in CBC mode?**

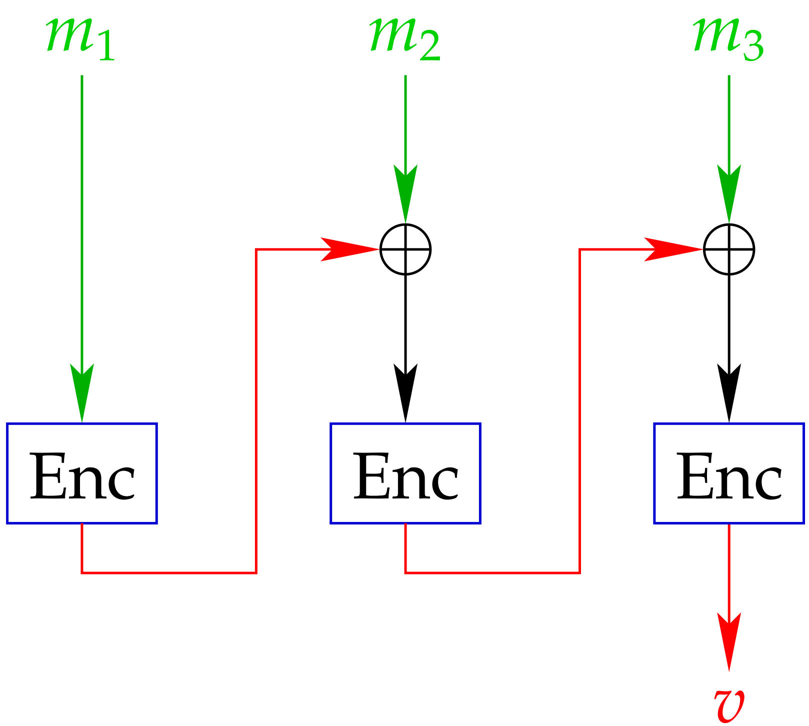
Problem: Integrity

- **Attacker can tamper with messages**
  - E.g., corrupt a block to flip a bit in next
- **What if you delete original file after transfer?**
  - Might have nothing but garbage at recipient
- **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, $\mathrm{MAC}(K,m) \rightarrow v$
  - Intractable to produce valid $\langle m,v \rangle$ without K
- **To send message securely, append MAC**
  - Send $\{m, \mathrm{MAC}(K,m)\}$ (m could be ciphertext, $E(K',M)$)
  - Receiver of $\{m,v\}$ checks $v \stackrel{?}{=} \mathrm{MAC}(K,m)$
- **Careful of Replay – don't believe previous** $\{m,v\}$

Example: CBC MAC



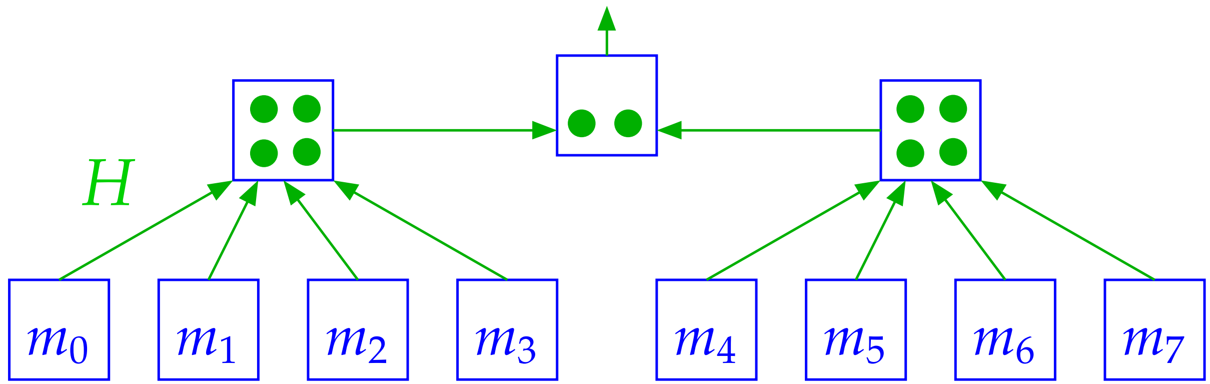
- **Encrypt M in CBC mode, keep only last block**
  - Or re-encrypt last block w. different key to strengthen
- **Do not use CBC MAC as encryption**
  - Must encrypt/MAC in two passes with two keys
  - More efficient single-pass "Authenticated encryption modes" such as OCB exist, but non-obvious; don't roll your own

Cryptographic hashes

- **Hash arbitrary-length input to fixed-size output**
  - Typical output size 160–512 bits
  - Cheap to compute on large input (faster than network)
- **Collision-resistant: Intractable to find** $x \neq y$, $H(x) = H(y)$
  - Of course, many such collisions exist
  - But no one has been able to find one, even after analyzing the alrogithm
- **Most popular hash SHA-1**
  - [Nearly] broken
  - Today should use SHA-256 or SHA-512

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
- **Hash tree [Merkle] lets you verify check small piece of large file/database with log number of nodes**



HMAC

- **Use cryptographic hash to produce MAC**
- $\textbf{HMAC}(K, m) = H(K \oplus \textsf{opad}, H(K \oplus \textsf{ipad}, m))$
  - H is a cryptographic hash such as SHA-1
  - ipad is 0x36 repeated 64 times, opad 0x5c repeated 64 times
- **Note: Don't just use H(K,M) as a MAC**
  - Say you have $\{M, \text{SHA-1}(K, M)\}$, but not K
  - Can produce $\{M', \text{SHA-1}(K, M')\}$ where $M' \neq M$

Order of Encryption and MACs

- **Should you Encrypt then MAC, or vice versa?**

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 $m \to m'$ by mapping each bit to two bits:
    Map $0 \to 00$ (always), $1 \to \{10, 01\}$ (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 randomized algorithms:**
  - *Generate* – $G(1^k) \to K, K^{-1}$
  - *Encrypt* – $E(K, m) \to \{m\}_K$
  - *Decrypt* – $D(K^{-1}, \{m\}_K) \to m$
- **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.

The RSA algorithm

- **Generation:**
  - Pick two primes, p and q, let $N = pq$
  - Pick random e that does not divide $(p-1)(q-1)$
  - Compute d such that $de \equiv 1 \pmod{(p-1)(q-1)}$
  - Public key: N, e, private key N, d
- **If $m \in \mathbf{Z}_N^*$, then $(m^e \bmod N)^d \bmod N = m$.**
- **Fact: For large enough p, q and random m**
  - Given N, e, and $m^e \bmod N$, but not p, q, d
  - No one knows practical algorithm to find m
- **To encrypt a message, just treat bits as number and compute $m^e \bmod N$.**

Wrong!

- **What if message is from a small set (yes/no)?**
- **What if I want to outbid you in secret auction?**
  - I take your encrypted bid c and submit $c\,(101/100)^e \bmod n$.
- **What if there's some protocol in which I can learn other message decryptions?**
  - E.g., people escrow ciphertexts, and get them back under certain circumstances (if an employee is fired or dies)
  - I take your ciphertext $c = m^e \bmod n$, and escrow $c2^e \bmod n$.
  - After I'm fired, my coconspirator gets back $2m$
- **Many people make this mistake, including SSL**
  - SSL didn't return decryptions, bur error messages had some information

Notions of security

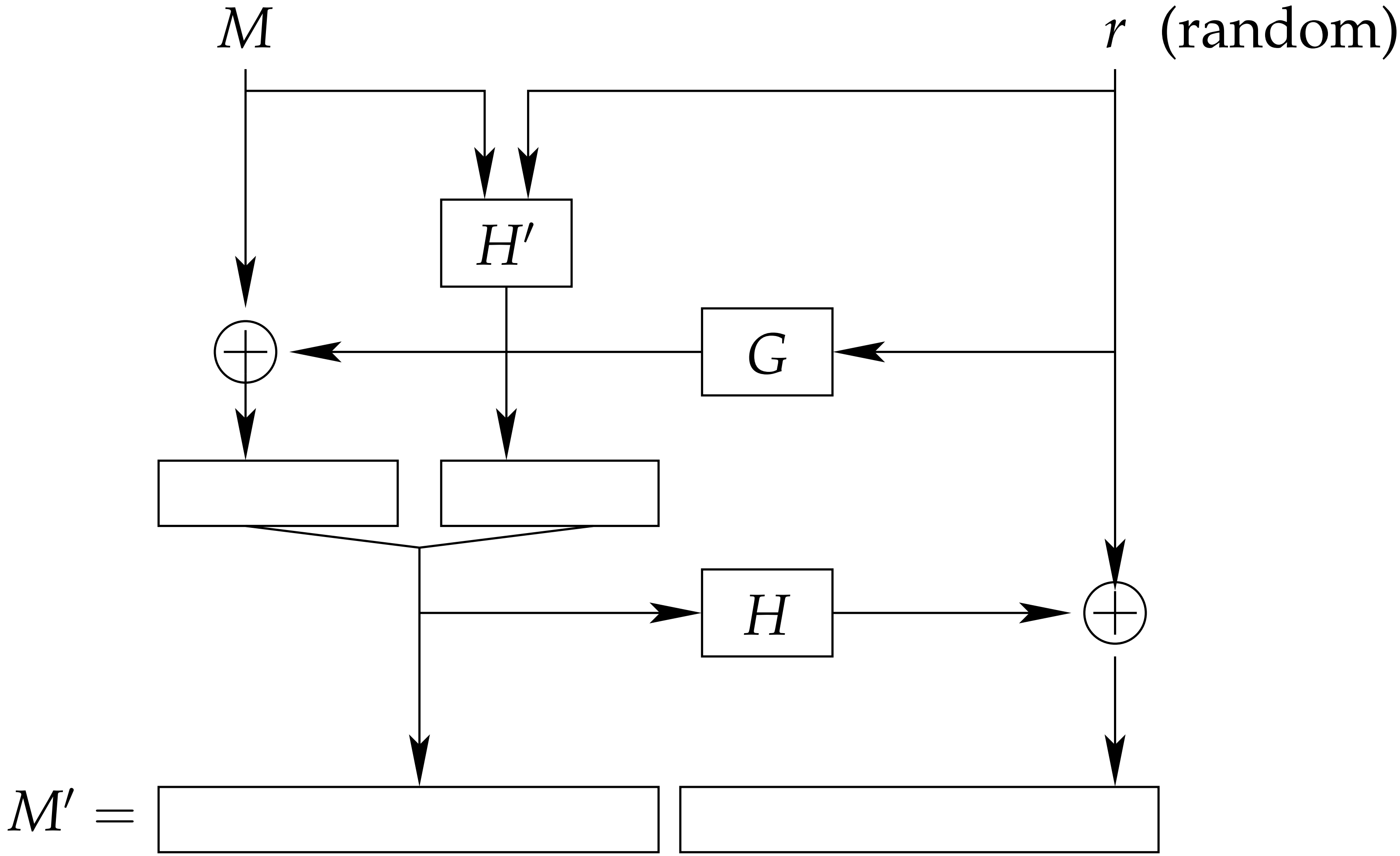
- **How do design systems using RSA?**
  - You don't want to think about interactions between your error messages, modular exponentiation, and lattice theory.
- **A PKS is adaptive chosen ciphertext secure if**
  - No attacker A can win the following game with probability more than $1/2$+negligible:
  - A can first ask for arbitrary messages to be decrypted
  - A then produces two messages, m_0 and m_1
  - The good guy flips a coin $b \leftarrow \{0,1\}$, returns $c = E(K, m_b)$.
  - A can ask for any messages except c to be decrypted
  - A guesses the value of b

Achieving Adaptive CC security

- **Good properties for message → integer mapping**
  - **Randomness**: unique ciphertext even for same message
  - **Redundancy**: make most strings invalid ciphertexts
  - **Entanglement**: partial information about integer should reveal nothing about message
  - **Invertibility**: of course, need to recover message
- **Note last two were achieved by Fiestel network**
- **Can use similar idea to construct a *padding scheme***

Practical solution: OAEP+ [Shoup]

- **Transforms plaintext M into number M' for RSA:**



- **Not provable, but heuristically secure**

Digital signatures

- **Three (randomized) algorithms:**
  - *Generate* – $G(1^k) \rightarrow K, K^{-1}$
  - *Sign* – $S\left(K^{-1}, m\right) \rightarrow \{m\}_{K^{-1}}$
  - *Verify* – $V\left(K, \{m\}_{K^{-1}}, m\right) \rightarrow \{\textbf{true}, \textbf{false}\}$
- **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

Digital signature security

- **Want signatures to be secure for all applications**
  - Analogous to strength of encryption definition
- **Existential unforgeability against chosen message attack $\Longrightarrow$ attacker has negligible chance of winning this game:**
  - Attacker asks you to sign $m_0, m_1, \ldots, m_n$
  - Attacker gets valid s_i after request for m_i
  - Attacker outputs (m', s'), where $m' \notin \{m_i\}$ and $\mathit{Verify}(K, m', s') = \textbf{true}$

Example: ElGamal signatures

- **Key generation:**
  - Chose large prime p, generator g of $\mathbf{Z}_p^*$ (p, g can be global)
  - Select x such that $1 \leq x \leq p-2$, compute $y \leftarrow g^x \bmod p$
  - Public key is (p, g, y), private key is (p, g, x)
- **Signature of m is (r, s), computed as follows:**
  - Chose random k s.t. $1 \leq k \leq p-2$ and $k^{-1} \bmod p-1$ exists
  - Set $r \leftarrow g^k \bmod p$, $s \leftarrow k^{-1}\,(H(m) - xr) \bmod (p-1)$
- **Verification:**
  - Sanity check: $1 \leq r \leq p-1$
  - Verify: $y^r r^s \stackrel{?}{\equiv} g^{H(m)} \pmod p$
  - $y^r r^s = (g^{xr})\left(g^{ks}\right) = g^{xr+ks} = g^{xr+k \cdot k^{-1}(H(m)-xr)} = g^{H(m)}$

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

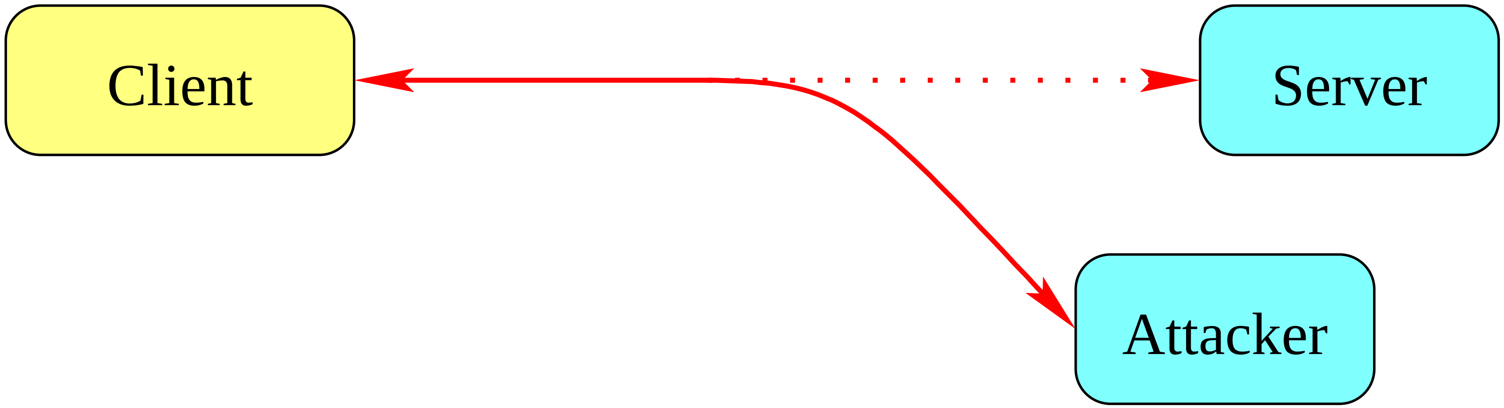
Hybrid schemes

- **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?

Danger: impersonating servers



- **Attacker pretends to be server, gives its own pub key**
- **Client sends sensitive data to fake server**
- **Attacker sends bad data back to client**

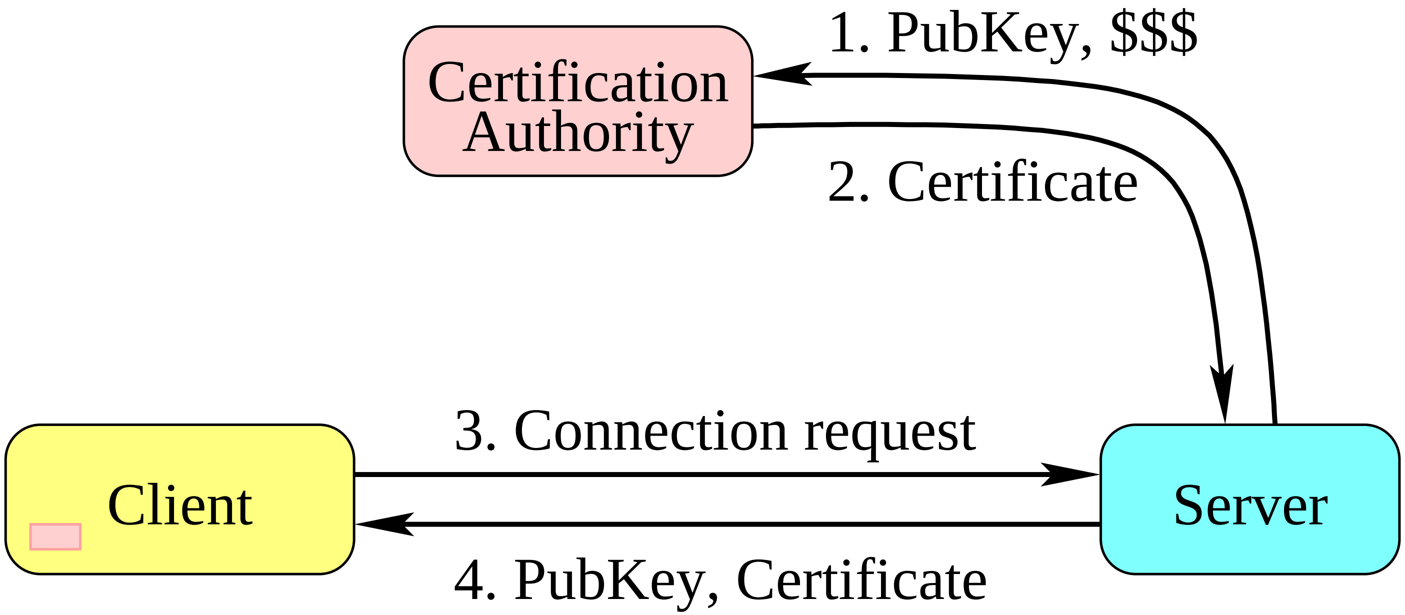
Man in the middle attacks

- **Attacker might not look like server**
  - E.g., user might notice different web site & not send password
- **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

SSL/TLS Overview

- **SSL offers security for HTTP protocol**
  - That's what the padlock means in your web browser

- **Authentication of server to client**
- **Optional authentication of client to server**
  - Incompatibly implemented in different browsers
  - CA infrastructure not in widespread use
- **Confidentiality of communications**
- **Integrity protection of communications**

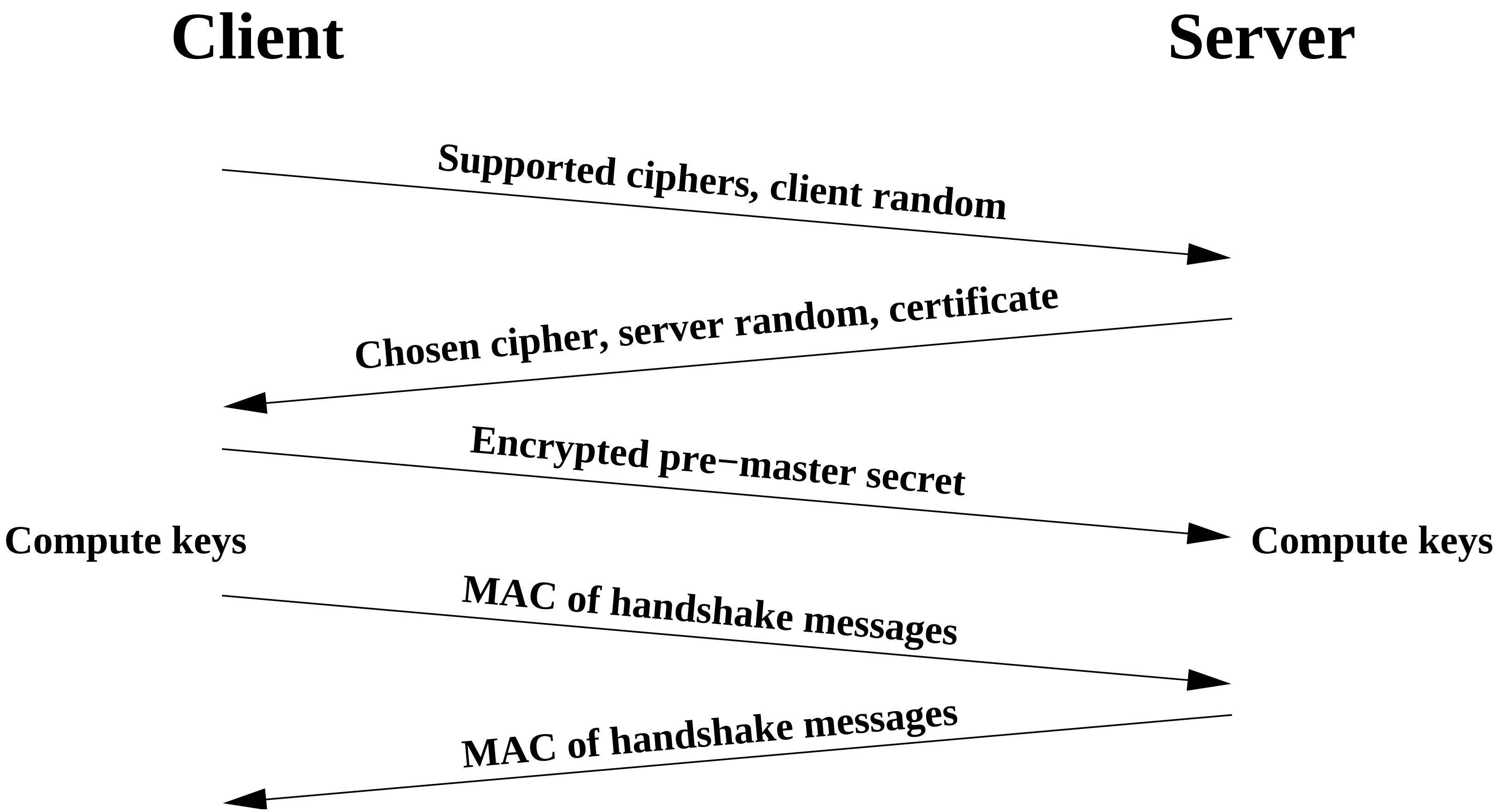
Purpose in more detail

- **Authentication based on certification authorities (CAs)**
  - Certifies who belongs to a public key (domain name and real name of company)
  - Example: Verisign
- **What SSL Does Not Address**
  - Privacy
  - Traffic analysis
  - Trust management

Ciphersuites: Negotiating ciphers

- **Server authentication algorithm (RSA, DSS)**
- **Key exchange algorithm (RSA, DHE)**
- **Symmetric cipher for confidentiality (RC4, DES)**
- **MAC (HMAC-MD5, HMAC-SHA)**

Overview of SSL Handshake



From “SSL and TLS” by Eric Rescorla

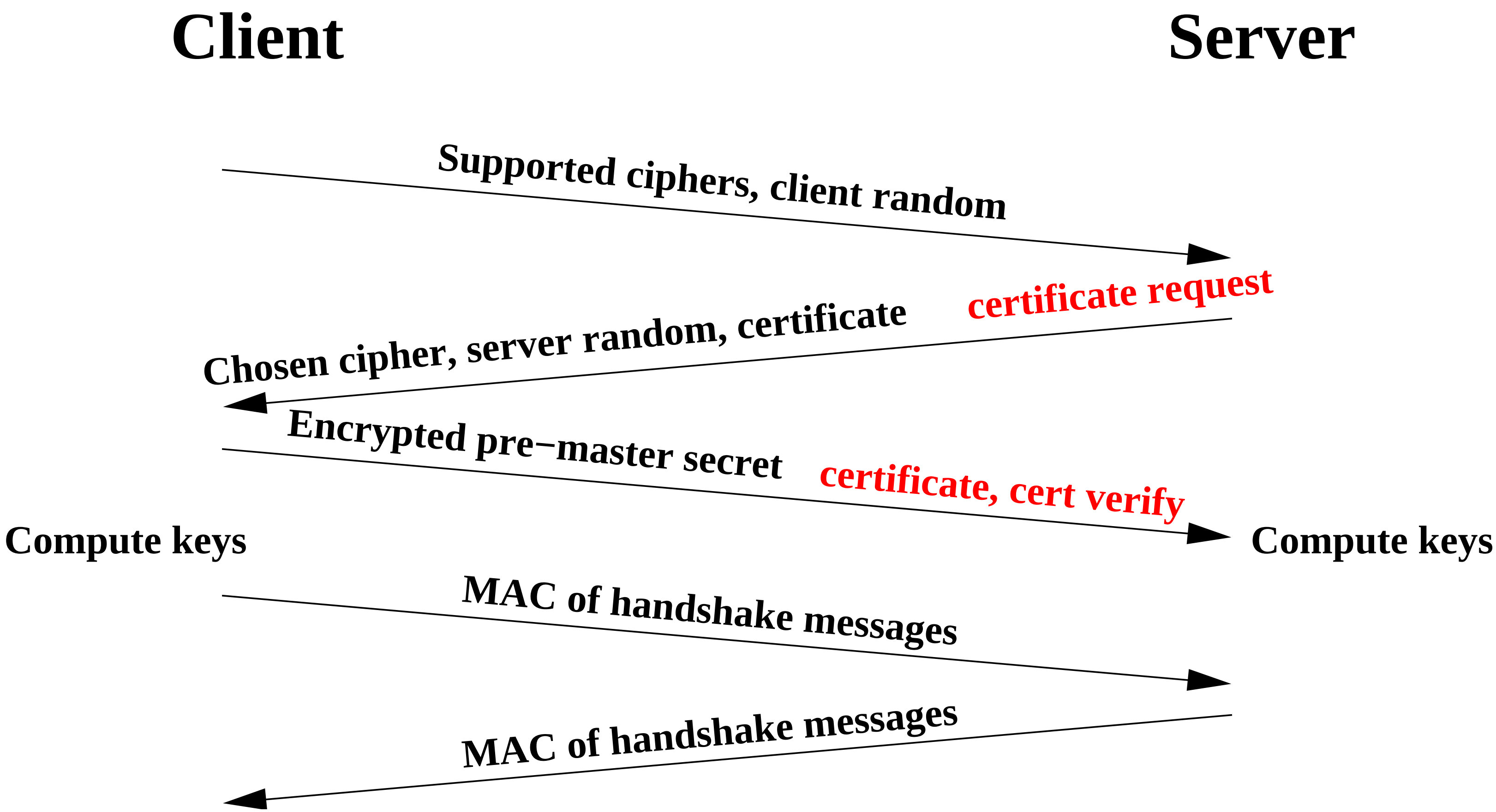
Simplified SSL Handshake

- **Client and server negotiate on cipher selection.**
- **Cooperatively establish session keys.**
- **Use session keys for secure communication.**

Client Authentication Handshake

- **Server requests that client send its certificate.**
- **Client signs a signed digest of the handshake messages.**

SSL Client Certificate

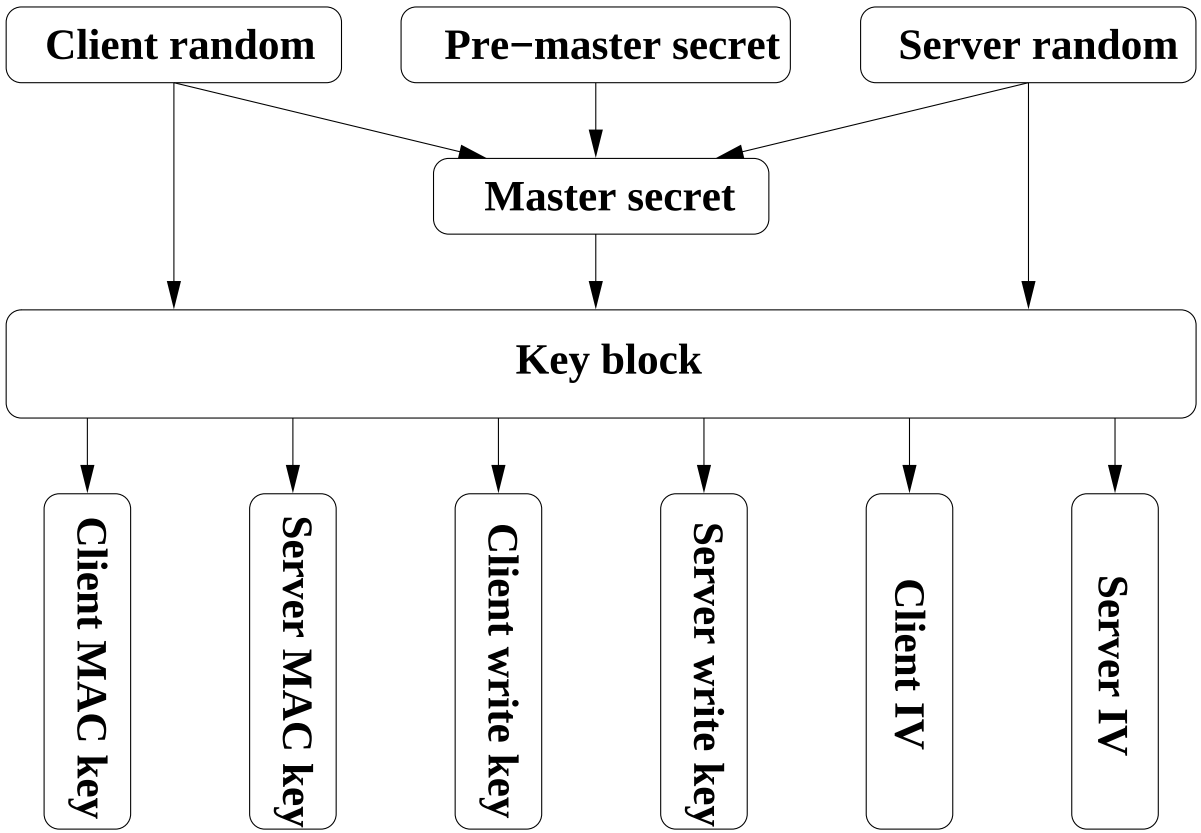


From “SSL and TLS” by Eric Rescorla

Establishing a Session Key

- **Server and client both contribute randomness.**
- **Client sends server a "pre-master secret" encrypted with server's public key.**
- **Use randomness and pre-master secret to create session keys:**
  - Client MAC
  - Server MAC
  - Client Write
  - Server Write
  - Client IV
  - Server IV

Establishing a Session Key



From “SSL and TLS” by Eric Rescorla

Session Resumption

- **Problem: Public key crypto expensive**
- **New TCP connection, reuse master secret.**
  - Avoids unnecessary public key cryptography.
- **Combines cached master secret with new randomness to generate new session keys.**
- **Works even when the client IP changes (servers cache on session ID, clients cache on server hostname).**

What does CA mean by certificate?

- That a public key belongs to someone authorized to represent a hostname?
- That a public key belongs to someone who is associated in some way with a hostname?
- That a public key belongs to someone who has lots of paper trails associated to a company related to a hostname?
- That the CA has **no liability**?
- >100-page Certification Practice Statement (CPS)

How to get a Verisign certificate

- **Pay Verisign ($300)**
- **Get DBA license from city call ($20)**
  - No on-line check for name conflicts…can I do business as Microsoft?
- **Letterhead from company ($0)**
- **Notarized document (need driver's license) ($0)**
- **Conclusions:**
  - Easy to get a fraudulent certificate
  - Maybe not so easy to avoid prosecution afterwards
- **But that's only Verisign's policy**
  - Many CAs can issue certificates

So many CAs…


Certificate Manager
Your Certificates
Other People's
Web Sites
Authorities
You have certificates on file that identify these certificate authorities:
Certificate Name
Security Device
Comodo CA Limited
Digital Signature Trust Co.
Entrust.net
Equifax
Equifax Secure
Equifax Secure Inc.
GTE Corporation
GeoTrust Inc.
GlobalSign nv-sa
Government Root Certification A...
IPS Internet publishing Services s.l.
IPS Seguridad CA
NetLock Halozatbiztonsagi Kft.
QuoVadis Limited
RSA Data Security, Inc.
RSA Security Inc
SECOM Trust.net
Sonera
View
Edit
Import
Delete

CA Convenience vs. Security

- **How convenient is a Verisign certificate?**
  - Need $300 + cooperation from Stanford IT to get one here
  - Good for credit cards, but shuts out many other people
- **How trustworthy is a Verisign certificate?**
  - In mid-March 2001, VeriSign, Inc., advised Microsoft that on January 29 and 30, 2001, it issued two… [fraudulent] certificates…. The common name assigned to both certificates is "Microsoft Corporation."

    VeriSign has revoked the certificates…. However… it is not possible for any browser's CRL-checking mechanism to locate and use the VeriSign CRL.

– Microsoft Security Bulletin MS01-017