G22.3033-003
Topics in Computer System Security

Instructor: David Mazières
TA: Antonio Nicolosi

New York University

Administrivia

- **Class home page:**
  `http://www.scs.cs.nyu.edu/css/`
  - No handouts in class
  - All assignments/handouts will be distributed by web
  - Check the web page frequently for updates
- **Office hours:**
  - David Mazières: 4:30pm-5:30pm Wednesday
    Other times okay, too, just send me mail.
  - Antonio Nicolosi: TBD
- **Contact: secstaff@scs.cs.nyu.edu**

Assignments & Grading

- **Main assignment: Reading**
  - We will be reading about 3 papers/week
  - Some papers labeled "Handout" or "skim," means I will discuss portions in lecture, you are responsible for those portions
- **Course grade mostly based on two exams:**
  - Midterm: March 1
  - Final: May 10?, May 2?, April 26? Hold vote.
  - Note: Both exams open book
- **Also a few programming assignments:**
  - You should know how to program in C

Lecture format

- **Some lecture, some discussion**
  - Read papers before class
- **People often learn best from their mistakes**
  - Security flaws by definition are unexpected
  - I may describe a system before talking about its flaws
  - Stay on your toes in lecture, and feel free to ask questions
  - If something looks wrong, it may be wrong
- **Lecture notes will go up on class web page**
  - Sometimes slides, sometimes just my own discussion notes

Topics

- Cryptography and applications
- Information flow & Mandatory Access Control
- Secure operating systems
- "Trusted" computing
- Detecting, avoiding, and tolerating software bugs
- Network security
- Anonymity & Privacy
- Unexpected/"Other" system failures

What this class is and is not

- **This class is not**
  - How to configure Microsoft IIS SSL server with SQL server back end
  - How to connect a Win98 box using the Win2K VPN protocol
  - How to design new encryption algorithms
  - Number theory, Lattice theory, Complexity theory…
- **The class will cover**
  - How to use cryptography in the systems you build
  - Issues involved if you want to design SSL
  - How to read research papers in the area of security

Definitions

- **Security**: Techniques to control who can access/modify system
- **Principal**: Unit of accountability in a system (e.g., a user)
- **Access control**: Techniques to restrict operations to certain principals
- **Authentication**: Verification of the identity of the principle making a request
- **Authorization**: The granting of a request to a principal

Attacks on security

- **Violation of secrecy**
  - Attacker reads data without authorization
- **Violation of integrity**
  - Attacker modifies data without authorization
  - E.g., Attacker modifies data on disk
  - E.g., Attacker modifies network reply to "read file" request
- **Denial of service**
  - Attacker makes system unavailable to legitimate users
  - Overload the system; cause a deadlock
  - Trigger security mechanism (e.g., wrong bank PIN 3 times)

Security is a negative goal

- **Ensure nothing happens without authorization**
  - How do you reason about what a system will *not* do?
- **Must first specify who is authorized to do what**
  - Boils down to a question of specifying *policy*
- **Trusted computing base (TCB): Collection of software upon which security relies**
  - Should have well-specified TCB
  - Minimize TCB—less code in which to find security holes

Protection mechanisms

- **Hardware protection**
  - ROM and/or read-only disks can provide booting
  - Processor separates user and kernel code
  - User code cannot access devices, write kernel memory, ...
  - Kernel boundary separates TCB
- **Software protection**
  - Kernel or server sanity-checks and restricts call arguments
  - Bytecode interpreter restricts functionality (e.g., JVM)
- **Protection of network traffic with cryptography**

Policy

- **Policy: The goal security must achieve**
  - Human intent—originates from outside the system
- **Often talked about in terms of subjects and objects**
  - Subject: Entity making requests (= principal)
  - Object: Abstraction to which access is requested (e.g., a file, a page of memory, a serial port)
  - Each object supports different kinds of access (e.g., read or write file, change permissions, . . .)
- **Access control: Should operation be allowed?**
  - What principal is making the request? (Authentication)
  - Is the operation permitted to the principal? (Authorization)

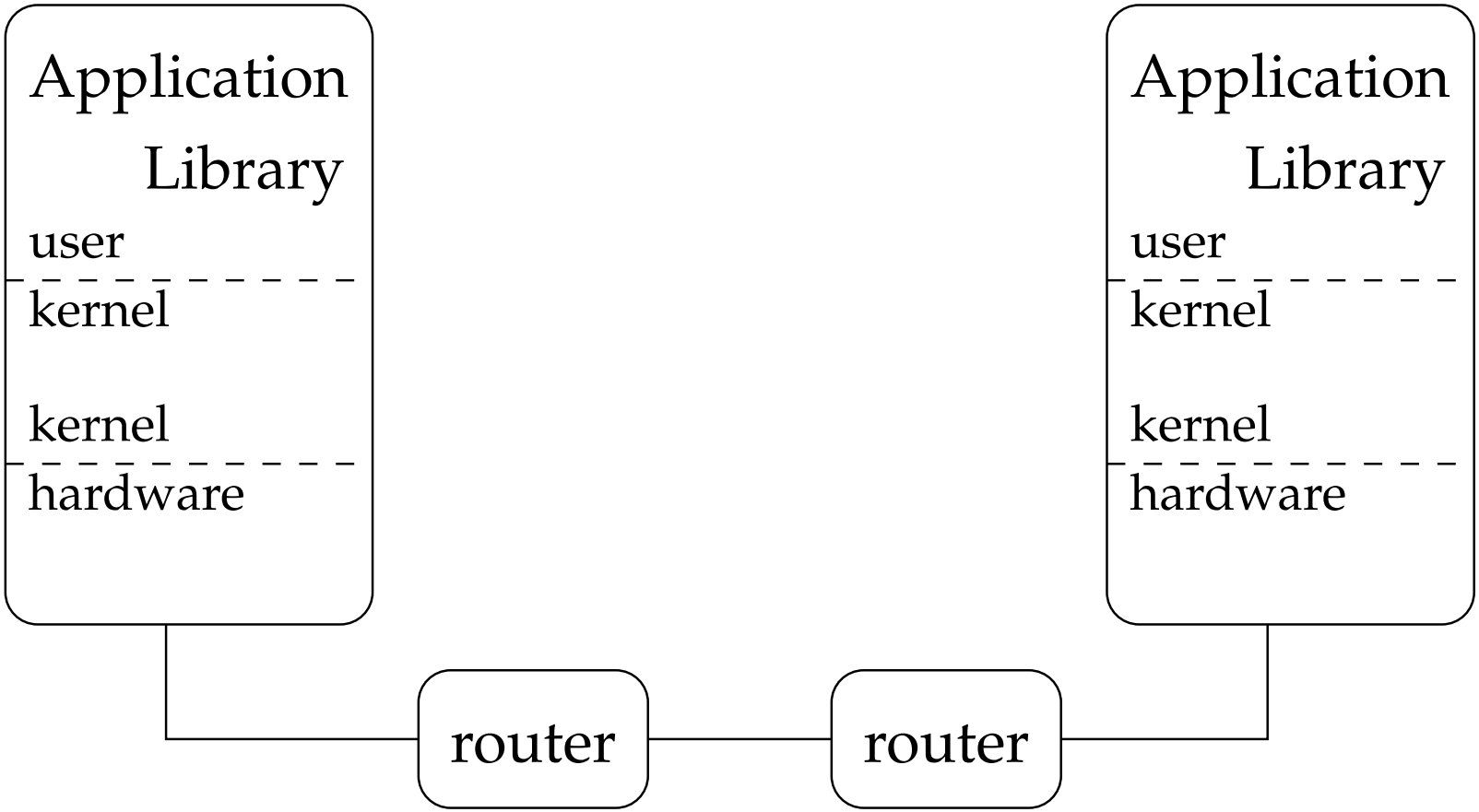
Examples of access control

- **Unnetworked machine in a locked room**
  - Policy: Only users with keys can access the computer (Don't overlook old-fashioned solutions if they apply!)
- **Bank ATM card**
  - Policy: Can only withdraw money if it's in your account
  - Authentication: Owner must possess card & know PIN
  - Authorization: Database tracks account balances
- **Private Unix file (only owner can read)**
  - Authentication: Password to run software as user
  - Authorization: Kernel checks permissions bits on file
- **Military classified data**
  - If process reads top-secret data, cannot write secret

Authentication mechanisms

- **User provides password to server**
- **User proves possession of device (e.g., smart card)**
- **Cryptographic protocol proves possession of private key**
- **Biometrics (e.g., fingerprint)—Often misused**
  - Should your laptop send your fingerprint to the server?
- **Don't forget server-to-user authentication!**
  - Fake login screen or ATM machine gets user's password

The end-to-end principal



- **Place functionality closer to the endpoints**

End-to-end security

- **Analogy: How to test the strength of a chain?**
  - Don't test individual links, pull two ends of chain!
- **Without end-to-end authentication:**
  - Many layers at which something could go wrong (e.g., bad guy tampers with "secure" network)
  - Policy often in terms of higher-level abstractions
- **Example: NFS over VPN**
  - NFS is insecure network file system (trusts clients and network)
  - VPN cryptographically protects network traffic between two machines
  - But policy states which *users* should access files, not which *machines*

Authorization mechanisms

- **Access control lists on objects**
  - A list of principals and permitted operations for each
- **Self-authenticating capabilities**
- **Signed digital certificates**
  - NYU certifies your status; eb.com allows NYU students
- **Object access in type-safe language**
  - Authentication: Some procedure returns a pointer to object
  - Authorization: Possession of pointer allows access
- **Issues of authorization mechanisms**
  - Can those with access futher delegate access?
  - How easy is it to revoke access to an object?

Interactions between objects

- **Must consider any operations could violate policy**
- **Example: Only professor can change grades**
  - OS refuses write requests from others to grades file (easy)
  - But can attacker change program professor runs to edit file?
  - Can attacker change directory so owner reads different file?
- **Example: setuid and ptrace (debugger system call)**
  - Ptrace lets one process modify another's memory
  - Setuid gives a program more privilege than invoking user
  - Don't let process ptrace more privileged process
  - But also must disable setuid if execing process ptraced

A linux security hole

- **Some programs acquire then release privileges**
  - E.g., `su user` setuid, becomes user if password correct
- **Consider the following:**
  - A and B unprivileged processes owned by attacker
  - A ptraces B
  - A executes "`su user`" to its own identity
  - While `su` is superuser, B execs `su root`
    (A is superuser, so this is not disabled)
  - A types password, gets shell, and is attached to `su root`
  - Can manipulate `su root`'s memory to get root shell

System design principals

- **Economy of mechanism**
  - mail.local (user software reimplemented kernel checks)
- **Principal of least privilege**
  - Separation of privilege
  - Bind priv TCP port→superuser, buggy server→disaster
- **Make the defaults secure**
  - MS email clients susceptible to worms
- **Open design**
  - GSM encryption failure, DVD CSS, MS VPN protocols, …

More system design principals

- **Make assumptions explicit to all parties**
  - Numerous protocol failures (next lecture)
- **Feedback and iteration**
  - You won't design it right the first time
  - Can't get rid of Kerberos IV
- **Psychological acceptability**
  - System only secure if used that way
  - E.g., users reuse passwords, tape password to monitor

Stretch break

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

Types of 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
  - 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:**

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

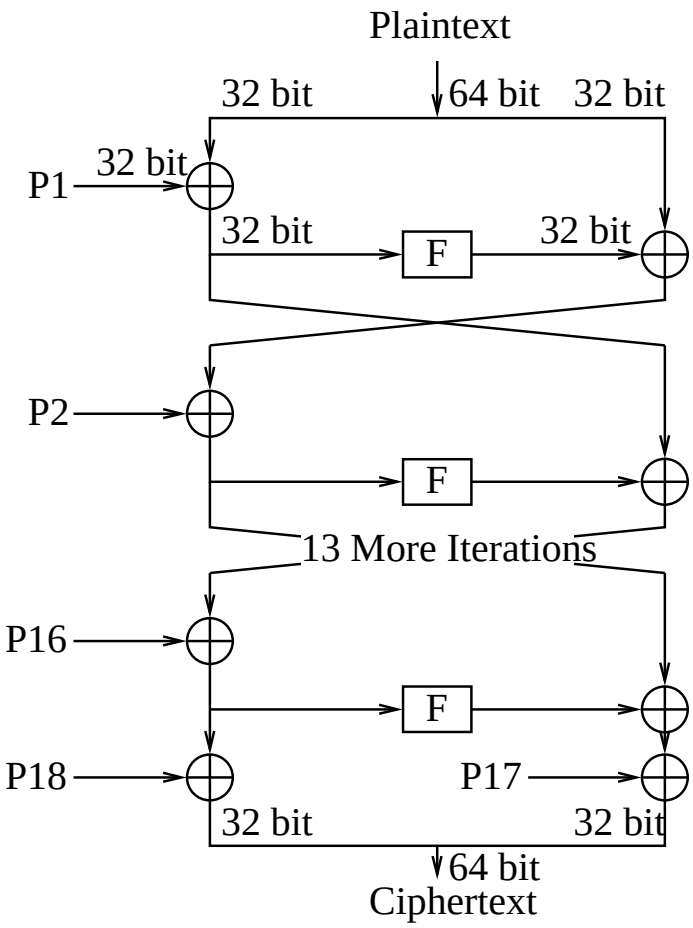
Example use of stream cipher

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

Wrong!

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



- **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)

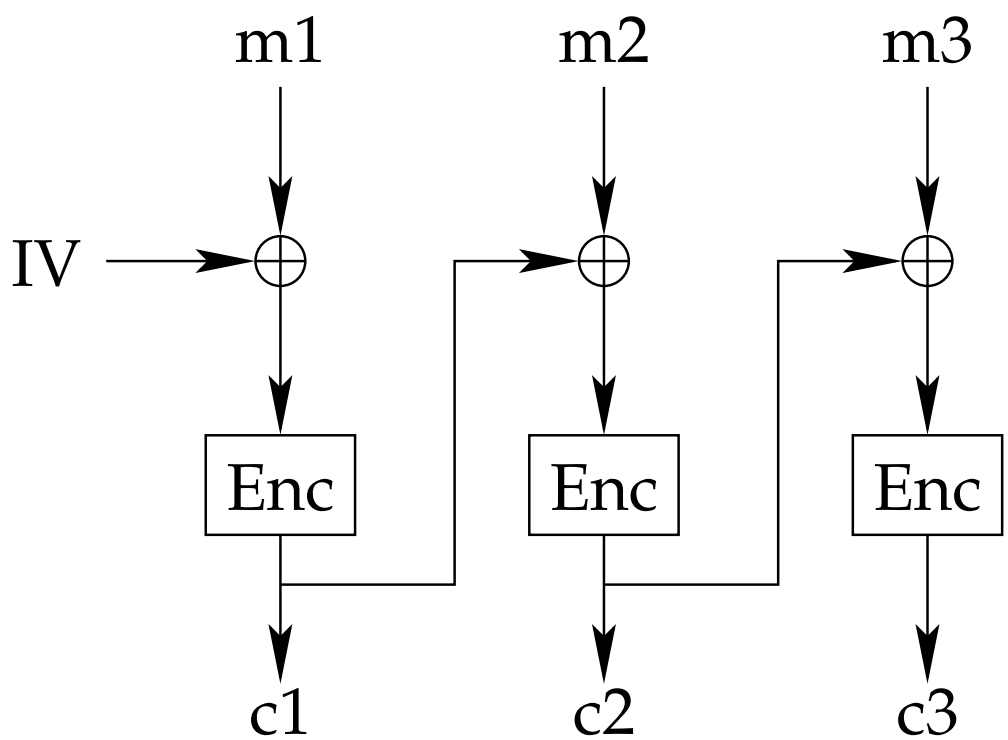
Using a block cipher

- **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 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
- **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, $\mathrm{MAC}(K, m) \rightarrow v$
  - Attacker cannot 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** $\{m, v\}$

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: 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 alrogithm
- **Most popular hash SHA-1**
  - Also SHA-256, SHA-512, more secure versions

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
- $\mathbf{HMAC}(K, m) = H(K \oplus \mathsf{opad}, H(K \oplus \mathsf{ipad}, m))$
  - 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?**
-
-
  -
  -
  -
  -

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 randomized 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.

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
- **Facts:**
  - If $m \in \mathbf{Z}_N^*$, then $(m^e \bmod N)^d \bmod N = m$.
  - For large enough p, q and random m, Given N, e, and $m^e \bmod N$, No one knows how to find m if they don't already know p, q, or d.
- **To encrypt a message, just treat bits as number and computer $m^e \bmod N$.**

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

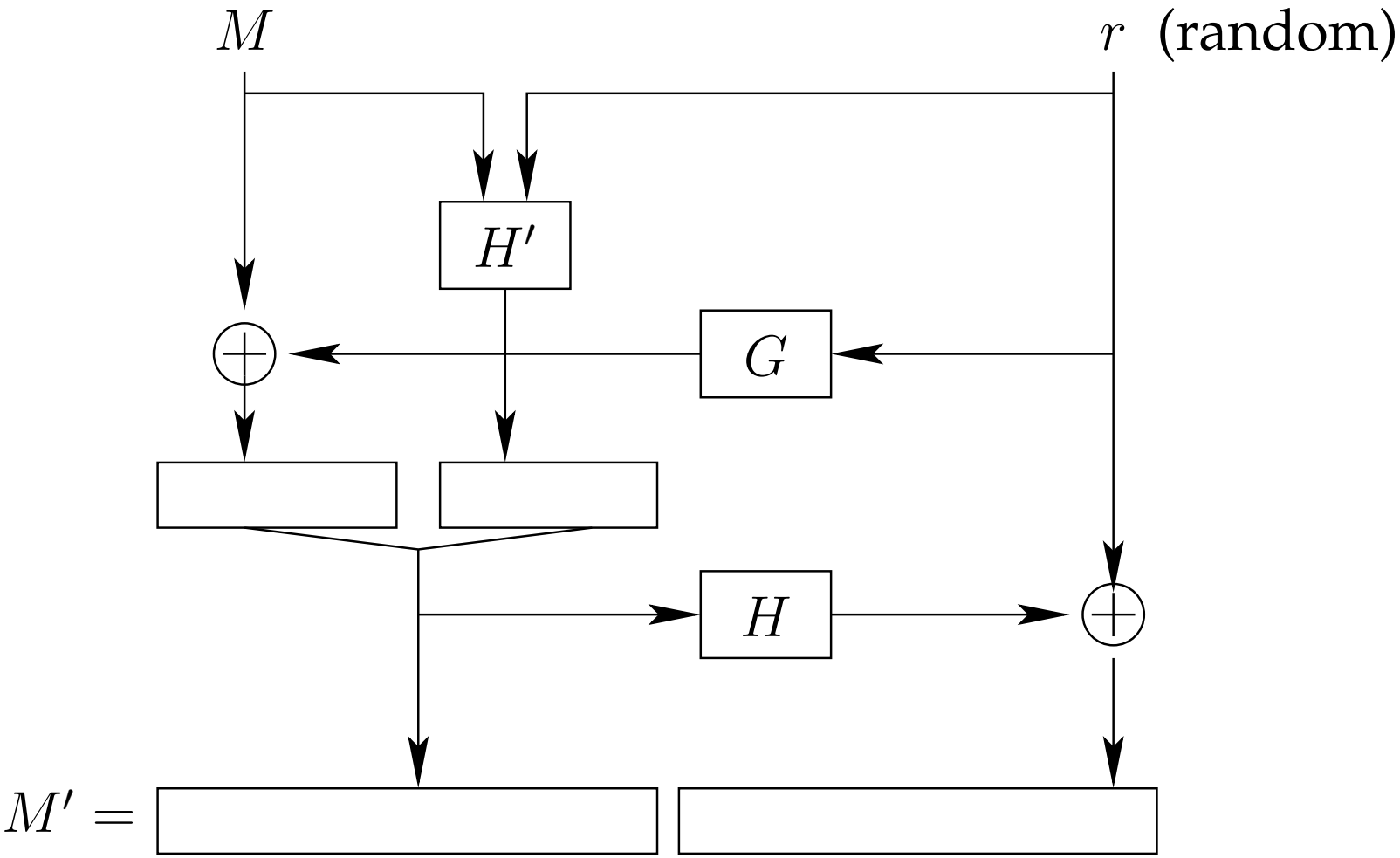
- **How do design systems using RSA?**
  - You don't want to think about interactions between your error messages, modular exponentiation, and lattice theory.
- **A PKS is adaptive chosen ciphertext secure if**
  - No attacker A can win the following game with probability more than $1/2$+negligible:
  - 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

How to achieve Adaptive CCA security?

- **Good properties for message $\rightarrow$ 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* – $V\left(K, \{m\}_{K^{-1}}, m\right) \rightarrow \{\mathbf{true}, \mathbf{false}\}$
- **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 $\mathit{Verify}(K, m', s') = \mathbf{true}$

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 $r \leftarrow g^k \bmod p$, $s \leftarrow k^{-1}\left(H(m) - xr\right) \bmod (p-1)$
- **Verification:**
  - Sanity check: $1 \leq r \leq p-1$
  - Verify: $y^r r^s \stackrel{?}{\equiv} g^{H(m)} \pmod p$
  - $y^r r^s = \left(g^{xr}\right)\left(g^{ks}\right) = g^{xr+ks} = g^{xr+k\cdot k^{-1}(H(m)-xr)} = g^{H(m)}$

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

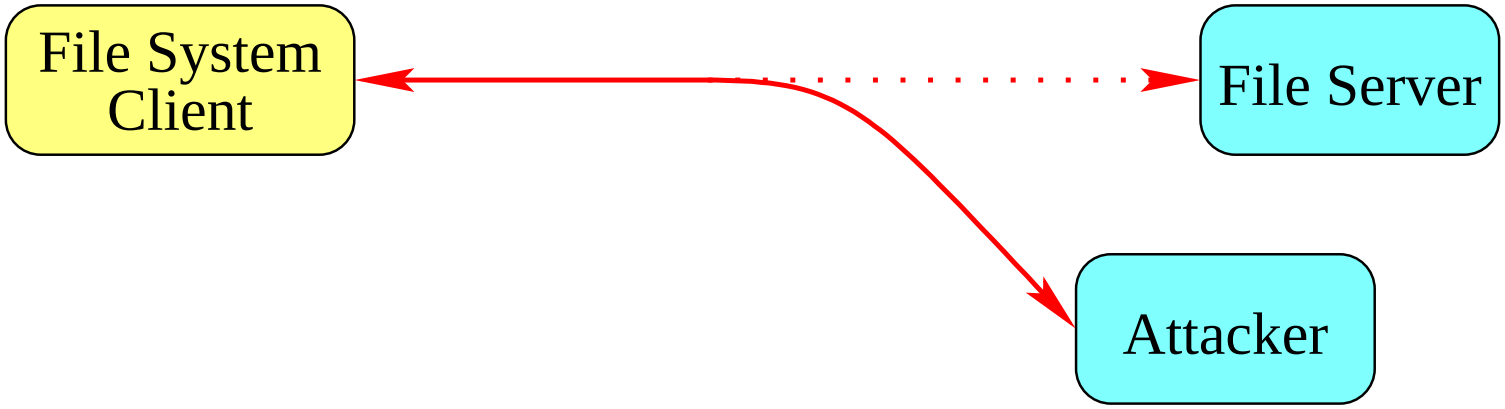
Hybrid schemes

- **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?

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

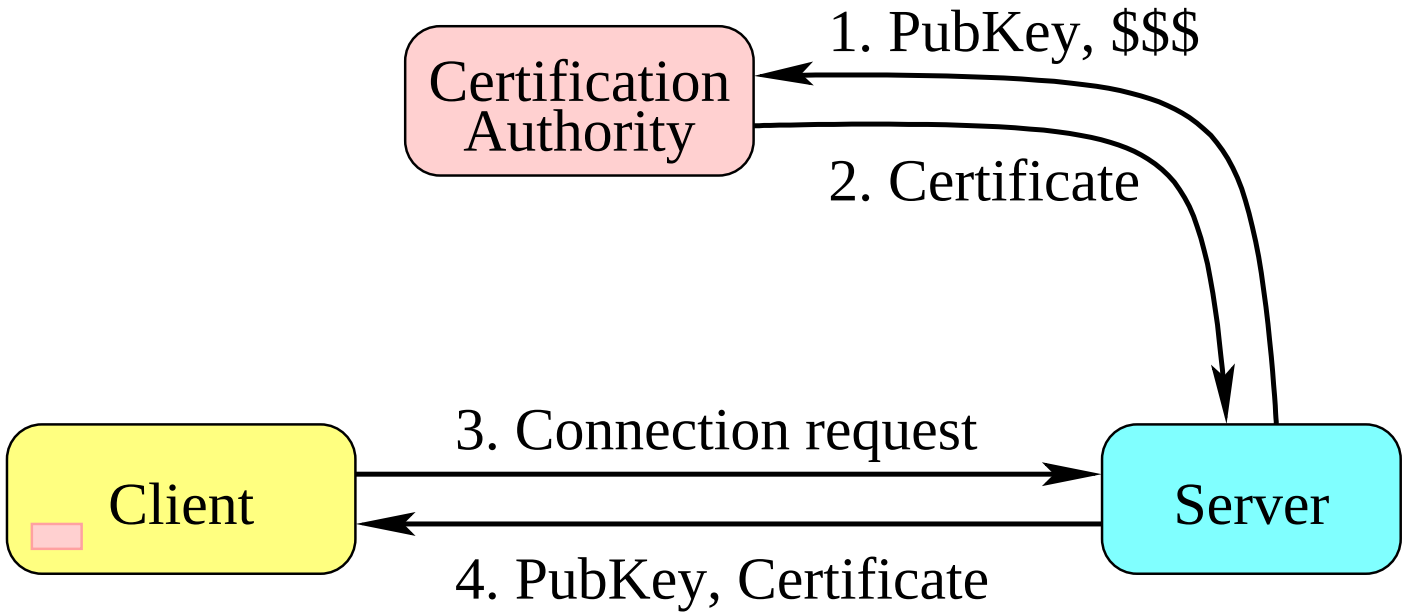
Man in the middle attacks

- **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