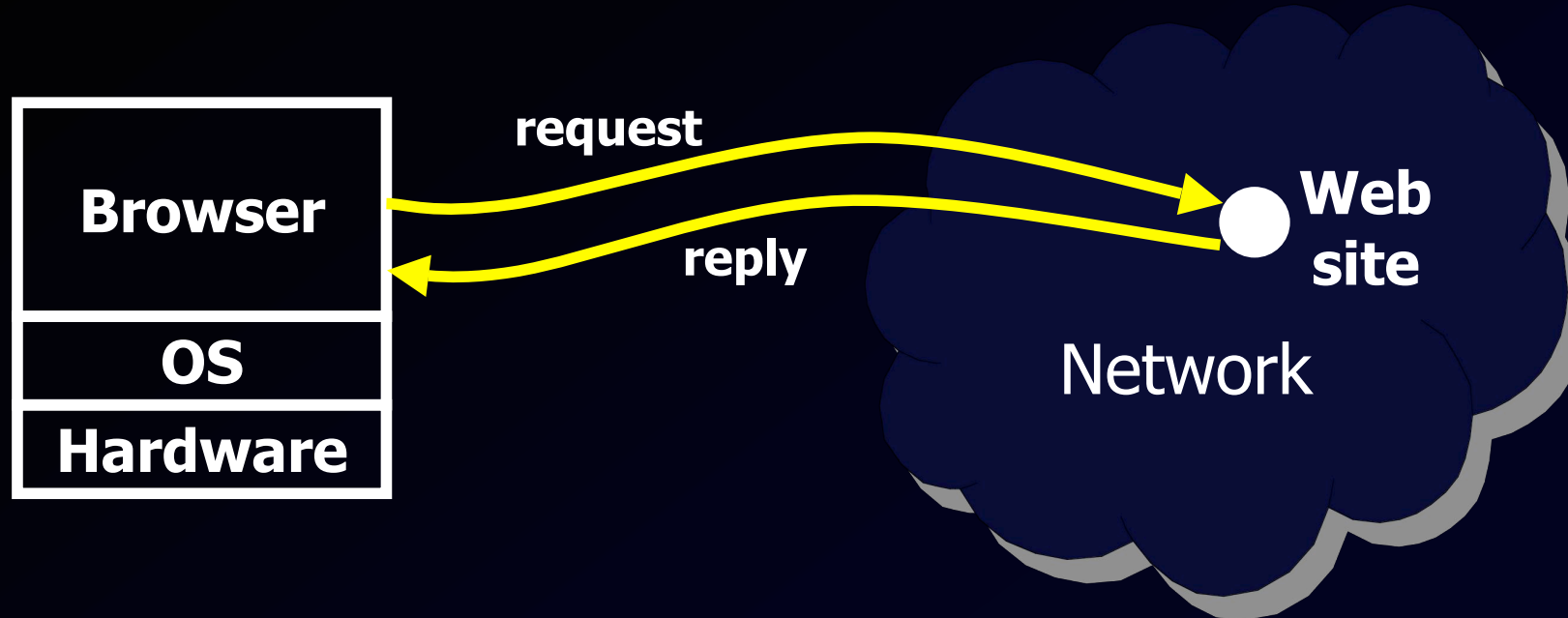


**UMSSIA**

**LECTURE VII: WEB FUN**

# WEB SECURITY



- **Browser sends requests to server**
- **Server processes requests**
- **Browser receives information and code**
- **Repeat as necessary.**

# HTTP

**Method**



**File**



**HTTP version**



**Headers**



```
GET /default.asp HTTP/1.0
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Connection: Keep-Alive
If-Modified-Since: Sunday, 17-Apr-96 04:32:58 GMT
```

**Blank line**



**HTTP version**



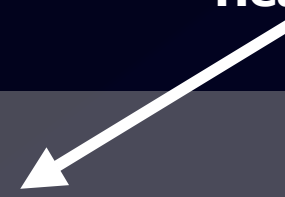
**Status code**

**Data – none for GET**

**Reason phrase**

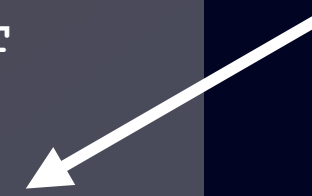


**Headers**



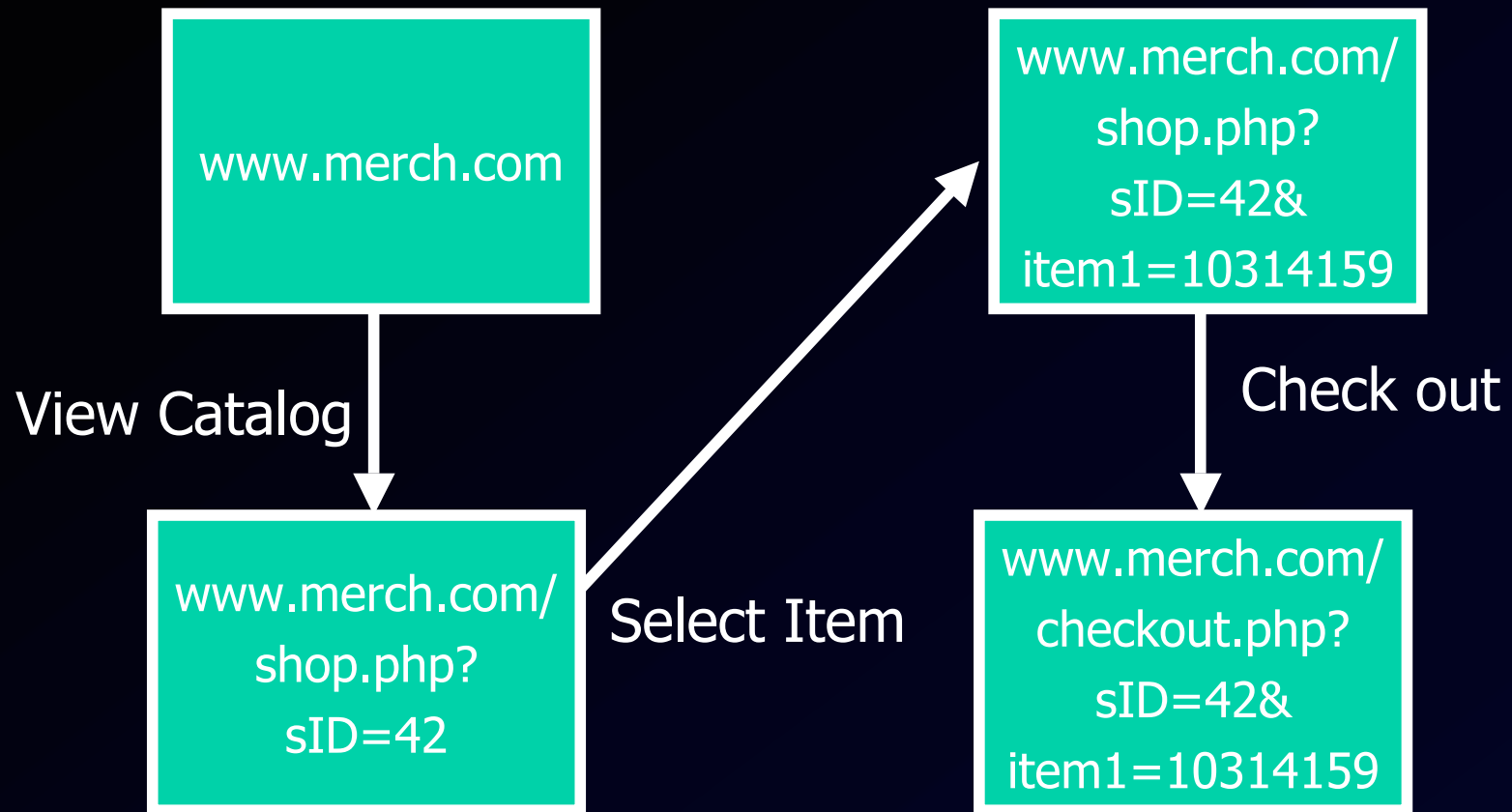
```
HTTP/1.0 200 OK
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Content-Length: 2543
```

**Data**



```
<HTML> Some data... blah, blah, blah </HTML>
```

# BROWSER SESSIONS



Store session information in URL

# COOKIES

A **cookie** is a persistent file created by a server to store information at the client

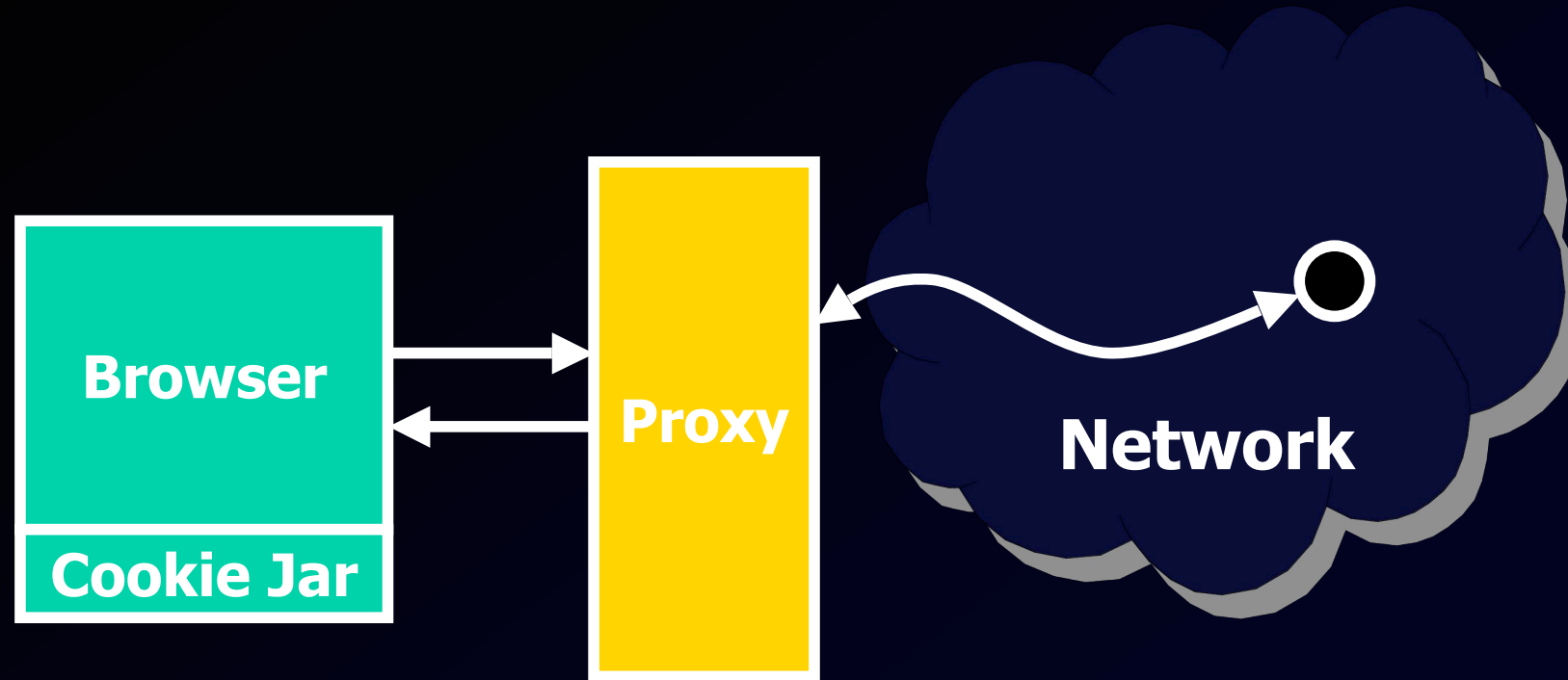


**HTTP is a stateless protocol; cookies add state**

# COOKIES

- ... specify which web sites can access them. Typically only the creating site can access the cookie.
- ... Come in persistent and temporary flavors.
- ... Are a privacy risk: can be used to store browsing habits, personal info, etc.
- ... Are often not secured against malicious attackers.

# PRIVACY PROXIES



... intercept HTTP requests and responses to help enforce privacy policy.

... e.g., modify cookies before sending to browser or server; filter out ads; block sites...

# PRIVACY POLICY

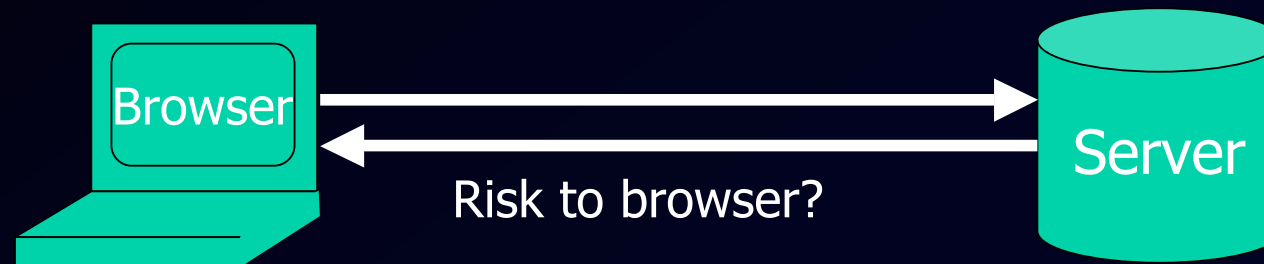
- Web sites collect a lot of personal information. What do they do with it?
- The P3P framework allows agreement on the use of this info, a formalized version of a "Privacy Policy."



- Enforcement at server side is another matter...  
beware the market for lemons!

# CLIENT-SIDE THREATS

- Given correct implementation, **data** is essentially “harmless...”
- Risks come from **code** received from web
  - Scripts in web pages
  - ActiveX controls and Browser extensions
  - Applets



# JAVASCRIPT

- **... is an interpreted language executed by browser, and used in many attacks**
- **... can run:**
  - **before the HTML is loaded**
  - **before the document is viewed**
  - **while the document is viewed**
  - **as the browser is leaving**
  - **when user changes focus**
- **... code has access to:**
  - **User inputs: Keyboard monitoring and logging**
  - **Cookies: hijack authentication, browsing data**
  - **Screen IO: Spoof parts of web browser UI**
  - **Network: Communicate across network**

# ACTIVEX

**ActiveX controls are programs that reside on clients and are activated by HTML pages. They are:**

- not interpreted by the browser**
- typically downloaded and installed on demand**

**The ActiveX “security model” relies on:**

- Digital signatures to verify the source of a binary**
- policy to reject controls from network zones**
- A type bit: *safe for initialization*, or *safe for scripting* which affects the way the control is used**

**Once an ActiveX control is installed, it runs with the privileges and in the memory context of the browser and there is no way to control or limit its execution.**

# JAVA

**... is a general purpose, strongly-typed programming language.**

**Browsers execute java programs in the Java Virtual Machine (JVM), which attempts to isolate remote code from sensitive system resources.**

**The JVM can also verify code security properties such as exception safety, modularity, etc.**

**This design allows for portable implementation, flexible sandboxing, etc...**

# JAVA SECURITY RISKS

**... include several types:**

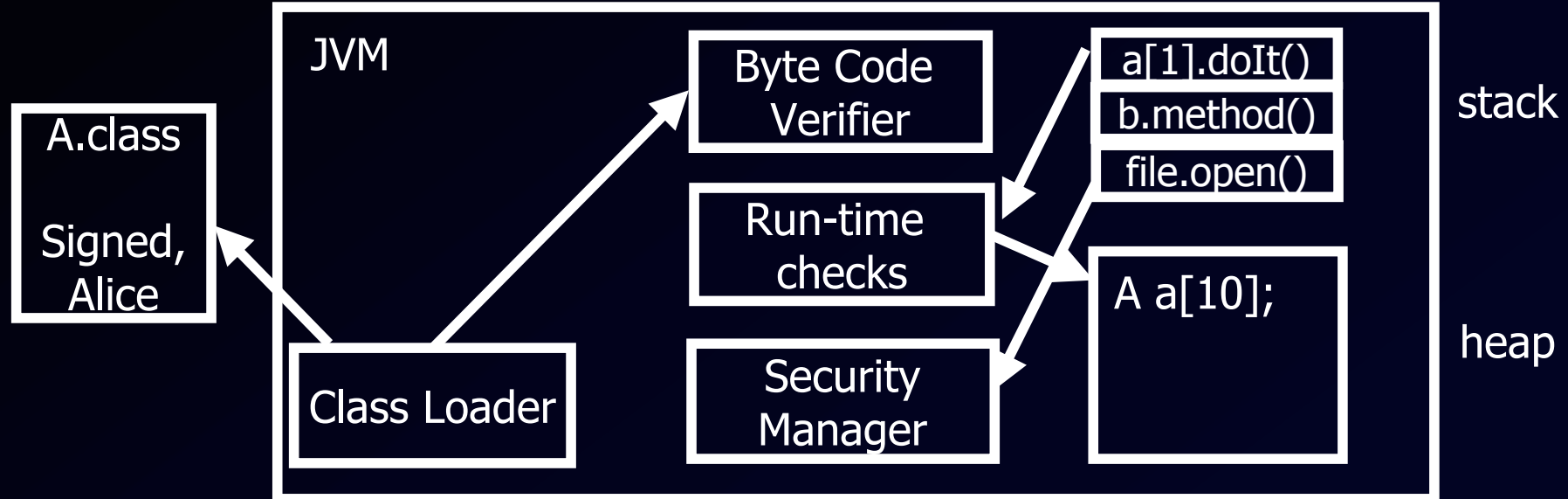
- **Annoyance or inconvenience:**
  - Display large window that ignores mouse input
  - Play irritating sound and do not stop
  - Consume CPU cycles, memory, network bandwidth ...
- **Information theft:**
  - Communication to remote machine
  - Access to password file, credit card number, ...
  - Subtle attack: trick dialog boxes, or trick UI
- **Modify or compromise system:**
  - Delete files, call system functions

# JAVA SANDBOX

**Class loaders** create new types, and enforce isolation through namespaces and **protection domains**.

The **Byte Code Verifier** and JVM **run-time tests** ensure that type safety, array bounds, and visibility levels are preserved.

The **Security Manager** checks access requests based on protection domains and **stack inspection**.



# WEB SITE SECURITY

**Securing web sites can be challenging for many reasons:**

- Sessions store state at the "client"**
- Inputs come from unknown, untrusted sources**
- The statelessness of HTTP allows replays, modification, injection, etc.**
- Mutually untrusted applications may use the same server**

# TRANSACTIONS OVER HTTP

**Potential for bugs come up in many places:**

- **Session creation and identification:  
Creating and assigning unique IDs**
- **Concurrency issues: contention and duplication of sessions**
- **Session termination and timeout:**
  - Clean up stale sessions
  - Handle stale requests
  - Concurrency in session termination
- **Session state storage**
  - Distributed or local? Performance issues
  - Fail-over, load balancing issues

# **EXAMPLE: COOKIE AUTHENTICATION**

- **Fu et al., 2002 study of cookies for 27 major web vendors:**
  - **Obtained access to other accounts on 8**
  - **Obtained access to arbitrary accounts on 1**
- **Common mistakes:**
  - **Weak or misused crypto**
  - **Username/UID = authentication**
  - **Session ID = authentication**

# EXAMPLES

## Bad Crypto:

- [www.ichat.com](http://www.ichat.com) : cookie =  
username || password © "secret string"
- [www.wsj.com](http://www.wsj.com) : cookie =  
UNIX crypt(username || "secret string")

## No Crypto:

- [www.highschoolalumni.com](http://www.highschoolalumni.com) cookie =  
user="name"&id="uid"
- [www.nebride.com](http://www.nebride.com) : cookie=  
userid="id"&email="blah"  
asks for password or **send via email.**

# EXAMPLES

## Use of nonsecrets:

- [www.fatbrain.com](http://www.fatbrain.com)

- User logs in, is assigned a session ID in URL.
- Knowing the sID allows access to a session.
- IDs are assigned as follows:

```
global_current_id = random_integer();  
while(more sessions)  
    next_id = global_current_id++;
```

- Same problem: [www.verizonwireless.com](http://www.verizonwireless.com)

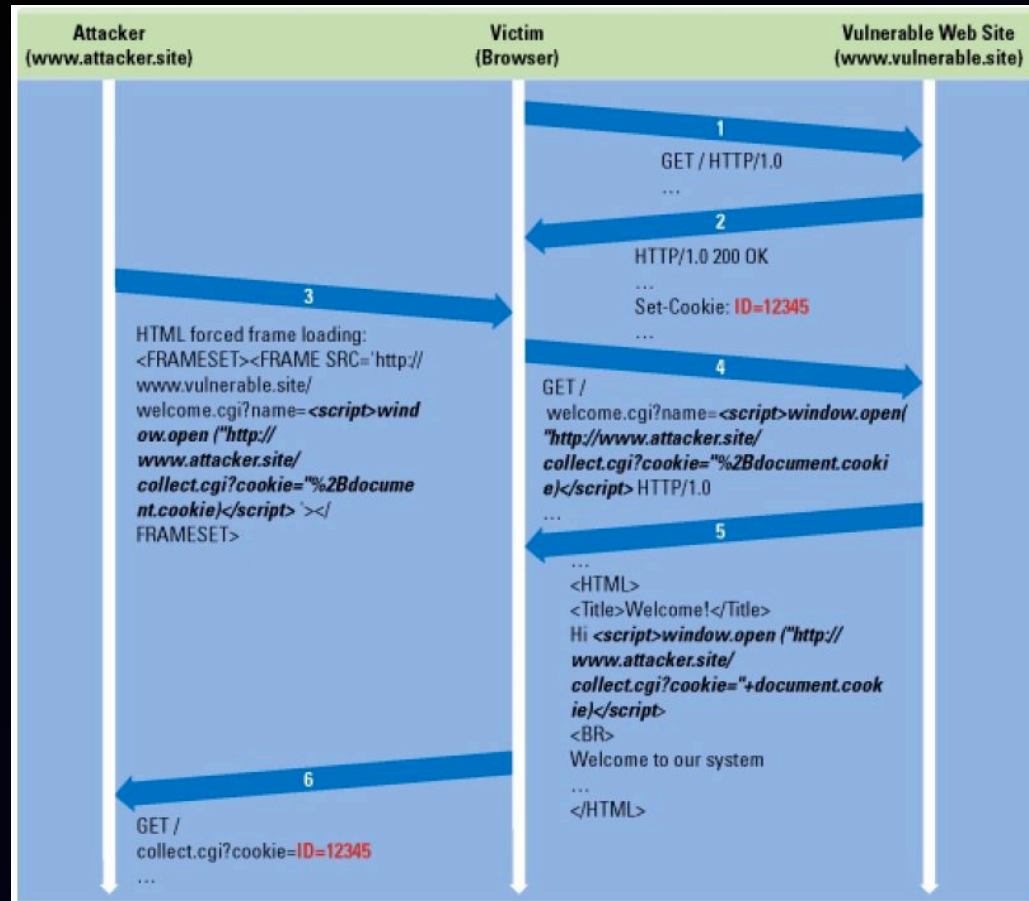
# DELICIOUS COOKIE RECIPE

- For safe cookies, use crypto integrity, e.g.
- Cookie: `exp=t`
  - `&data1=<blah1>`
  - `&data2=<blah2> ...`
  - `&auth=MACK(exp=t&data1=...)`
- Where K is a secret key held by the server (shared between servers if multiple)
- Session numbers should be chosen randomly
- If `data[i]` should be secret, apply encryption to `<blah[i]>`. (MAC the encrypted `<blah>`)

# SERVER-SIDE ATTACKS

- ... include the typical buffer overflows, format strings, integer overflows, etc. But also:
- **Command-line injection:**
  - “cat reply | mail \$user” =>  
“cat reply | mail user@example.com | rm -rf /”
- **“SQL/XPath Injection”**
  - Alter database queries to reveal or modify application database
  - Change user passwords, login as bogus user, learn entire tables, run commands as DBMS user, etc....
- **URL injection:**
  - PHP fopen(“/local/file”, “r”) opens /local/file.
  - PHP fopen(“http://www.example.com/remote/file” , “r”)

# CLIENT-SIDE ATTACKS



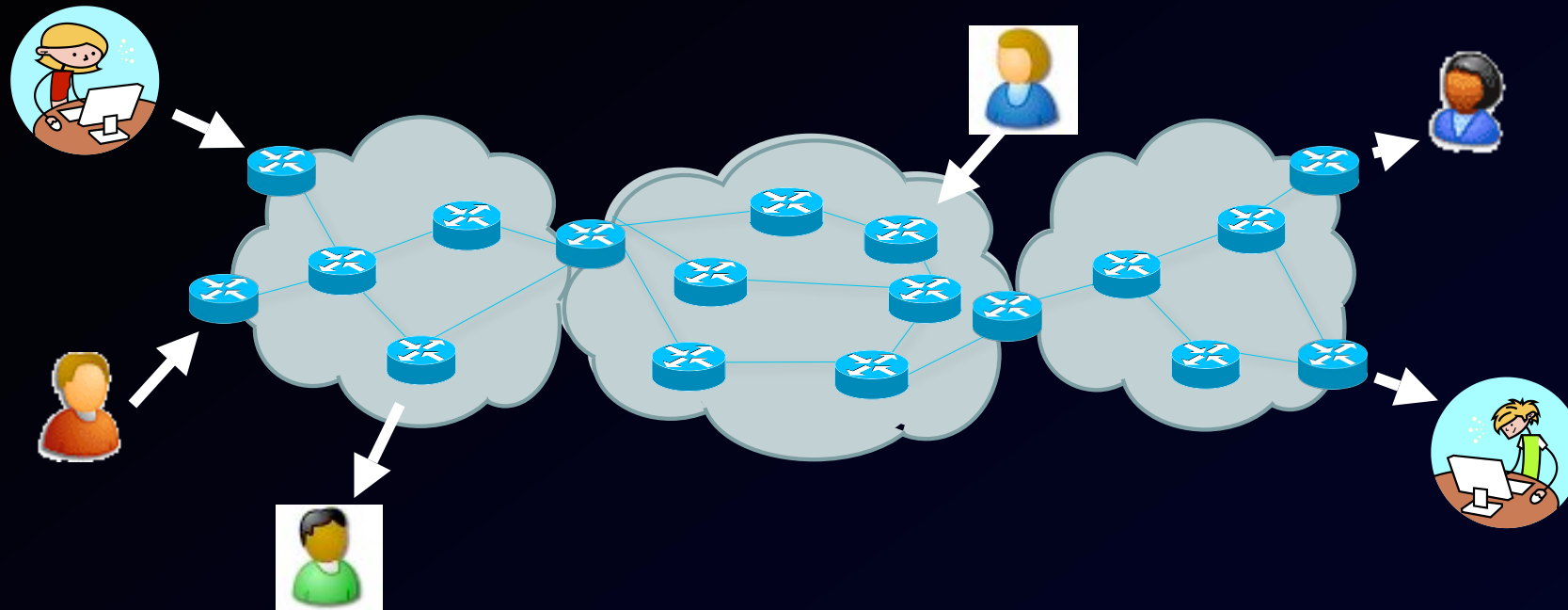
- **Cross-Site Scripting**
  - Run my script on your site's page
- **Cache poisoning**
  - My site/script loads as your URL
- **Cookie poisoning**
  - Set my cookies for your site

# DEFENSE

## INPUT VALIDATION

- **Check to make sure inputs are valid**
  - Eg. Email addr: only alphanumeric, `_`, `.`, `@`
- **Whitelist, don't blacklist**
  - **DON'T** look for `|` and `;`
- **Apply checking before and after all decoding and conversions**
  - IIS unicode bug, etc...
- **Use defenses available, e.g. Perl tainting (`perl -w`), etc...**

# ANONYMITY AGAINST WHOM?

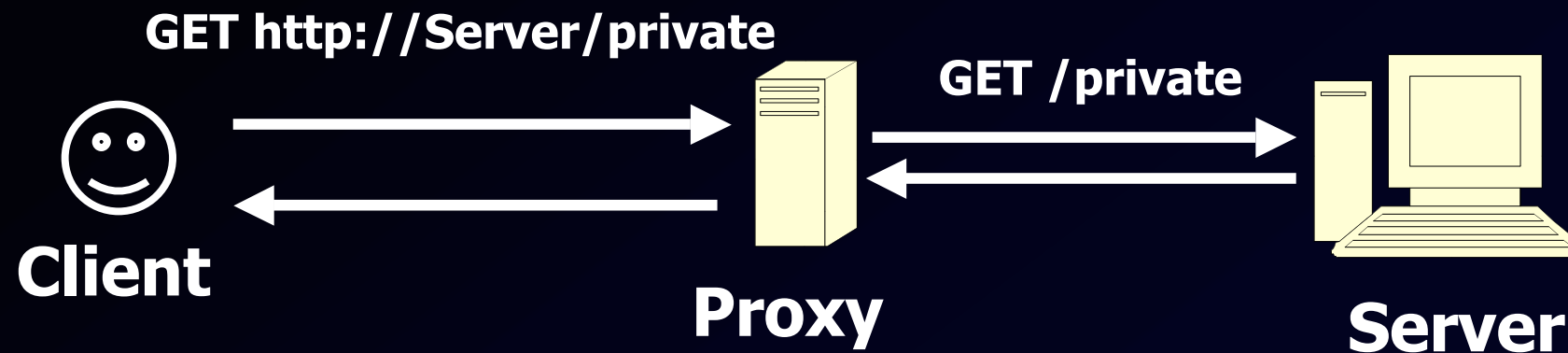


**Passive** adversaries (local, global, ISP) eavesdrop only

**Active** adversaries participate, inject, or drop messages.

# SIMPLE ANONYMITY

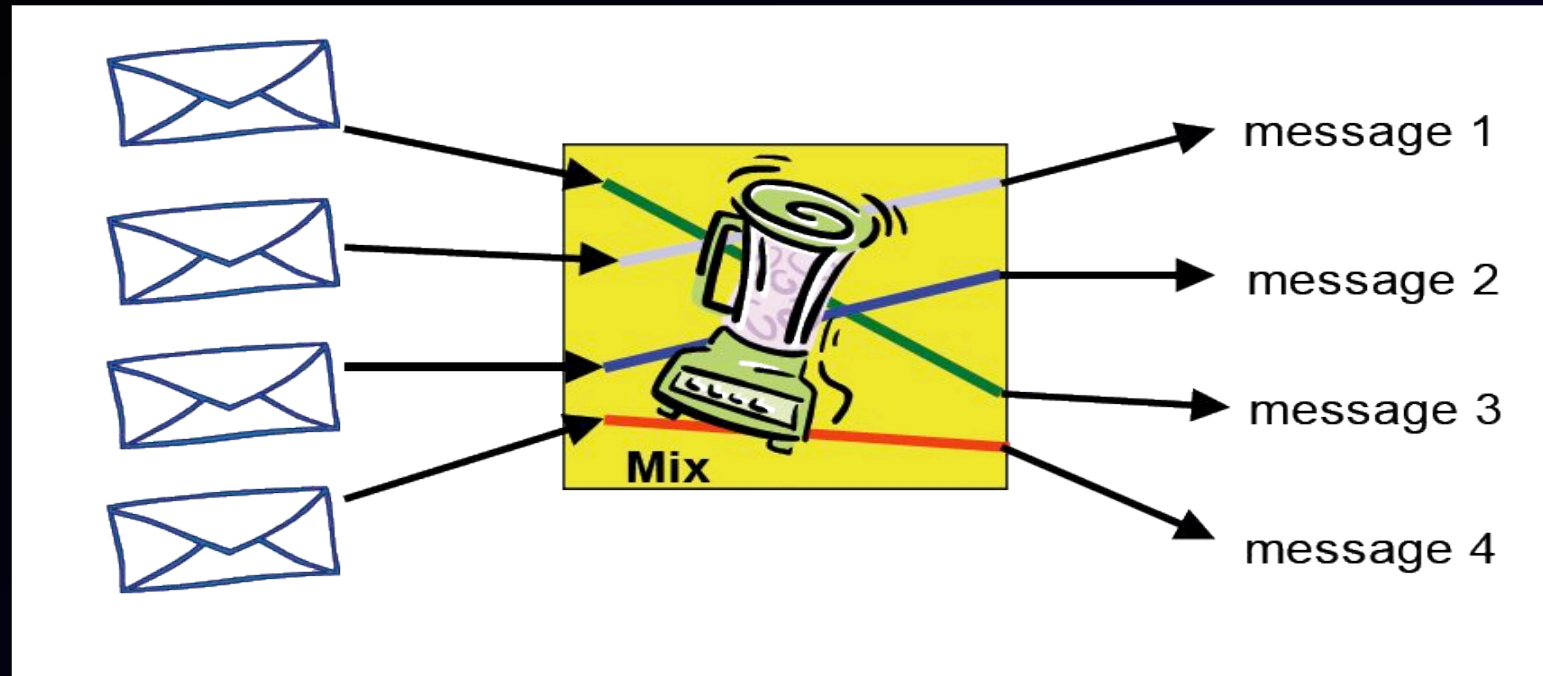
- **Email: pick a web mailer (check headers)**
- **Web: "open proxies" are machines running http proxies with no access control.**



**Some problems: plaintext, timing,...**

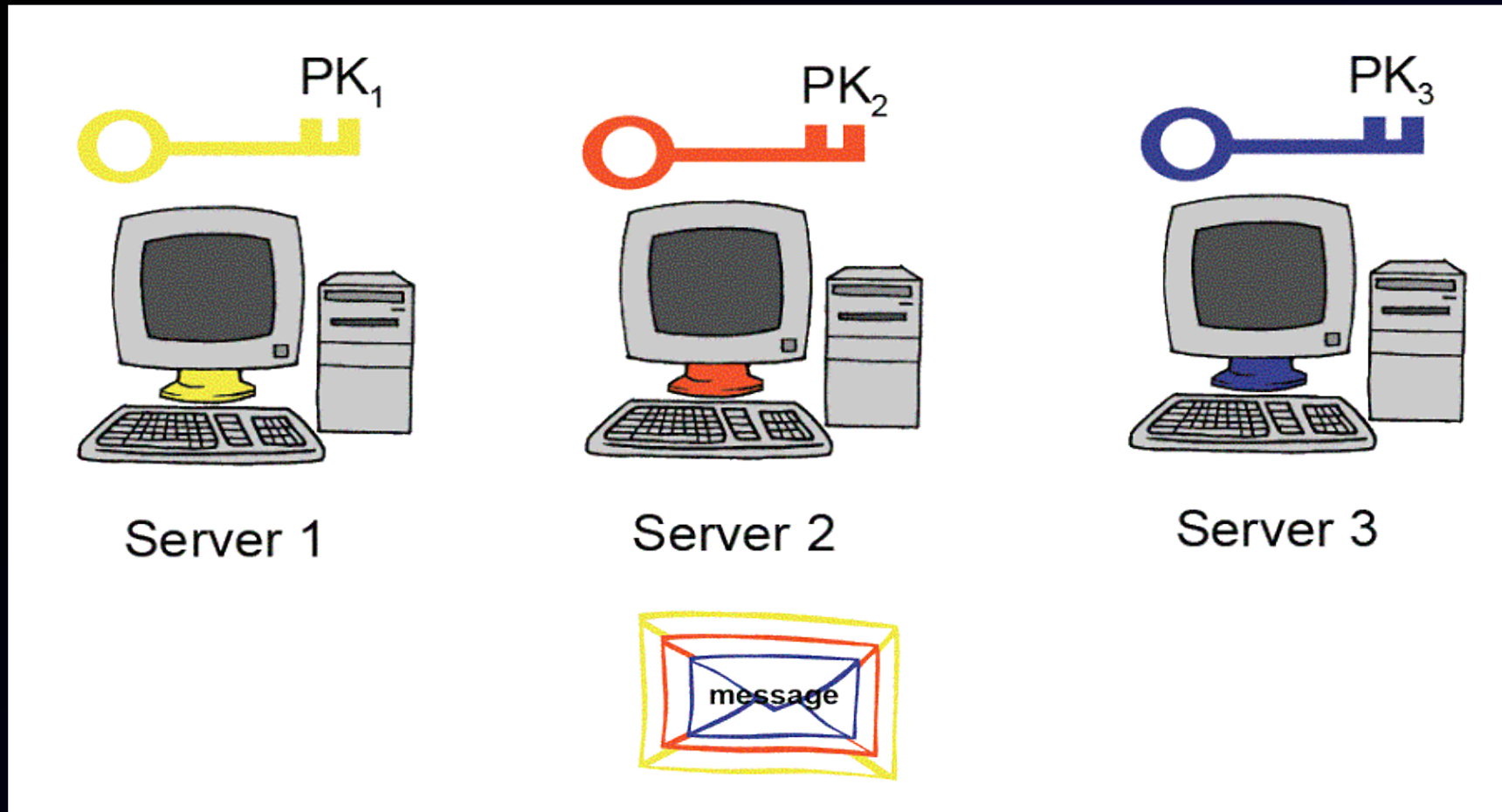
# MIX NETS

A **Mix net** hides the correspondence between a **batch** of input and output messages



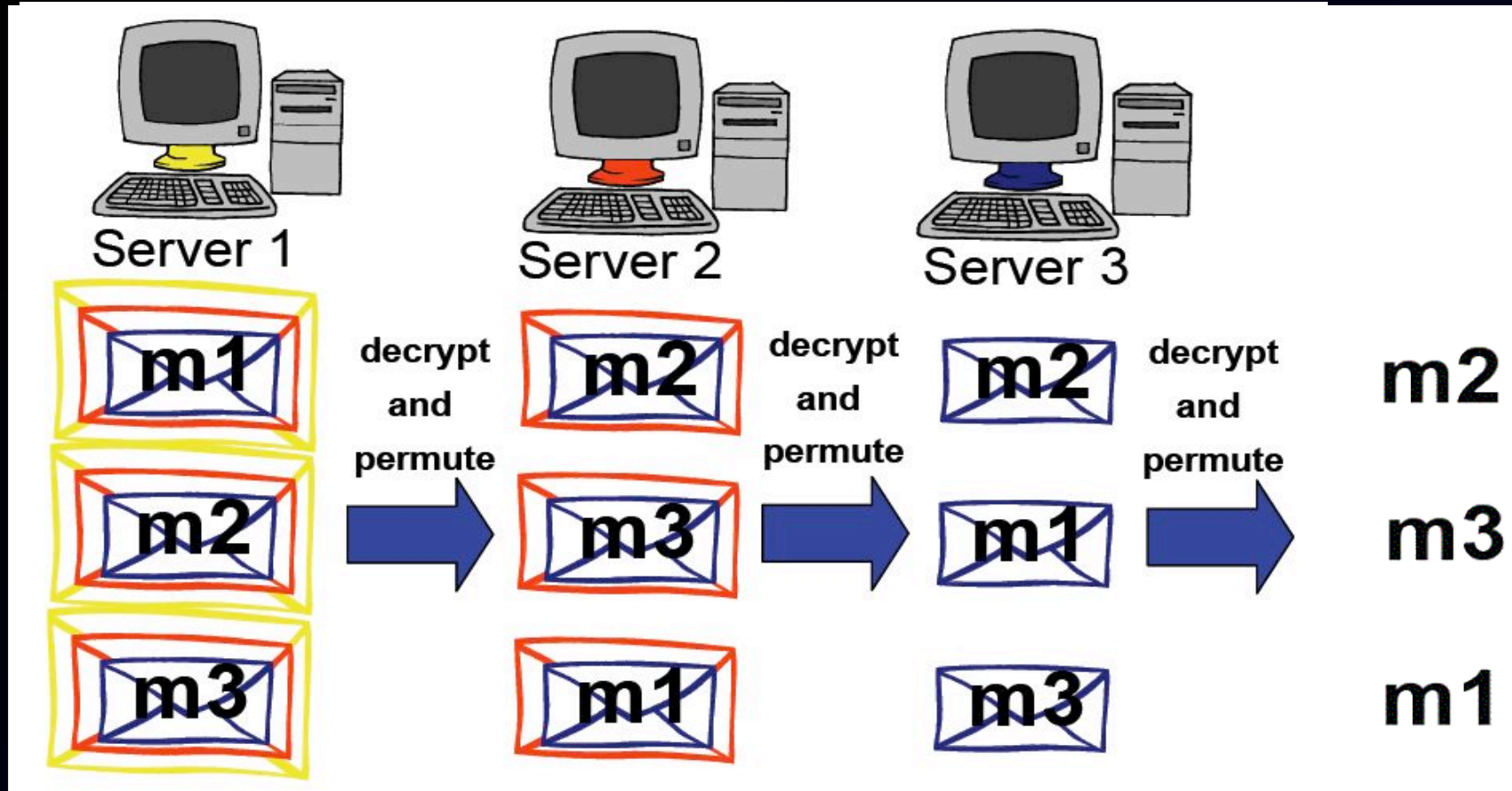
Active attacks include "N-1" and corrupting the mix.

# CASCADE MIX



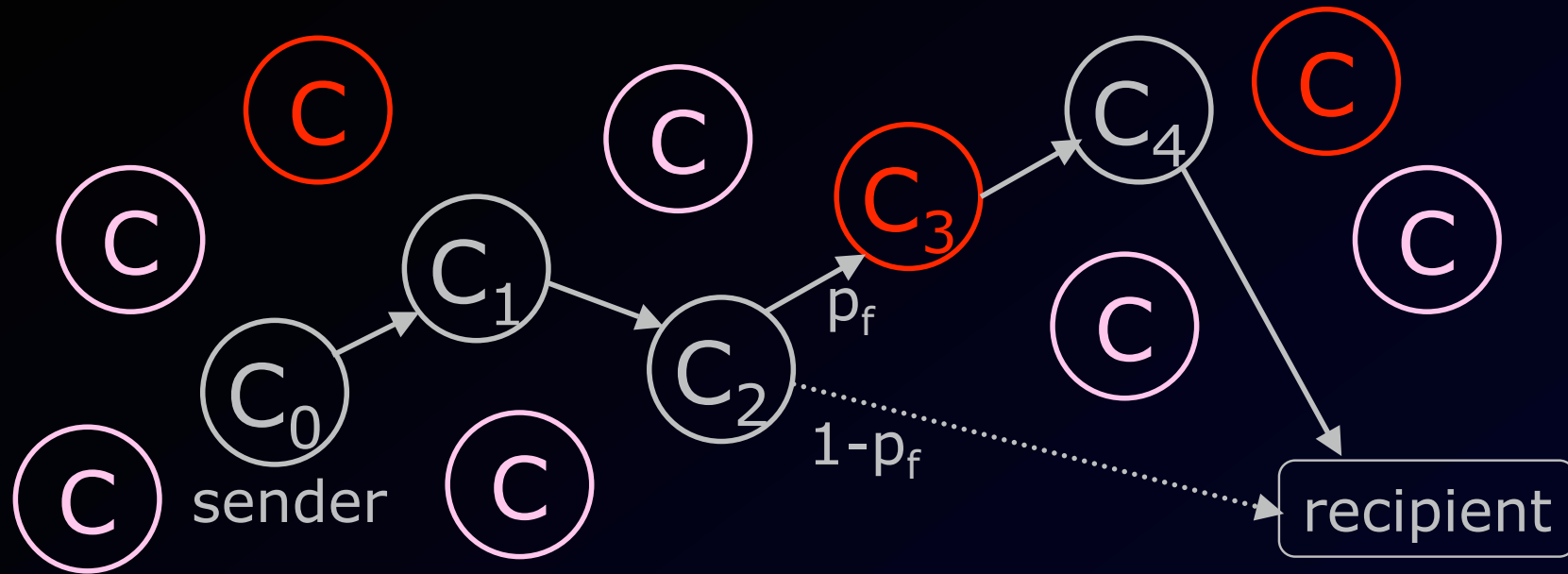
$$\text{Ciphertext} = E_{PK_1}[E_{PK_2}[E_{PK_3}[\text{message}]]]$$

# CASCADE MIX



**Examples: Type II (Mixmaster) and III (Mixminion) remailers. Java Anon Proxy.**

# LOW LATENCY: CROWDS



**Crowds uses peers to provide anonymity:**

- **The sender randomly chooses another member of the crowd, and forwards the message.**
- **After receiving a message, an honest peer:**
  - **With probability  $P_f$  routes to another member of the crowd**
  - **With probability  $1- P_f$  sends directly to the recipient**

# CROWDS AND ANONYMITY

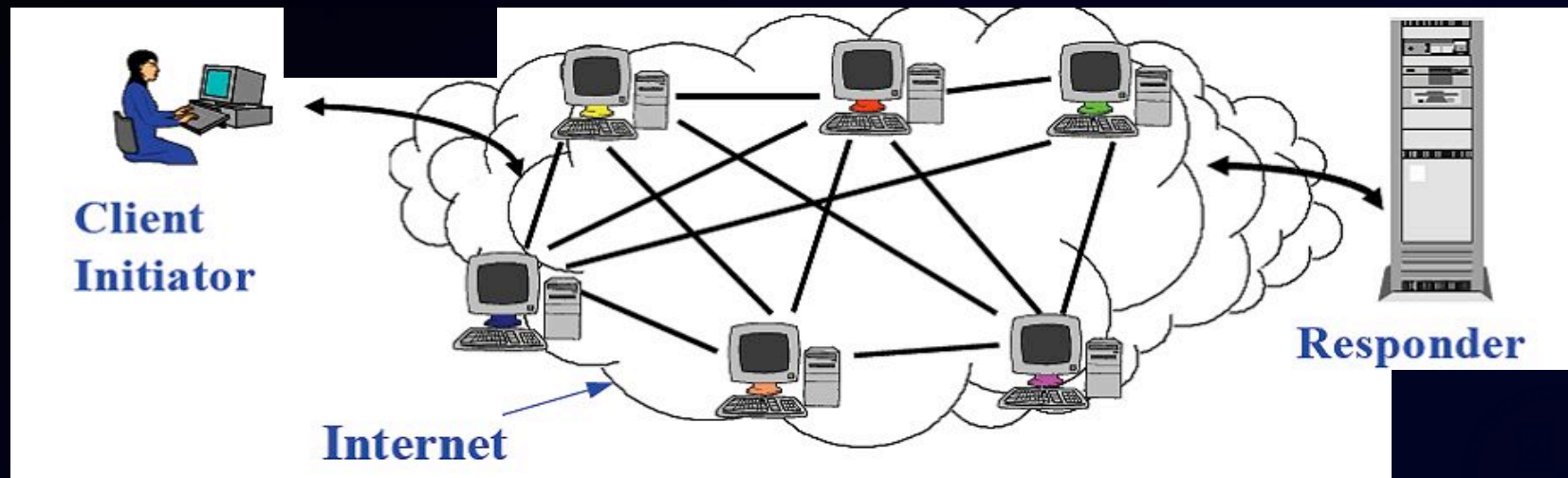
Crowds gives **probable innocence\*** against an adversary who controls a few peers.

\*When  $P_f$  is set correctly, the probability that a peer forwarding a message is the actual sender is at most 50%.

An adversary can defeat this by making up many "identities" or controlling many IP addresses – this is called the **Sybil** attack.

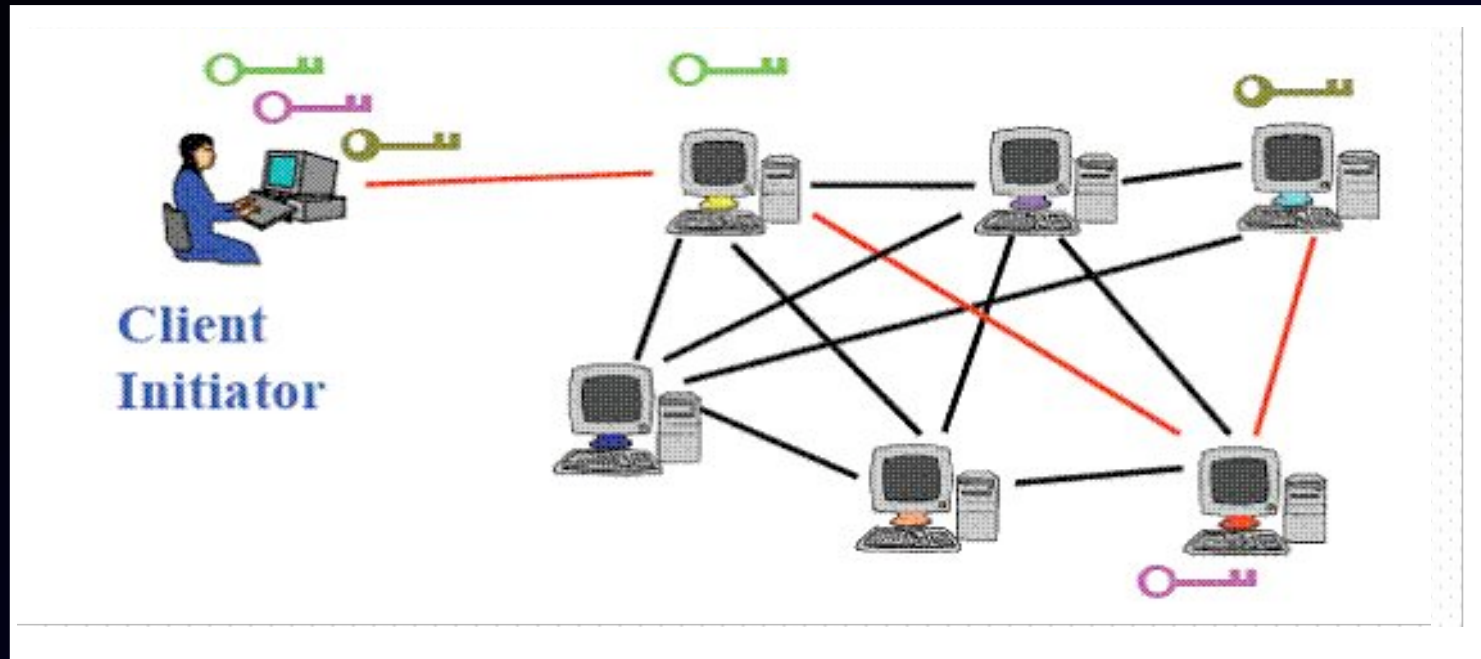
# ONION ROUTING

- **Onion routing** schemes use crypto to make eavesdropping harder.
- To send message  $m$ :
  - pick a random subset of onion routers  $R_1 \dots R_n$ ;
  - get their public keys  $PK_1 \dots PK_n$ .
  - Form an **onion**:  $E_{PK_1}(R_2, E_{PK_2}(R_3, \dots E_{PK_n}(\text{Bob}, M)))$



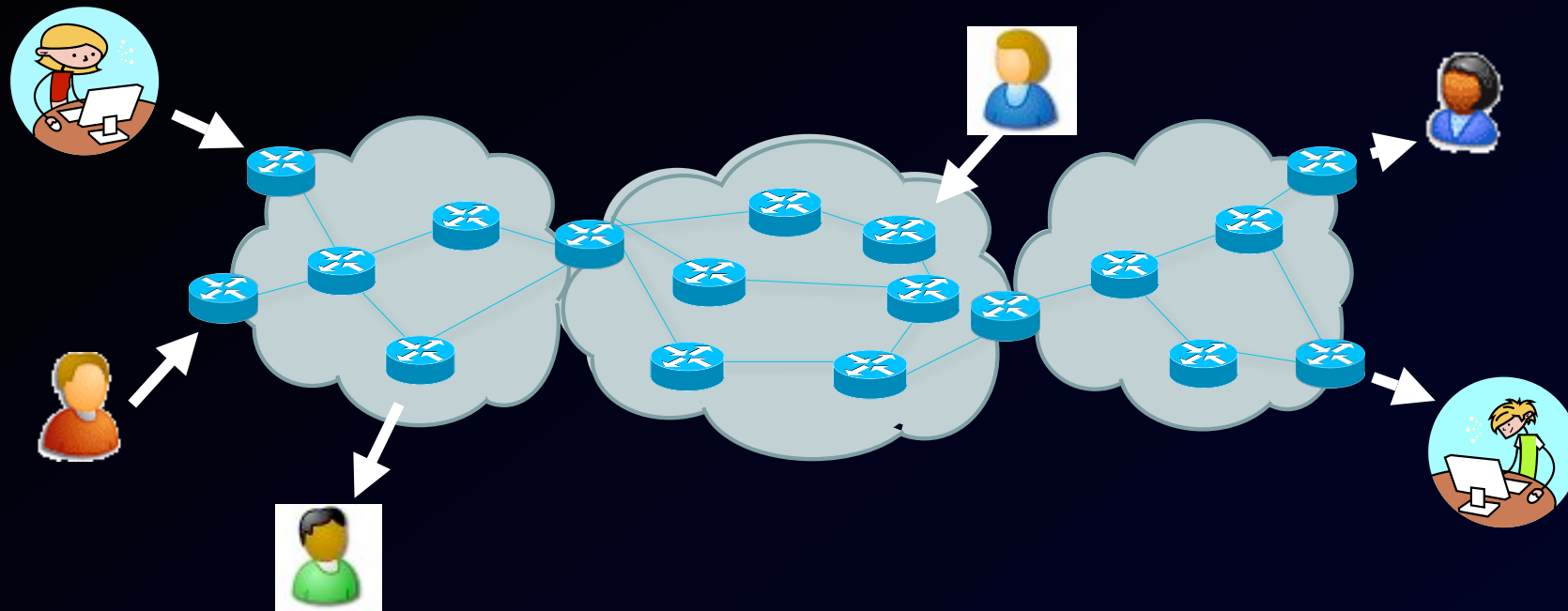
# TOR

- ... is an efficient, large-scale onion-routing scheme.
- ... uses authenticated DH key exchange to build **circuits** that are encrypted with symmetric keys.
- ... aims for security against compromised routers.



# INTERSECTION ATTACKS

Eve can see a few peoples' communications, but not all.



**Intersection attacks** allow Eve to decide which pairs are probably communicating from when they send and receive messages only.

# ANONYMOUS BROADCAST



**Problems: efficiency, spoofing, forward security, deniability, reliability...**

# DINING CRYPTOGRAPHERS

$$\begin{aligned} X &= S_A \oplus S_B \oplus S_C \oplus S_D \\ &= X_A \oplus X_B \oplus X_C \oplus X_D \\ &= M \end{aligned}$$



**A**

$$\begin{aligned} X_A &= M; K_{AB}, K_{AC}, K_{AD} \\ S_A &= K_{AB} \oplus K_{AC} \oplus K_{AD} \oplus X_A \end{aligned}$$



**B**

$$\begin{aligned} X_B &= 0; K_{AB}, K_{BC}, K_{BD} \\ S_B &= K_{AB} \oplus K_{BC} \oplus K_{BD} \oplus X_B \end{aligned}$$



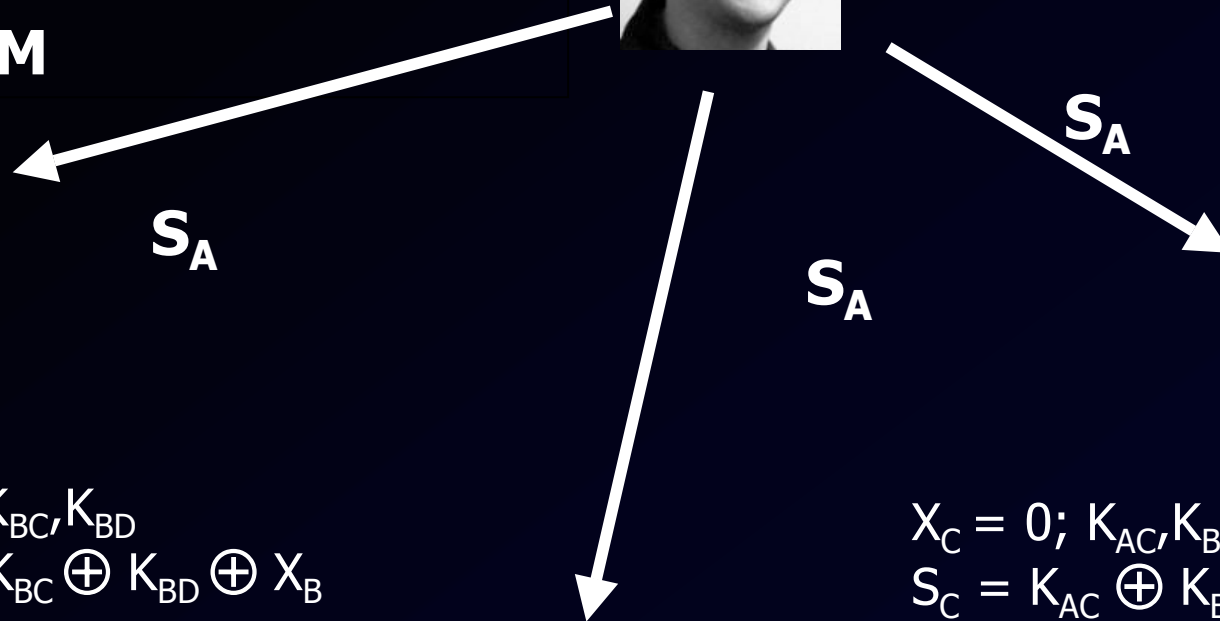
**C**

$$\begin{aligned} X_C &= 0; K_{AC}, K_{BC}, K_{CD} \\ S_C &= K_{AC} \oplus K_{BC} \oplus K_{CD} \oplus X_C \end{aligned}$$



**D**

$$\begin{aligned} X_D &= 0; K_{AD}, K_{BD}, K_{CD} \\ S_D &= K_{AD} \oplus K_{BD} \oplus K_{CD} \oplus X_D \end{aligned}$$



# ANONYMITY PROBLEMS

- **Denial of Service becomes an attack on confidentiality**
- **Popularity (usability, efficiency, reliability, cost)**
- **Abuse**
  - **flooding discussion forums, mailbombing, etc.**
  - **security disclosures, etc...**
  - **swapping CDs/DVDs; credit cards; harassing emails, illicit content...**
- **Connection anonymity vs content anonymity**