# Applied Cryptography and Network Security CS 1653

Summer 2023

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Prof. Adam Lee's CS1653 slides.)

## Announcements

- Please schedule Project Phase 3 Demo with Pratik as soon as possible
- Phase 3 Peer Evaluation Survey is up on CATME
  - Due this Friday
- Homework 9 due this Friday @ 11:59 pm
- Programming Assignment 2 due this Friday
- Project Phase 4 Due on 7/31 @ 11:59 pm
  - Teams must meet with me on or before Thursday 7/27
- Midterm grades have been posted
  - Question reattempts up to 10 points

## Final Exam

- Take home: download from GradeScope, (print), solve, (scan), and upload to GradeScope
- 72 hours time frame
  - No class on Wednesday
- open-book and open-notes
- some overlap with midterm topics
- Please check study guide on Canvas

## So what? A high-value target will always fall eventually

Sometimes compromising one machine/service is just the beginning

- Compromising a service can violate the assumptions of other entities in the system
- Easily bootstrap the compromise of any machine that trusts the compromised service
- Spread to more vulnerable machines

Once integrity of mechanism is violated, all bets are off

Viruses and worms are malware that propagate

## Viruses and Worms

Definitions

***Case studies:***

- The Brain virus
- The Morris worm
- Code Red

Into the future

## Malicious logic is a set of instructions that cause an organization's security policy to be violated

One common type of malicious logic is the Trojan horse, which is a program that has a well-known overt effect, and an unknown covert effect.

*Example:* NetBus

- Allows an attacker to remotely administer a Windows NT box
- Remote admin code was bundled with games/"fun" programs

Not all Trojan horses are so easy to spot…

*Example:* Thompson's login program

- Modification to UNIX login program that accepted a default password
- Modify compiler to insert default case when compiling the login program
  - Backdoor is not visible in login source code
  - Backdoor code is visible in compiler code
- Modify compiler to insert above code if the compiler is being compiled
- Install malicious compiler binary, and benign compiler source
  - Backdoor invisible in all source code

## The problem with a Trojan horse is that you need to convince the victim to install the host program

A computer virus is a program that inserts itself into one or more files and then performs some action

The term computer virus was coined by Fred Cohen and his advisor Len Adelman at USC in 1983

- Cohen's thesis is one of the few theoretical results regarding viruses
- Key result: No algorithm can detect computer viruses precisely
- This is why virus scanners are largely heuristic…

Why are viruses called viruses?

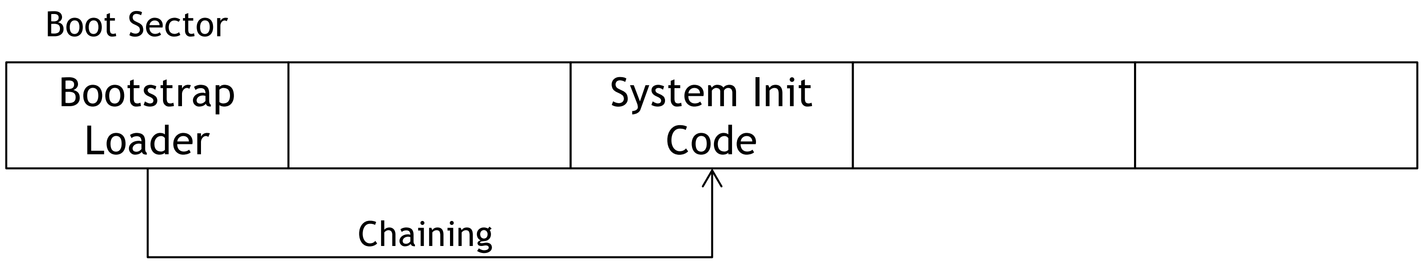
Because they self-replicate by attaching themselves to host programs!

In the days before widespread Internet availability, virus writers had to get creative to enable the spread of a virus beyond one system

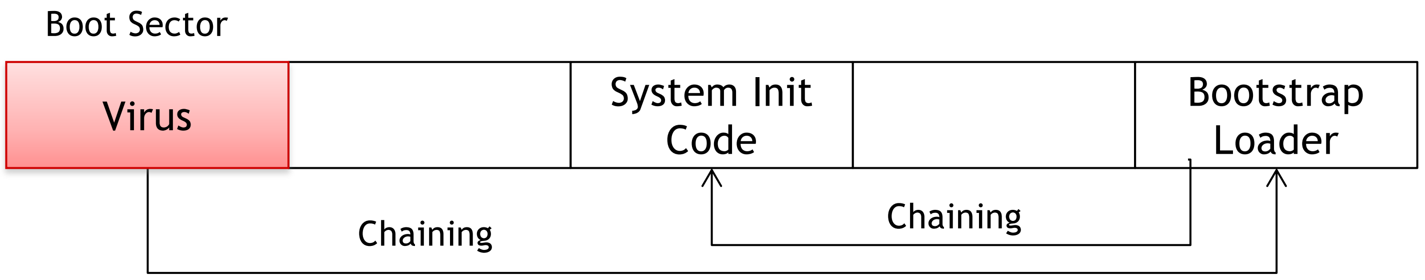
# Some viruses spread by infecting a drive’s boot sector

How does the boot process work?

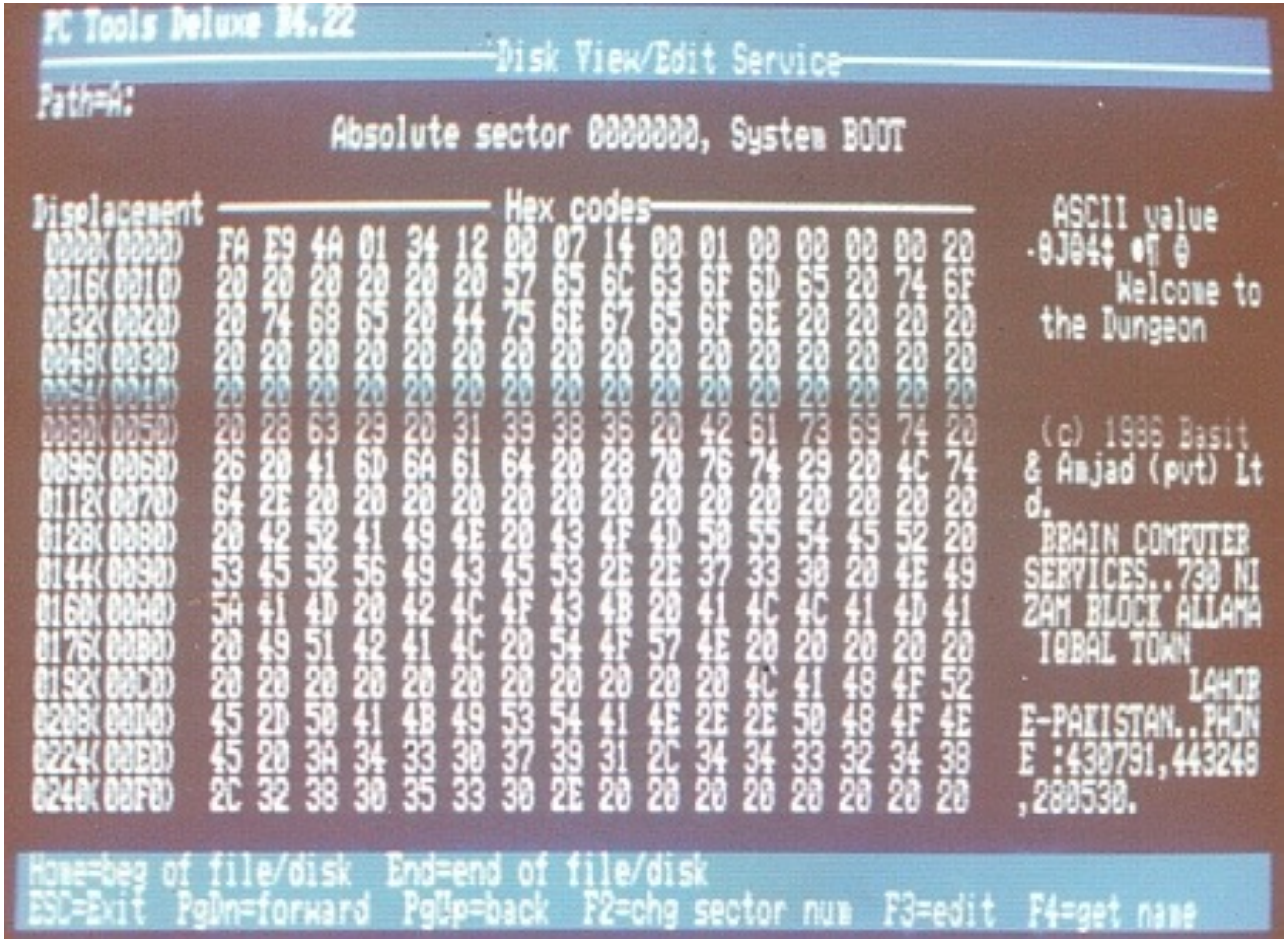
- System firmware reads a specific disk sector (the boot sector) into memory
- The system then jumps to that memory location
- The boot program then begins loading the OS



A boot sector virus hijacks this process to facilitate its spread



# Case Study:  The Brain virus



```
PC Tools Deluxe R4.22
                              Disk View/Edit Service
Path=A:
                    Absolute sector 0000000, System BOOT

Displacement  ------------ Hex codes -------------------      ASCII value
 0000(0000)  FA E9 4A 01 34 12 00 07 14 00 01 00 00 00 00 20  ·ΘJ0♦$ ♦¶ Θ
 0016(0010)  20 20 20 20 20 20 57 65 6C 63 6F 6D 65 20 74 6F        Welcome to
 0032(0020)  20 74 68 65 20 44 75 6E 67 65 6F 6E 20 20 20 20   the Dungeon
 0048(0030)  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
 0064(0040)  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
 0080(0050)  20 28 63 29 20 31 39 38 36 20 42 61 73 69 74 20   (c) 1986 Basit
 0096(0060)  26 20 41 6D 6A 61 64 20 28 70 76 74 29 20 4C 74  & Amjad (pvt) Lt
 0112(0070)  64 2E 20 20 20 20 20 20 20 20 20 20 20 20 20 20  d.
 0128(0080)  20 42 52 41 49 4E 20 43 4F 4D 50 55 54 45 52 20   BRAIN COMPUTER
 0144(0090)  53 45 52 56 49 43 45 53 2E 2E 37 33 30 20 4E 49  SERVICES..730 NI
 0160(00A0)  5A 41 4D 20 42 4C 4F 43 4B 20 41 4C 4C 41 4D 41  ZAM BLOCK ALLAMA
 0176(00B0)  20 49 51 42 41 4C 20 54 4F 57 4E 20 20 20 20 20   IQBAL TOWN
 0192(00C0)  20 20 20 20 20 20 20 20 20 20 20 4C 41 48 4F 52             LAHOR
 0208(00D0)  45 2D 50 41 4B 49 53 54 41 4E 2E 2E 50 48 4F 4E  E-PAKISTAN..PHON
 0224(00E0)  45 20 3A 34 33 30 37 39 31 2C 34 34 33 32 34 38  E :430791,443248
 0240(00F0)  2C 32 38 30 35 33 30 2E 20 20 20 20 20 20 20 20  ,280530.

Home=beg of file/disk  End=end of file/disk
ESC=Exit  PgDn=forward  PgUp=back  F2=chg sector num  F3=edit  F4=get name
```

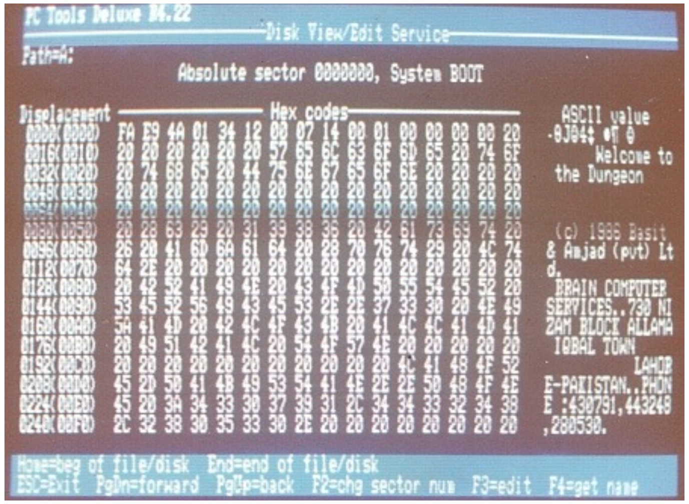
http://www.noulakaz.net/weblog/2007/02/24/welcome-to-the-dungeon/

# The Brain virus was one of the first, and most carefully studied, computer viruses

Brain was a boot sector virus that originated in Pakistan in 1986

Two brothers, Basit and Amjad Farooq Alvi, supposedly wrote Brain to protect medical software that they wrote from copyright infringement

However, the Brain virus contained no malicious payload and actually advertised the brothers’ contact information!



```
PC Tools Deluxe R4.22
                         Disk View/Edit Service
Path=A:
                 Absolute sector 0000000, System BOOT

Displacement  ------------------ Hex codes ----------------------    ASCII value
 0000(0000)   FA E9 4A 01 34 12 00 07 14 00 01 00 00 00 00 20    ·ΘJ☺4‡ •¶ Θ
 0016(0010)   20 20 20 20 20 20 57 65 6C 63 6F 6D 65 20 74 6F          Welcome to
 0032(0020)   20 74 68 65 20 44 75 6E 67 65 6F 6E 20 20 20 20     the Dungeon
 0048(0030)   20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
 0064(0040)   20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
 0080(0050)   20 28 63 29 20 31 39 38 36 20 42 61 73 69 74 20     (c) 1986 Basit
 0096(0060)   26 20 41 6D 6A 61 64 20 28 70 76 74 29 20 4C 74    & Amjad (pvt) Lt
 0112(0070)   64 2E 20 20 20 20 20 20 20 20 20 20 20 20 20 20    d.
 0128(0080)   20 42 52 41 49 4E 20 43 4F 4D 50 55 54 45 52 20     BRAIN COMPUTER
 0144(0090)   53 45 52 56 49 43 45 53 2E 2E 37 33 30 20 4E 49    SERVICES..730 NI
 0160(00A0)   5A 41 4D 20 42 4C 4F 43 4B 20 41 4C 4C 41 4D 41    ZAM BLOCK ALLAMA
 0176(00B0)   20 49 51 42 41 4C 20 54 4F 57 4E 20 20 20 20 20     IQBAL TOWN
 0192(00C0)   20 20 20 20 20 20 20 20 20 20 20 4C 41 48 4F 52               LAHOR
 0208(00D0)   45 2D 50 41 4B 49 53 54 41 4E 2E 2E 50 48 4F 4E    E-PAKISTAN..PHON
 0224(00E0)   45 20 3A 34 33 30 37 39 31 2C 34 34 33 32 34 38    E :430791,443248
 0240(00F0)   2C 32 38 30 35 33 30 2E 20 20 20 20 20 20 20 20    ,280530.

Home=beg of file/disk  End=end of file/disk
ESC=Exit  PgDn=forward  PgUp=back  F2=chg sector num  F3=edit  F4=get name
```

It is suspected that Brain was really a cute gimmick to draw attention to their business!

# How did the Brain virus work?

When a system boots from an infected disk:

1. The virus loads itself in upper memory
2. It then resets the user accessible memory boundary to just below itself
3. Brain then mangles the system interrupt table
   - Interrupt 19 (disk read) is reset to point to the Brain code in memory
   - Interrupt 6 (unused) is then set to point to the old code for interrupt 19

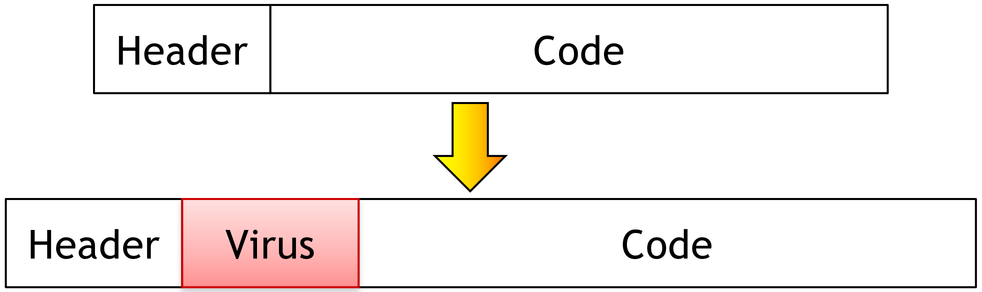
Whenever a disk read occurs

1. Interrupt 19 is triggered and transfers control to the virus
2. If the disk being read is not yet infected, the virus infects it
3. The virus then triggers interrupt 6 to allow the disk read to occur

Although Brain contained no malicious payload, it was used as a template for several more destructive viruses

***Question:*** How do you think that users detected this virus?

# There are many other types of viruses…



***Executable viruses***

- Attach to executable code
- Invoked when code is invoked
- *Example:* Jerusalem

***TSR viruses***

- TSR = Terminate and stay resident
- Virus stays in memory even after host process terminates (syscall interception)
- *Examples:* Brain and Jerusalem

***Macro viruses***

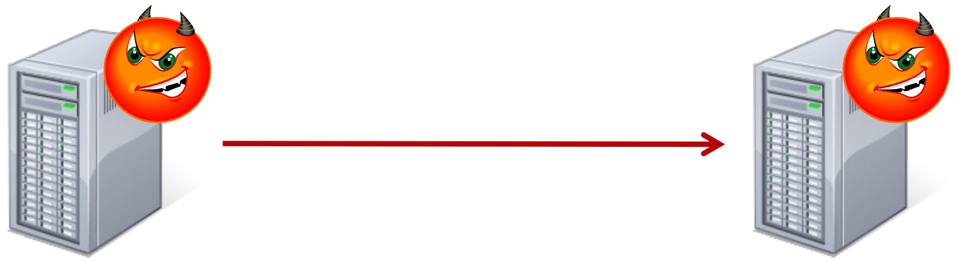
- Infect documents, not executables
- Not bound by system architecture
- *Examples:* Melissa

## How fast can viruses spread?

In the early days, the speed with which a virus could spread was limited to the speed of human interactions

- Trading floppy disks
- Sharing spreadsheets or other documents at work
- Installing Trojan programs passed along by a friend
- …

The growing prevalence of the Internet during the late 80s and early 90s gave rise to computer worms, which are viruses capable of spreading themselves across many machines



The result: Much speedier infection rates!

## Case Study: The Morris worm

The first major Internet worm was released in 1988

- Written by Robert Tappan Morris
- Launched around 5:00 PM on 2nd November 1988
- Originated at Cornell University

The worm was purportedly written to assess the size of the Internet and had no malicious payload

Unfortunately, unbounded replication of the worm brought many systems down

This worm used many techniques that we have already studied…

Eugene H. Spafford, "The Internet Worm: Crisis and Aftermath," Communications of the ACM, 32(6): 678-687, June 1989.

## How did it spread?

Rather than reinvent the wheel, the Morris worm leveraged three well-known vulnerabilities to enable its spread across machines running Berkeley and Sun UNIX

***Method 1:*** fingerd

- fingerd provides a lookup service for users' public contact information
- The version of fingerd running on many systems had a buffer overflow due to the use of the unchecked `gets()` routine

***Method 2:*** A bug in sendmail

- sendmail is a popular e-mail routing program
- If operating in DEBUG mode, sendmail allows system commands to be transmitted over SMTP

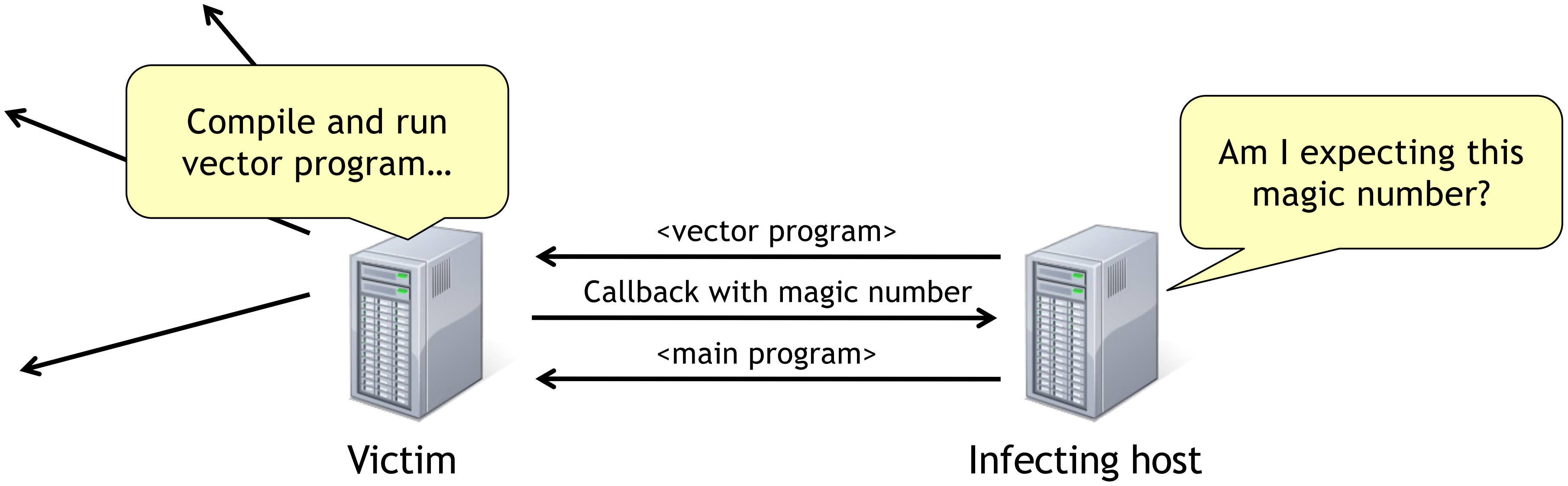
***Method 3:*** Weak password security

- Many passwords are weak and can be broken with simple guessing
- The rsh protocol allows trusted users/hosts to bypass authentication

# How did an infection proceed?

The worm consisted of two programs:  a short vector (i.e., infection) program, and the main program

The vector program was 99 lines of C code, transferred using one of the previous three known exploit techniques


Compile and run vector program...
Am I expecting this magic number?
<vector program>
Callback with magic number
<main program>
Victim
Infecting host

If the (binary) attack programs would not run on the victim, the vector would delete everything to cover its tracks

## What did the main program do?

The main program first gathered information about network interfaces and hosts on the local network, which was randomized to provide a set of hosts to attack

The main program then entered a simple state machine

- Read `/etc/hosts.equiv` and `/.rhosts` to look for trusted hosts to add to table
- Try to break user passwords using simple choices
- Try to break user passwords using a dictionary of 432 words
- Brute force user passwords using entire UNIX online dictionary

**If a password was broken, it was used in conjunction with rsh to infect other hosts**

Each state was run for short periods of time, between which the main program attempted to infect other hosts in the list to attack

# The Morris worm actually took steps to prevent overloading a particular host

The worm would periodically check for copies of the worm running on the same host by connecting to a predetermined TCP socket

If another worm was found, one of the two would randomly quit

However, this didn't work terribly well…

- Race conditions in the code sometimes prevented worms from connecting
- Furthermore, 1 in 7 worms became immortal to prevent fake kills

Result: Many hosts had several copies of the worm running

---

It is interesting to note that the worm occasionally forked itself

- This gave the worm a new PID to prevent detection
- Also prevented the worms priority from downgrading over time

# Outcomes of the Morris worm

Experts think that upwards of 6,000 hosts were infected with the Morris worm, though this number is an estimate

The Morris worm brought network security to the forefront

- More regular software patching/updating
- Broader proliferation of shadow password files
- Inspired much intrusion detection research

In response to this incident, DARPA funded CERT/CC to monitor computer vulnerabilities, and manage incident response

Robert T. Morris

- Was the first person convicted under the 1986 Computer Fraud and Abuse Act
- Is now a professor at MIT ☺

## Case Study: Code Red

Although the Morris worm was the first widespread worm, it was certainly not the last!

Software is being developed at an astounding rate

- Software is getting more complex
- Higher complexity leads to more bugs
- More bugs means more potential for exploits

The Code Red worm (v2) was launched on July 19, 2001

- Exploited a bug in Microsoft IIS server
- Infected over 359,000 hosts in under 14 hours!

Although this worm was released 13 years after the Morris worm, it is structurally very similar…

## How did Code Red work?

Like the Morris worm, Code Red utilized a buffer overflow to propagate

- Microsoft IIS version 4.0 or 5.0 with the Indexing service installed

-
  ```
  /default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
  NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
  NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
  NNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%
  u7801%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3%u0003%u8b00%u531
  b%u53ff%u0078%u0000%u00=a
  ```

After a host was infected, it spawned 300-600 threads that would:

- Randomly choose an IP address and attempt to connect
- If success, attempt the above buffer overflow

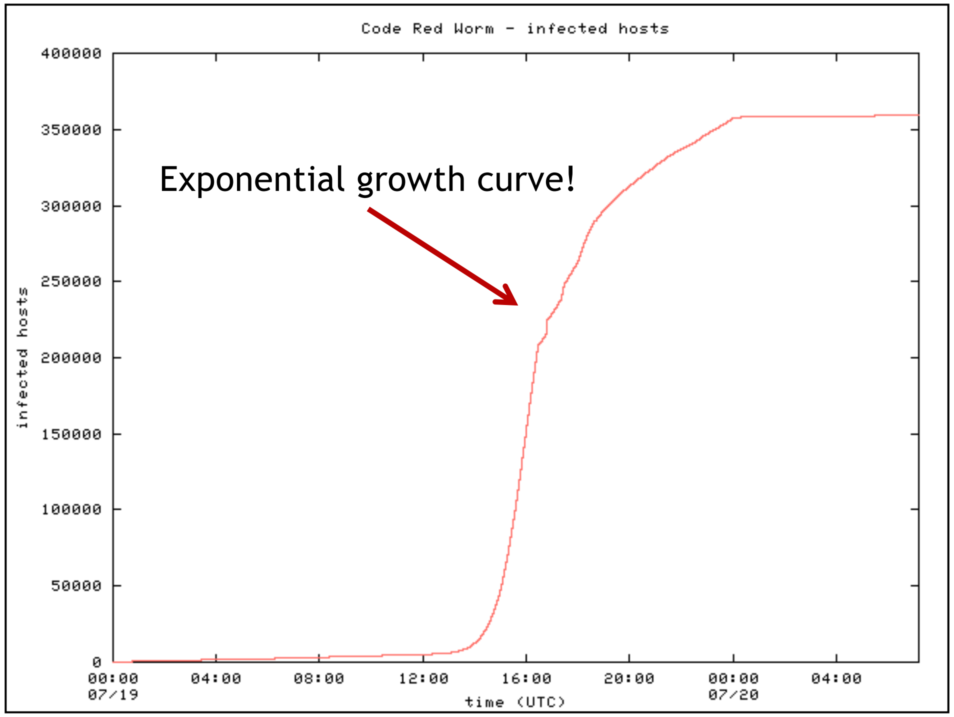
Code Red’s behavior was dependent on the day of the month

- Day 1 - 19: Randomly infect other hosts
- Day 20 - 27: Carry out a packet-flooding denial of service attack on the IP address of www1.whitehouse.gov
- Day 28 - <end of month>: Sleep

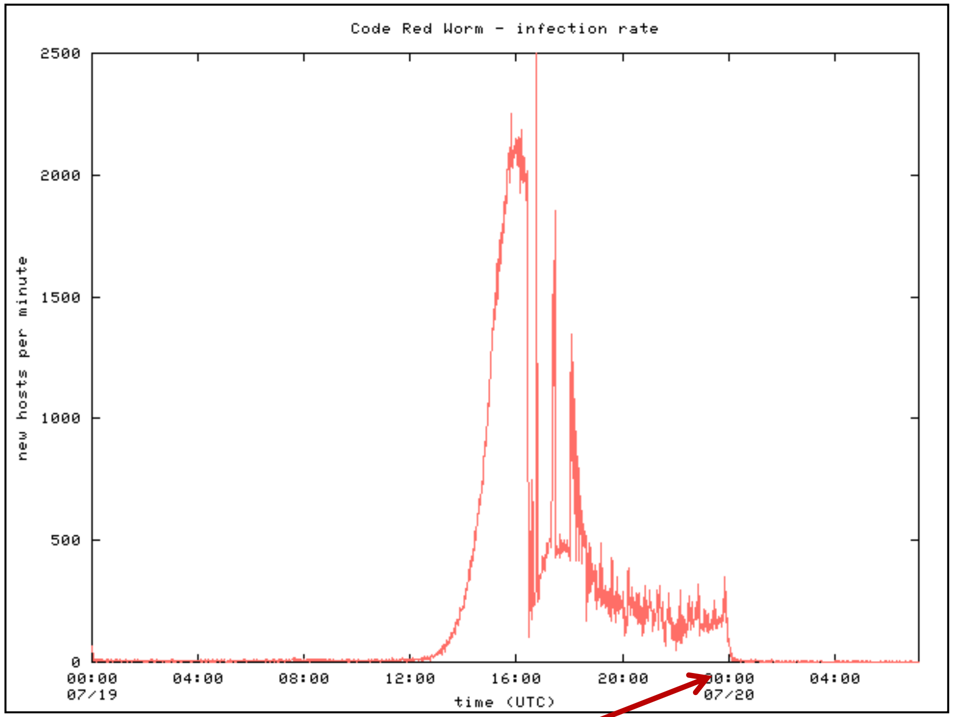
CERT® Advisory CA-2001-19 "Code Red" Worm Exploiting Buffer Overflow In IIS Indexing Service DLL
http://www.cert.org/advisories/CA-2001-19.html

# Code Red spread at an alarming rate

http://www.caida.org/research/security/code-red/coderedv2_analysis.xml



Exponential growth curve!

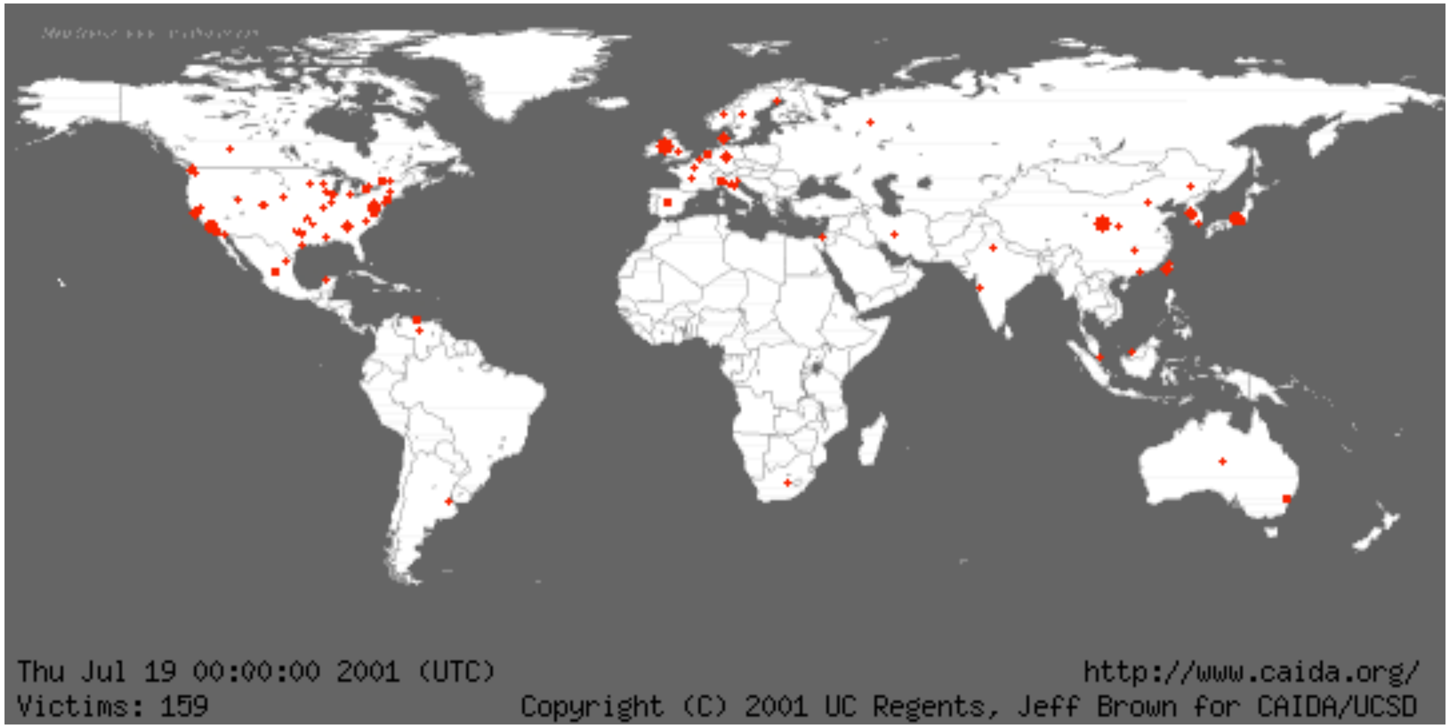


Only 13 hours were needed to infect over 359,000 hosts!

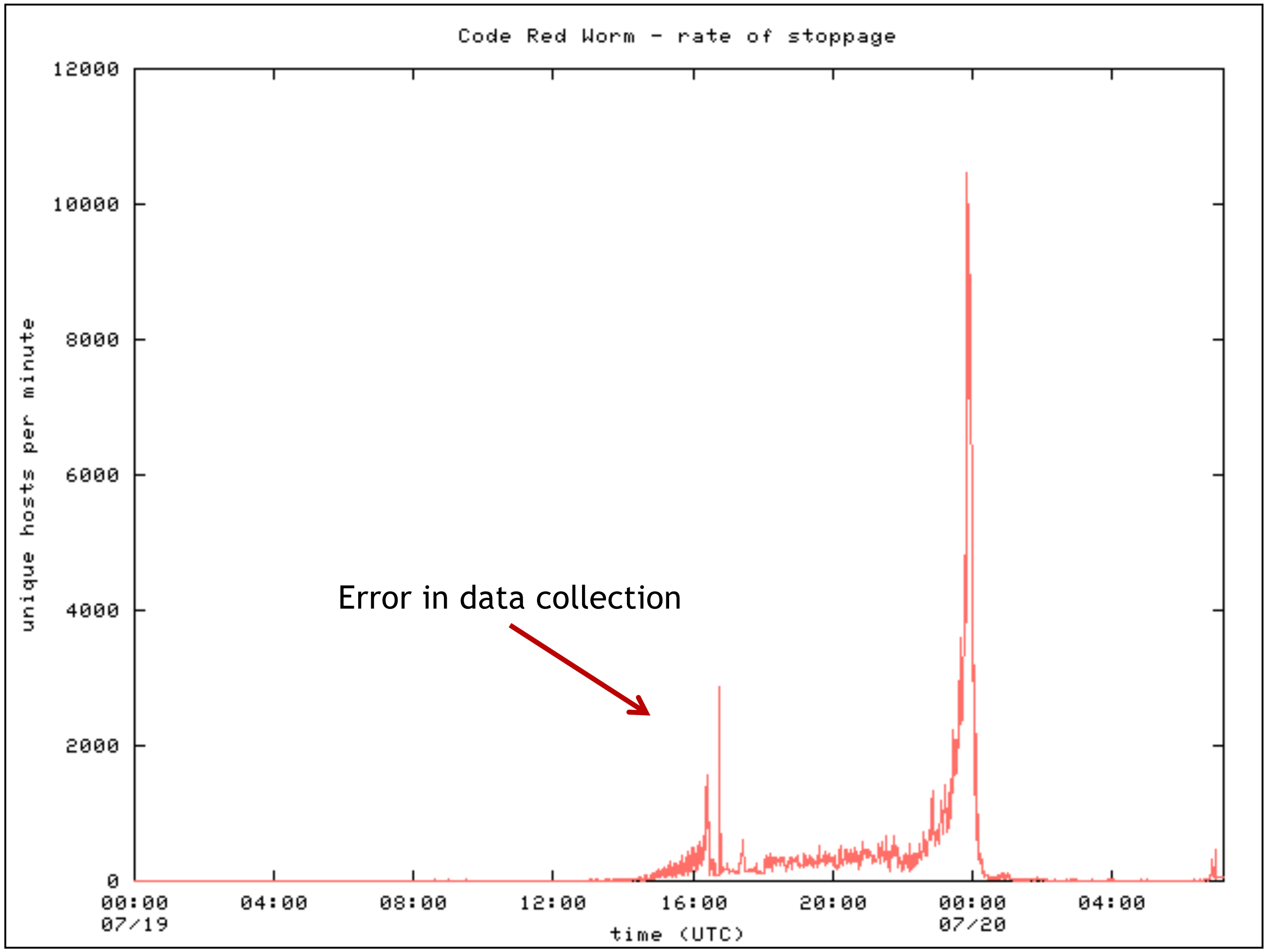
Recall: Code Red stopped infecting on the 20th of each month

---

***Question:*** Why was the spread rate of Code Red so much higher than that of the Morris worm?


Thu Jul 19 00:00:00 2001 (UTC)
Victims: 159
http://www.caida.org/
Copyright (C) 2001 UC Regents, Jeff Brown for CAIDA/UCSD

## Throughout the day, many hosts were patched or firewalled to help prevent the spread of Code Red



We got lucky here…

# Characterizing the Attack

*Top 10 Countries*

| *Country* | *Hosts Infected* | *Percentage* |
|---|---:|---:|
| United States | 157694 | 43.91 |
| Korea | 37948 | 10.57 |
| China | 18141 | 5.05 |
| Taiwan | 15124 | 4.21 |
| Canada | 12469 | 3.47 |
| United Kingdom | 11918 | 3.32 |
| Germany | 11762 | 3.28 |
| Australia | 8587 | 2.39 |
| Japan | 8282 | 2.31 |
| Netherlands | 7771 | 2.16 |

# Characterizing the Attack

*Top 10 TLDs*

| *TLD* | *Hosts Infected* | *Percentage* |
|---|---:|---:|
| Unknown | 169584 | 47.22 |
| net | 67486 | 18.79 |
| com | 51740 | 14.41 |
| edu | 8495 | 2.37 |
| tw | 7150 | 1.99 |
| jp | 4770 | 1.33 |
| ca | 4003 | 1.11 |
| it | 3076 | 0.86 |
| fr | 2677 | 0.75 |
| nl | 2633 | 0.73 |

No reverse lookup entry

Consistent with the overall representation of these TLDs on the Internet

## Characterizing the Attack

*Top 10 Domains*

| *Domain* | *Hosts Infected* | *Percentage* |
|---|---:|---:|
| Unknown | 169584 | 47.22 |
| home.com | 10610 | 2.95 |
| rr.com | 5862 | 1.63 |
| t-dialin.net | 5514 | 1.54 |
| pacbell.net | 3937 | 1.10 |
| uu.net | 3653 | 1.02 |
| aol.com | 3595 | 1.00 |
| hinet.net | 3491 | 0.97 |
| net.tw | 3401 | 0.95 |
| edu.tw | 2942 | 0.82 |

Note: Home and small business ISPs played a huge role in the spread of Code Red!

# Outcomes of Code Red

Main point: We got lucky ☺

- Code Red was a fairly benign worm
- The automatic cut-off date eased the disinfection process
- The worm relied on a flawed DDoS attack strategy

Code Red taught us a number of important lessons

- Home users play a big role in worm propagation
- Homogeneity makes the Internet susceptible to widespread attack
- Even a worm that randomly guesses IP addresses can spread at an alarming rate
- A "release and patch" mentality is detrimental

***Question:*** If a random scanning worm can infect over 359,000 hosts in 13 hours, what could a more directed worm do?

## Researchers have predicted that the spread of flash worms could happen too fast to stop

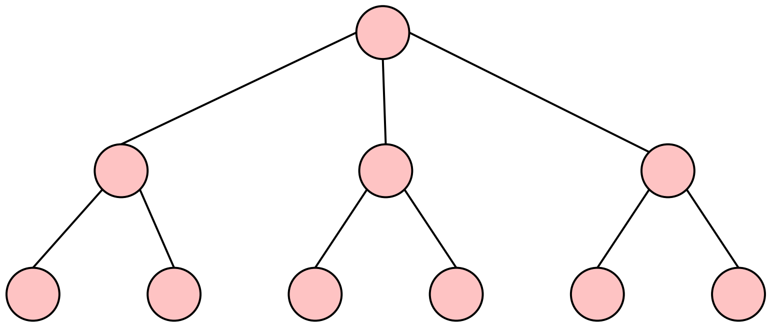
Random probing slows worms down

- Attempts to attack non-existent hosts
- Infecting hosts with minimal ability to infect others
- Hosts can be infected multiple times

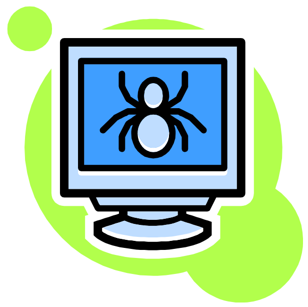
Flash worms seek to spread as quickly as possible!

***Phase 1:*** Find Vulnerable Hosts

***Phase 2:*** Build an optimized attack tree



***Phase 3:*** Infect!



So, how fast could a flash worm spread in the wild?

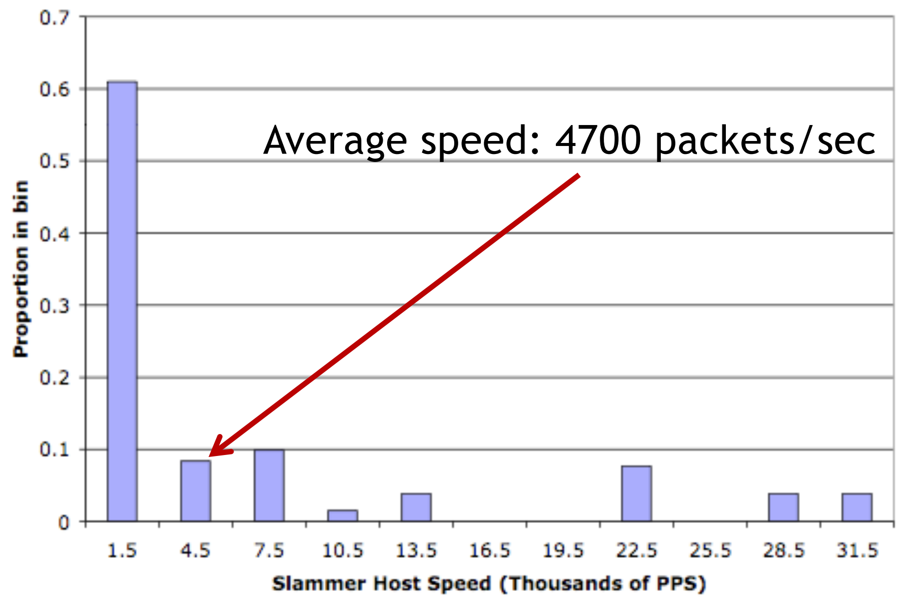
Stuart Staniford, David Moore, Vern Paxson, and Nicholas Weaver, "The Top Speed of Flash Worms," Proceedings of the ACM Workshop on Rapid Malcode (WORM), Oct. 2004.

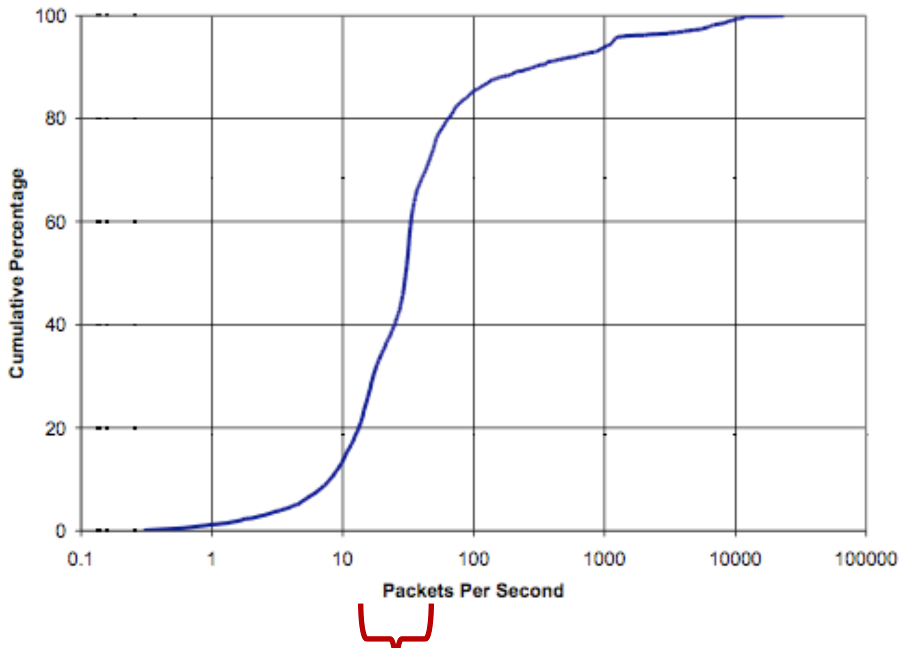
# Predicting a worm means that we first need to characterize it

How small could a flash worm be?

- The Slammer worm used a single 404-byte UDP packet!

How fast could a flash worm propagate?





60% of nodes infected by Witty sent between 11 and 60 1090-byte packets/sec. This would be between 29.67 and 161.88 Slammer-sized packets/sec.

# Flash worms use an optimized distribution tree

Average Internet latency distribution is 103ms.

- Sending Slammer sized packets at 2700pps, 227 packets can be sent before the first infection
- This motivates a wide and shallow infection tree

Time to infection can be modeled as follows:

Number of hosts to infect = N/(A+1)
xA + x = N ➔ x = N/(A+1)

Size of worm packet = W + 4A

Number of addresses sent to each node = A

Latency

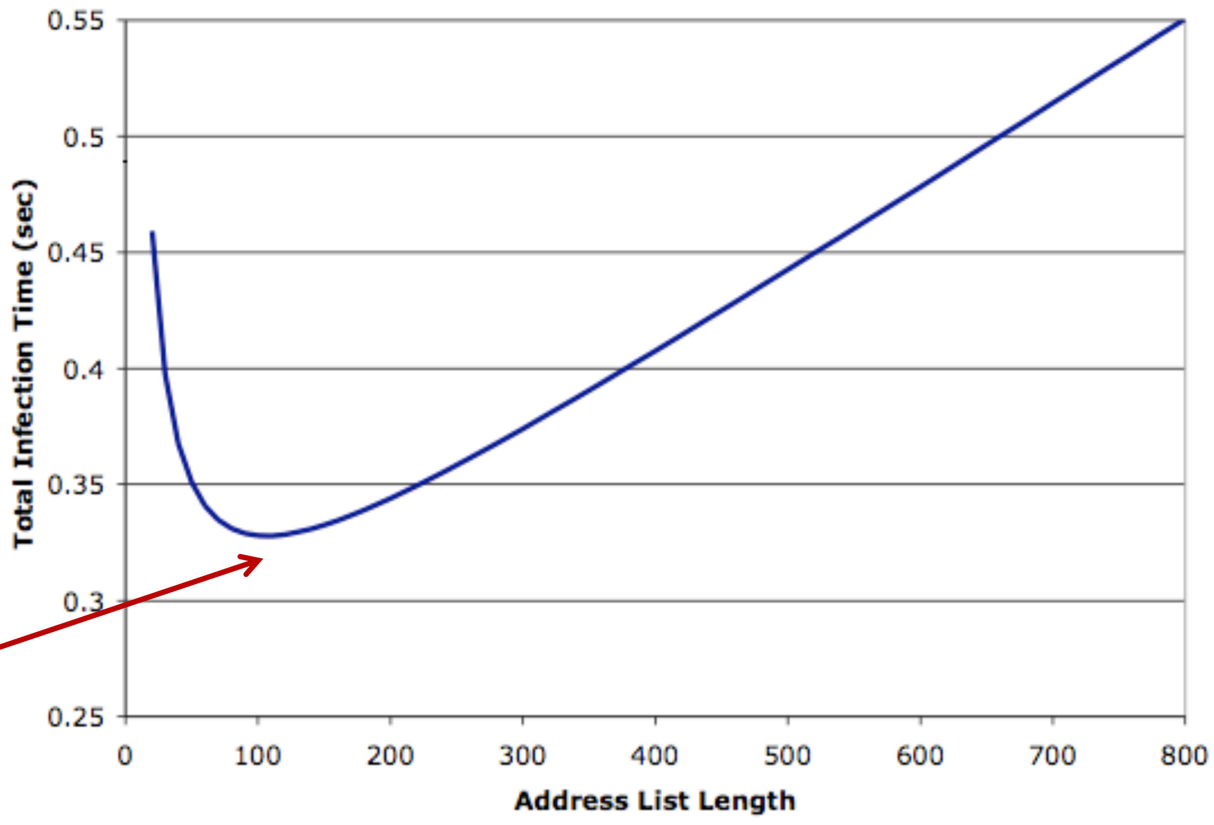
$$t_I = \underbrace{\frac{N(\overbrace{W+4A})}{(A+1)B}}_{\text{Time to infect first level in tree}} + \underbrace{\frac{AW}{b}}_{\text{Time to infect second level in tree}} + \underbrace{2L}_{\text{Parallelized latency}}$$

Bandwidth (B bytes/sec for root node and b for intermediate nodes)

## What is the optimal number of addresses to send to each 2nd level host?

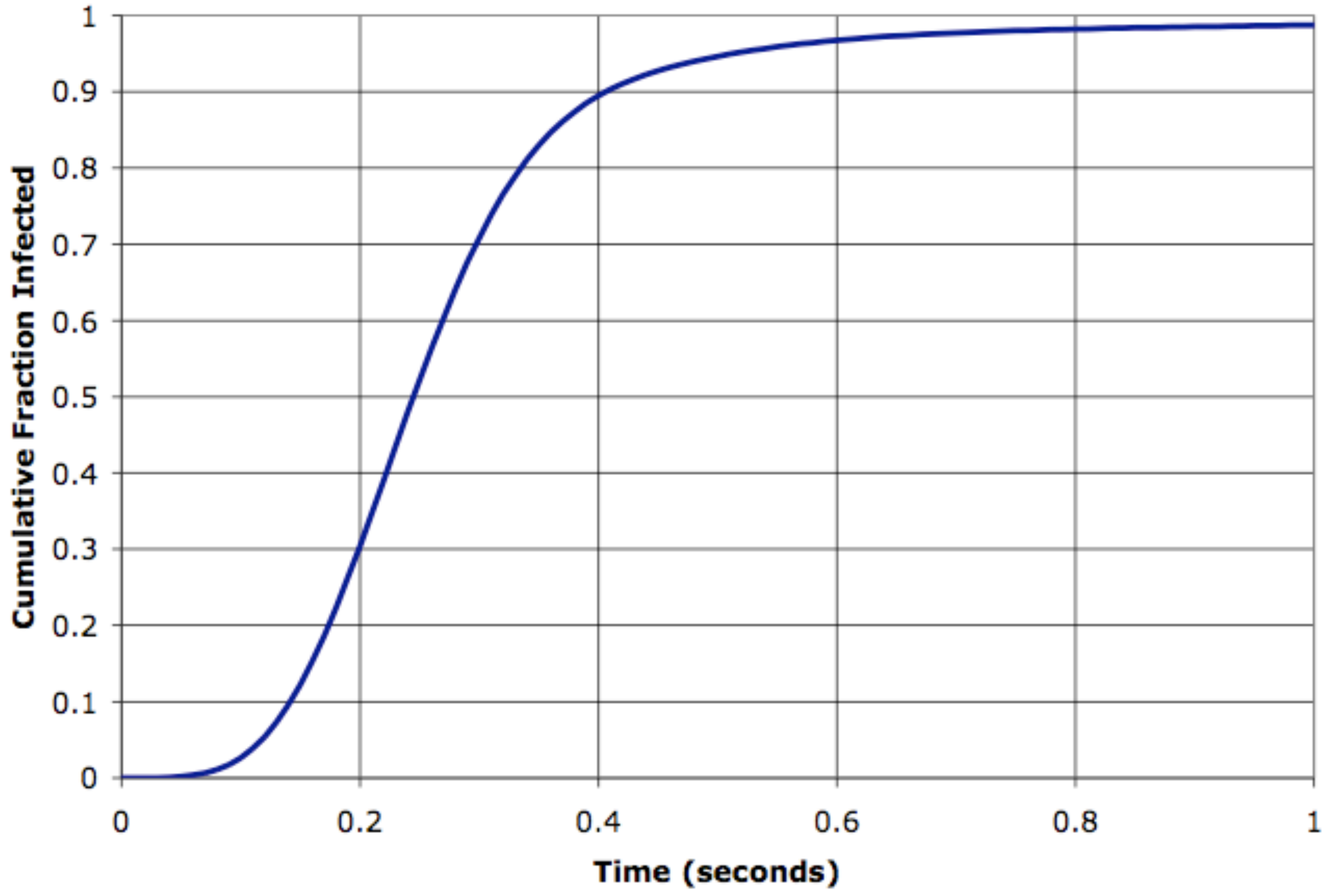
Assumptions:

- N = 1,000,000 hosts to infect
- W = 404 bytes
- Initial link can send 240,000 Slammer-sized packets/sec
  - 75% of a 1 Gbps link
  - B = .75 Gbps
- L = 103 ms
- b = 1 Mbps

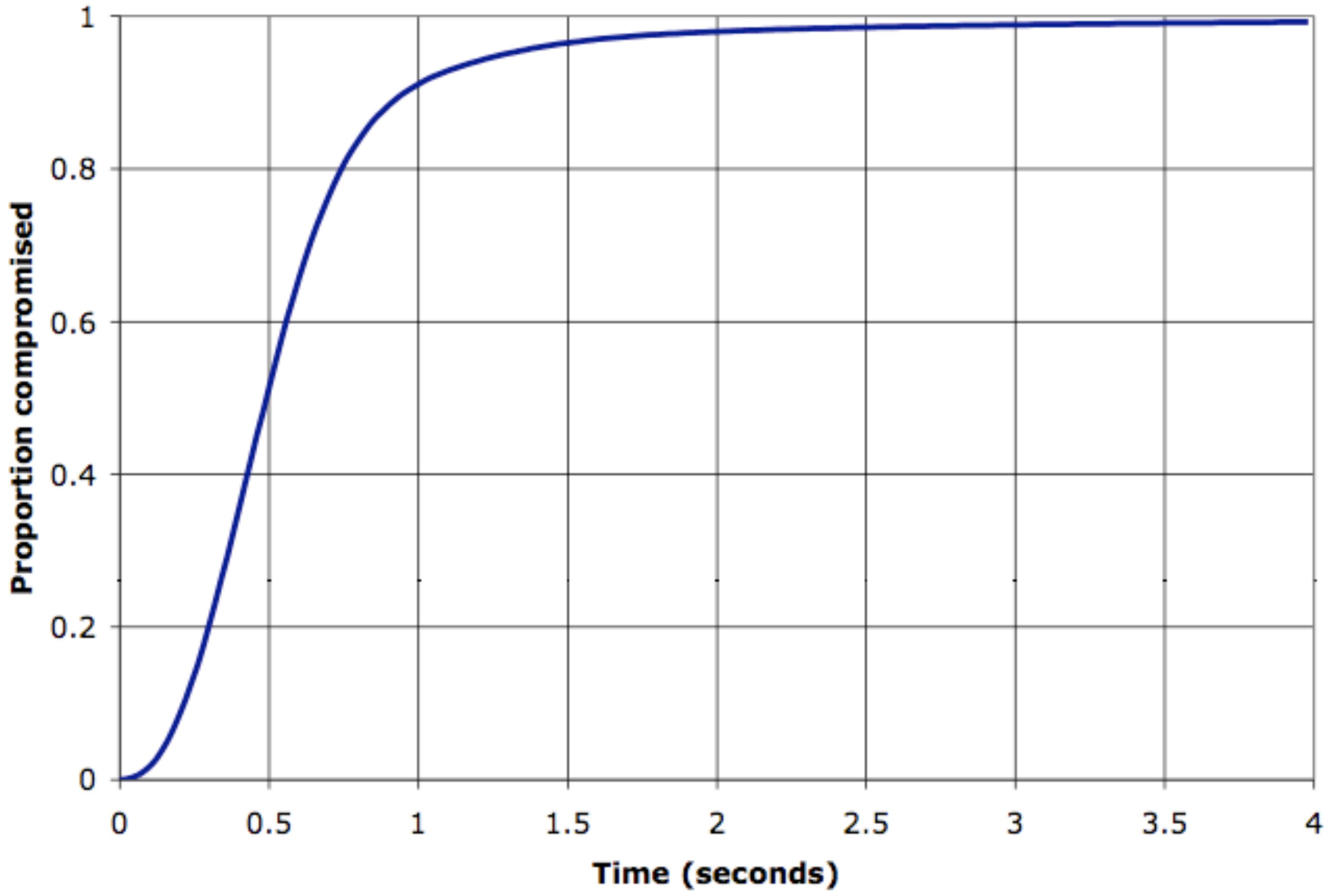


Optimum: 107 addresses to each 2nd level host

# A UDP flash worm could infect 95% of 1 million susceptible hosts in 510ms!



## A TCP flash worm could infect 95% of 1 million susceptible hosts in 3.3 seconds!



## What about more sophisticated, targeted attacks?

### Stuxnet

- Detected in 2010, developed starting in 2005
- Now believed to be developed by US and Israel
- Spread via Windows, targeted Siemens industrial systems
- Reportedly ruined 20% of Iran’s nuclear centrifuges

### Flame

- Discovered in 2012, in the wild since at least 2007
- Now known developed by NSA, Israel, and GCHQ for cyber espionage
- Mutated and self-destructed to evade detection
- Exploited weaknesses in MD5 to counterfeit certificate
- Records audio, screenshots, key presses, network traffic, nearby bluetooth devices’ contacts, etc.
- Reportedly infected at least 1,000 government, education, and private computers

## Summary

Malicious code has been around for a very long time

In the early days, computer viruses and Trojan horses moved at the speed of human-to-human interactions

Worms spread much faster by leveraging constant node connectivity

- Over the years, the propagation techniques used by worms have not changed
- More aggressive propagation mechanisms allow newer worms to spread faster

Flash worms are quite scary, but sensitive to minor problems in the network

- Excellent “worse case” for analyzing worm defenses

## Network Security

Private Communication

Private Routing

## Private Communication

Overview of the email system

Desirable properties for secure communication

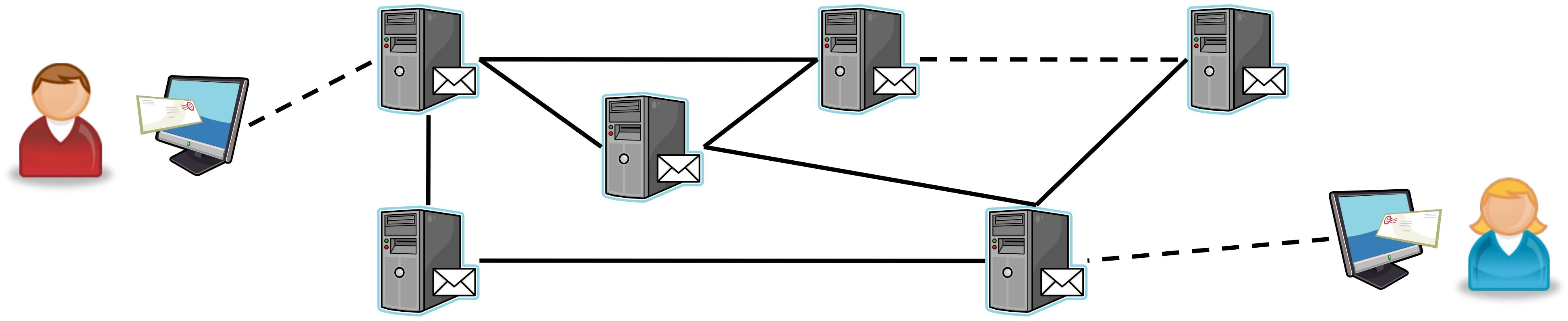
Attaining these properties

- Basic properties
- Confidentiality properties
- Delivery properties

Case study: PGP/GPG

Case study: OTR

## Electronic mail is forwarded through a mesh of servers



User agents (UAs) are used to compose, send, retrieve, and view mail

- *Examples:* Thunderbird, Outlook, Pine, Apple Mail, etc.
- UAs are not always connected to the email network

Mail transfer agents (MTAs) are used to route e-mail between users

- *Example:* The `sendmail` program
- MTAs are typically always online, but historically, they need not be

Question: Why might we have paths involving multiple MTAs? Why not just send email point to point?

## Like most older protocols, the email system was not built with security in mind

The Simple Mail Transfer Protocol (SMTP) is used to send mail from a UA to an MTA and to relay mail between MTAs

UAs typically use either the Post Office Protocol (POP) or the Internet Message Access Protocol (IMAP) to pull messages from their MTA

These protocols are very basic:

- Commands are sent as ASCII text strings
- Messages must be ASCII text
  - Non-ASCII text needs to be encoded prior to transmission
  - See Multipurpose Internet Mail Extensions (MIME) standards
- Authentication (if in place) is username/password authentication

To provide some sense of security, these protocols are often run over SSL/TLS

# SMTP Example

```
S: 220 smtp.example.com ESMTP Postfix
C: HELO relay.example.org
S: 250 Hello relay.example.org, I am glad to meet you
C: MAIL FROM:<bob@example.org>
S: 250 Ok
C: RCPT TO:<alice@example.com>
S: 250 Ok
C: RCPT TO:<theboss@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: From: "Bob Example" <bob@example.org>
C: To: Alice Example <alice@example.com>
C: Cc: theboss@example.com
C: Date: Tue, 15 Jan 2008 16:02:43 -0500
C: Subject: Test message
C:
C: Hello Alice. This is a test.
C: Your friend, Bob
C: .
S: 250
Ok: queued as 12345
C: QUIT
S: 221 Bye {The server closes the connection}
```

Message is from Bob

Sent to Alice and the boss

Message is just plain text, terminated with a return, a period, and a return

Question: What is to stop users from pretending to be other users?

## Discussion

*What are some desirable properties that we might want a secure communication system to have? Work in groups to identify properties, their definitions, and possible enforcement mechanisms.*

## Properties I (Basic)

Authentication: Is the sender who they claim to be?

- Might want Sender/MTA authentication to ensure appropriate use of an organization’s mail facilities
- User to user authentication is nice if email has real world implications

Integrity: Users should be convinced that they receive the same message that was originally sent

Non-repudiation: The recipient should be able to prove to a third party that the sender really did send the message

- Why is this useful?
  - The email in question might authorize some sort of atypical action
  - The message might contain a contract or purchase order
  - …

# Properties II (Data hiding)

Message privacy: The contents of a message should only be readable by the intended recipient

- Email is more like a postcard than a sealed letter
- Message privacy provides senders/receivers with the envelope that is otherwise missing from this analogy

Message flow confidentiality: The fact that two parties are communicating with one another should remain confidential

- Traffic analysis
  - Lots of communication between the Pentagon and late-night delivery places implies that a military operation may be starting up
- The ability to protect message flow confidentiality hides potentially sensitive behaviors from prying eyes

Anonymity: The recipient of a message should have no way to determine who sent the message

When might this be a useful property?

# Properties III (Delivery)

Proof of submission: The sender should be able to prove to a third party that a particular message was sent

- This is like certified mail in the physical world
- Useful for proving that you took required actions in a timely manner
  - i.e., It's not my fault! I sent the message and the delivery system failed!

Proof of delivery: The sender should be able to obtain proof that a message was delivered to its intended recipient

- Analogous to delivery confirmation at the post office
- Useful when messages need to be tracked

Sequencing: The system should provide assurance that messages arrive in their intended order

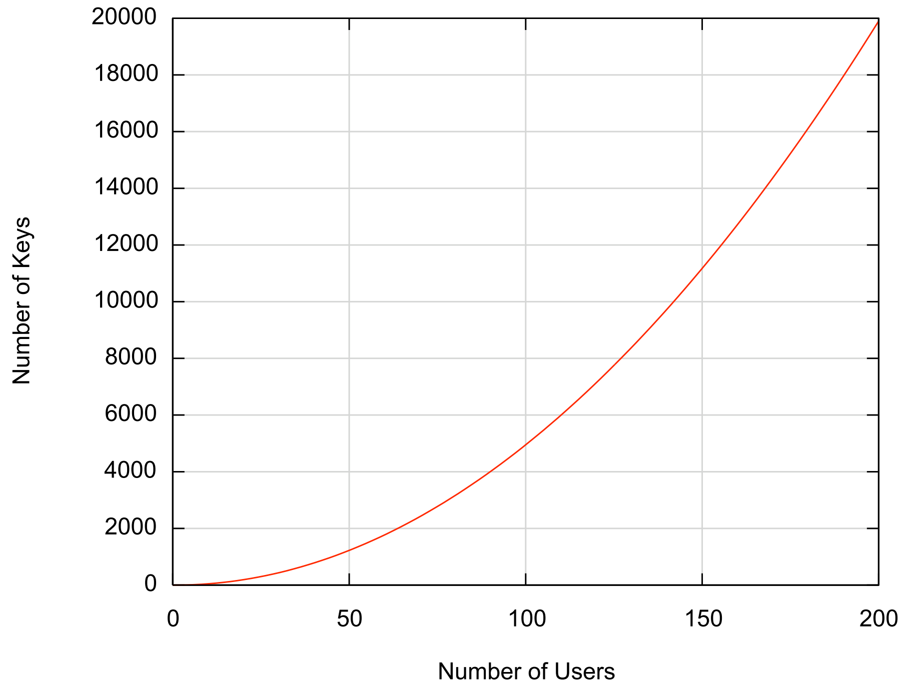
Why should we care about message sequencing?

# How can we attain these properties?

*By using cryptography!*

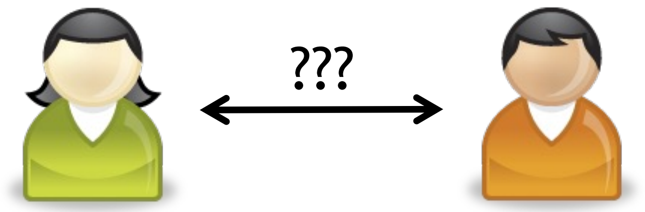
## Problem 1: Key management

- In a network with *n* participants, $\binom{n}{2}$ = n(n-1)/2 keys are needed!
- This number grows very rapidly!



## Problem 2: Key distribution

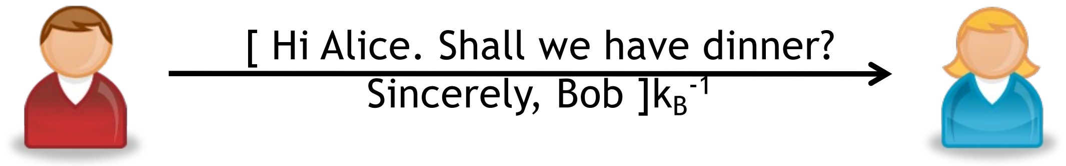
- How do Alice and Bob share keys in the first place?



- What if Alice and Bob have never met in person?
- What happens if they suspect that their shared key $K_{AB}$ has been compromised?

*Most secure email systems are based on public key cryptography, but use hybrid cryptosystems for efficiency*

## Digital signatures can provide us with many of our basic properties



Authentication

- The message body says that the letter is from Bob
- Alice can verify that Bob signed the message by using $k_B$
- If the signature checks out, only Bob could have produced it
- Note: Digital signatures can also provide UA → MTA authentication (How?)

Integrity

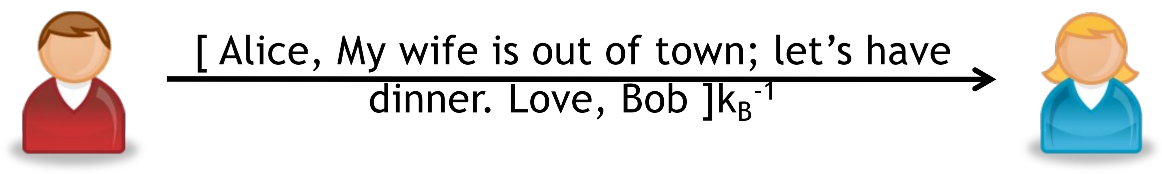
- If the message is modified in transit, the signature will not check out!
- This protects against transient failures and malicious modifications

Non-repudiation

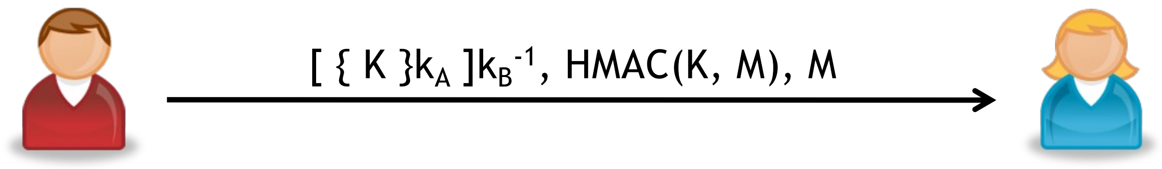
- All that is needed to verify Bob's signature is his public key
- Since the public key is public knowledge, anyone can verify Bob's signature

## Interesting twist: Plausible deniability using public key cryptography

Bob may not always want message non-repudiation…


[ Alice, My wife is out of town; let's have dinner. Love, Bob $]k_B^{-1}$

Hybrid cryptography gives us a means of achieving both authentication and deniability:



Question: Why does this authenticate Bob?

- Alice knows that she received the message from Bob
- The digital signature verifies this fact

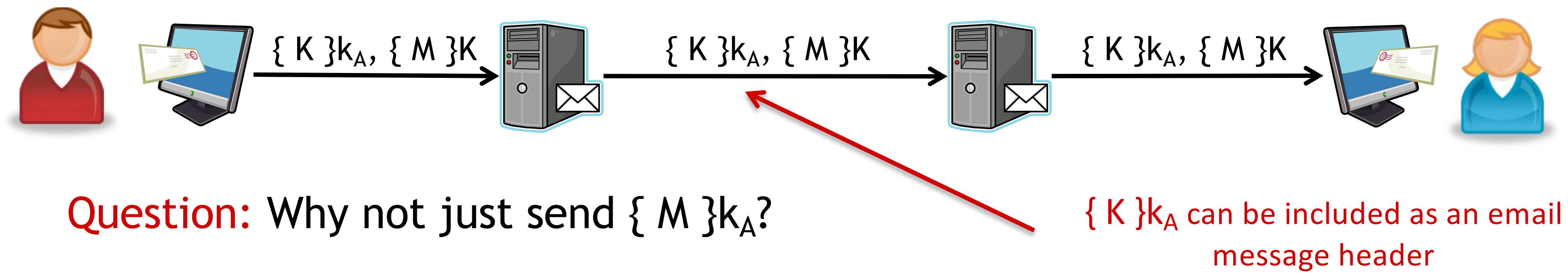
Question: Why does this protocol provide message integrity?

- The key K is only known to Alice and Bob

Question: Why does this protocol provide deniability?

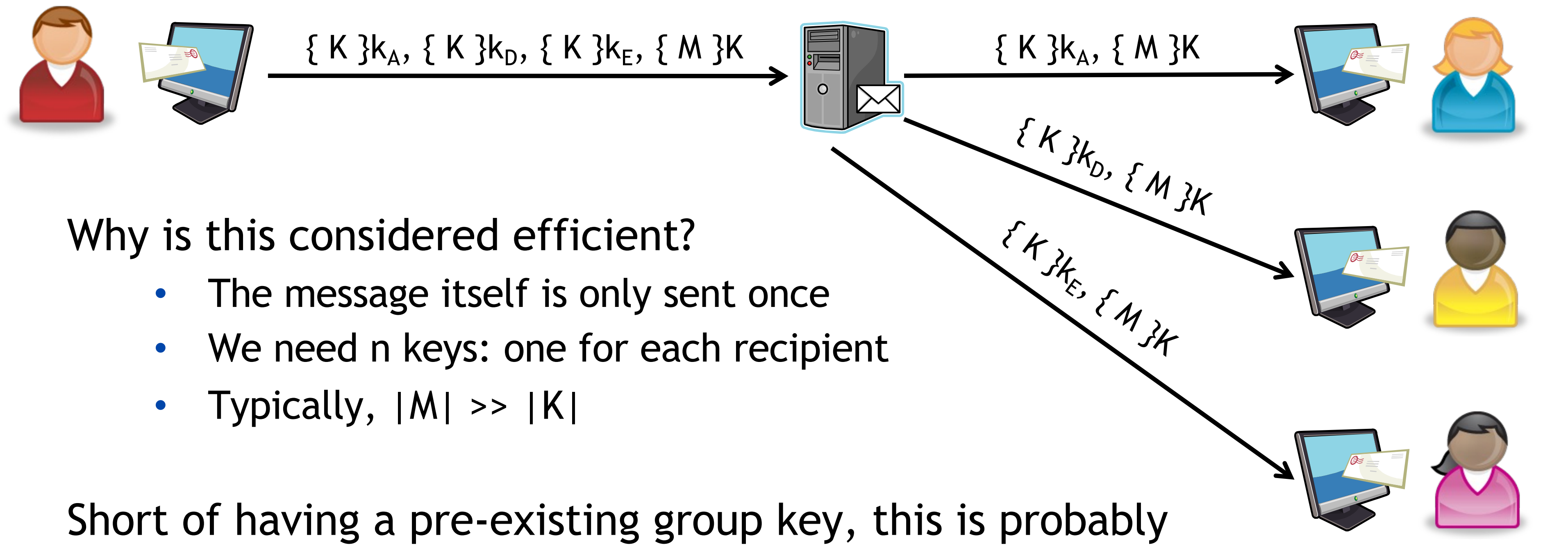
- After Alice recovers the key K, she can compute the HMAC for any message

## Given what we've seen so far, achieving message-level privacy is actually pretty easy



Question: Why not just send $\{ M \}k_A$?

This also allows us to efficiently send mail to multiple recipients!

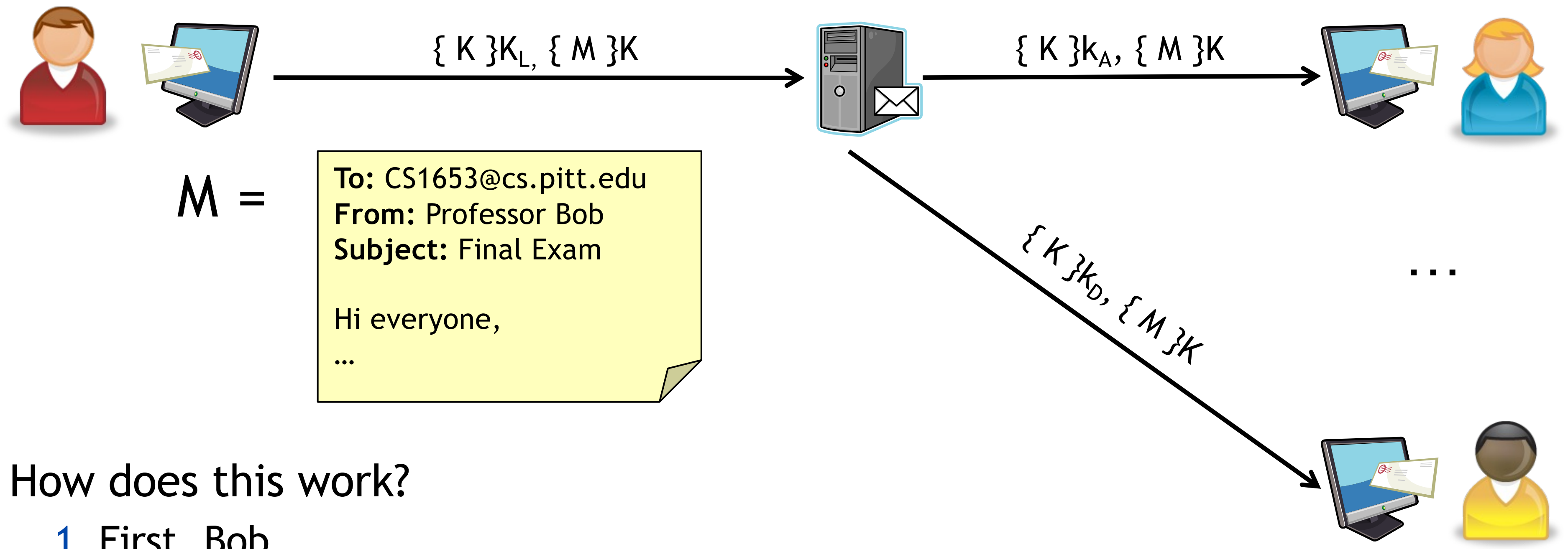


Why is this considered efficient?

- The message itself is only sent once
- We need n keys: one for each recipient
- Typically, $|M| >> |K|$

Short of having a pre-existing group key, this is probably as good as we could expect to do

## This can even be extended to work with mailing lists!



**To:** CS1653@cs.pitt.edu
**From:** Professor Bob
**Subject:** Final Exam

Hi everyone,
…

How does this work?

1. First, Bob
   - Generates a random symmetric key K
   - Computes { M }K and { K }$k_L$
   - Sends these values to the list server

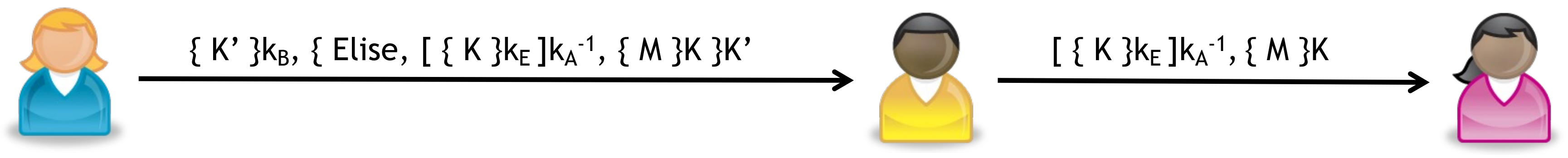
2. Then, the list server
   - Decrypts { K }$k_L$ to recover K
   - Explodes the list "CS1653@cs.pitt.edu"
   - Encrypts the key K to each member
   - Transmits { K }$k_i$ { M }K to each principal $p_i$

3. Finally, each principal $p_i$
   - Decrypts the { K }$k_i$ to recover K
   - Decrypts { M }K

# Message flow confidentiality and anonymity are a little bit harder to attain…

Note: All MTAs along the delivery path for a message know both the sender and the recipient of that message

One way to hide message flows is to use a trusted intermediary


$\{ K' \}k_B, \{ \text{Elise}, [ \{ K \}k_E ]k_A^{-1}, \{ M \}K \}K'$
$[ \{ K \}k_E ]k_A^{-1}, \{ M \}K$

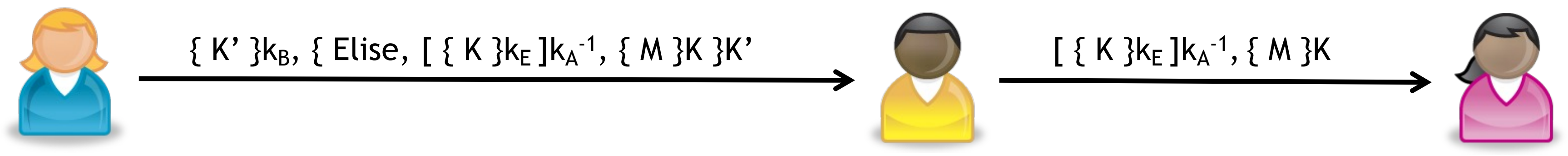
Why does this work?

- Correctness: Bob is able to decrypt Alice's original message and forward the message that it contains
- Confidentiality: Bob cannot recover the key K, since he does not know $k_E^{-1}$
- Integrity: Elise can verify the signature on K
- Flow confidentiality: Only Bob knows that Alice and Elise are talking

# Message flow confidentiality and anonymity are a little bit harder to attain…

Note: All MTAs along the delivery path for a message know both the sender and the recipient of that message

One way to hide message flows is to use a trusted intermediary


$\{ K' \}k_B, \{ \text{Elise}, [ \{ K \}k_E ]k_A^{-1}, \{ M \}K \}K'$
$[ \{ K \}k_E ]k_A^{-1}, \{ M \}K$

Question: Why might Alice want to use multiple levels of intermediary?

- No single intermediary knows the whole path
- Messages can be batched to hide flows from global monitors
- This is how Mixmaster (anonymous remailer) works
  - https://en.wikipedia.org/wiki/2012_University_of_Pittsburgh_bomb_threats

Question: How could trusted intermediaries facilitate anonymous communication?

# *Case study:* PGP

Pretty Good Privacy (PGP) is a hybrid encryption program that was first released by Phil Zimmerman in 1991

In the PGP model

- Users are typically identified by their email address
- Users create and manage their own digital certificates
- Certificates can be posted and discovered by using volunteer “key servers”
- Key servers also serve a function similar to an OLRS

Email addresses are GUIDs, so they effectively disambiguate identities

However, note that there are no CAs in the system! As such, users can create certificates for any email address or identity that they want!

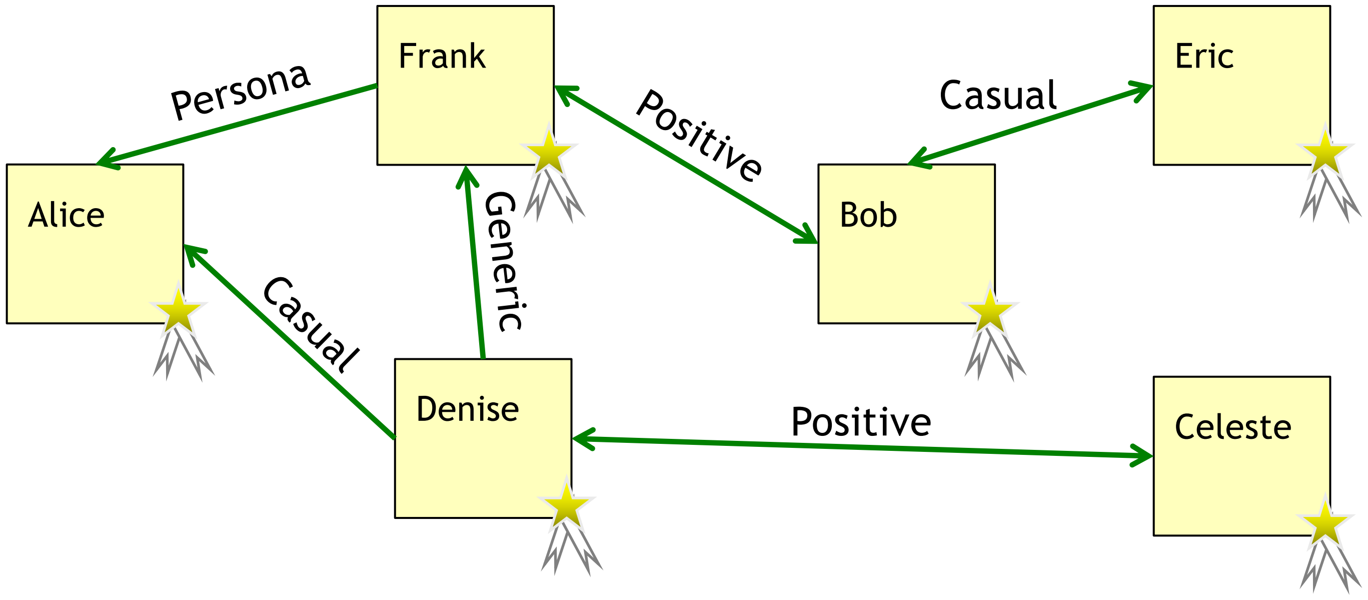
*How can we deal with this anarchy?!*

# PGP takes a grassroots approach to certificate validation

Users create their own certificates, and sign the certificates of others

- “I am Alice and I have verified that this certificate belongs to Bob”
- Four levels of certification: Generic, Persona, Casual, and Positive

This is essentially a cryptographic social network!



## GPG is an implementation of the OpenPGP standard

GPG stands for the Gnu Privacy Guard

- The OpenPGP standard is defined in RFC 4880
- As such, GPG interoperates with current versions of PGP

GPG supports a number of unencumbered cryptographic algorithms

- Symmetric key ciphers: CAST5, Triple DES, AES, Blowfish, and Twofish
- Asymmetric key ciphers: RSA and ElGamal
- Hash functions: MD5 (ack!), SHA-1, RIPEMD-160, and Tiger
- Digital signatures: DSA

The core GPG program is command-line driven, and is available for a number of popular operating systems

## How does GPG work?

Step 1: Adding a signature

- Compute the hash, H(M), of the message
- Append the name of the hash function and a signature over H(M) to the message

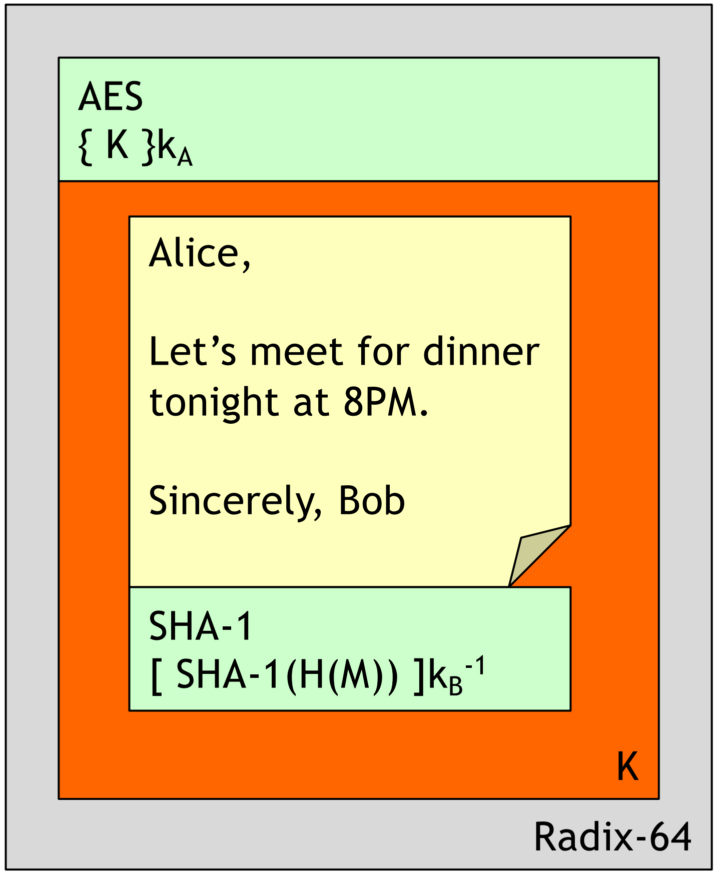
Step 2: Compress the message

Step 3: Encryption

- Generate a random key K
- Encrypt K and its corresponding algorithm name with the recipients public key
- Encrypt the whole message with K
- Prepend the encrypted key block to the newly encrypted message

Step 4: Encoding and Transmission

- Radix-64 encode the message (Why?)
- Send via email


AES
$\{ K \}k_A$
Alice,
Let's meet for dinner tonight at 8PM.
Sincerely, Bob
SHA-1
$[ \text{SHA-1}(H(M)) ]k_B^{-1}$
K
Radix-64

# Receiving a Message

Step 1: Decode the message

- Undo Radix-64 encoding to recover bytes

Step 2: Key recovery

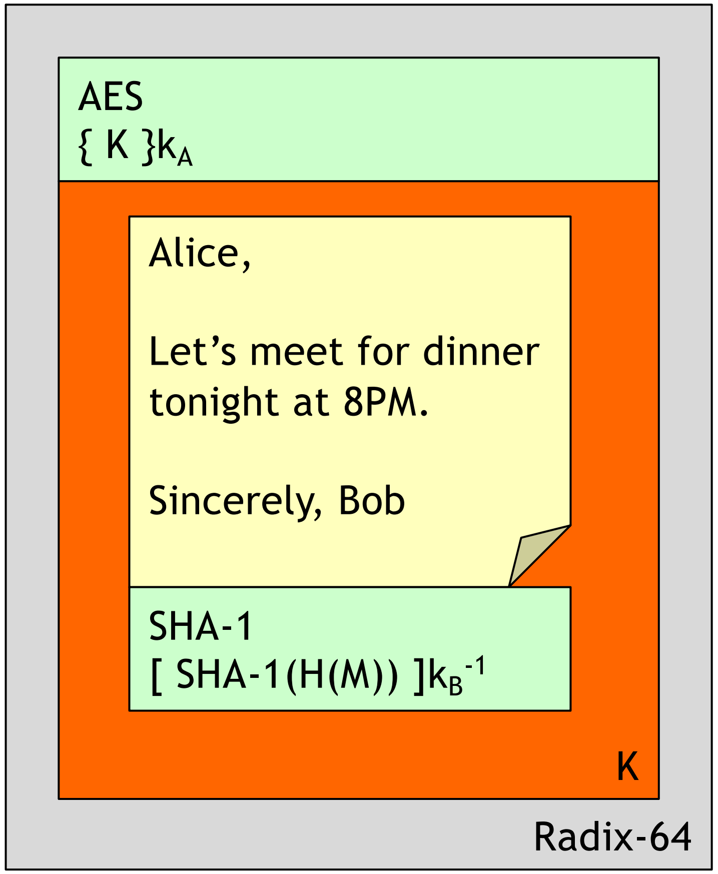
- Extract symmetric key algorithm name
- Recover session key K

Step 3: Decrypt and decompress

- Decrypt the message block
- Decompress message block

Step 4: Verification

- Extract hash function name
- Compute H(M) and validate sender signature



AES
$\{ K \}k_A$
Alice,
Let's meet for dinner tonight at 8PM.
Sincerely, Bob
SHA-1
$[ \text{SHA-1}(H(M)) ]k_B^{-1}$
K
Radix-64

## *Case study:* OTR

Off-the-Record Messaging (OTR) was designed by cryptographers Ian Goldberg and Nikita Borisov to mirror talking in a soundproof room

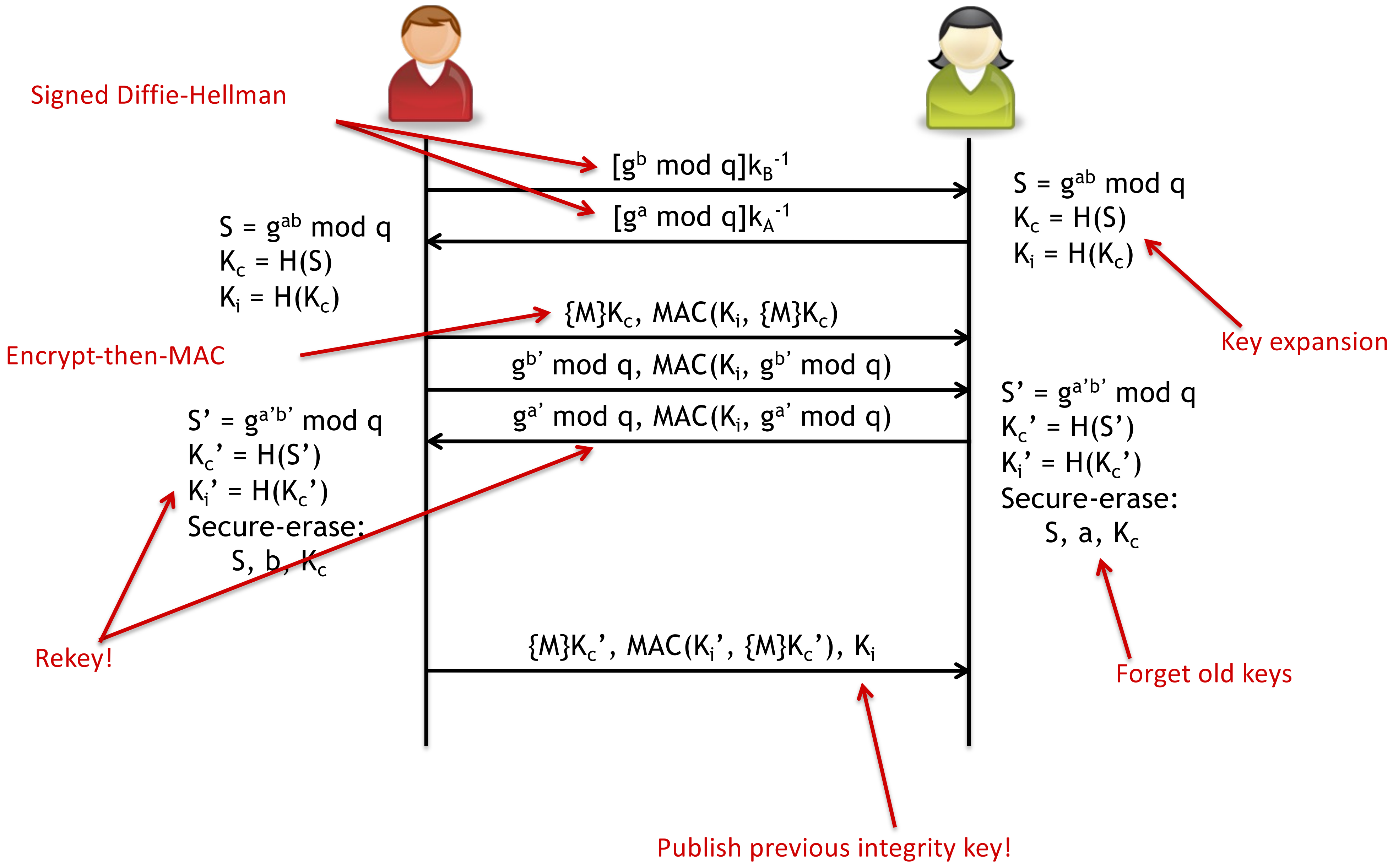
OTR tackles problems with PGP

- Alice and Bob must both know how to use PGP
- Signed messages are potentially incriminating: Alice's privacy relies on Bob securing his machine and records

OTR provides the following security properties:

- Mutual (repudiable) authentication
- Confidentiality
- Perfect forward secrecy
- Deniability (through malleable encryption)

# OTR uses techniques we've looked at earlier in the course



## Why is OTR secure?

Confidentiality: All messages are encrypted with a fresh key

Authentication: Key exchange is signed with asymmetric crypto

Repudiable authentication: Only keys are signed, not content

Perfect forward secrecy: Even if long-term keys are leaked, no confidentiality keys are compromised

Deniability: Since integrity keys are public after use, messages are forgeable by any party

## OTR has been implemented in a wide variety of chat applications

Native support:

- Adium
- IM+
- Kopete
- ChatSecure

Plugin available:

- pidgin
- Miranda
- Trillian
- Gajim

Several techniques used for key exchange

1. Originally, exchanged SSH-style
   Trade key fingerprints, verify by eye on first chat
2. Mobile clients can exchange via local connections
   Bluetooth, NFC, QR codes
3. Socialist millionaire protocol can be used with plaintext "secret"
   Verify natural-language challenge-response

The Axolotl (Double) ratchet extends these ideas to an asynchronous setting

- Forward and backward secrecy, repudiability, etc.

## Summary

Email was designed in *much* more trustworthy times

Modern secure communication systems should satisfy additional properties beyond email's "best effort delivery"

- End to end confidentiality, message path confidentiality, integrity, non-repudiation, deniability…

Many of these goals can be attained using cryptographic means (not always simultaneously)

GPG is an open-source implementation of the OpenPGP standard

OTR Messaging is a standard protocol for repudiable, forward-secret chat (like a verbal conversation in a soundproof room)