Android and Anonymity

Tadayoshi Kohno

Thanks to Vitaly Shmatikov, Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Bennet Yee, and many others for sample slides and materials ...

Goals for Today

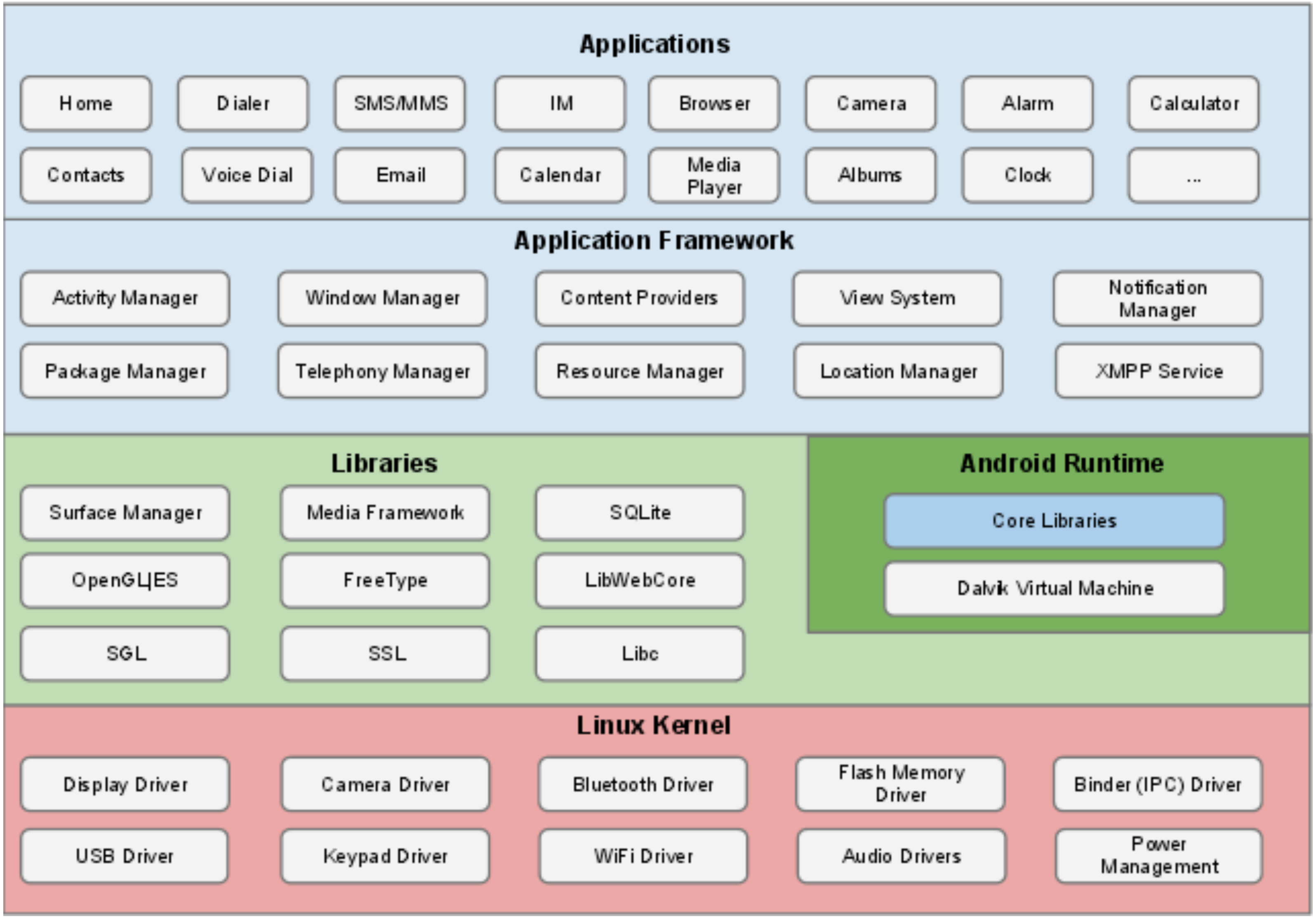
- Lab 3 discussion
- Android
- Anonymity

- HW 3 now out (due Friday)
- Lab 3 out just now

Mobile Device Security (Android)

- Android
  - Based on Linux
  - Layers:
    - Android Application Runtime (generally written in Java, run in the Dalvik virtual machine; sometimes native applications or native libraries)
    - Android OS
    - Device Hardware
  - Applications
    - Pre-installed
    - User-installed
      - Via app stores
      - Via over the air (OTA) updates.

Android Software Stack



http://source.android.com/tech/security/index.html

Application Sandboxes

- Based on Linux: Has clear notion of users and permissions
- Each application
  - Assigns unique user ID (UID)
  - Runs as that user in a separate process
  - **Different than traditional operating systems** where multiple applications run with the same user permissions

Application Sandboxes (II)

- Desktop browser sandbox: language specific
- Android sandbox: baked into the OS, via the kernel
  - No restriction on how applications are written
  - Native code
  - Java code
- Conventional systems: memory corruption errors lead to complete compromise
- Android: memory corruption errors only lead to arbitrary code execution in the context of the **particular** compromised application
- (Can still escape sandbox -- but must compromise Linux kernel to do so)

File permissions

- Files written by one application cannot be read by other applications
  - Not true for files stored on the SD card
- It is possible to do full filesystem encryption
  - Key = Password combined with salt, hashed with SHA1 using PBKDF2.

Memory Management

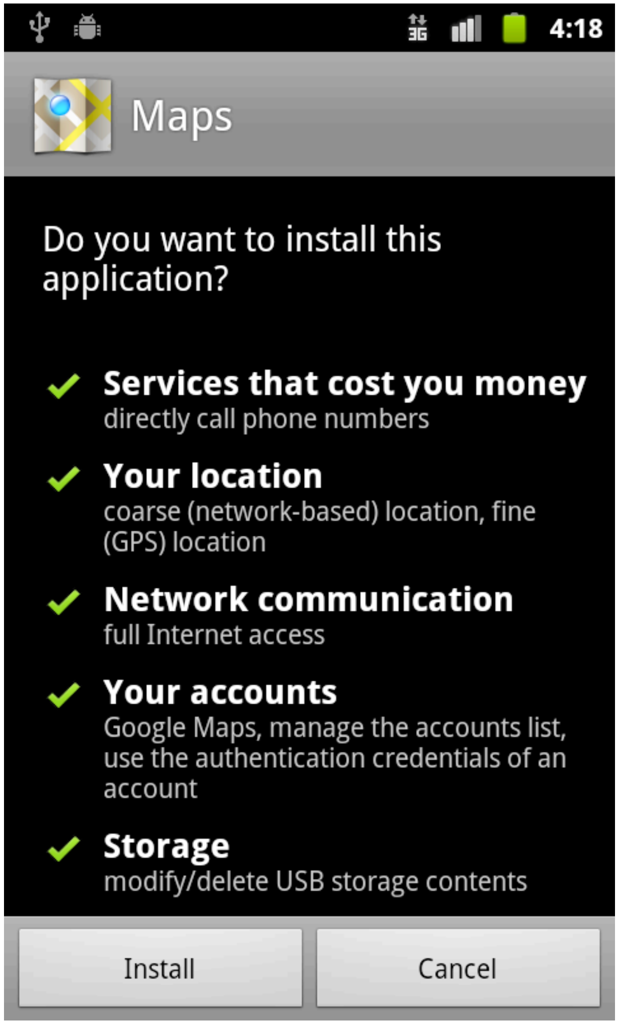
- Address Space Layout Randomization to randomize addresses on stack
- Hardware-based No eXecute (NX) to prevent code execution on stack/heap
- Stack guard derivative
- Some defenses against double free bugs (based on OpenBSD's dmalloc() function)
- …
- (See http://source.android.com/tech/security/index.html)

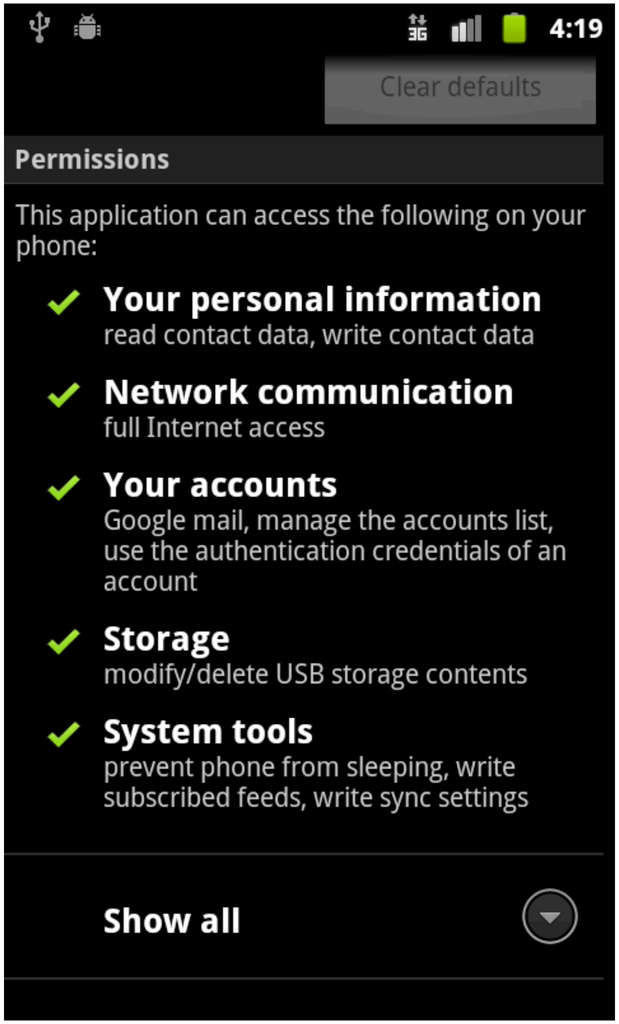
Applications

- Activity: Code for single, user-focused task
- Services: Code that runs in the background
- Broadcast Receiver: Receive Intents (messages from other applications)

- AndroidManifest.xml
  - Overall information about application (activities, services, ...)
  - Also specifies which **permissions** are required by applications

Permissions / Manifests





http://source.android.com/tech/security/index.html

Permissions

- Example permissions
  - Camera
  - Location (GPS)
  - Bluetooth
  - SMS functions
  - Network capabilities
- Cannot grant / deny individual permissions
- Once accepted, users not notified of permissions again
- Security exception thrown if attempt to access resource not declared in manifest

Obtaining User Consent for Permissions

- General options:
  - At install time (manifests)
  - At time of use (prompts)
- Why manifests
  - Users are evaluating the application, the developers, etc, to see if they want the app
  - Prompts slow down user; hinder user experience
  - Users may just say "OK" to all dialogs without reading them
- Why prompts
  - At time of resource access
  - Opportunity for user to be more in control of actual resource use (app with GPS permissions should only actually access the GPS when the user wishes -- but can't tell with manifest model)
- (Alternative: User-driven access control, Roesner et al (2012))

Application Signing

- Apps are signed
  - Often with self-signed certificates
- Signed application certificate defines which user ID is associated with which applications
  - Different apps run under different UIDs
- Shared UID feature
  - Shared Application Sandbox possible, where two or more apps signed with same developer key can declare a shared UID in their manifest

Shared UIDs

- App 1: Requests GPS / camera access
- App 2: Requests Network capabilities

- Generally:
  - First app can’t exfiltrate information
  - Second app can’t exfiltrate anything interesting
- With Shared UIDs (signed with same private key)
  - Permissions are a superset of permissions for each app
  - App 1 can now exfiltrate; App 2 can now access GPS / camera

Questions

- Q1: How might malware authors get malware onto phones?
- Q2: What are some goals that mobile device malware authors might have?
- Q3: What technical things might malware authors do?

Malware

- Legitimacy of apps
  - Self-signing means that signers can claim to be whoever they wish
- Installation vector
  - (Seems to be) "drive-by-downloads" and exploits for infection, and more social engineering (tricking users to install)
  - E.g., "sideloading" sites: distribute pirated versions of popular applications, which can be decompiled and modified to include malicious behavior
  - Utilities, games, adult-oriented apps [Lookout Mobile Threat Report, August 2011]

Malware techniques

- Add background Service
- Modify existing application source code
- Component library replacement

- To avoid basic signature detection:
  - Dynamically download new Dalvik bytecode
  - Use DexClassLoader API to run the downloaded code

- Use exploit to obtain root access
- Many other techniques

Malware Functions

- Make a profit
  - Premium number dialers
  - Aggressive adware
  - Data collection (obtain personally-identifiable information that can be sold)
  - Banking trojans (e.g., FakeToken.A to bypass two-factor authentication)
- Bot clients (phone have limited resources, so more useful as a mechanisms to support other goals, e.g., later targeted data collection)
  - Internet C&C
  - SMS C&C
- Privileged Operations Trojans (obtain root)
- Disruptive Trojans (denial of service, destroy data)
  - Not stealthy; no profit

Privacy on Public Networks

- Internet is designed as a public network
  - Machines on your LAN may see your traffic, network routers see all traffic that passes through them
- Routing information is public
  - IP packet headers identify source and destination
  - Even a passive observer can easily figure out who is talking to whom
- Encryption does not hide identities
  - Encryption hides payload, but not routing information
  - Even IP-level encryption (tunnel-mode IPSec/ESP) reveals IP addresses of IPSec gateways

Questions

- Q1: Why might people want anonymity on the Internet?

- Q2: Why might people **not** want anonymity on the Internet?

Questions

- Q1: How might one go about trying to obtain anonymity? What technical approaches might we use?
- Q2: How might one go about trying to violate someone else's anonymity?

Applications of Anonymity

- Privacy
  - Hide online transactions, Web browsing, etc. from intrusive governments, marketers and archivists
- Untraceable electronic mail
  - Corporate whistle-blowers
  - Political dissidents
  - Socially sensitive communications (online AA meeting)
  - Confidential business negotiations
- Law enforcement and intelligence
  - Sting operations and honeypots
  - Secret communications on a public network

Applications of Anonymity (II)

- Digital cash
  - Electronic currency with properties of paper money (online purchases unlinkable to buyer’s identity)
- Anonymous electronic voting
- Censorship-resistant publishing

What is Anonymity?

- Anonymity is the state of being not identifiable within a set of subjects
  - You cannot be anonymous by yourself!
    - Big difference between anonymity and confidentiality
  - Hide your activities among others' similar activities
- Unlinkability of action and identity
  - For example, sender and the email he or she sends are no more related after observing communication than they were before
- Unobservability (hard to achieve)

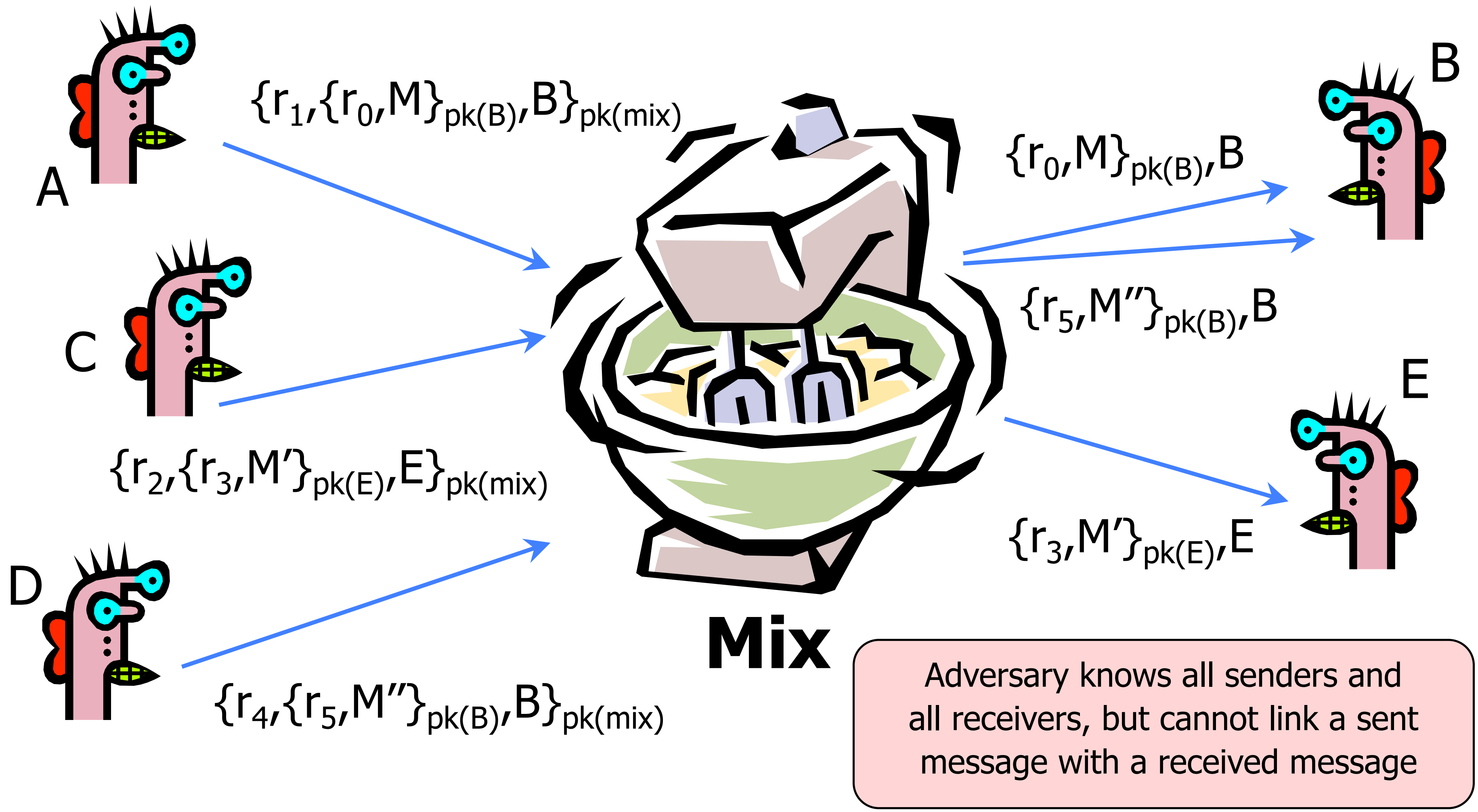
Chaum's Mix

- Early proposal for anonymous email
  - David Chaum. "Untraceable electronic mail, return addresses, and digital pseudonyms". Communications of the ACM, February 1981.

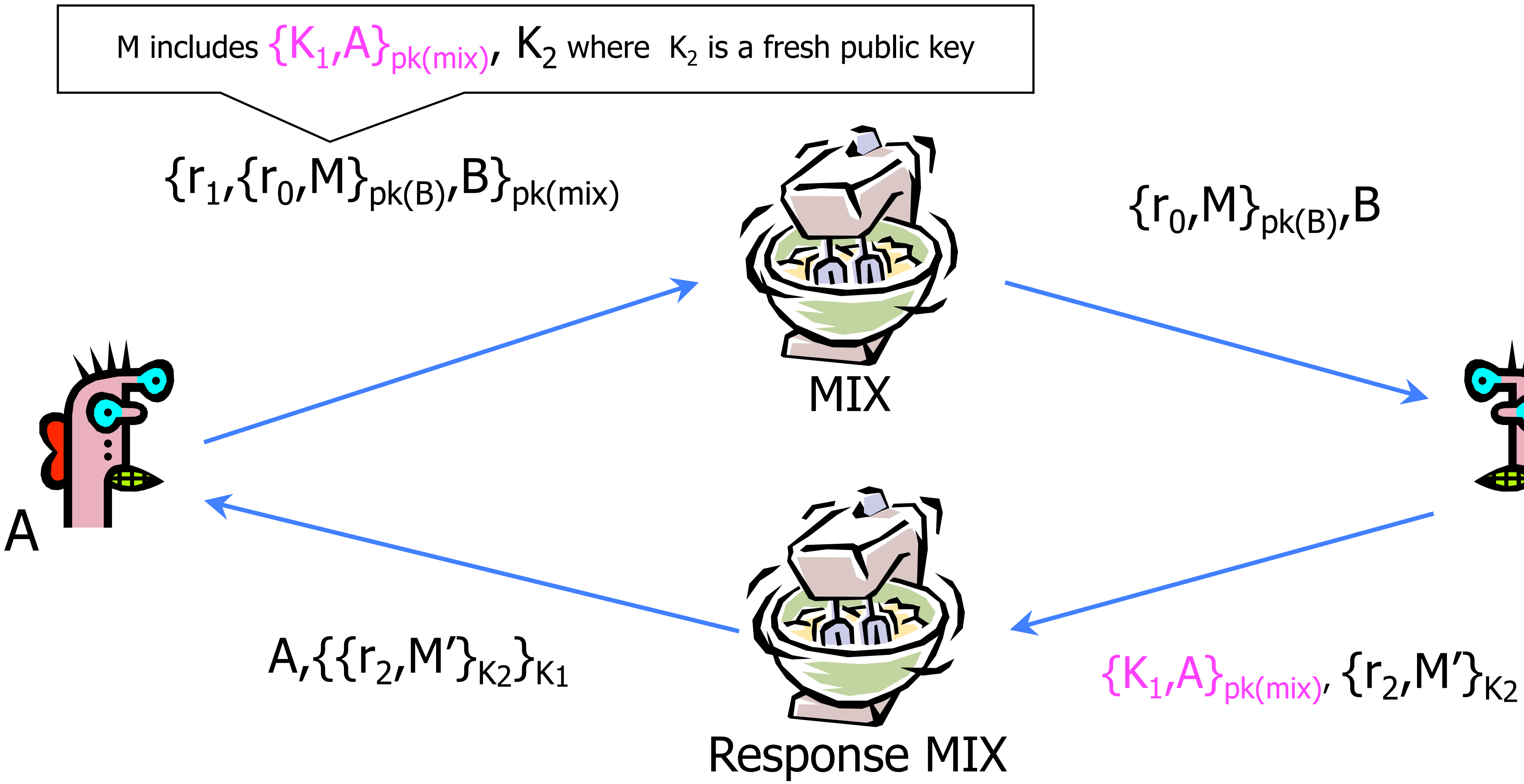
Before spam, people thought anonymous email was a good idea ☺

- Public key crypto + trusted re-mailer (Mix)
  - Untrusted communication medium
  - Public keys used as persistent pseudonyms
- Modern anonymity systems use Mix as the basic building block

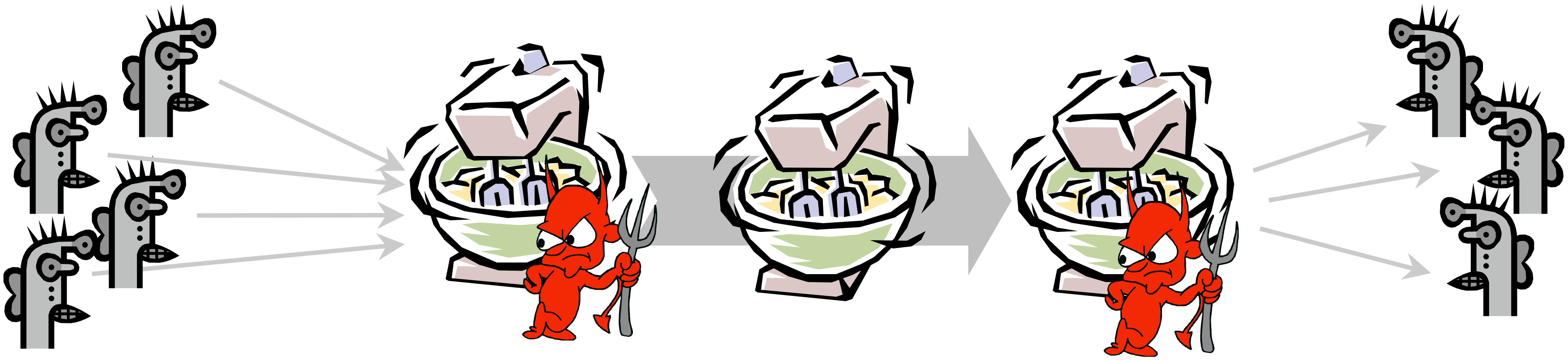
Basic Mix Design



Anonymous Return Addresses



Mix Cascade

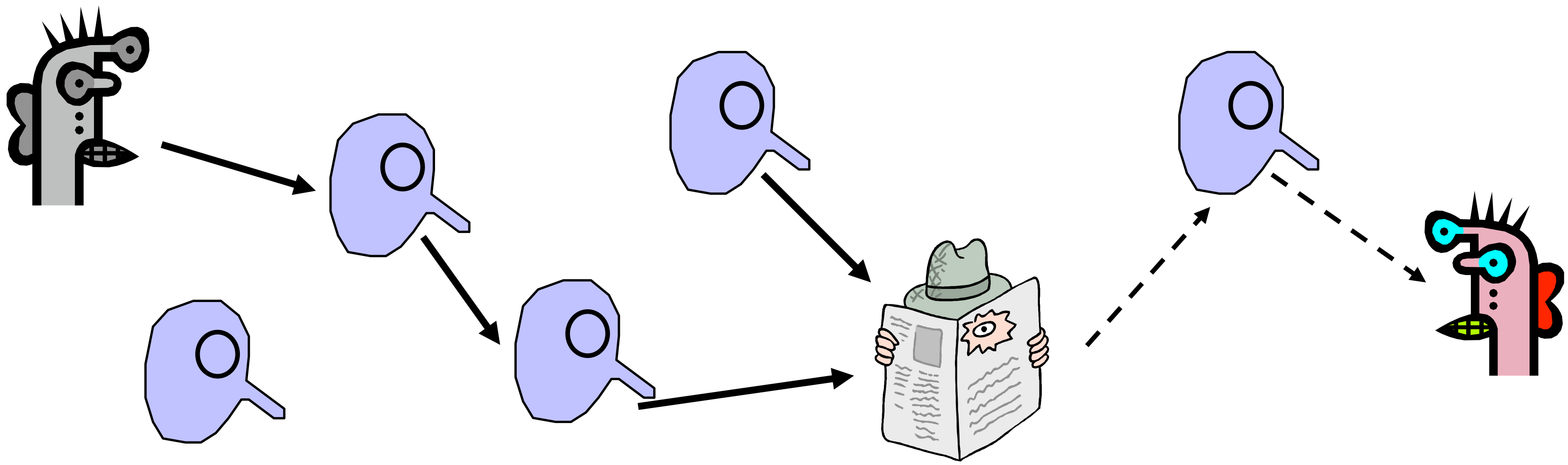


- Messages are sent through a sequence of mixes
  - Can also form an arbitrary network of mixes ("mixnet")
- Some of the mixes may be controlled by attacker, but even a single good mix guarantees anonymity
- Pad and buffer traffic to foil correlation attacks

Disadvantages of Basic Mixnets

- Public-key encryption and decryption at each mix are computationally expensive
- Basic mixnets have high latency
  - Ok for email, not Ok for anonymous Web browsing
- Challenge: low-latency anonymity network
  - Use public-key cryptography to establish a "circuit" with pairwise symmetric keys between hops on the circuit
  - Then use symmetric decryption and re-encryption to move data messages along the established circuits
  - Each node behaves like a mix; anonymity is preserved even if some nodes are compromised

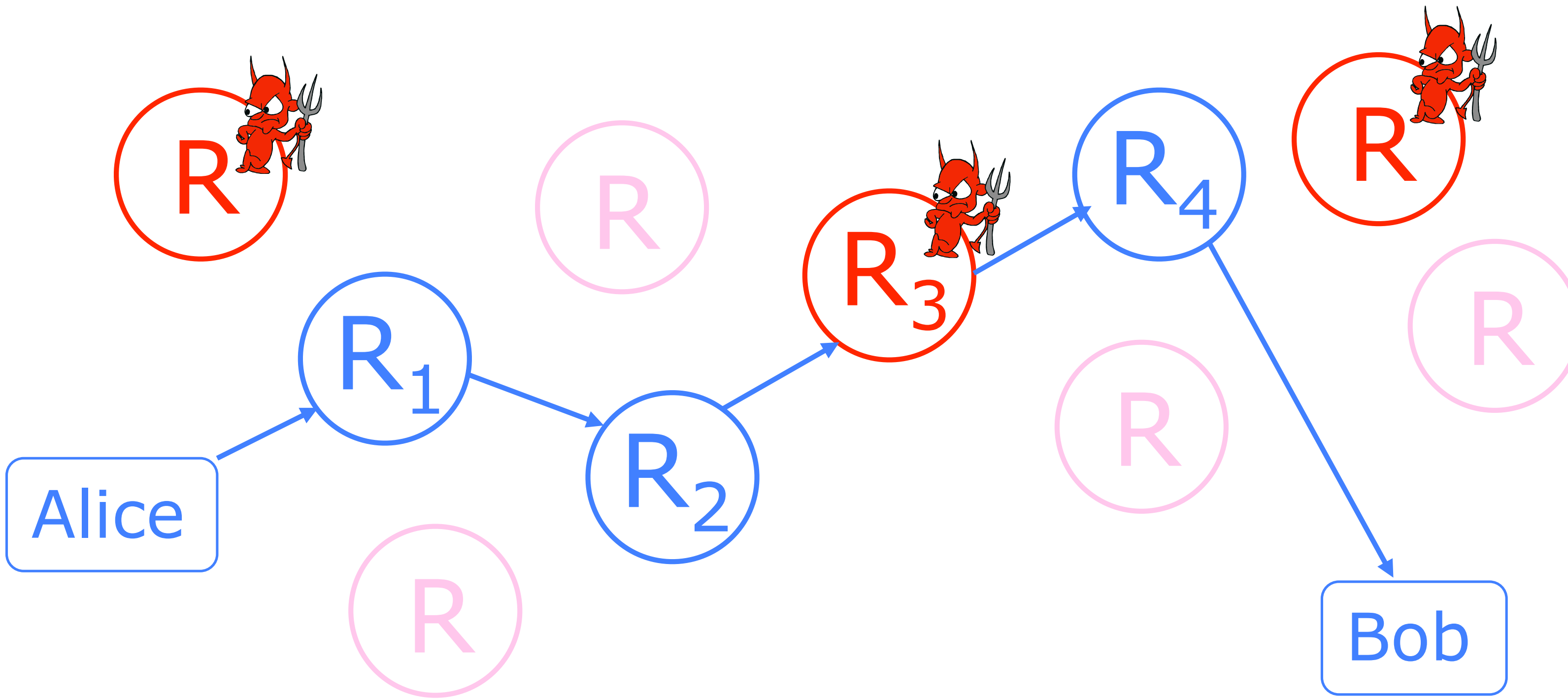
Another Idea: Randomized Routing



- Hide message source by routing it randomly
  - Popular technique: Crowds, Freenet, Onion routing
- Routers don't know for sure if the apparent source of a message is the true sender or another router

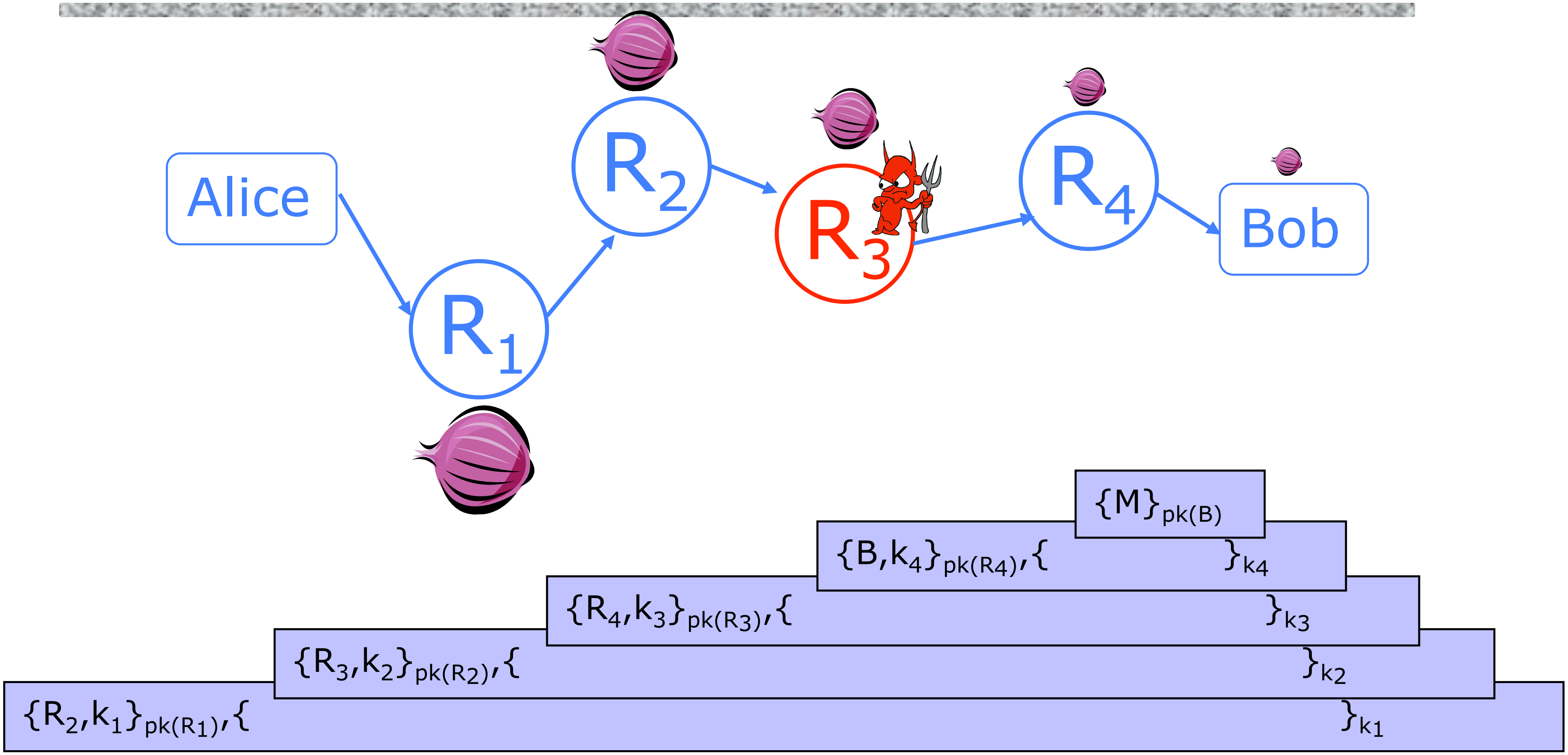
Onion Routing

[Reed, Syverson, Goldschlag '97]



- Sender chooses a random sequence of routers
  - Some routers are honest, some controlled by attacker
  - Sender controls the length of the path

Route Establishment



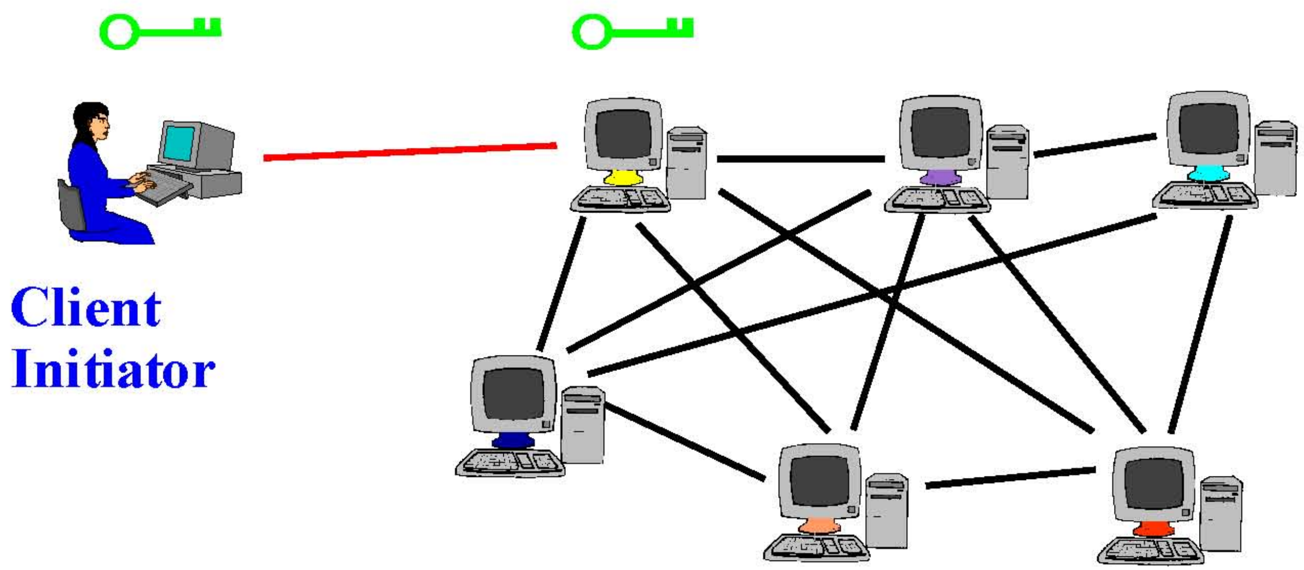
- Routing info for each link encrypted with router's public key
- Each router learns only the identity of the next router

Tor

- Second-generation onion routing network
  - http://tor.eff.org
  - Developed by Roger Dingledine, Nick Mathewson and Paul Syverson
  - Specifically designed for low-latency anonymous Internet communications
- Running since October 2003
- "Easy-to-use" client proxy
  - Freely available, can use it for anonymous browsing

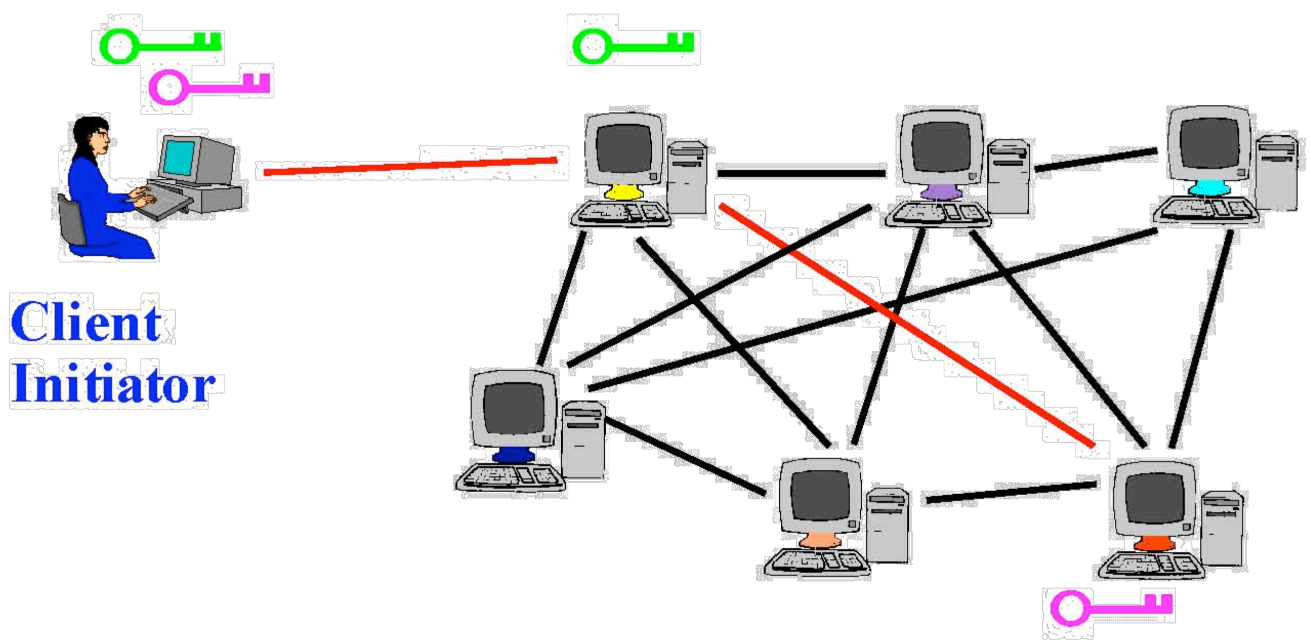
Tor Circuit Setup (1)

- Client proxy establish a symmetric session key and circuit with Onion Router #1



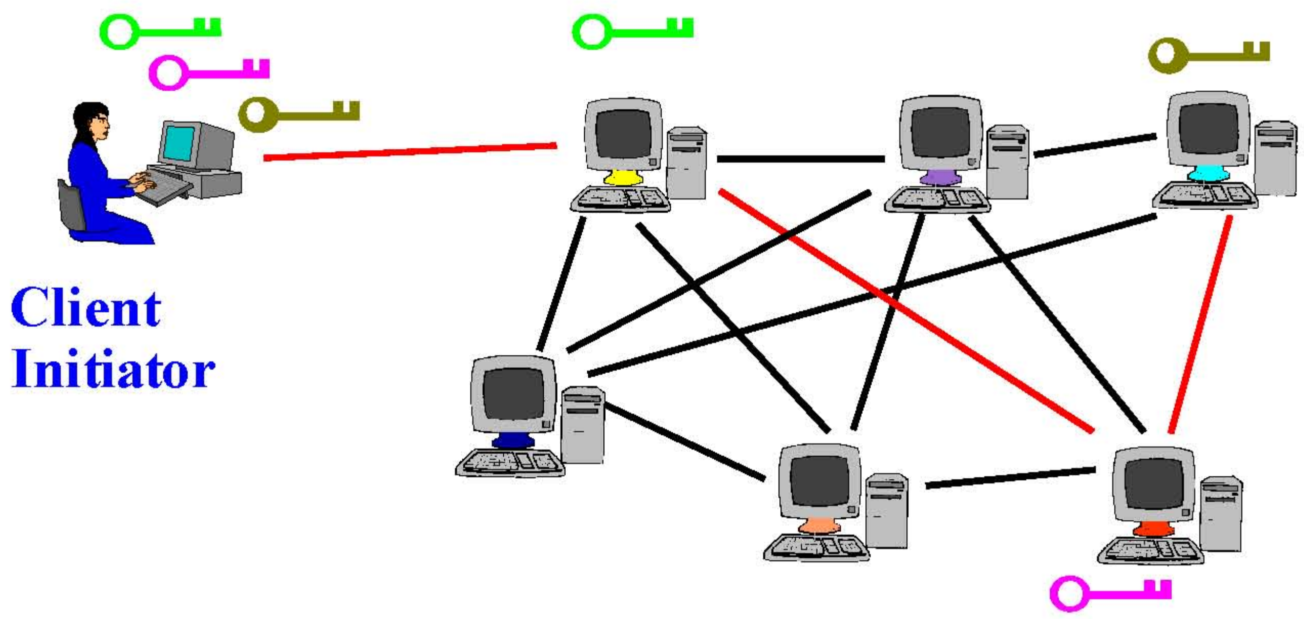
Tor Circuit Setup (2)

- Client proxy extends the circuit by establishing a symmetric session key with Onion Router #2
  - Tunnel through Onion Router #1 (don't need



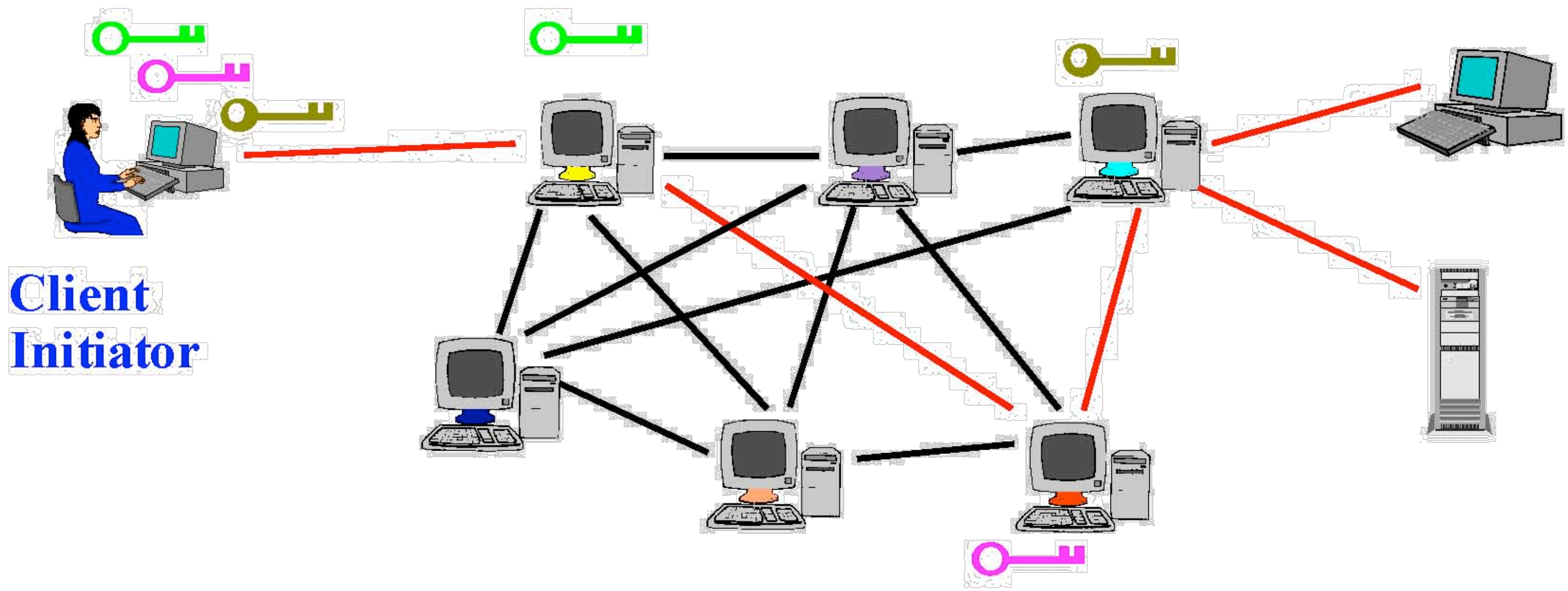
Tor Circuit Setup (3)

- Client proxy extends the circuit by establishing a symmetric session key with Onion Router #3
  - Tunnel through Onion Routers #1 and #2



Using a Tor Circuit

- Client applications connect and communicate over the established Tor circuit



Tor Management Issues

- Many applications can share one circuit
  - Multiple TCP streams over one anonymous connection
- Tor router doesn’t need root privileges
  - Encourages people to set up their own routers
  - More participants = better anonymity for everyone
- Directory servers
  - Maintain lists of active onion routers, their locations, current public keys, etc.
  - Control how new routers join the network
    - “Sybil attack”: attacker creates a large number of routers
  - Directory servers’ keys ship with Tor code

Attacks on Anonymity

- Passive traffic analysis
  - Infer from network traffic who is talking to whom
  - To hide your traffic, must carry other people's traffic!
- Active traffic analysis
  - Inject packets or put a timing signature on packet flow
- Compromise of network nodes
  - Attacker may compromise some routers
  - It is not obvious which nodes have been compromised
    - Attacker may be passively logging traffic
  - Better not to trust any individual router
    - Assume that some fraction of routers is good, don't know which

Deployed Anonymity Systems

- Tor (http://tor.eff.org)
  - Overlay circuit-based anonymity network
  - Best for low-latency applications such as anonymous Web browsing
- Mixminion (http://www.mixminion.net)
  - Network of mixes
  - Best for high-latency applications such as anonymous email

Some caution

- Tor isn't completely effective by itself
  - Challenges if you have cookies turned on in your browser, are using JavaScript, etc.
  - Exit nodes can see everything!