Mixminion and Tor: Two deployed anonymity networks

Nick Mathewson

E-Privacy / Firenze / 2005

About this talk

- Background
- Mixminion
- Tor
- The Future

Topic

Two deployed networks for anonymous communication.

- Tor: low latency, less anonymous.
- Mixminion: more latency, so less useful, but more anonymous.

Acknowledgements

- Mixminion project
  - Two other designers
- Tor project
  - Many other designers
  - One other programmer

1. Background

What is Anonymity?

- Technical definition:
  - “The state of being unidentifiable within a set (the ‘anonymity set’).”
- Informal meaning: Alice uses system S to interact with Bob.
  - *Forward anonymity:*
    Nobody can tell who Alice is.
  - *Reverse anonymity:*
    Nobody can tell who Bob is.

What is anonymity *not*?

- Cryptography (hides what, not who)
- Steganography
- Ordinary non-collection / non-retention
- “I didn’t write my name on it.”

Who needs anonymity? (I)

Citizens

- Avoid profiling by advertisers (DoubleClick, etc)
- Avoid identification by communications partners
- Avoid retribution for unpopular opinions

Who needs anonymity? (II)

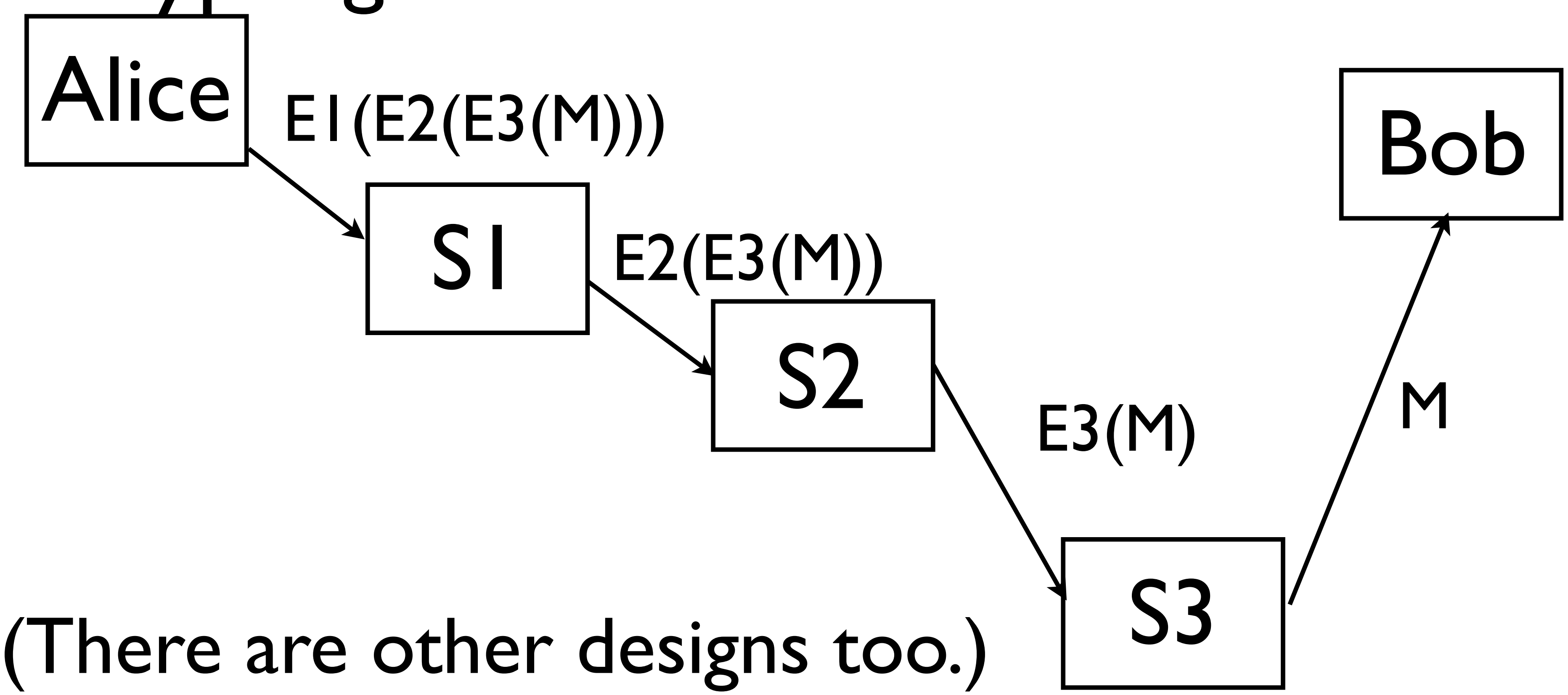
- Businesses (anonymity="security")
  - Investigate competition
  - Hide strategic relationships
- Governments (anonymity="traffic-analysis resistance")
  - Investigation / Intelligence gathering
  - Prevent traffic analysis

Who needs anonymity? (III)

- And, yes, bad people too...
  - ...but they already *have* good means of anonymous communication.

How does it work?

Relay communications through network of decrypting servers.



(There are other designs too.)

Anonymity needs users

- You can't be anonymous alone.
  - (contrast to cryptographic systems)
- Need diverse users to hide own interests.
  - (private networks are useless)

Threat model and usability

- Key choice: Low or high latency?
- Many applications need low-latency
- But to defeat a global eavesdropper, you need high latency.
  - Why? End-to-end correlation...

Correlation attacks

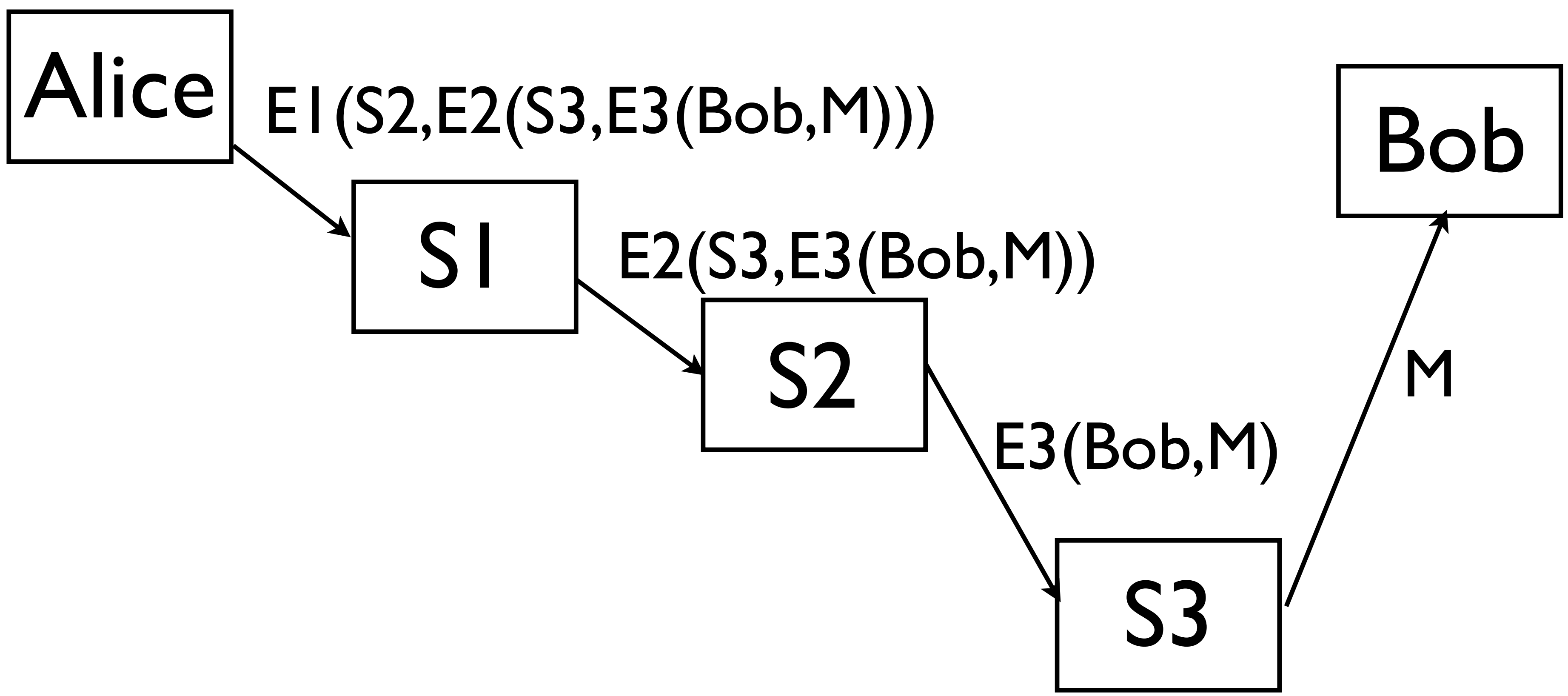
- Attacker can see whole network
- Observe when messages are sent/received
- Correlate timing: Observe that when A sends, B receives.
- Deduce that A is talking to B.
- (Known low-latency defenses are too expensive.)

II. Mixminion

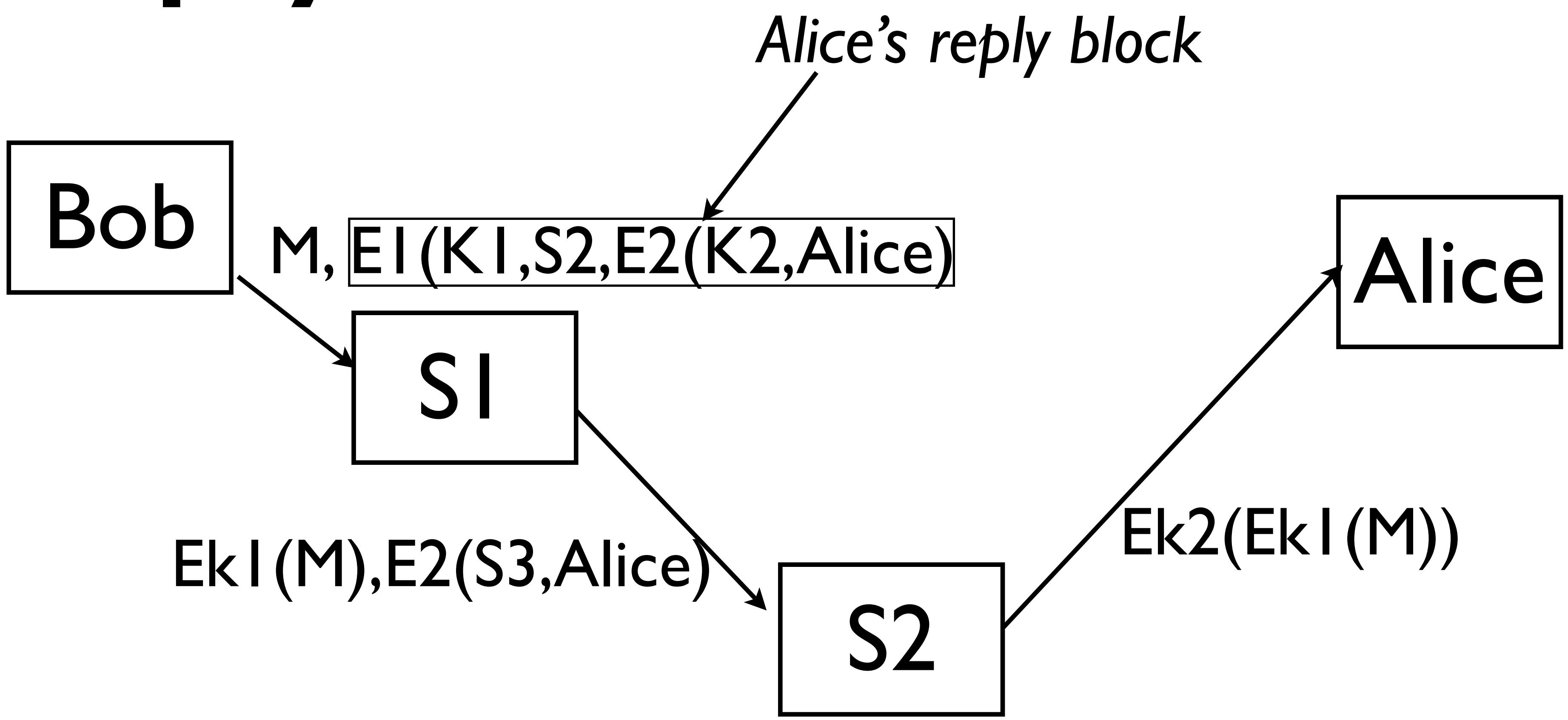
Project overview

- High-latency system
- Tries to defend against all known attacks
  (Assume global adversary who can send msgs and controls some servers.)
- Builds on earlier "Anonymous Remailers"
  Cypherpunk a.k.a. Type I: old and insecure
  Mixmaster a.k.a. Type II: more secure, no replies
  Mixminion a.k.a. Type III: more secure, replies

Routing



Reply blocks & routing



(Stop here, and you have Type I)

Remaining attacks

- Relay attack: Capture Alice's message, send another copy.
- Reply block flooding
- Size correlation: 2MB in == ~2MB out
- Partition on PGP version or other options
- Partition replies from forward messages
- Blending attacks: flooding, n-1, etc.

Mixmaster's Defenses

- Replay: remember message digests for a few days; then use expiration date in msg.
- Reply flooding: no reply blocks
- Size correlation: all messages same size
- Partitioning: no algorithm choice
- Blending... timed dynamic pool mixing

Timed dynamic pool

- Every N minutes, decide whether to relay messages...
- ...but don't relay if too few messages in pool   (prevents trickle attacks)
- ...and don't relay more than a fraction of the messages   (slows flushing attacks)

Attacks on Mixmaster

- Knowledge partioning attacks
- Post-message key exposure
- Users still use Cypherpunk anyway

Mixminion Contributions (I)

- Integrated server directories:
  - Enables key rotation
    - Enables easier replay prevention
- No SMTP for transport
- K/N message fragmentation

Mixminion Contributions (II)

- Forward security
- Single Use Reply Blocks (SURBs)
- Client integration

Integrated directories

- All clients have same network view;
  no knowledge partitioning
- Servers can update info automatically...
- ...including keys!
- So replay caches only need to last as long as keys.

Built-in SSL transport

- Problems with SMTP:
  - not always encrypted
    (even when encrypted, not always authenticated or forward-secure)
  - often unreliable or filtered or clipped

K-of-N fragmentation

- When messages are bigger than packet size
- Tolerates packet loss
- Hides number of packets in large messages

(note patent issues)

Forward security

- Definition: prevent future attacks from exposing current data.
- How:
  - Key rotation
  - DH in SSL protocol

Single-use reply blocks

- Can only be used once: prevents flooding
  - (Uses same trick as replay cache)
- Indistinguishable from forward messages.
  -
- (Need tricky cryptography to beat tagging; see paper for more information.)

Current status

- Free, open-source software
- Written in Python; cross-platform; 40kloc
- Public specification, published design
- Current version is 0.0.7.1
- 34 servers worldwide

III. Tor

Tor: **T**he **o**nion **r**outer

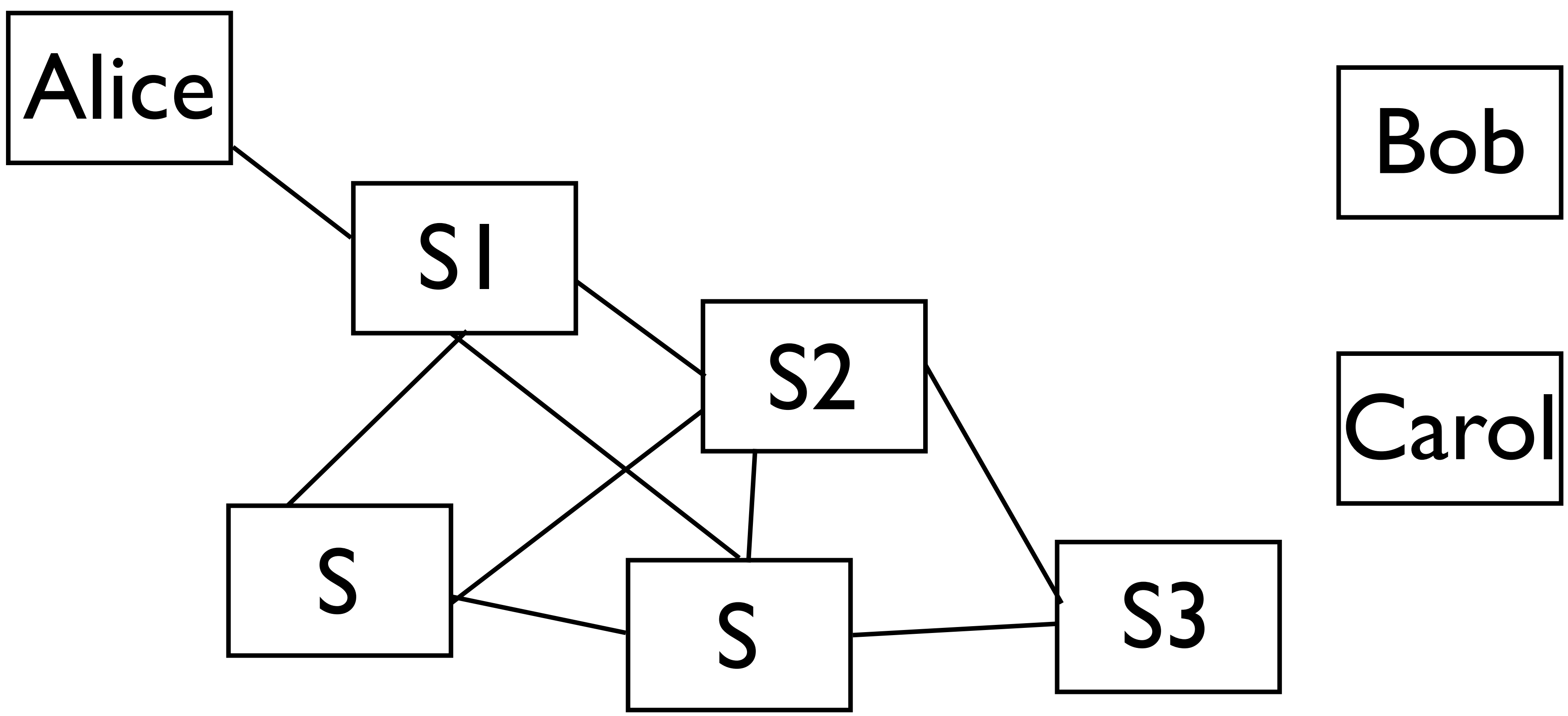
- Onion routing invented ~1996 by Syverson, Goldschlag, Reed at NRL. Test system temporarily deployed.
- 2001: Roger Dingledine joins as external programmer/designer. New implementation.
- 2002: I join.
- 2003: Released as open source
- 2004: NRL funding stalls; EFF begins funding

Onion Routing Goals

- Support existing protocols:
  Must be low-latency
- Resist attacker who can't see both ends
  (Stronger adversary wins because of correlation attacks.)
- Support many users efficiently

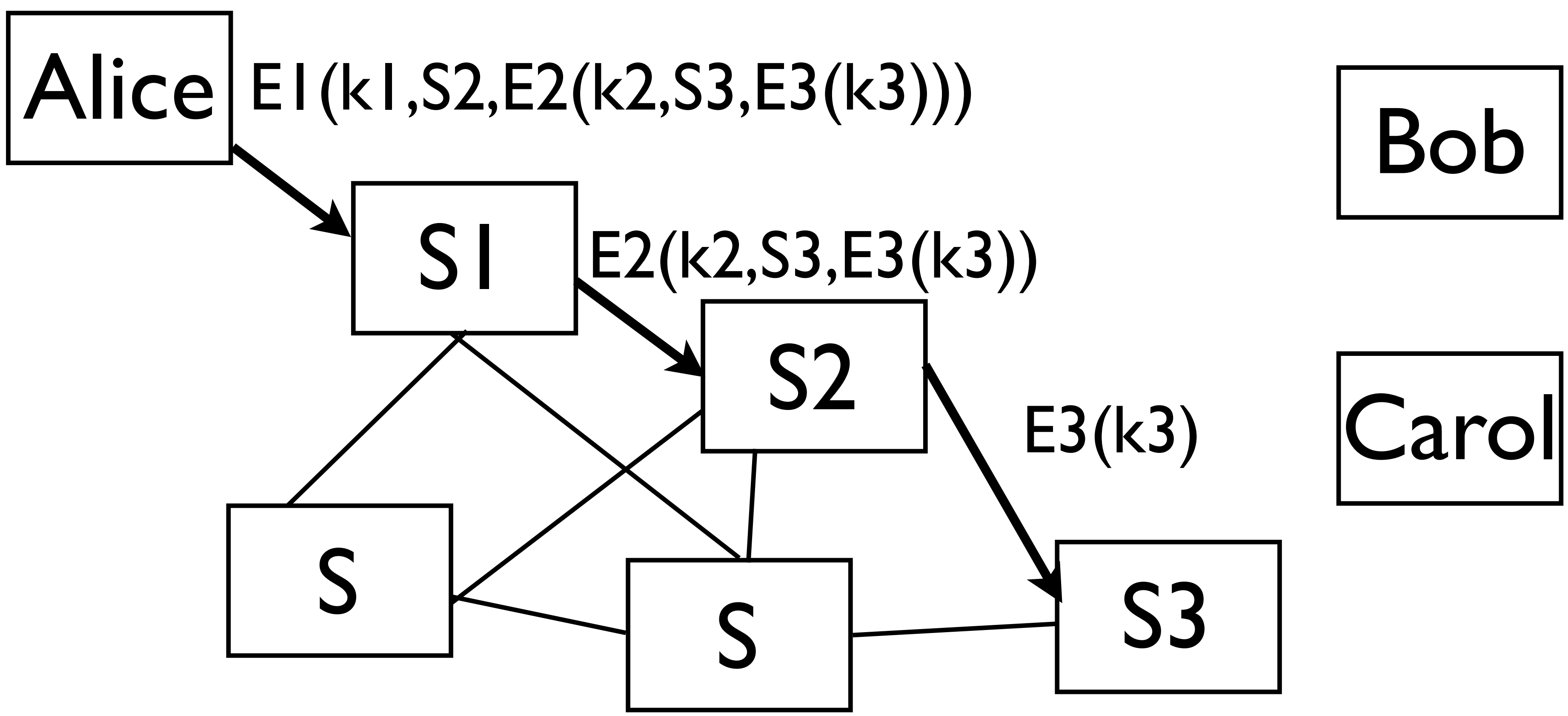
Onion Routing v1

0. Network of encrypted links



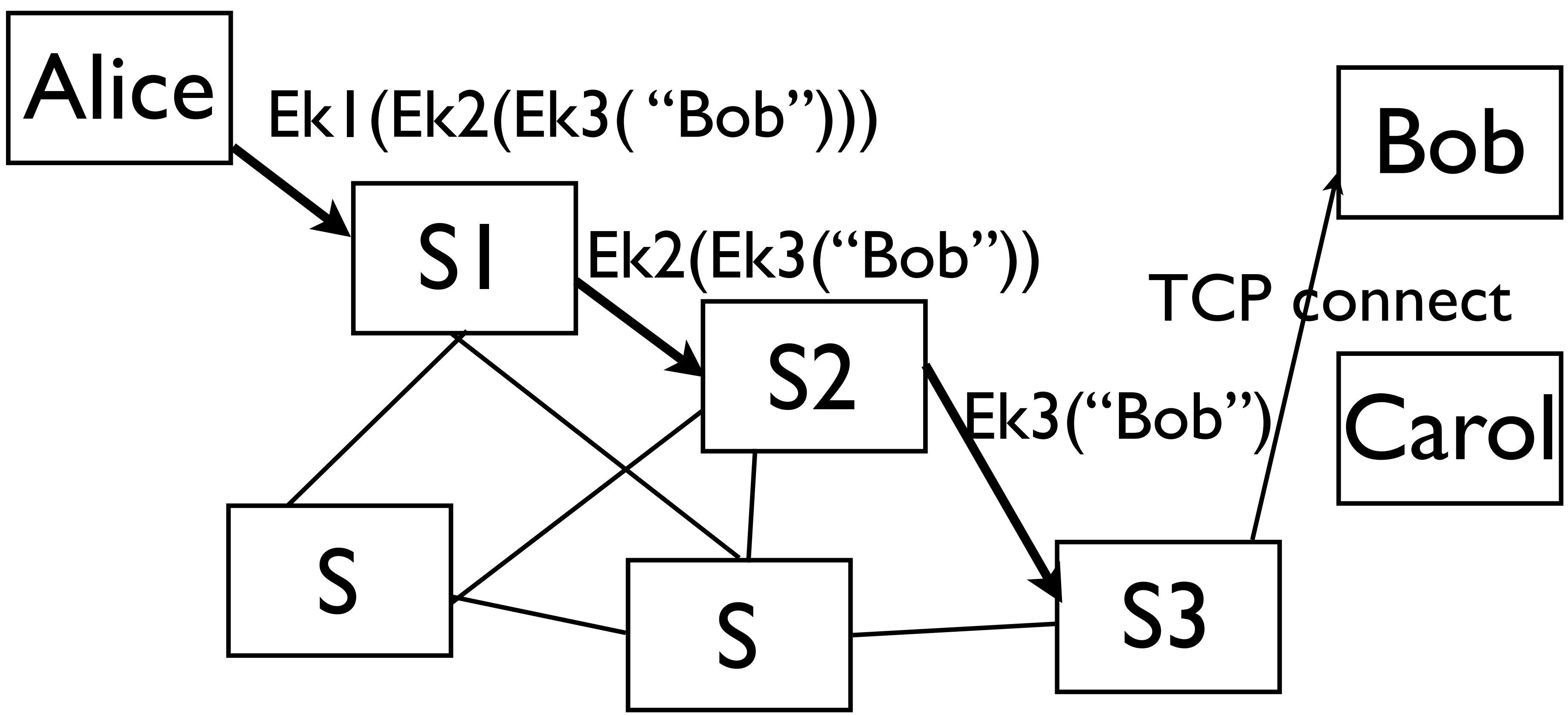
Onion Routing v1

1. Construct a circuit



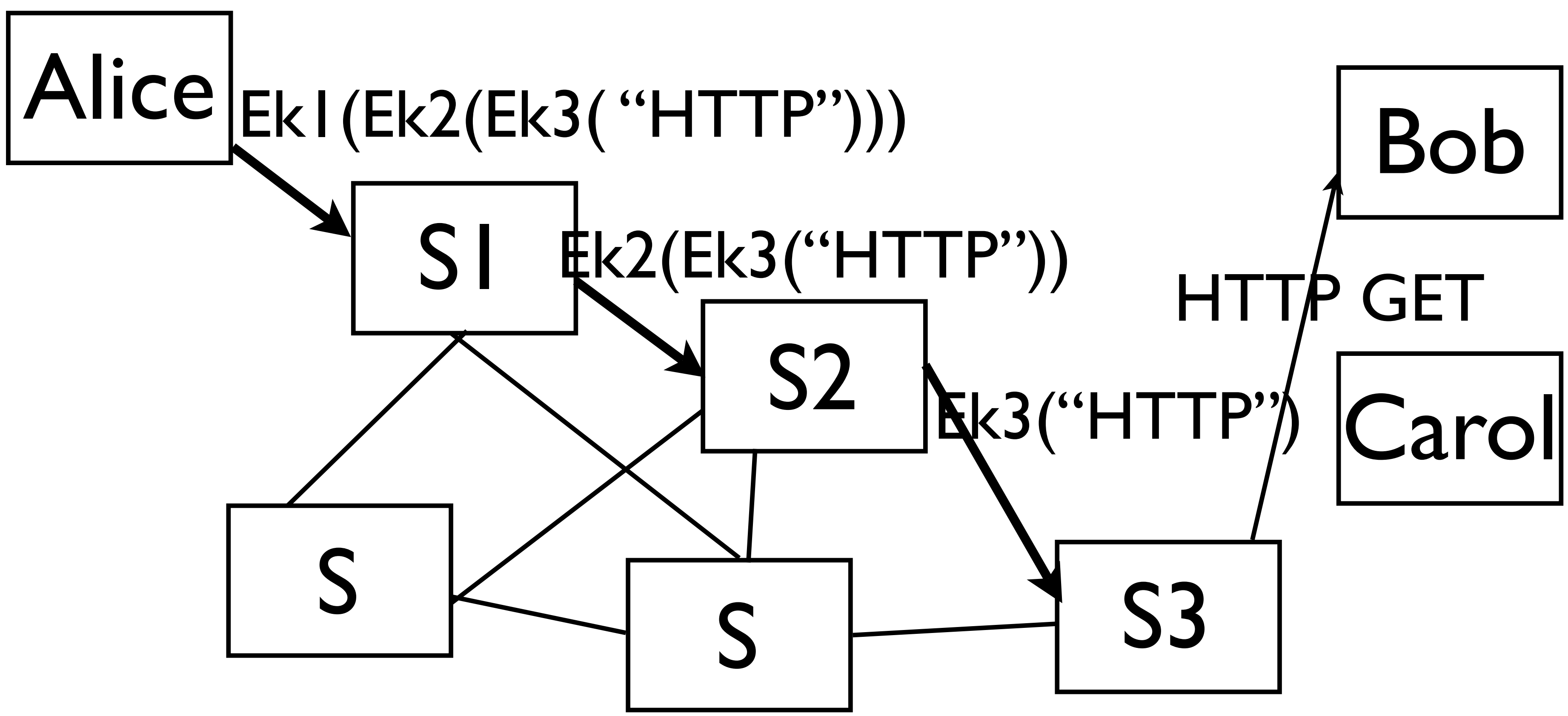
Onion Routing v1

2. Open a stream



Onion Routing v1

3. Communicate data in cells



Problems with OR v1

- Too many proxies
- Too much public-key
- No exit restrictions
- Not forward-secure
- Patented in USA :p

Tor goals

- No protocol scrubbing
- No modifications to existing apps
- More efficient
- End-to-end integrity checking
- Deployable

Tor improvements

- Establish circuits step-by-step:
  forward secure, not patented.
- Exit policies
- Directory service
- Many streams per circuit: less CPU for PK
- SOCKS

Building circuits

- Extend circuit step-by-step
- Diffie-Hellman handshake with each step
- After circuit closed, attacker can’t get keys

Directory servers

- As in Mixminion: servers publish keys and other signed info; clients download.
- Servers cache for efficiency
- (Currently revising.)

Exit policies

- Nobody is willing to run an open proxy
  (Especially for port 25!)
- Servers declare which IPs, ports to support
- Clients choose servers that support targets

SOCKS proxy

- Old OR required new proxy for each protocol
- Tor uses SOCKS; most apps support
- DNS issues

Feature: Location-hidden services

- Bob runs service; Alice can't learn IP address.
- Bob builds circuit to 'intro points'; advertises as "Robert".
- Alice asks for "Robert".
- Alice's Tor builds circuits to 'rendezvous point' and intro point; tells IP about RP.
- IP tells Bob; Bob builds circuit to RP; connects circuits.

Feature: Controller protocol

- (We don't do GUIs; others do.)
- Tor listens for connections from local controller program; controller can observe and adjust Tor.
- Also for scripting: *google.a.b.c.path*
- (Want to write a controller?)

Feature: Improved circuit logic

- For HTTP/FTP: pick high-bandwitdth servers
- For SSH/IRC: pick long-lived servers
- Build likely circuits ahead of time.

Current Status

- Free, open-source software
- Written in C; cross-platform; 40 kloc
- Public specification, published design
- >100 verified servers; >10,000 users
- Latest is 0.1.0.8-rc

IV. The Future

- Development plans
- Technical challenges
- Social/policy challenges

Next in Mixminion

- Client API
- Integrated server status probes
- Pseudonym service
- Decentralized voting directories

Voting directories

- Single directory is single point of failure
- So have multiple directory servers vote on contents
- (What if: directories lie? directories fail? directories disagree on who is a directory?)

Next in Tor

- More efficient (reduce need for PK by 25%)
- Better directory system (allow 10k servers)
- Better DNS proxying
- GUI contest

Technical challenges

- For Mixminion:
  - Long-term intersection attacks (hard)
  - Better nym service design
  - Stylometry (very hard)
  - Collusion-aware introduction

Technical challenges

- For Tor:
  - Incentives to relay
  - Fingerprinting attacks
  - Can mid-latency slow correlation?
  - How to scale to 10k servers?
  - Location diversity (when building circs)
  - Anti-censorship (?)

Social challenges: Blacklisting

- Many services use IPs to punish bad users
- So all of Tor gets blocked.
- Current approach: improved security on service, blacklisting on Tor.
  - Controversial: be easy to blacklist?

Increasing adoption

- How to get people to use Tor?
- So far, promotion seems to work.
- Target diverse groups.

Public perception

- (Ordinary users avoid disreputable nets)
- Targeting good users helps
- Discouraging illegal use helps
- Having good sponsors helps
- More education needed!

Bandwidth, sustainability

- [Tor is first widely deployed *volunteer* low-latency network.]
- Social problem is getting more servers
- Idea: give server operators better service?
  - (This has anonymity problems)

Legal challenges

- Warning: I am not a lawyer. Especially not here.
- Are server operators liable for traffic? (unlikely)
- Must server operators aid wiretaps? (barely)
- Must designers aid wiretaps? (no)
  (Under US law, no way.)
- *Should* network be backdoored? (IMO, no!)
- Circuits cross jurisdictions, so local policies are limited.

Policy challenges

- Free expression needs anonymous speech? What about ability to exercise this right?
- Must educate law enforcement
- Must educate policy-makers:
  Privacy **is** security.
  Allow data holders to protect their own data.
  Never let one party to a compromise choose the compromise.

Q&A

- **Tor** http://tor.eff.org/
- **Mixminion** http://mixminion.net/
- **Anonymity Bibliography**
  http://freehaven.net/anonbib/
- **Email**
  Nick Mathewson <nickm@freehaven.net>
  PGP: B35B F85B F194 89D0 4E28 C33C 2119 4EBB 1657 33EA