Dissent: Accountable Anonymous Group Communication

Bryan Ford

Joan Feigenbaum, David Wolinsky,

Henry Corrigan-Gibbs, Shu-Chun Weng, Ewa Syta

Yale University

Vitaly Shmatikov, Aaron Johnson

University of Texas at Austin

Presented at MPI-SWS – Jan 4, 2012

Why do we want anonymity online?

Many motivations:

- Discuss sensitive/controversial topics safely; protect freedom of speech
- Citizens of authoritarian states evading repression
- Voting in elections or deliberative organizations
- Collaborative content creation/editing, e.g., Wikipedia
- Protect secrecy of bids in commercial auctions
- Law-enforcement “tip” or whistleblowing hotlines
- Peer review processes for research, journalism
- …

Motivating Scenario

Alice, Bob, Charlie, Dave, & friends

- Citizens of Repressistan
- Wish to connect, organize online safely

Government is powerful but not all-powerful

- Can't just “turn off Internet” indefinitely or throw *all* protesters in jail: cost is too high
- Must identify and make examples of the movement's outspoken “activist leaders”

Alice & friends need “strength in numbers”

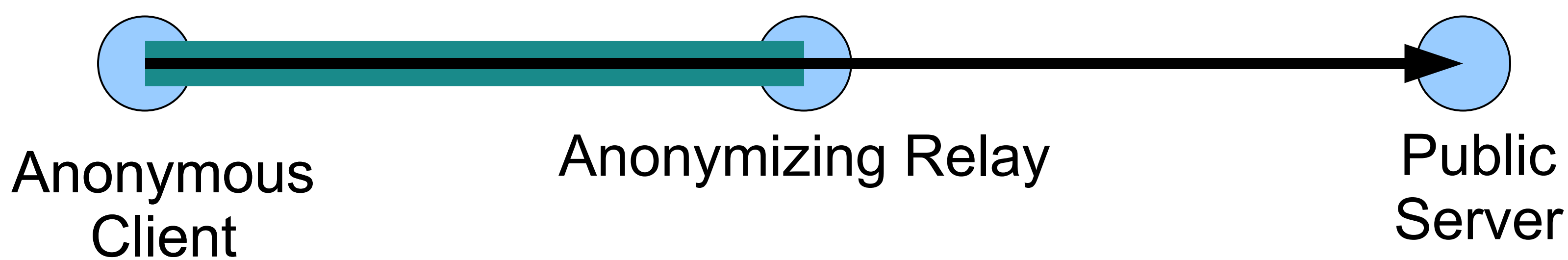
Being Anonymous: Naive Ways

Assume the Internet is “anonymous enough”

– IP addresses never provided real anonymity; many ways to track users, machines, browsers

Use centralized anonymizing relays/proxies

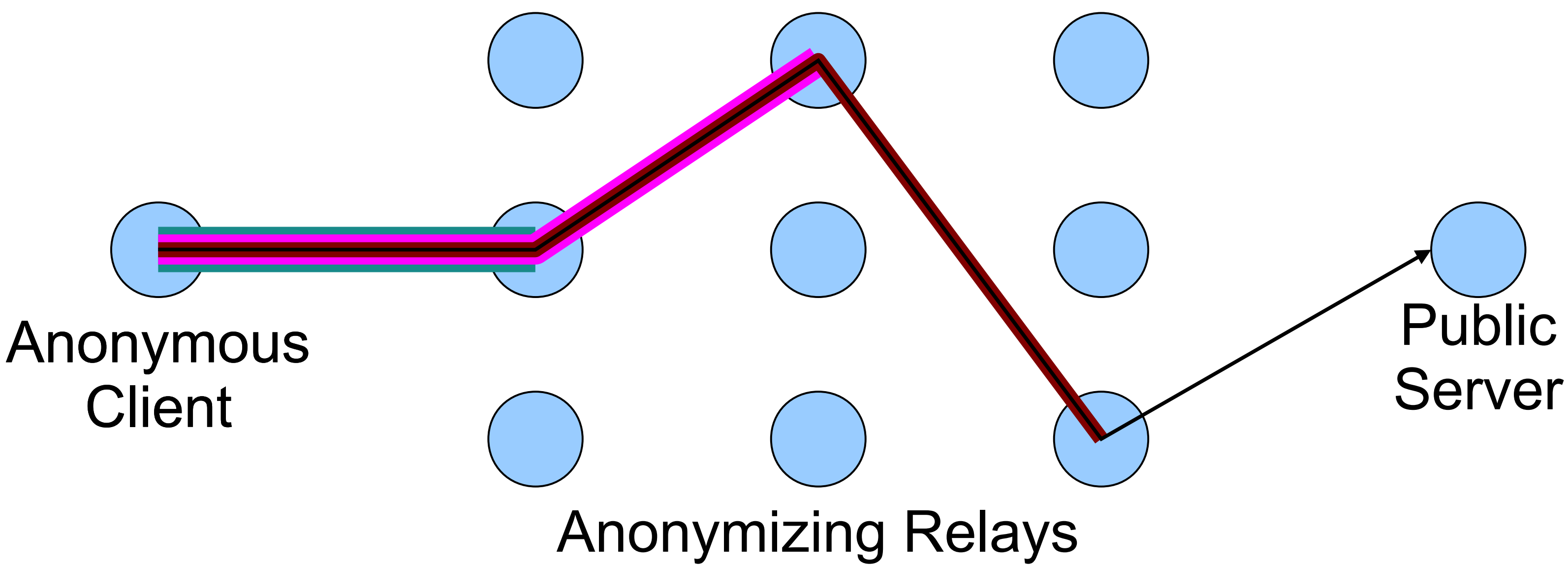
– Central point of failure, prime compromise target



Being Anonymous: Better Ways

MIX networks, onion routing systems: e.g., Tor

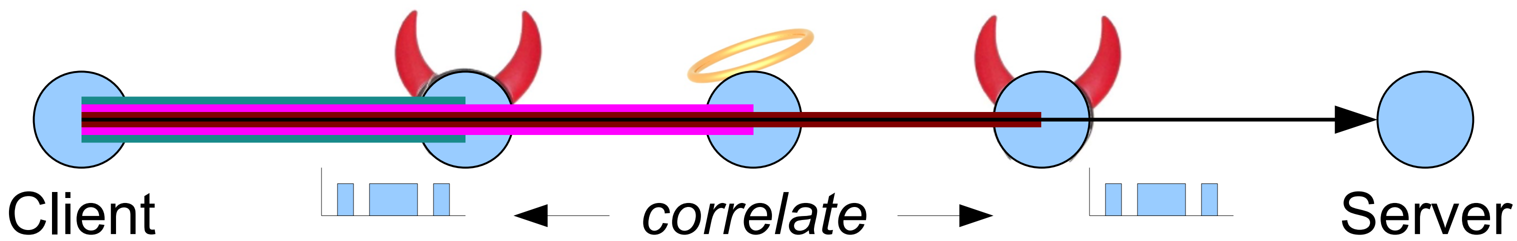
- Tunnel through a series of anonymizing relays
- Protects even if any one is malicious or hacked



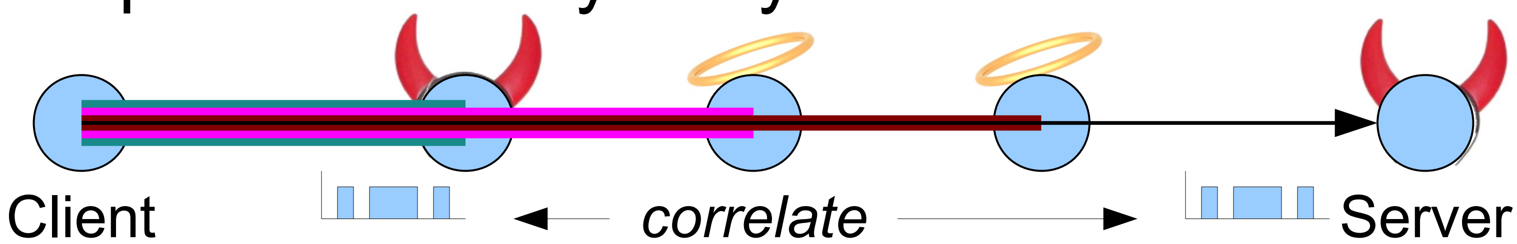
Limitations of Onion Routing

Vulnerable to traffic analysis, correlation attacks

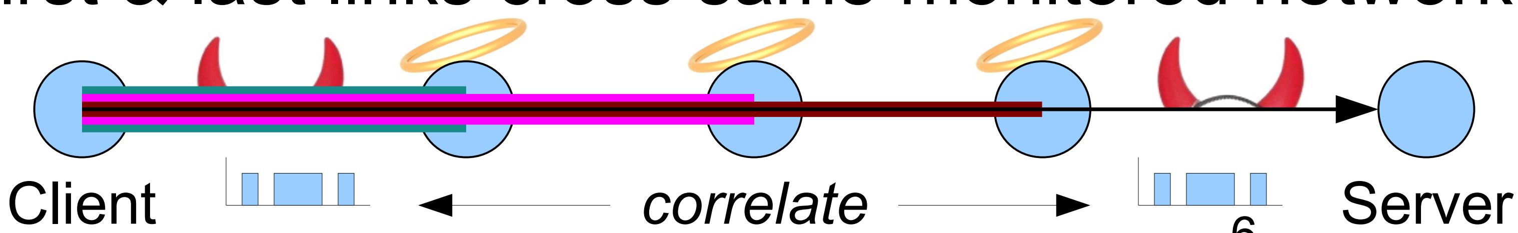
– compromised, colluding first & last hop relays



– compromised entry relay and server



– first & last links cross same monitored network path



Limitations of Onion Routing

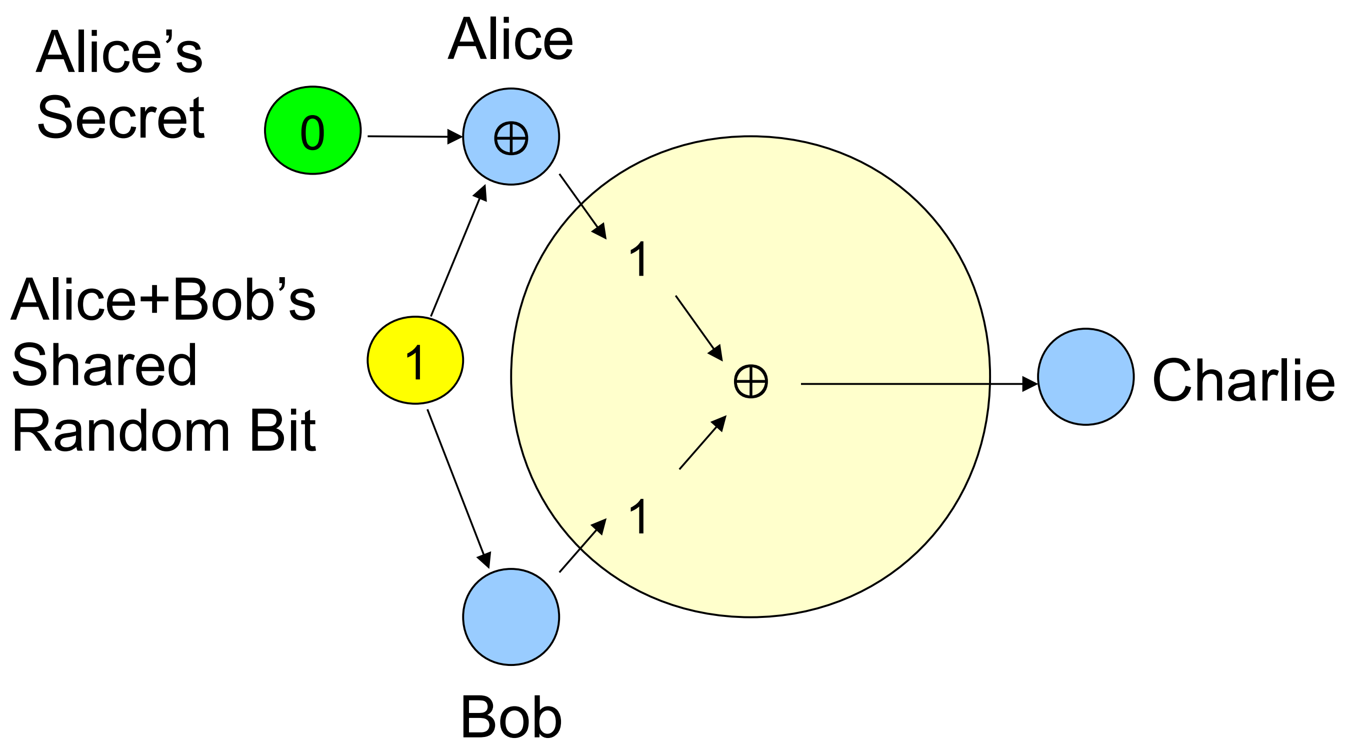
Vulnerable to anonymous abuse

- No accountability for misbehavior
  - No one knows you're a dog,
    so everyone gets to behave like a dog
- Unlimited supply of “fresh” pseudonyms
  - Create sock-puppet “supporters” in online forums
  - Vote many times in online polls, elections
  - Get banned, come back with new IP address
    (loser is *next* user of old IP address or exit relay)

Dining Cryptographers (DC-nets)

Another fundamental Chaum invention from the 80s...

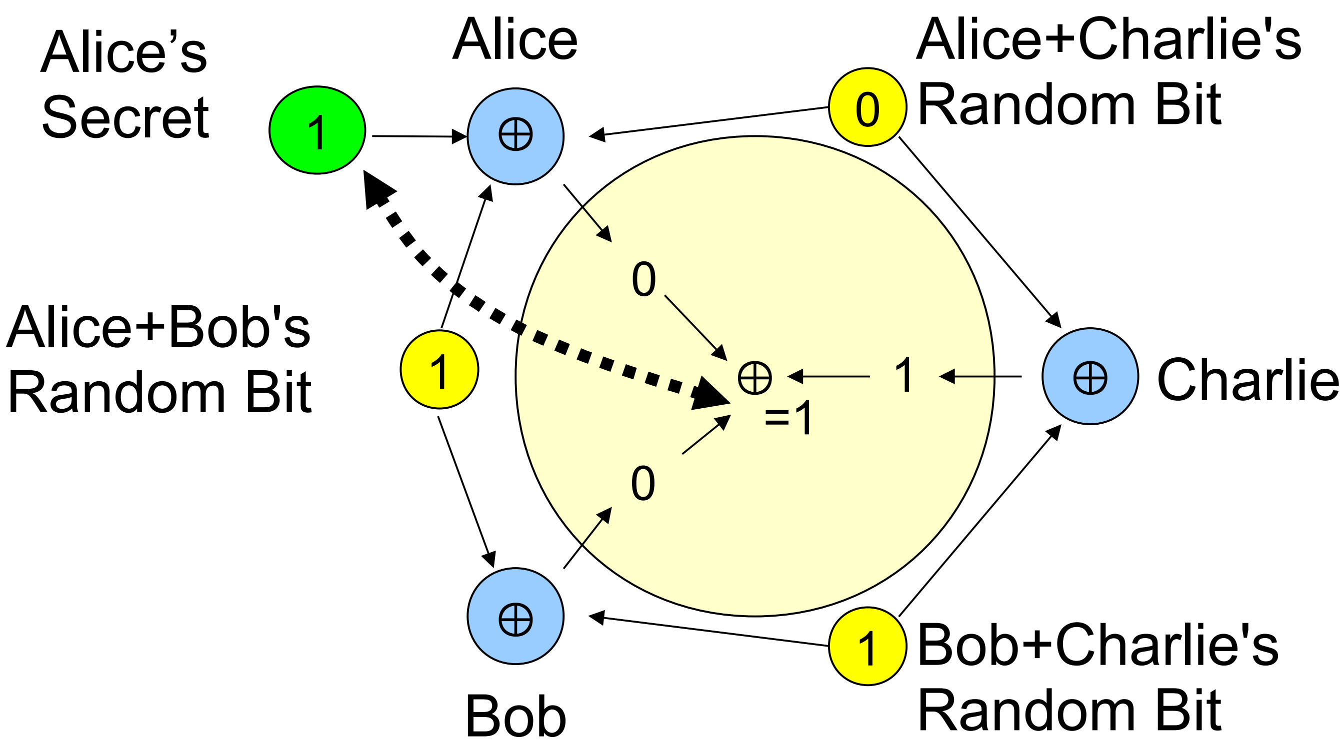
- Ex. 1: “Alice+Bob” sends a 1-bit secret to Charlie.



Dining Cryptographers (DC-nets)

Another fundamental Chaum invention from the 80s...

- Ex. 2: Homogeneous 3-member group anonymity



Dining Cryptographers (DC-nets)

Tantalizingly strong anonymity guarantees:

- Unconditional information-theoretic anonymity
  - (if we use “real” random coins, which we won't)
- Optimal security against traffic analysis & collusion
  - Anonymity set = nodes *not* colluding with adversary

Never successfully used in practical systems:

- No provision for accountability or proportionality
  - Malicious member can jam by sending random bits
- Not readily scalable to large groups
  - Especially with node failure, network churn

Talk Outline

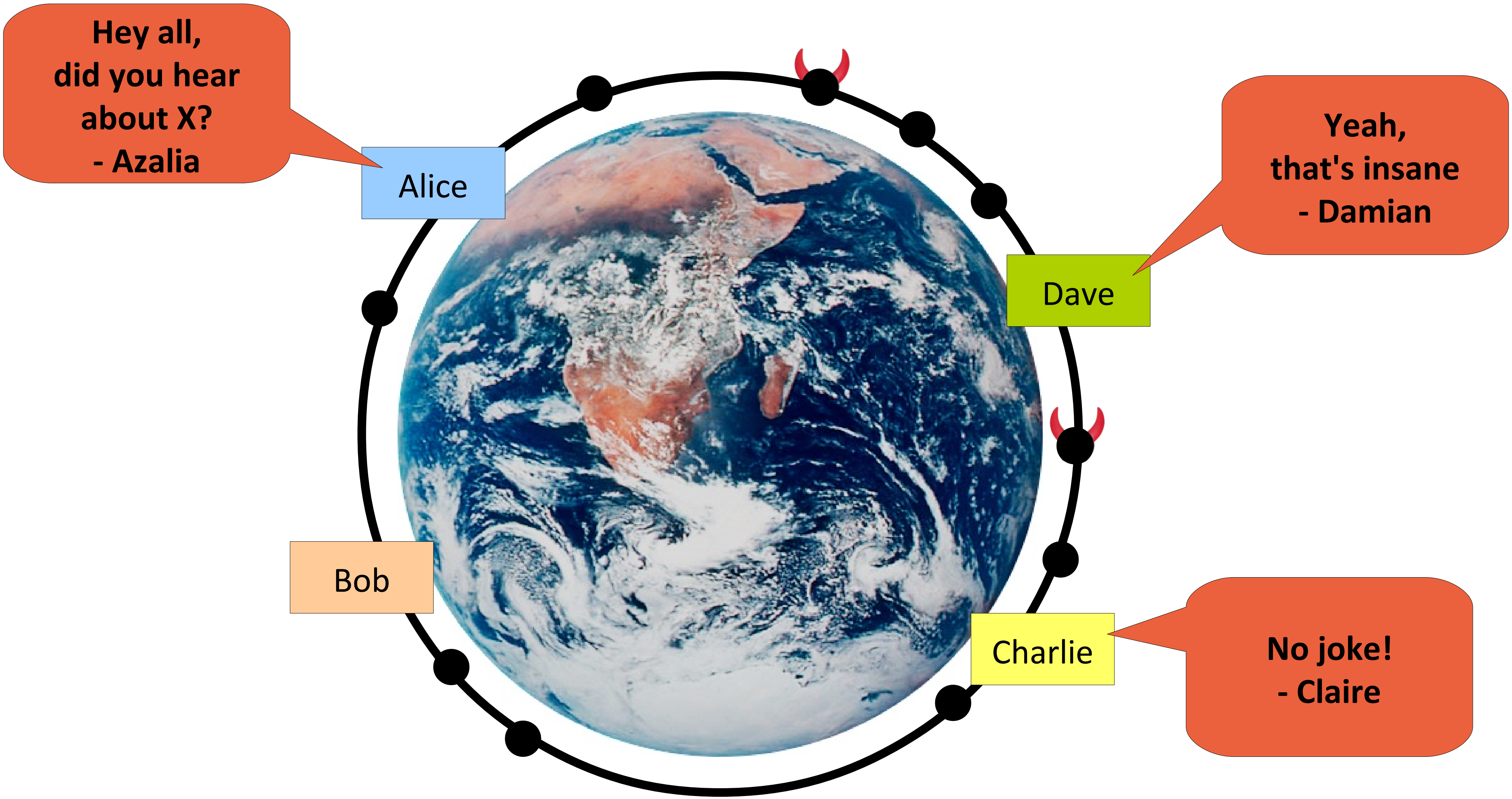
- ✔ Online Anonymity: What and Why?
- **An Accountable Group Anonymity Model**
- A MIX-based Accountable Shuffle
- Dining On Schedule: Accountable DC-nets
- Current Results and Ongoing Work

Dissent: Accountable, Proportional Group Anonymity

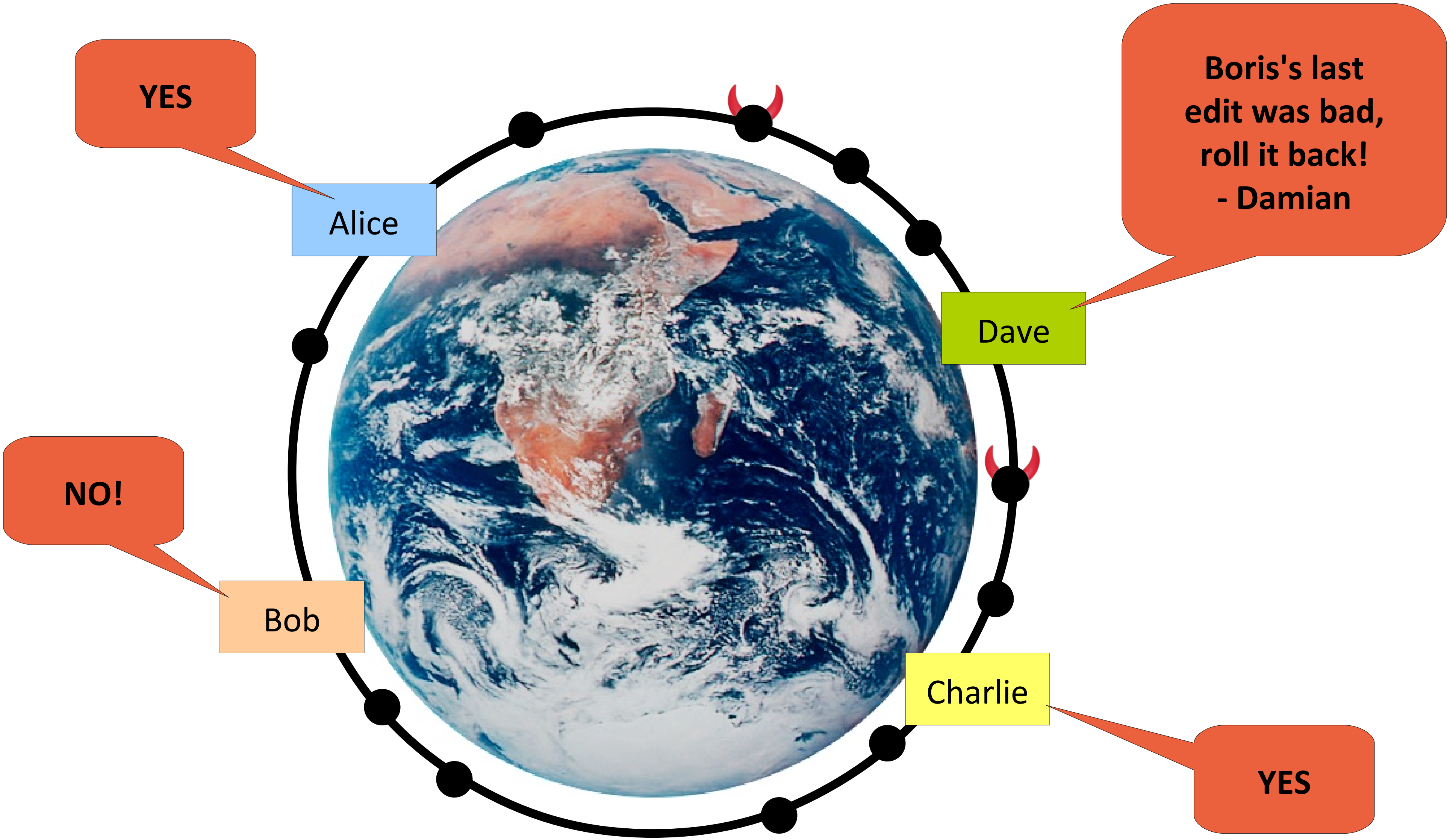
A **group communication** model akin to DC-nets

- Assumes a well-defined *group* of $N \geq 2$ parties wishes to communicate with each other online
- Members have persistent *identities* known to each other – may or may not be “public”
- Wish to hide *which member* sent a message, but make it clear that *some member* sent it
- Wish to maintain *proportionality:* each member gets one message, vote, bid, etc. per “round”

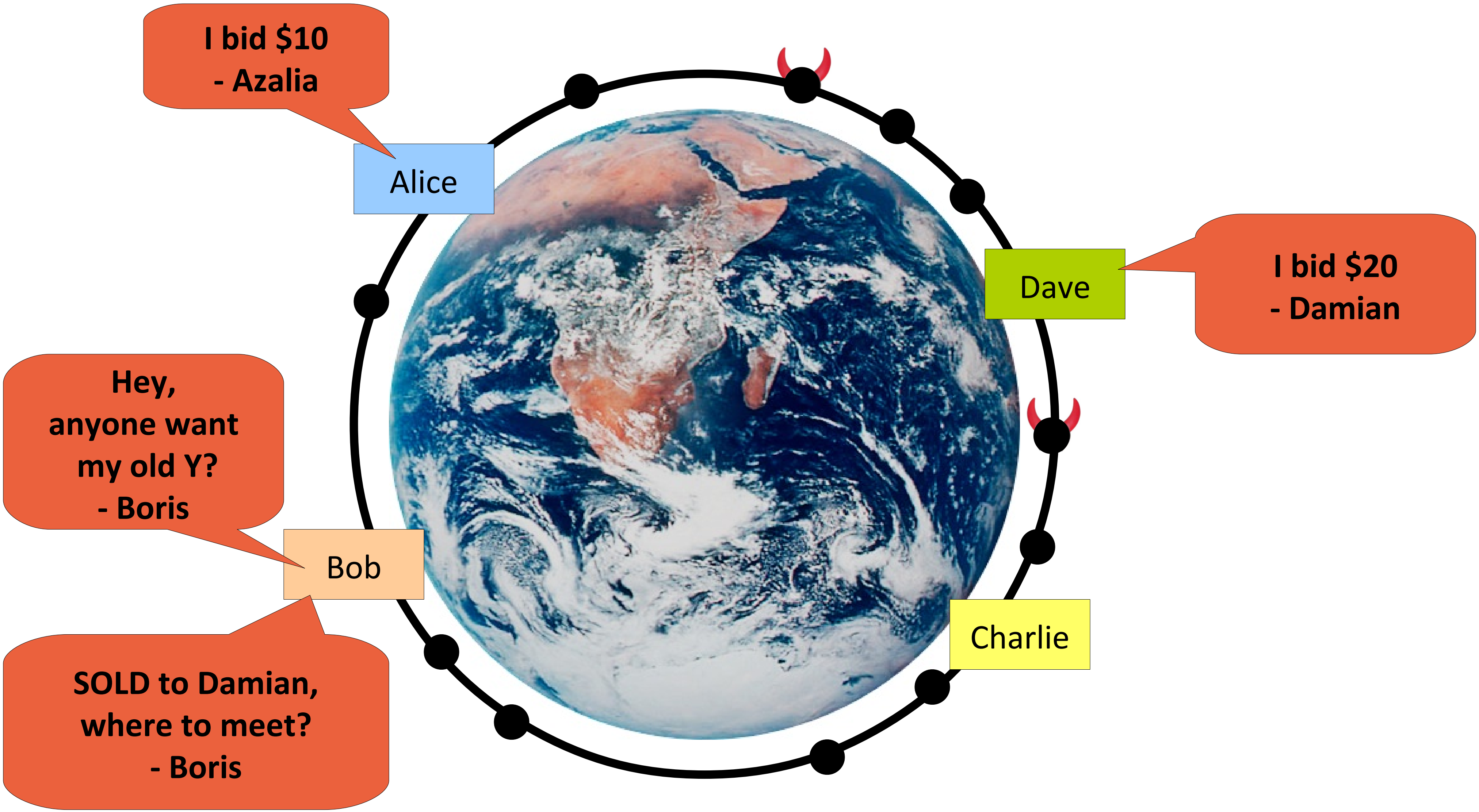
Bulletin Boards, Chat Rooms



Group Voting, Deliberation

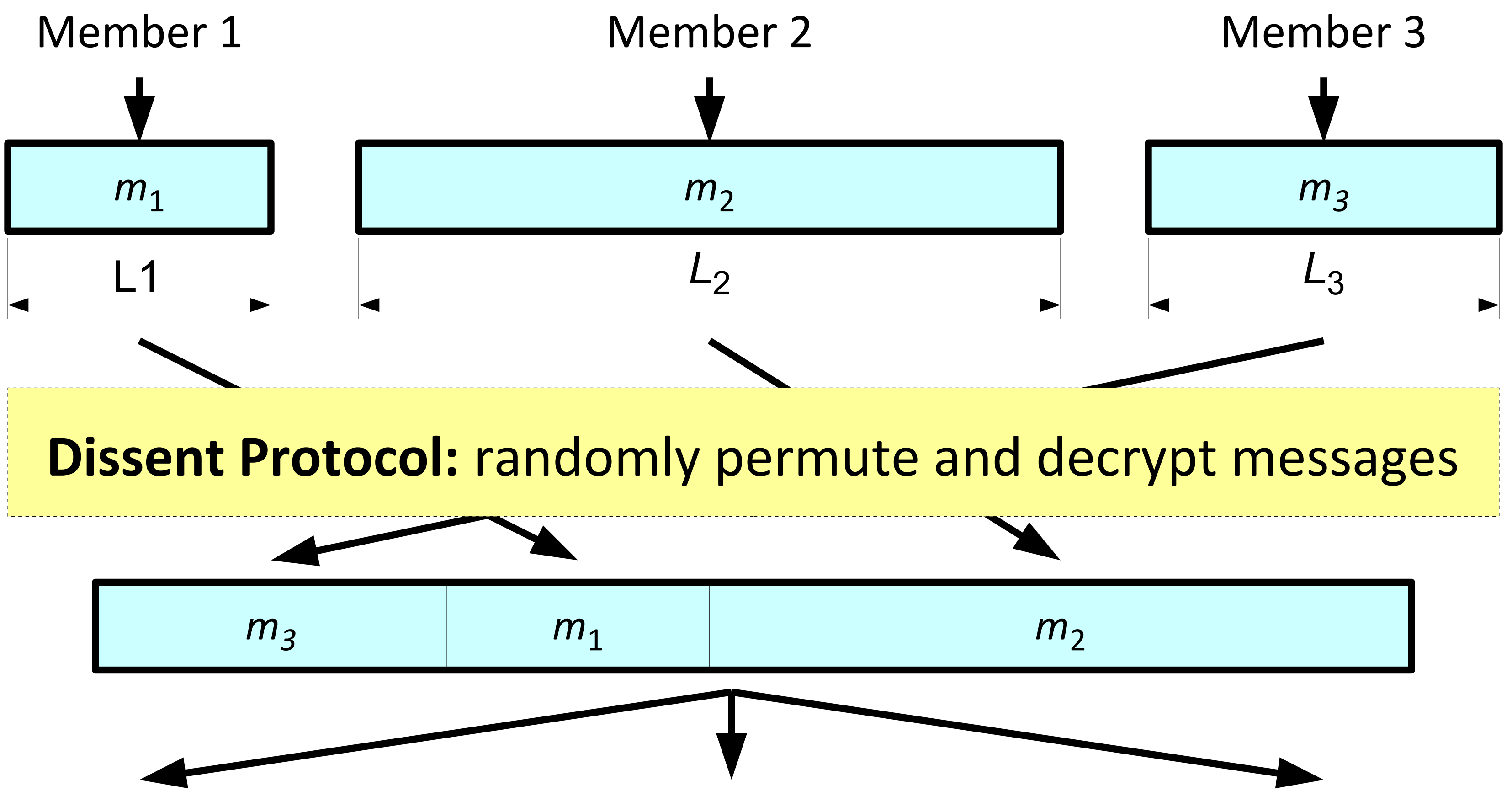


Anonymous Auctions



“Shuffling” Into Group Anonymity

Input: secret message $\boldsymbol{m_i}$ of length $\boldsymbol{L_i}$ from each group member i



Output: send all members' messages to target(s) in shuffled order

Anonymity with Accountability

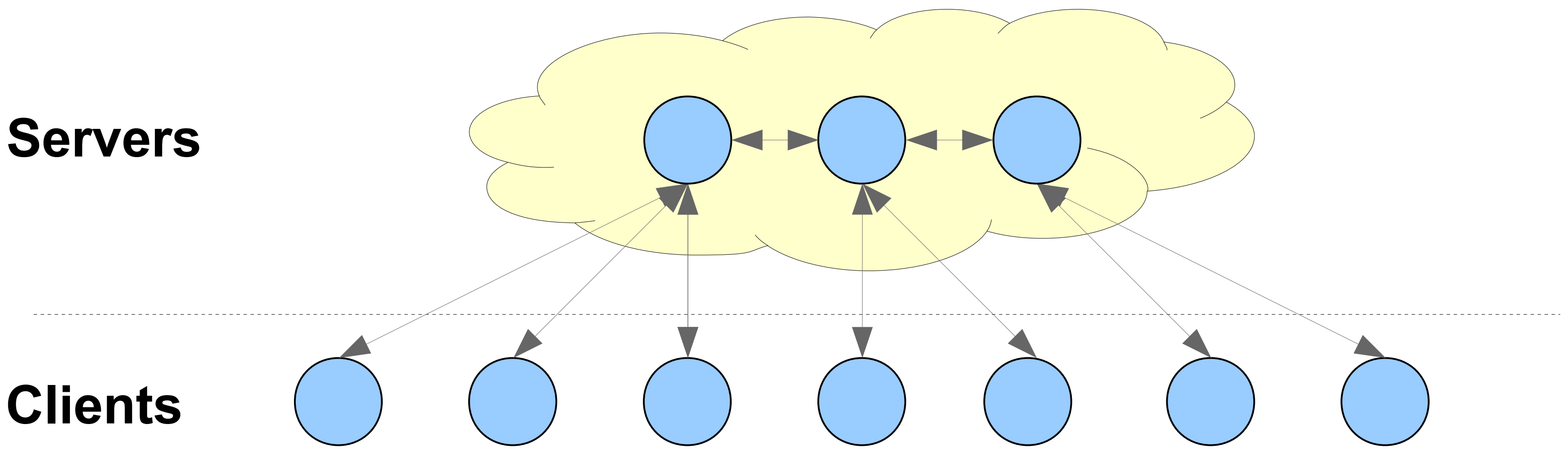
Dissent's group model facilitates ***accountability:***

- Not "just anyone" may send, vote, bid, etc. → group membership can reflect "credentials"
  - Board members, journalists, etc., acting collectively
- 1-to-1 shuffle prevents Sybil attacks
  - Each "real" group member can send *exactly one* message per shuffle, cannot act like many users
- Resistant to anonymous disruption attacks
  - If any member attempts to jam or block protocol, Dissent exposes attacker's *real* identity, can expel

Dissent Network Model

Quasi-Client/Server Model:

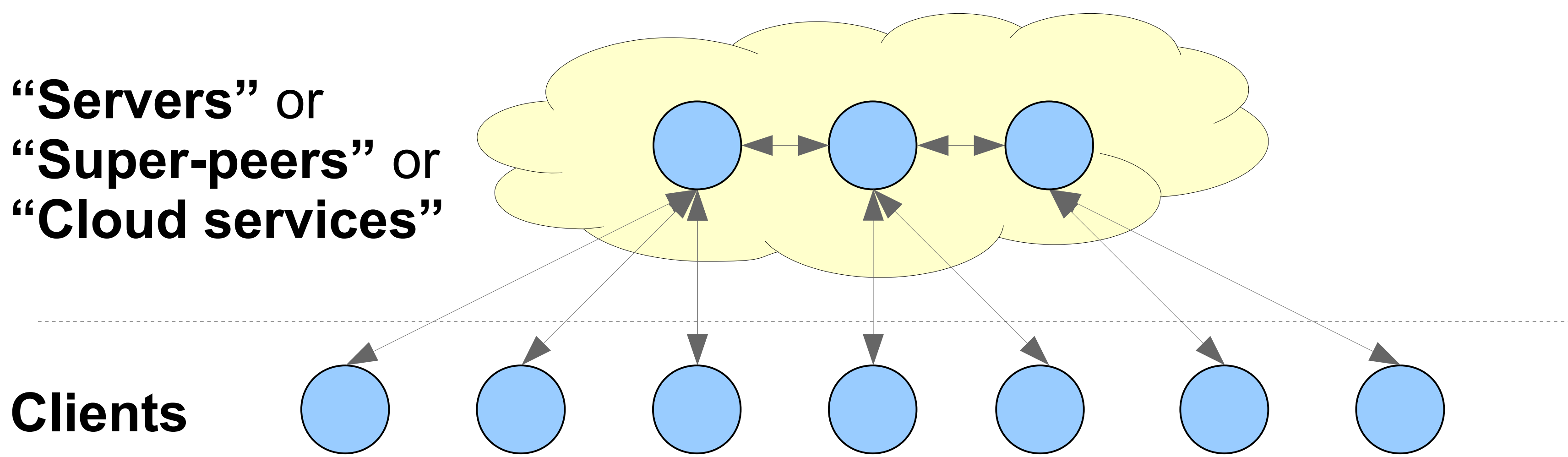
- **Client** nodes represent group members (users) wishing to post messages anonymously
- **Server** nodes are intermediaries that facilitate anonymous group communication



Dissent Network Model

Dissent "**servers**" could actually be:

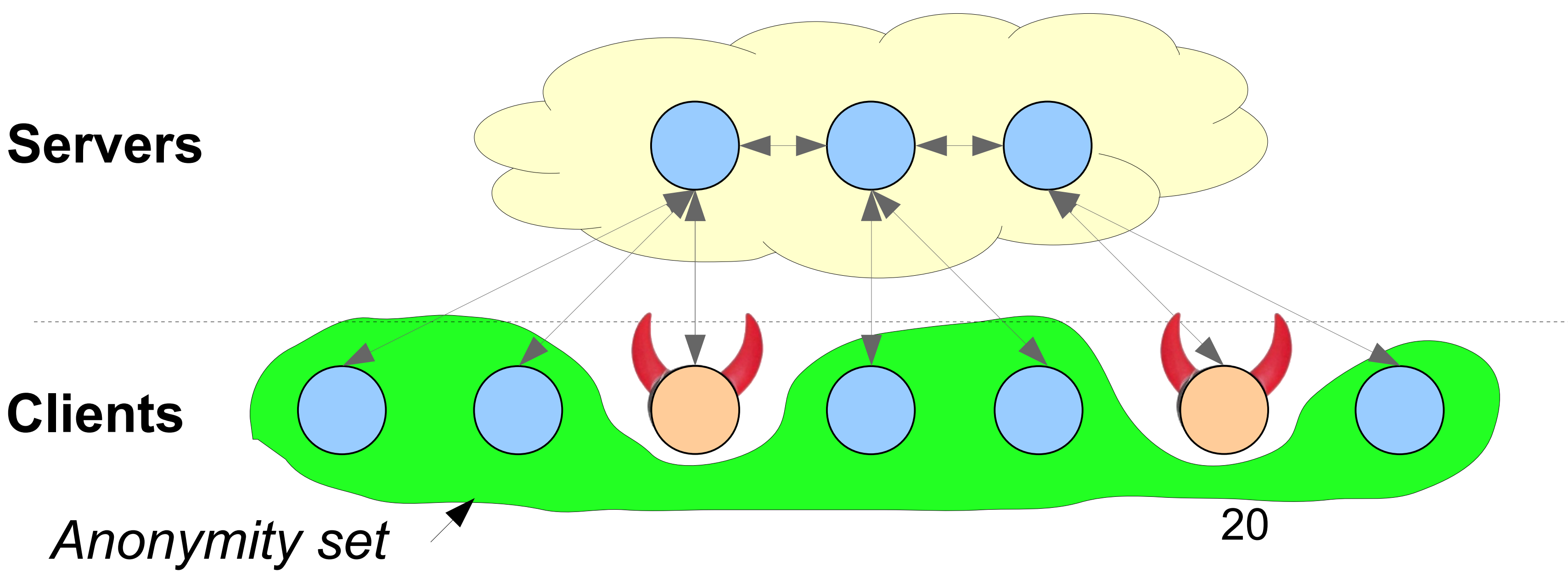
- Dedicated or volunteer servers, like Tor relays
- Super-peers chosen from clients, P2P-style
- Cloud-based virtual services run professionally



Dissent Trust Model

Any number of clients may be malicious, collude

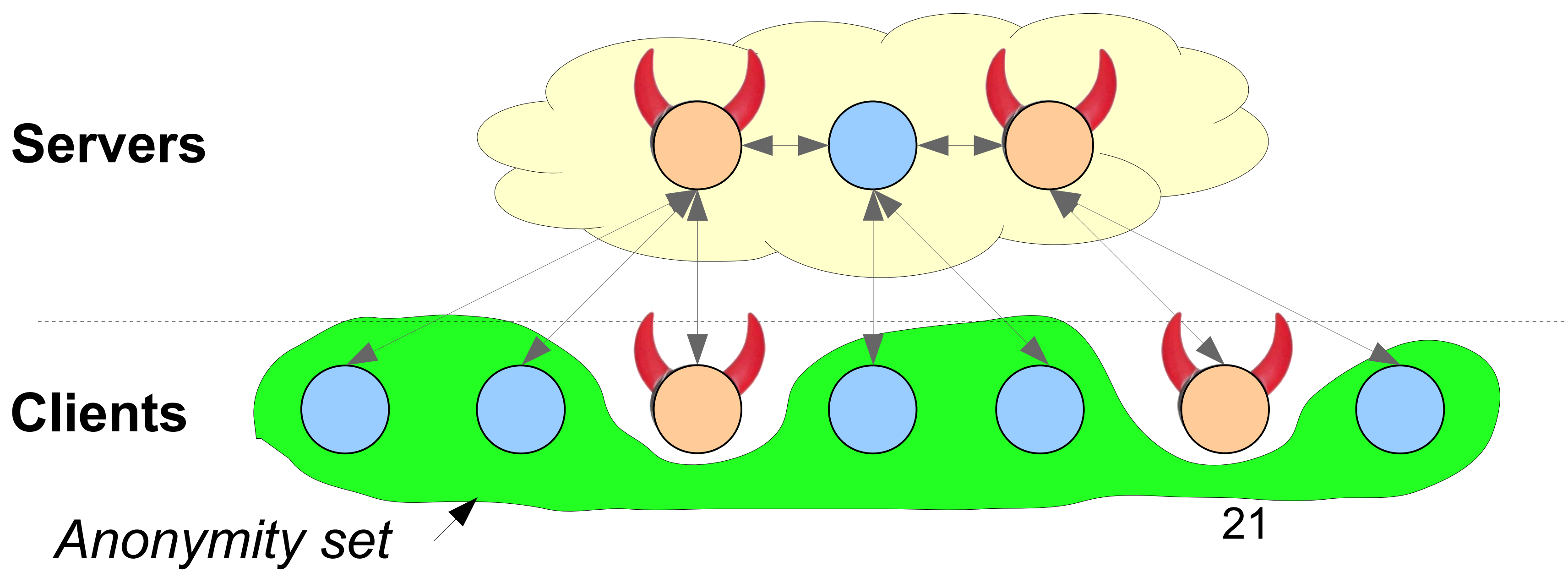
- A client's effective "anonymity set" includes all clients *not* colluding with adversary
  - (trivally optimal for colluding adversary model)



Dissent Trust Model

All but one server may be malicious, collude

- Clients **need not know** which server(s) to trust
- Clients of dishonest servers are *still protected!*
  - Unlike existing MIX, DC-nets cascading schemes



Talk Outline

- ✓ Online Anonymity: What and Why?
- ✓ An Accountable Group Anonymity Model
- **A MIX-based Accountable Shuffle**
- Dining On Schedule: Accountable DC-nets
- Current Results and Ongoing Work

The Dissent Shuffle – Overview

Assuming *N* clients, *M* servers:

1. Servers choose **inner**, **outer** encryption keys
2. Clients encrypt messages in 2*M* “onion layers”
3. Servers MIX and decrypt **outer** layers
4. Clients validate result, broadcast **go/no-go**
5. Servers either:
   - Decrypt **inner** layers of permuted messages, *or*
   - Reveal permutation and trace at least one disruptor

Phase 1: Setup

- Assume at outset:
  - All nodes know each others' public signing keys
  - All nodes sign and verify all messages
- Each **client** *i* chooses:
  - Secret message m_i padded to fixed length L
- Each **server** *j* creates **outer** and **inner** encryption key pairs: (O_j, O'_j) and (I_j, I'_j)
- Each **server** *i* broadcasts public encryption keys O'_j, I'_j to all clients

Phase 2: Onion Encryption

Each **client** *i*:

- Encrypts m_i with **inner** keys $I'_M, \ldots, I'_1$ to create m'_i
- Encrypts m'_i with **outer** keys $O'_M, \ldots, O'_1$ to create m''_i

Example with $N = 3$ clients, $M = 2$ servers:

$$m''_1 = \{\{\ m'_1 = \{\{\ m_1\ \}I'_2\ \}I'_1\ \}O'_2\ \}O'_1$$

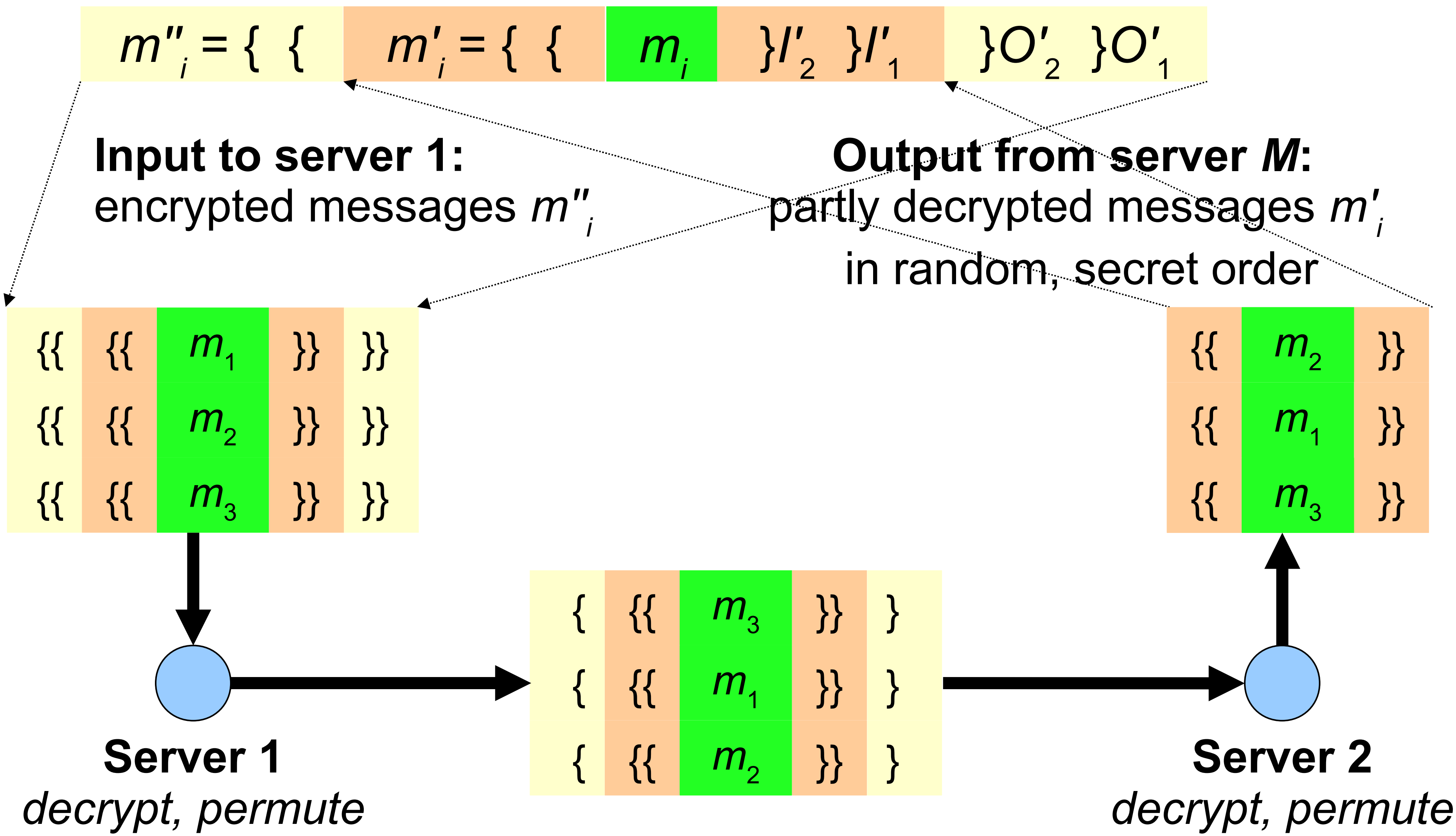
$$m''_2 = \{\{\ m'_2 = \{\{\ m_2\ \}I'_2\ \}I'_1\ \}O'_2\ \}O'_1$$

$$m''_3 = \{\{\ m'_3 = \{\{\ m_3\ \}I'_2\ \}I'_1\ \}O'_2\ \}O'_1$$

Phase 3: Anonymization (1)

- Server 1 collects messages (m''_l, …, m''_N).
- For $j \leftarrow 1$ to M, server j
  - Decrypts j^{th} **outer** encryption layer with key O_j
  - Randomly permutes the list of partially decrypted messages, *temporarily* saves the random permutation
  - Forwards permuted message list to server $j+1$ (if j < N)
- Server M broadcasts permuted m'_i list to all nodes

Phase 3: Anonymization (2)



Phase 4: Validation

After anonymization, **no client or server** knows the final permutation, but every client *i* should see his own m'_i in the list!

Each client *i* looks for m'_l in the permuted list

- **Present** → client *i* broadcasts "**GO**"
- **Absent** → client *i* broadcasts "**NO-GO**"

Phase 5: Decryption or Blame

Each server j collects all GO/NO-GO messages

- **GO messages from *all* clients:**
  - Each server j broadcasts his private **inner** key I_j
  - All clients use **inner** keys $I_1,...,I_M$ to decrypt all m'_i revealing all cleartext messages m_i
- **NO-GO message from *any* client:**
  - Each server j broadcasts proof that he decrypted and permuted messages properly in Phase 3
  - All servers use these proofs to uncover disruptor(s)

Anonymity of the Shuffle

Every server secretly randomizes the shuffle

- Even if *any subset* of servers collude, **any single honest server protects all clients**
- Resists traffic analysis, correlation attacks: traffic reveals *nothing* about who sent what
- Malicious ciphertext substitution or duplication always detected at GO/NO-GO if not before
- All key security properties provable (see CCS '10 paper for details)

Accountability of the Shuffle

Any NO-GO message obliges *all* nodes to “prove their innocence”, i.e., that they:

- Correctly encrypted messages in phase 2
- Correctly decrypted/permuted in phase 3
- Correctly validated final list in phase 4

This process reveals the “secret” permutation, **but** leaves permuted cleartexts (m_j) undecipherable

- Protected by all honest servers' **inner** keys

Scalability of the Shuffle

All phases parallelizable *except* 3: anonymization

- *M* servers "take turns" permuting & decrypting
- Not show-stopping if *N* clients $\gg$ *M* servers

Scenario 1: servers are *M* independently run, cloud-based "Dissent service providers"

- Each "server" is a parallel, fault-tolerant cluster

Scenario 2: servers are *M* super-peers chosen randomly from $N \gg M$ clients in P2P setting

- If $f{\cdot}N$ clients are faulty, f^M chance of failure

Talk Outline

- ✓ Online Anonymity: What and Why?
- ✓ An Accountable Group Anonymity Model
- ✓ A MIX-based Accountable Shuffle
- **Dining On Schedule: Accountable DC-nets**
- Current Results and Ongoing Work

Limitations of MIX-based Shuffle

Inefficient when only *some* clients are “talking”

- All clients must pad messages to same length
- If one “talker” wants to send useful *L*-byte file, each “listener” must send *L* bytes of garbage

M-server serialized path still incurs latency

- We'd prefer if *entire* protocol was parallelizable

Shuffle must “start over” if a client comes or goes

- Natural or malicious churn could cause DoS

Solution

Use MIX-based shuffle to bootstrap & schedule a **dining cryptographers (DC-nets)** phase

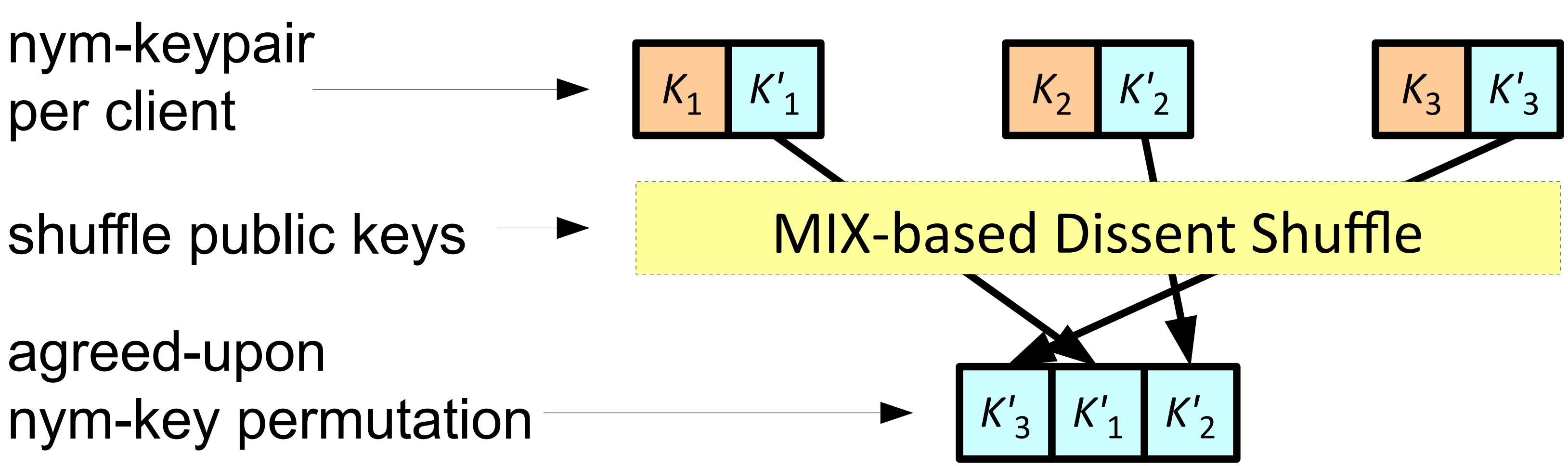
Outline

1. Setup pseudonym keys and transmit schedule
2. Share secrets between all client/server pairs
3. Transmit any number of DC-nets rounds

Setup 1: Nym-Key Schedule

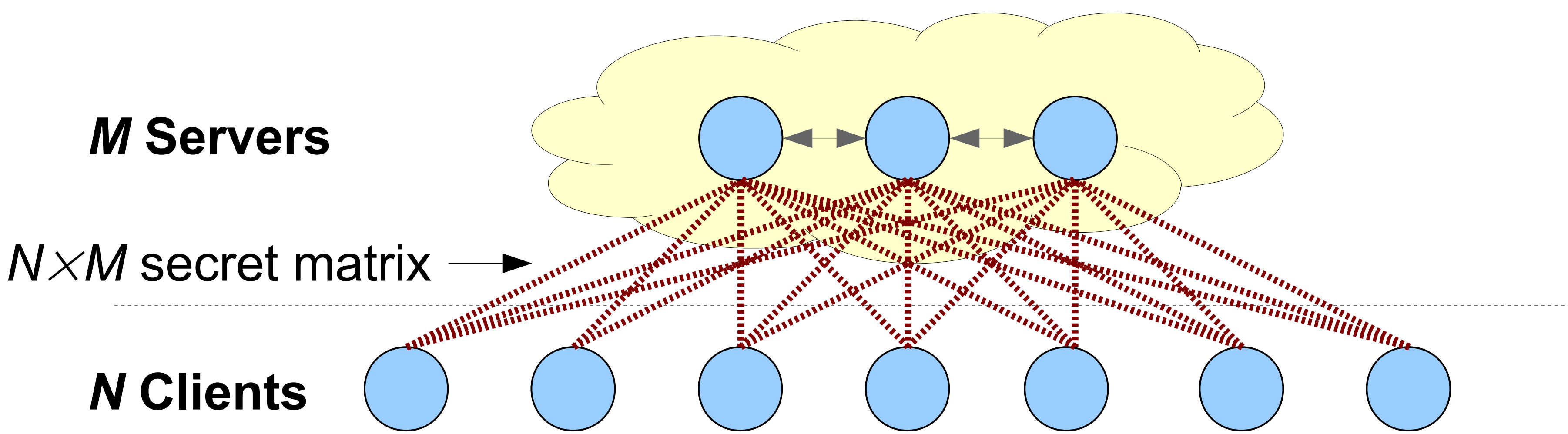
Use MIX-based shuffle above to "bootstrap"

1. Each client creates fresh **pseudonym keypair**
2. Servers MIX-shuffle public nym-keys
   - Yields an agreed-upon permutation of nym-keys
   - Each client knows all keys but only *its own position*



Setup 2: Shared Secrets

- Each node publishes a Diffie-Hellman (DH) key
  - Owner of each key well-known, *not anonymous*
- Each client forms DH secret with each server, each server forms DH secret with each client
- Use each secret as seed for shared PRNG

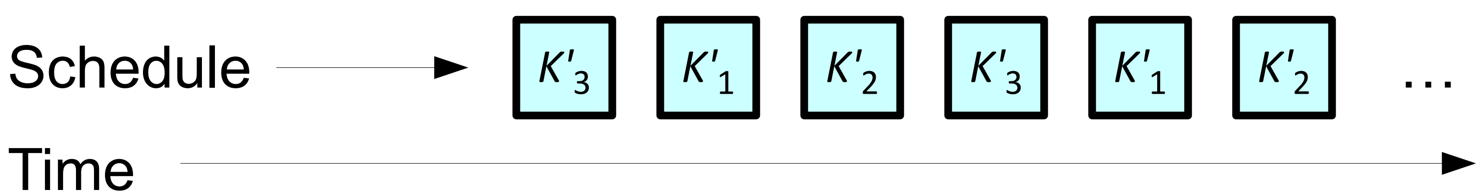


Transmission Rounds

Transmission proceeds in **rounds** scheduled by agreed-upon nym-key permutation

1. All clients send bits on behalf of slot 1's owner
2. All clients send bits on behalf of slot 2's owner
3. ….

Clients know which nym-key owns a given slot, but not which client owns which (other) slots.



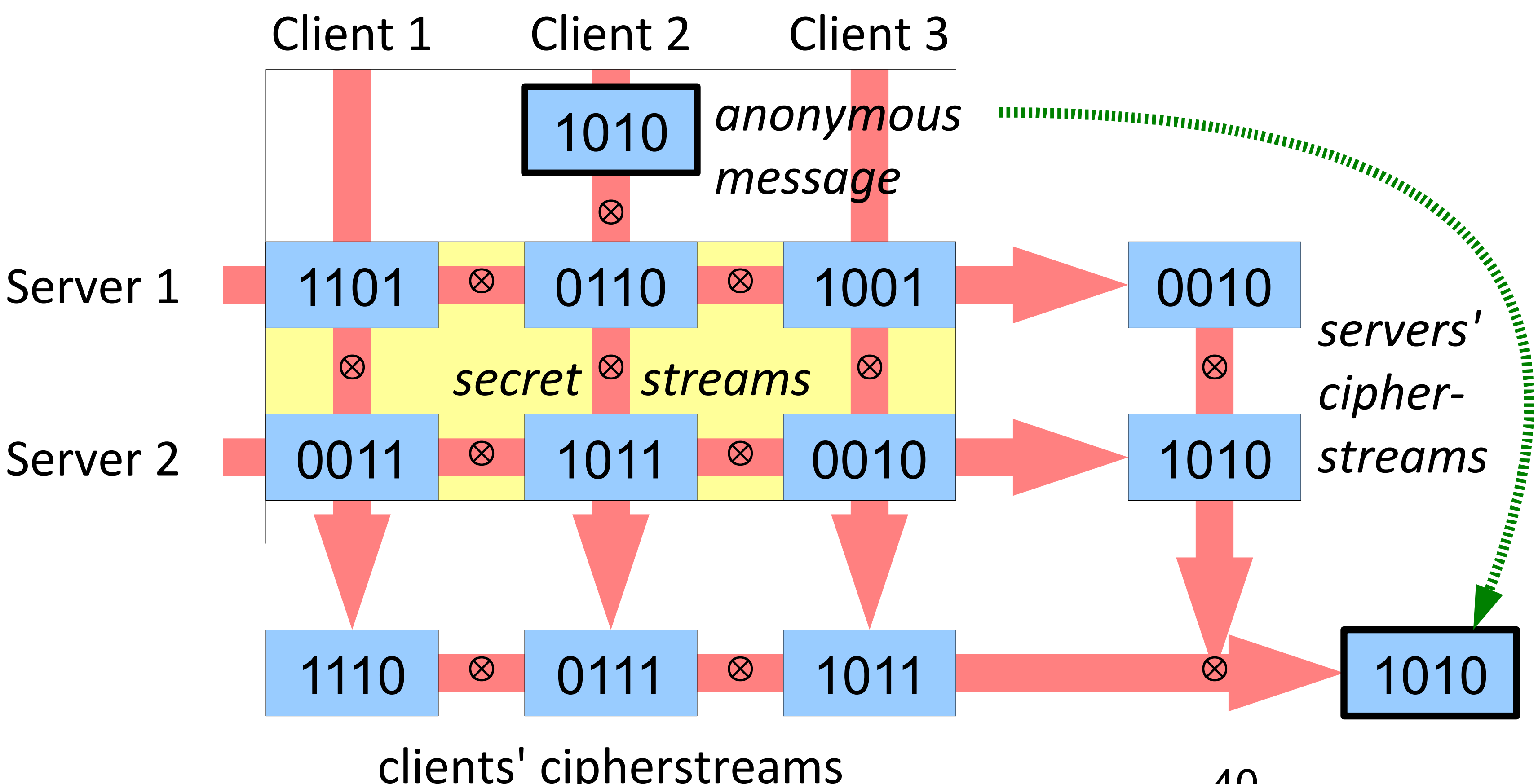
Computing Cipherstreams

In an *L*-bit transmission round:

- Each server *j* XORs together next *L* bits of PRNG streams shared with each client
- Each client *i* XORs together next L bits of PRNG streams shared with each server
  - If client *i* holds nym-key for current round, further XORs in his L-bit message
- Servers collect, XOR all cipherstreams
  - Shared PRNG streams cancel, leaving message

The Cipherstream Matrix

Ex: N=3 clients, M = 2 servers, L = 4 bits, client 2 owns nym-key for current round



Anonymity Properties

Assuming at least 1 server is honest:

- Each honest client shares secret PRNG stream with that server
  - These shared streams unknown to adversary
  - Different combination XORed in each cipherstream
  - Adversary can't distinguish any honest client's cipherstream (individually) from random bits
- Guarantee depends on strength of DH, PRNG

Accountability Properties

Owner of transmission slot must sign message with private nym-key for that slot

- Nodes can verify signature, corruption obvious

On corruption, nontrivial multi-round *blame protocol* required to identify source of corruption

- Guaranteed to succeed within “a few rounds”

Ongoing work: DC-nets protocol with proactive zero-knowledge proofs of correctness

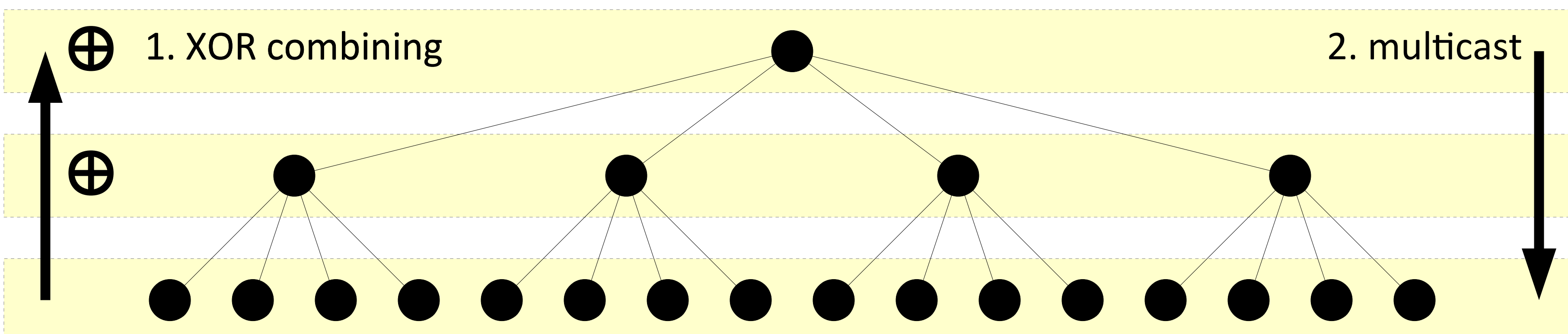
Scalability Properties

All computation, transmission fully parallelizable

P2P multicast-trees can optimize network usage:

1. XOR all cipherstreams on way up tree
2. Root multicasts plaintext result back down

Low latency/load, bandwidth 2× “plain” multicast



Robustness Properties

We assume servers are “reliable”

- Must ensure either via selection or design
- Server churn requires re-shuffle, restart

Clients may join/leave frequently, however

- Servers must recompute their cipherstreams
- Clients don't need to, as their cipherstreams depend only on secrets shared with servers

Talk Outline

- ✓ Online Anonymity: What and Why?
- ✓ An Accountable Group Anonymity Model
- ✓ A MIX-based Accountable Shuffle
- ✓ Dining On Schedule: Accountable DC-nets
- **Current Results and Ongoing Work**

Current Status

- Two working but less-scalable prototypes built, tested already (one Python, one C++)
  - See CCS '10 paper for performance/evaluation
- Third, more scalable prototype mostly done, implementing the techniques discussed here
  - Evaluation not yet done → no concrete results yet

Further Ongoing Work

- Proactive handling of network churn
  - e.g., ECC-encode messages across multiple redundant combining/distribution trees
- Handling intersection attacks
  - Risk if user retains a pseudonym “a long time”
  - Defense: hide *which members are online*
  - Exploring use of ring signatures, anonymous deniable authentication schemes...
- Deployment in “real” group communication apps!

Conclusion

The Dissent project is exploring anonymity in a **group communication** context

- Stronger security compared to onion routing
  - Anonymity: resistant to traffic analysis
  - Accountability: resistant to sybil attacks, disruption
- Early prototypes working, but many challenges remain before realistic deployment!

http://dedis.cs.yale.edu/2010/anon/