Dissent: Accountable Anonymous Group Messaging

Henry Corrigan-Gibbs and Bryan Ford

Department of Computer Science
Yale University

17th ACM Conference on
Computer and Communications Security
October 6, 2010

“Wikileaks” Problem

Alice

Bob

Eve

Chris

Image courtesy NASA Johnson Space Center


Alice
Bob
Eve
Chris


Alice
Bob
Has a 646 MB classified military video from Iraq
Eve
Chris


Alice
Bob
Is this video authentic?
Eve
Chris


Alice
Bob
Wants to:
1. Anonymously publish video to the group
2. Solicit anonymous comments
Eve
Chris

Alice

Bob

Eve

Chris

Alice

ob

Wants to:

1. Break anonymity
2. Stop initial video publication
3. Alter Alice and Bob's reviews
4. Submit *many* bad reviews

Eve

Chris

The Question

How can group members communicate efficiently and anonymously when:

1. all network communication is public,
2. Eve wants to block communication, and
3. group members' messages are of vastly different lengths?

Limitations of Existing Schemes

Method	Weakness
Mix Nets, Tor	Traffic analysis attacks
Group and Ring Signatures	Traffic analysis attacks
Voting Protocols	Short, fixed-length messages
DC Nets	Anonymous DoS attacks
Brickell-Shmatikov Shuffle	Anonymous DoS attacks and fixed message length

Outline

- Introduction to Dissent
- How Dissent works
  - Overall protocol: Variable-length shuffle
  - Key component: Fixed-length shuffle
- Prototype and experimental results
- Goals for future work

Outline

- **Introduction to Dissent**
- How Dissent works
  - Overall protocol: Variable-length shuffle
  - Key component: Fixed-length shuffle
- Prototype and experimental results
- Goals for future work

Dissent

- Dissent is a protocol for **latency-tolerant sender-anonymous broadcast** within a pre-defined group of nodes:
  1. Each group member secretly submits one message per protocol round
  2. Group members run the protocol
  3. The protocol reveals to all members a permutation of the message set
  4. No group member knows the permutation
- We call this a *shuffle* protocol

Dissent guarantees…

- **Integrity:** Messages are received unmodified
- **Anonymity:** To identify the sender of a message, all other members must collude
- **Accountability:** Members interfering with message transmission will eventually be identified

*N.B.: These definitions are **very** informal. Please refer to our paper for precise definitions.*

Our Contributions

Dissent builds upon the Brickell-Shmatikov anonymous data collection protocol (KDD 2006), adding:

1. **Accountability**: Alice, Bob, and Chris can identify Eve if she tries to alter messages or block protocol progress
2. **Communication efficiency with variable-length messages**: Group members do not have to pad their messages to a fixed length

Outline

- Introduction to Dissent
- **How Dissent works**
  - **Overall protocol: Variable-length shuffle**
  - Key component: Fixed-length shuffle
- Prototype and experimental results
- Goals for future work

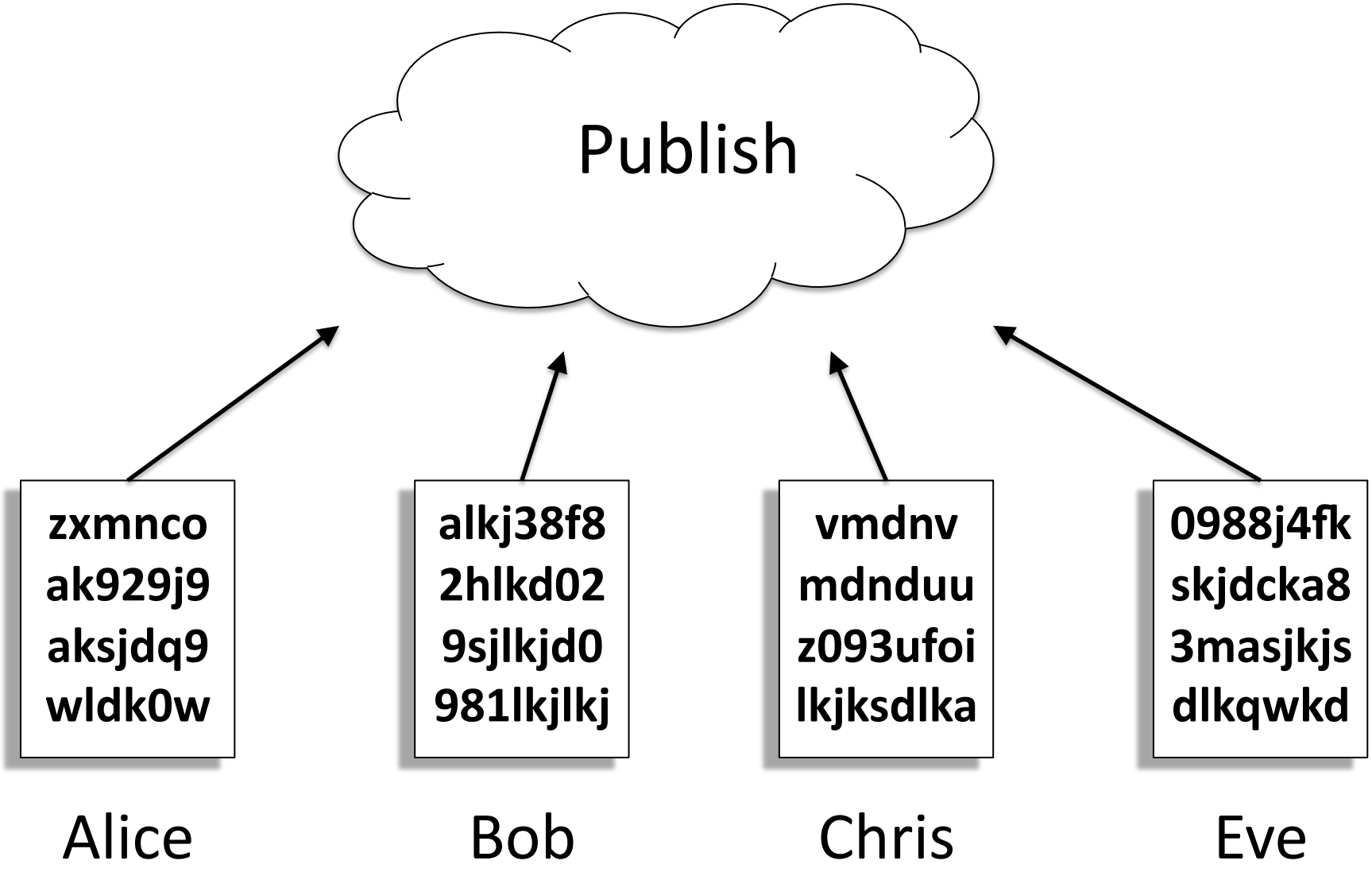
Conceptual Description

- Follows one group member (Chris) and his 646 MB video
  - **Every other** group member follows the same steps as Chris in parallel
- We assume:
  - Every member has a signature verification key for every other group member
  - Existence of a *wrapper protocol* handling group membership, liveness, protocol initiation, etc. (*See paper for details on the wrapper.*)

Issue 1: Traffic Analysis

- How can Chris broadcast his 646 MB video anonymously if attackers are listening in?
- Neither message structure nor length should identify Chris as the true sender
- Therefore:
  - All other group members must also broadcast a 646 MB message
  - Messages should look like random strings

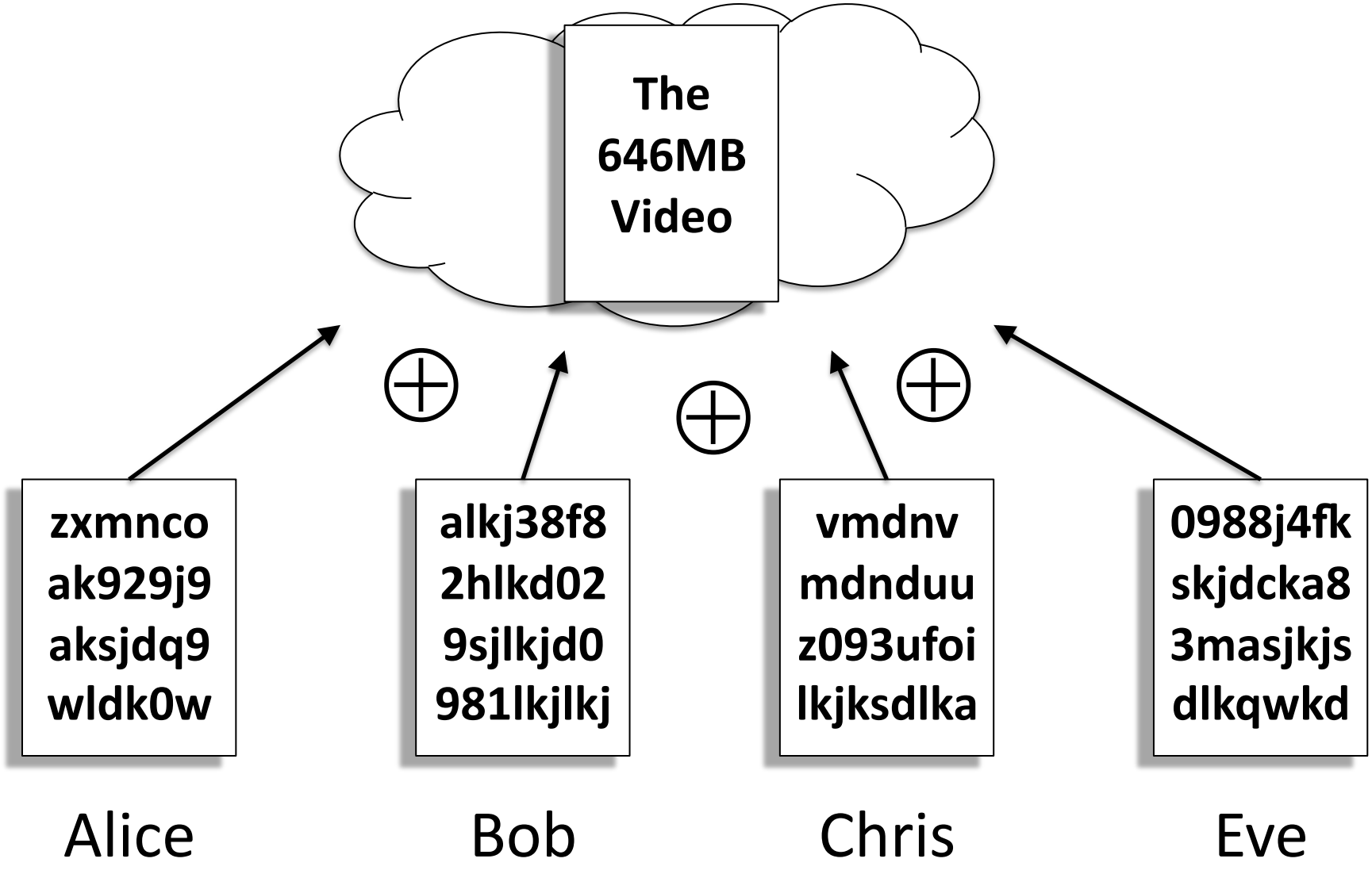
Issue 1: Traffic Analysis



Issue 2: Recovery

- How can members recover Chris' video from the 646 MB random-looking strings?
- Assume Alice, Bob, and Eve send pseudo-random strings known to Chris
- Then:
  - Chris sends the **XOR** of his video with Alice, Bob, and Eve's pseudo-random strings
  - i.e., three serial one-time pad encryptions
- Reminiscent of Chaum's DC net

Issue 2: Recovery



Issue 3: Assignment

- How can Chris efficiently assign 646 MB strings to Alice, Bob, and Eve?
- Chris anonymously broadcasts (somehow) **a table of assignments** along with the length of his message
- Each assignment contains:
  - A PRF seed encrypted for each member
  - A cleartext hash of the string assigned to each member

Issue 3: Assignment

Length: 0 bytes		
	Seed	*Hash*
A	$\{dfwv\}_A$	td82
B	$\{ert3\}_B$	3flk
C	$\{09fg\}_C$	df3f
E	$\{fg4h\}_E$	vce3

Alice

Length: 0 bytes		
	Seed	*Hash*
A	$\{v9ek\}_A$	2d2t
B	$\{2fva\}_B$	nve0
C	$\{afbf\}_C$	3ren
E	$\{d2g5\}_E$	sdvz

Bob

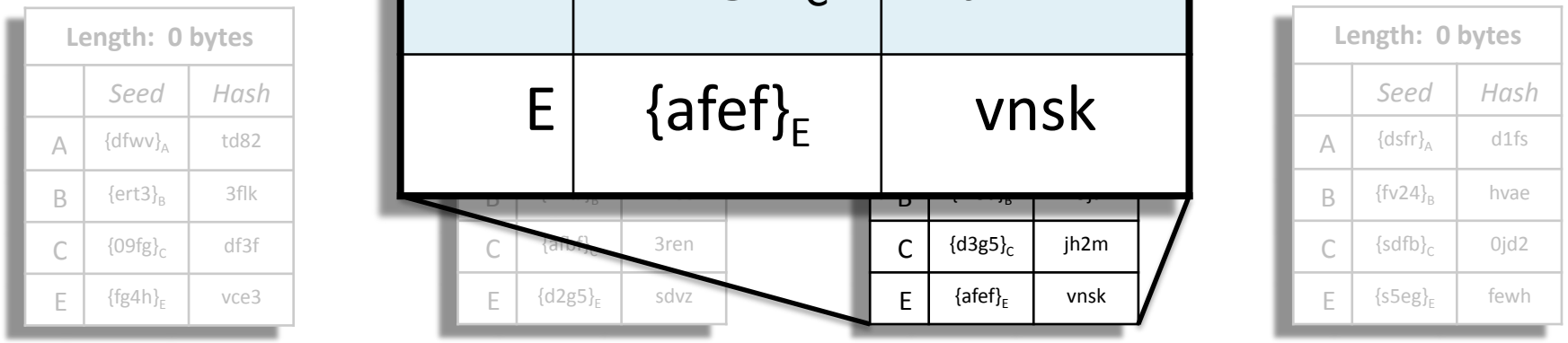
Length: 646 MB		
	Seed	*Hash*
A	$\{dkad\}_A$	092f
B	$\{f23d\}_B$	f9ja
C	$\{d3g5\}_C$	jh2m
E	$\{afef\}_E$	vnsk

Chris

Length: 0 bytes		
	Seed	*Hash*
A	$\{dsfr\}_A$	d1fs
B	$\{fv24\}_B$	hvae
C	$\{sdfb\}_C$	0jd2
E	$\{s5eg\}_E$	fewh

Eve

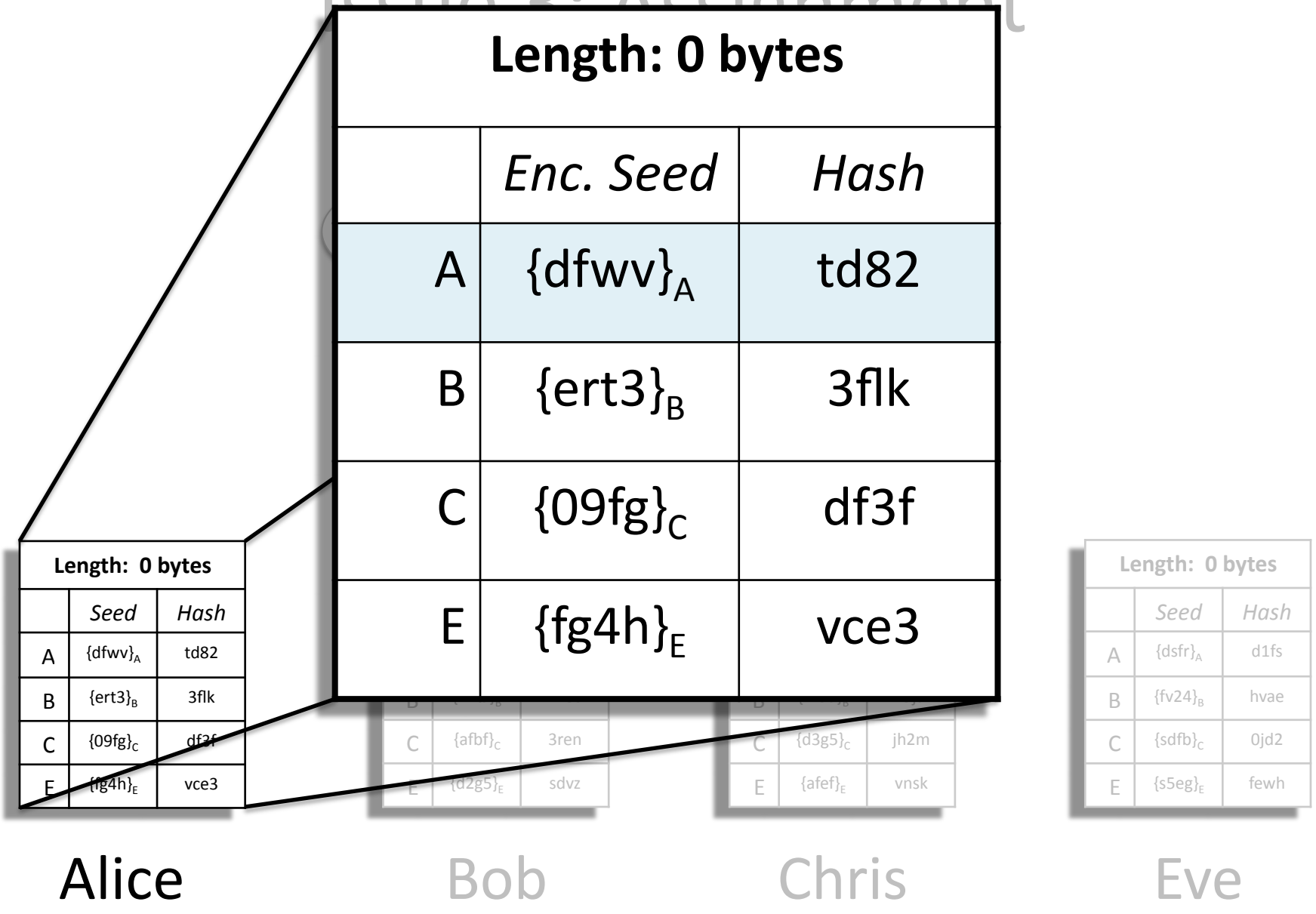
Issue 3: Assignment



Length: 646 MB

	Enc. Seed	*Hash*
A	$\{dkad\}_A$	092f
B	$\{f23d\}_B$	f9ja
C	$\{d3g5\}_C$	jh2m
E	$\{afef\}_E$	vnsk

Alice Bob Chris Eve


Issue 3: Assignment
Length: 0 bytes
Enc. Seed
Hash
A
{dfwv}A
td82
B
{ert3}B
3flk
C
{09fg}C
df3f
E
{fg4h}E
vce3
Length: 0 bytes
Seed
Hash
A
{dfwv}A
td82
B
{ert3}B
3flk
C
{09fg}C
df3f
E
{fg4h}E
vce3
C
{afbf}C
3ren
E
{d2g5}E
sdvz
C
{d3g5}C
jh2m
E
{afef}E
vnsk
Length: 0 bytes
Seed
Hash
A
{dsfr}A
d1fs
B
{fv24}B
hvae
C
{sdfb}C
0jd2
E
{s5eg}E
fewh
Alice
Bob
Chris
Eve

Issue 3: Assignment
Length: 0 bytes
Enc. Seed
Hash
A
{v9ek}A
2d2t
B
{2fva}B
nve0
C
{afbf}C
3ren
E
{d2g5}E
sdvz
Length: 0 bytes
Seed
Hash
A
{dfwv}A
td82
B
{ert3}B
3flk
C
{09fg}C
df3f
E
{fg4h}E
vce3
C
{afbf}C
3ren
E
{d2g5}E
sdvz
C
jh2m
E
{afef}E
vnsk
Length: 0 bytes
Seed
Hash
A
{dsfr}A
d1fs
B
{fv24}B
hvae
C
{sdfb}C
0jd2
E
{s5eg}E
fewh
Alice
Bob
Chris
Eve

Issue 3: Assignment
Length: 0 bytes
Enc. Seed
Hash
A
{dsfr}A
d1fs
B
{fv24}B
hvae
C
{sdfb}C
0jd2
E
{s5eg}E
fewh
Length: 0 bytes
Seed
Hash
A
{dfwv}A
td82
B
{ert3}B
3flk
C
{09fg}C
df3f
E
{fg4h}E
vce3
C
{afbf}C
3reh
E
{d2g5}E
sdvz
C
{d3g5}C
jh2m
E
{afef}E
Length: 0 bytes
Seed
Hash
A
{dsfr}A
d1fs
B
{fv24}B
hvae
C
{sdfb}C
0jd2
E
{s5eg}E
fewh
Alice
Bob
Chris
Eve

Issue 3: Assignment

Length: 0 bytes		
	Seed	*Hash*
A	$\{dfwv\}_A$	td82
B	$\{ert3\}_B$	3flk
C	$\{09fg\}_C$	df3f
E	$\{fg4h\}_E$	vce3

Alice

Length: 0 bytes		
	Seed	*Hash*
A	$\{v9ek\}_A$	2d2t
B	$\{2fva\}_B$	nve0
C	$\{afbf\}_C$	3ren
E	$\{d2g5\}_E$	sdvz

Bob

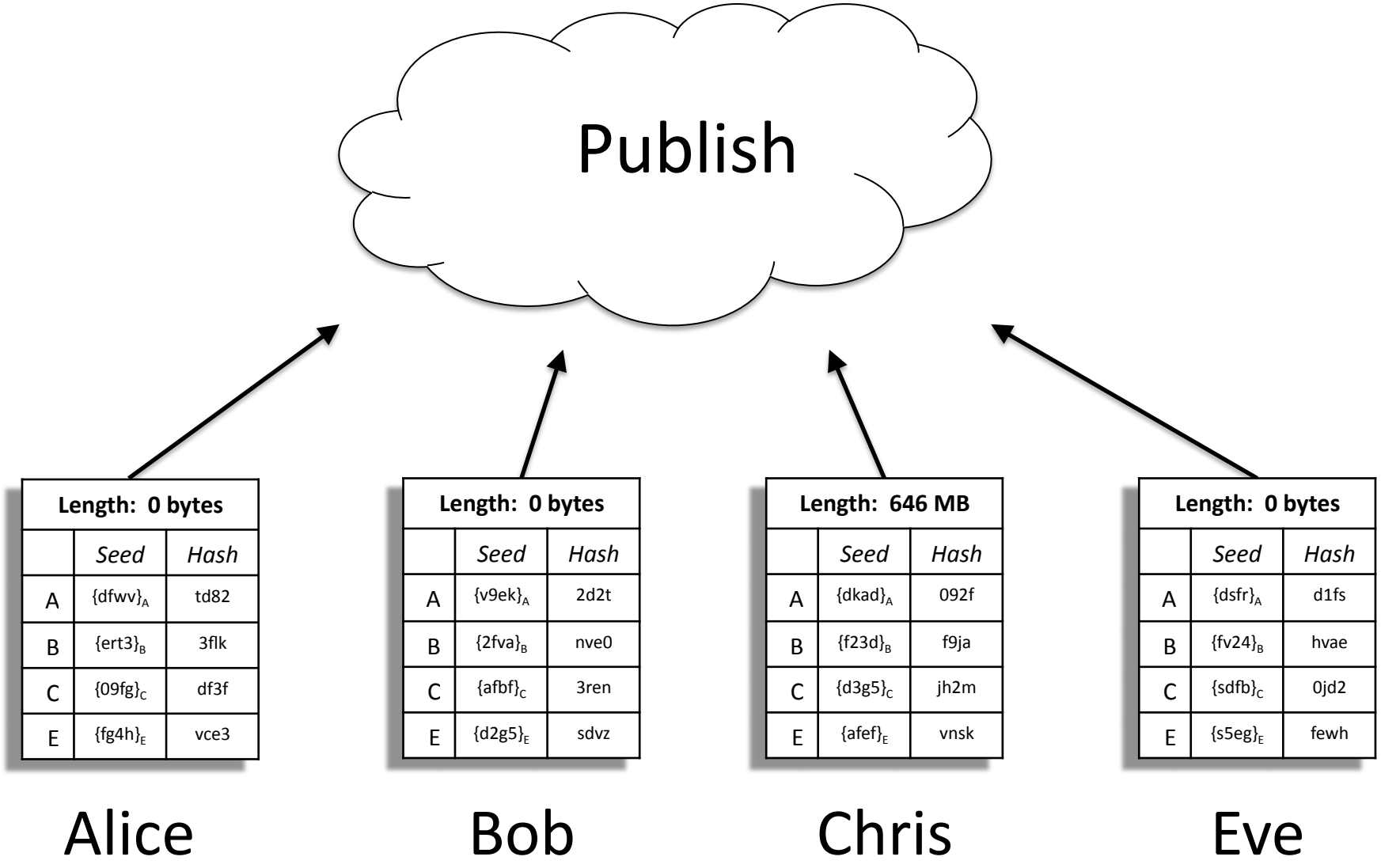
Length: 646 MB		
	Seed	*Hash*
A	$\{dkad\}_A$	092f
B	$\{f23d\}_B$	f9ja
C	$\{d3g5\}_C$	jh2m
E	$\{afef\}_E$	vnsk

Chris

Length: 0 bytes		
	Seed	*Hash*
A	$\{dsfr\}_A$	d1fs
B	$\{fv24\}_B$	hvae
C	$\{sdfb\}_C$	0jd2
E	$\{s5eg\}_E$	fewh

Eve

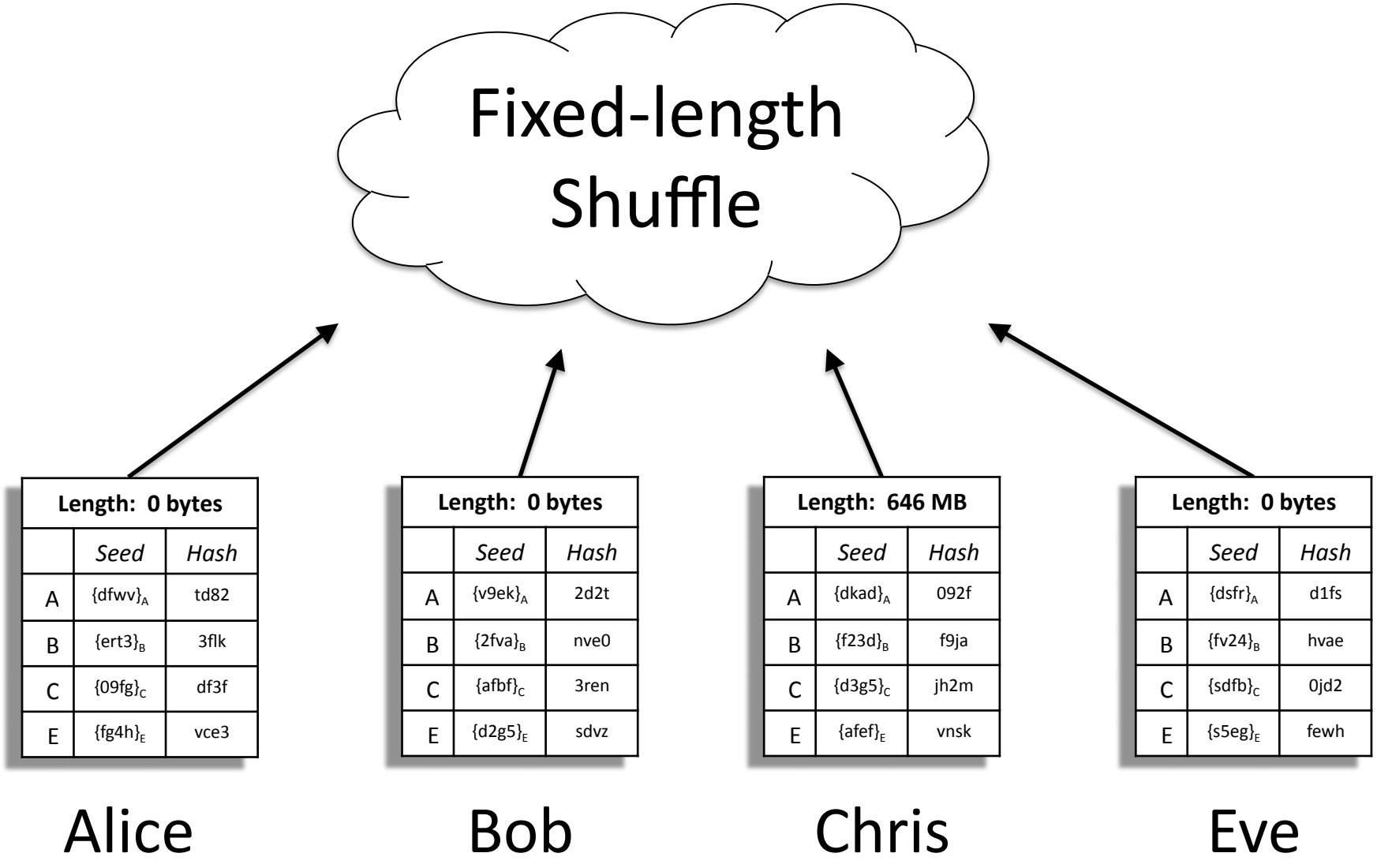
Issue 3: Assignment



Issue 4: Anonymity

- How can Chris transmit his assignment table anonymously?
- The assignment contains the length of Chris' message, so it must not be traceable to him
- Dissent uses a modification of the Brickell-Shamikov data collection protocol to shuffle the fixed-length assignment tables
- Final ordering of tables determines ordering in which members broadcast their pseudo-random strings

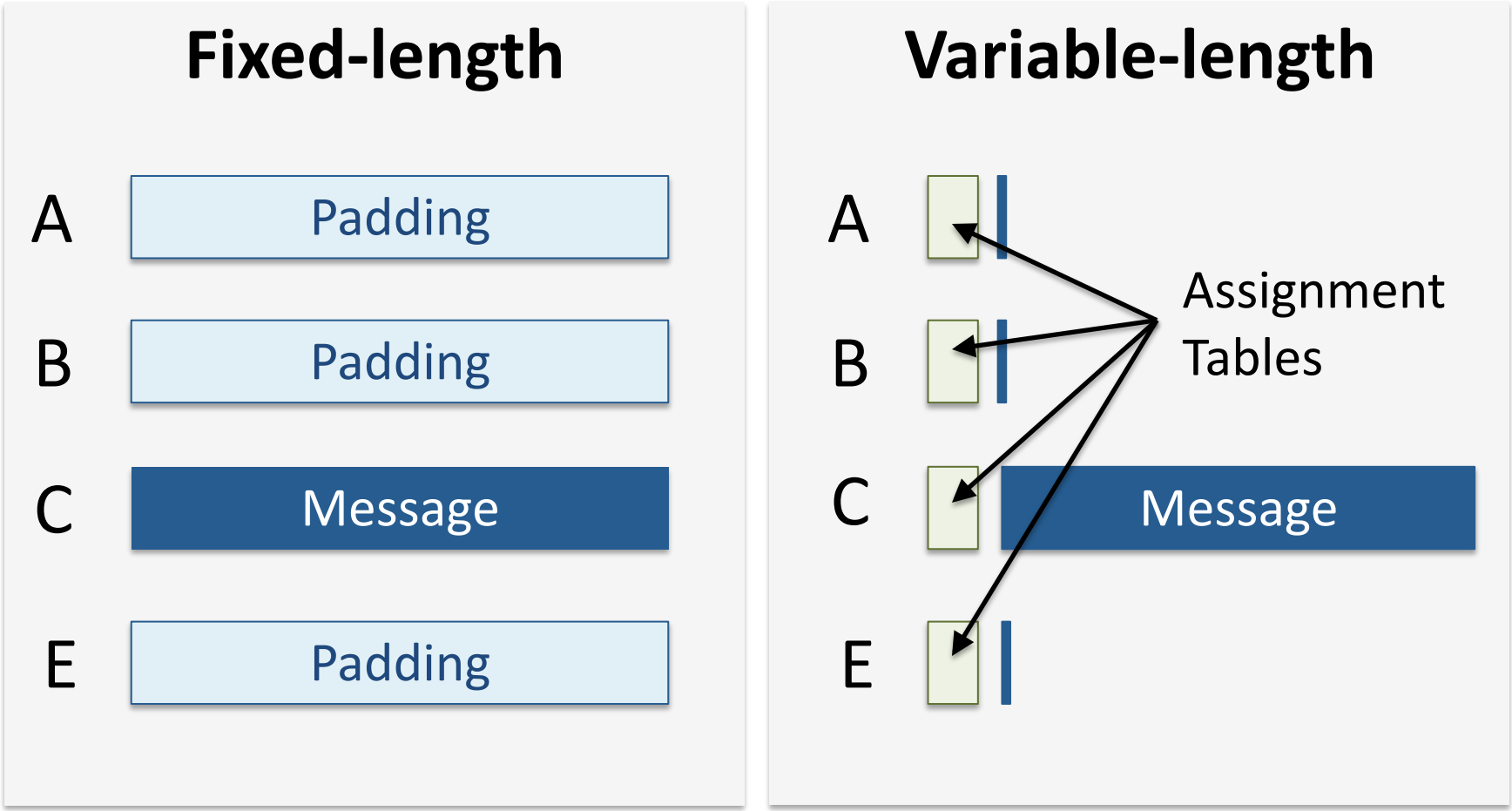
Issue 4: Anonymity



Putting it together…

1. Each group member generates a **fixed-length assignment table**
2. Members use **fixed-length shuffle** to anonymize these assignment tables
3. Members (except sender) use assigned PRF seeds to generate a **psuedo-random string** for each anonymous message
4. Strings corresponding to each message XOR to the sender's message

Variable-length is better



NL_{max} bits

$L_{total} + kN$ bits

Outline

- Introduction to Dissent
- **How Dissent works**
  - Overall protocol: Variable-length shuffle
  - **Key component: Fixed-length shuffle**
- Prototype and experimental results
- Goals for future work

Fixed-Length Shuffle

- A verifiable secret shuffle
- Adds **accountability** to Brickell-Shmatikov shuffle
- Two possible outcomes:
  1. Shuffle succeeds, messages delivered, secret permutation unrecoverable
  2. Shuffle fails, messages unrecoverable, at least one attacker exposed

Issue 1: Anonymity

- How do members anonymize their messages?
- We use onion routing / mix network to provide anonymity:
  1. Members serially encrypt their message with the public key of *each* group member in a pre-determined order
  2. Each group member sequentially decrypts, permutes, and forwards the message set

Message set under

onion encryption

$$\{\{\{\{\mathbf{A}\}_E\}_C\}_B\}_A$$

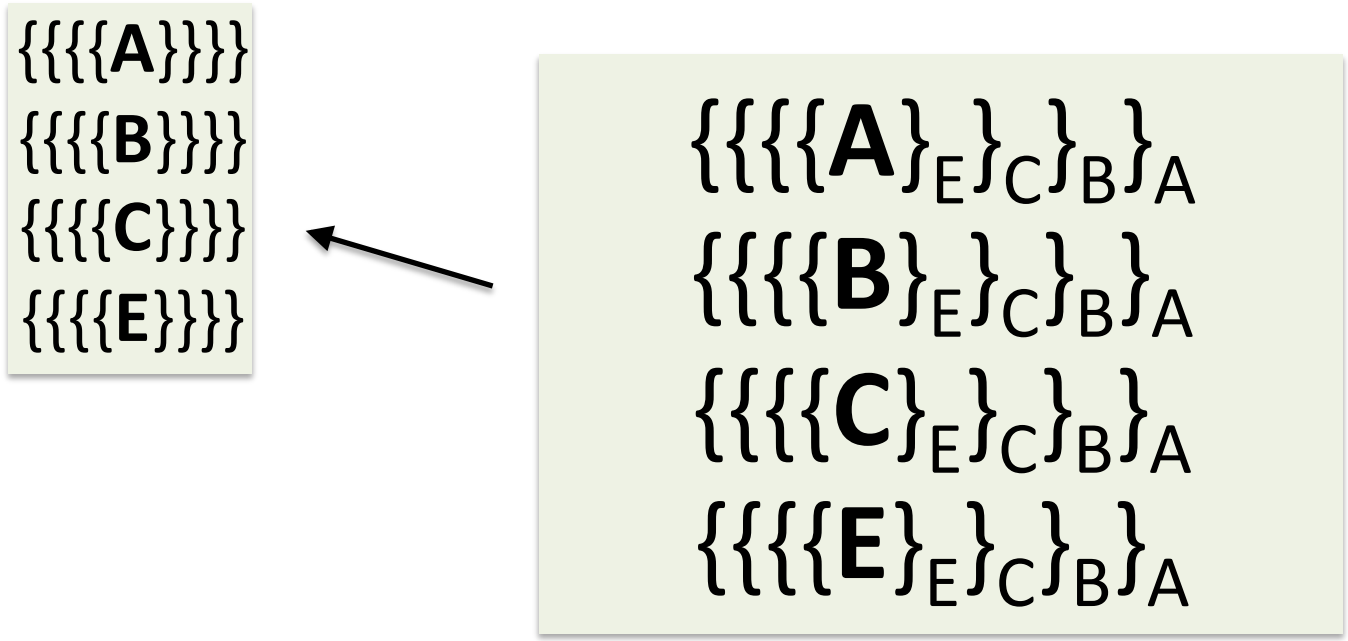
$$\{\{\{\{\mathbf{B}\}_E\}_C\}_B\}_A$$

$$\{\{\{\{\mathbf{C}\}_E\}_C\}_B\}_A$$

$$\{\{\{\{\mathbf{E}\}_E\}_C\}_B\}_A$$

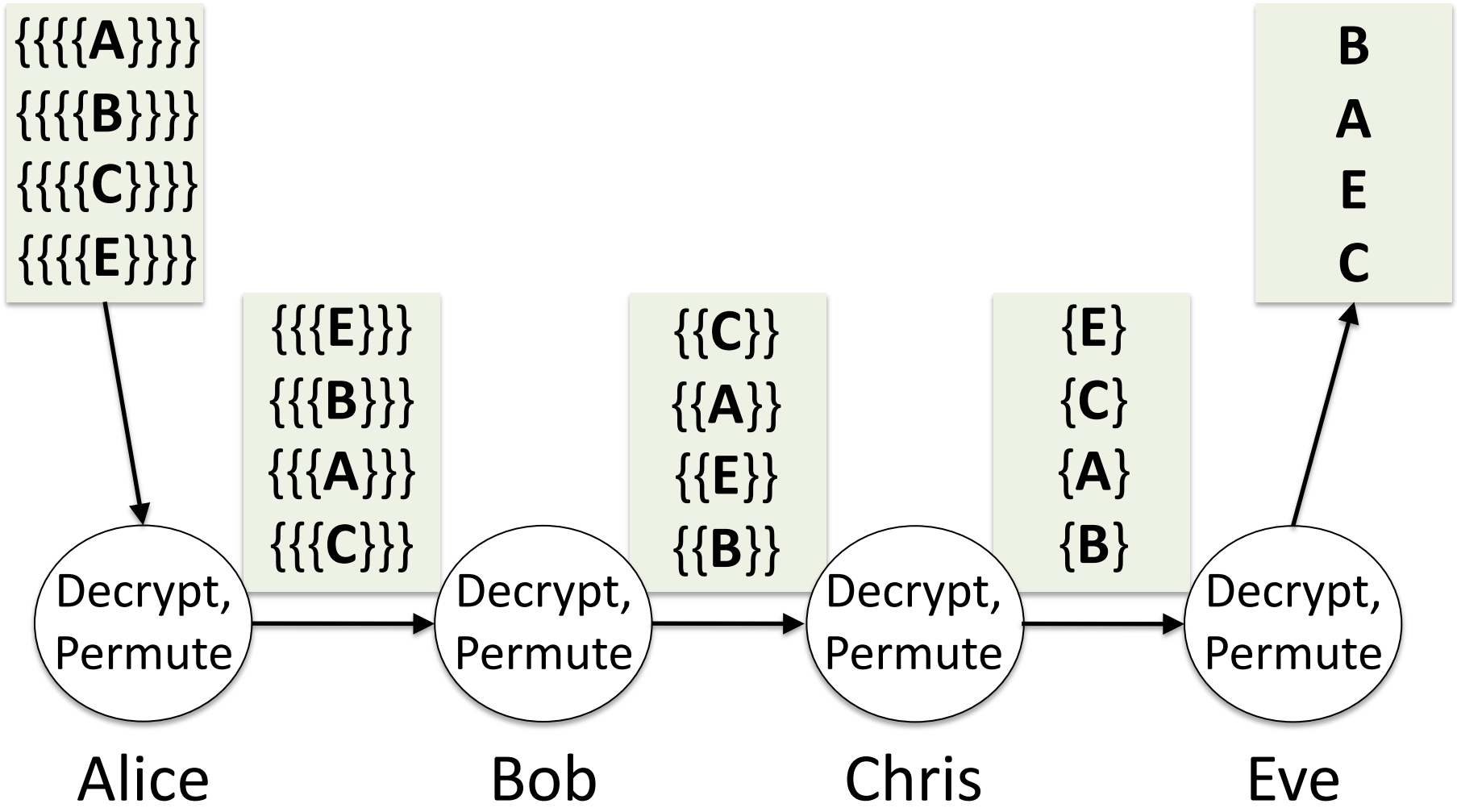
Message set under

onion encryption



Message set under **onion encryption**

Permuted **plaintext** message set



Issue 2: Integrity

- How do members ensure that their message is in the output message set?
- Group members submit a “Go” or “No-go” message after seeing the permuted message set to confirm that their message is in the set
- Members use a **second layer of onion encryption** so that Go/No-Go messages don’t break anonymity
  - One-time-use secondary keypair

Message set under
primary and **secondary**
onion encryption

$$\{\{\{\ \alpha = [[[[\mathbf{A}]_E]_C]_B]_A\ \}_E\}_C\}_B\}_A$$
$$\{\{\{\ \beta = [[[[\mathbf{B}]_E]_C]_B]_A\ \}_E\}_C\}_B\}_A$$
$$\{\{\{\ \gamma = [[[[\mathbf{C}]_E]_C]_B]_A\ \}_E\}_C\}_B\}_A$$
$$\{\{\{\ \varepsilon = [[[[\mathbf{E}]_E]_C]_B]_A\ \}_E\}_C\}_B\}_A$$

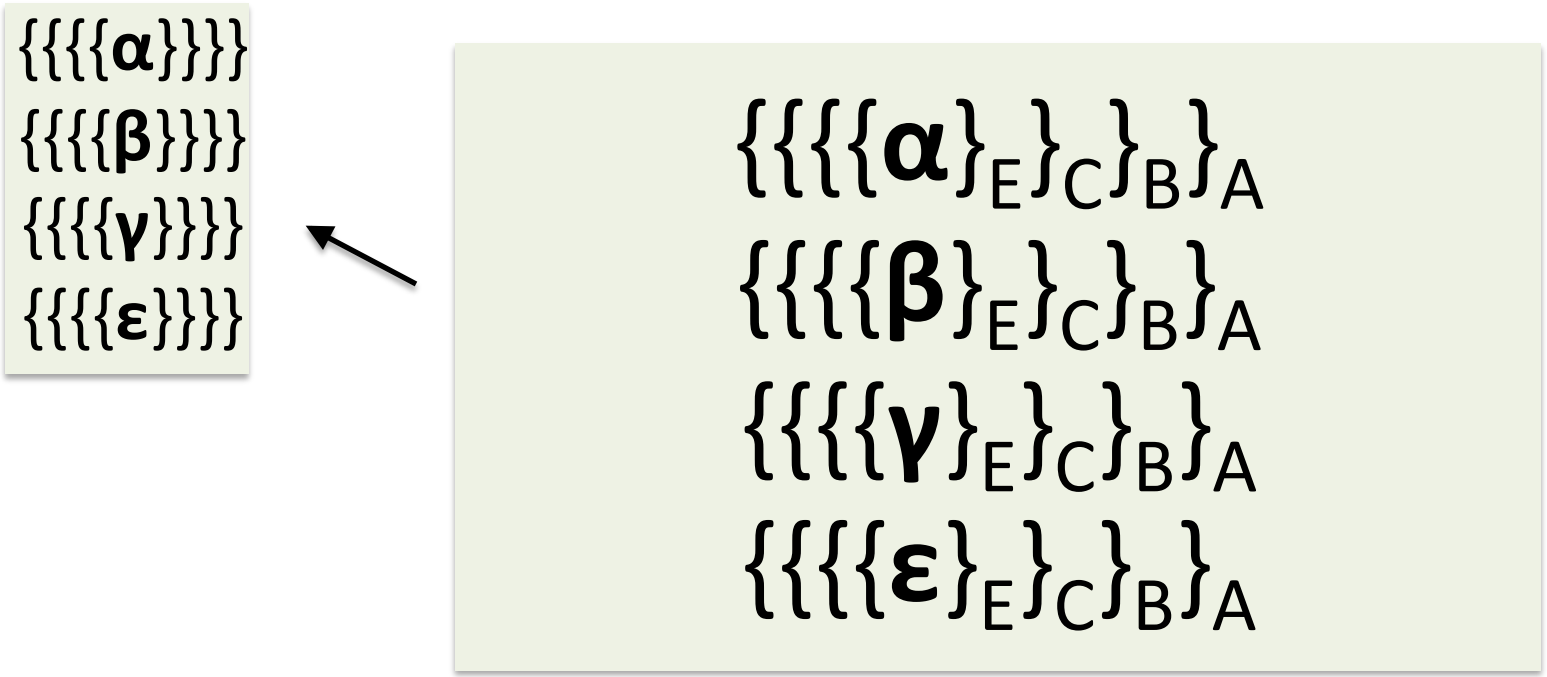
Message set under
primary and **secondary**
onion encryption

$$\{\{\{\{\boldsymbol{\alpha}\}_E\}_C\}_B\}_A$$
$$\{\{\{\{\boldsymbol{\beta}\}_E\}_C\}_B\}_A$$
$$\{\{\{\{\boldsymbol{\gamma}\}_E\}_C\}_B\}_A$$
$$\{\{\{\{\boldsymbol{\varepsilon}\}_E\}_C\}_B\}_A$$

Message set under

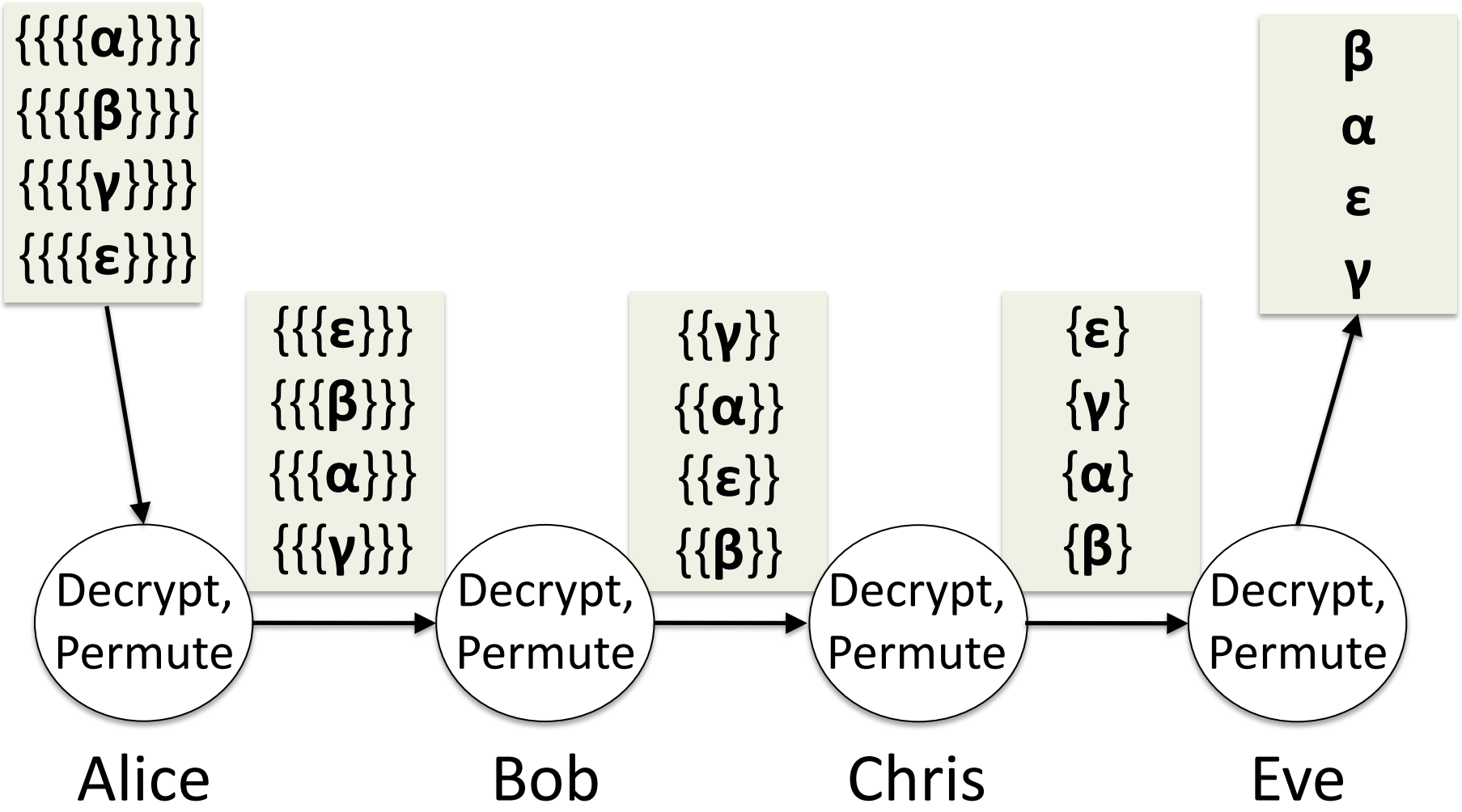
primary and **secondary**

onion encryption



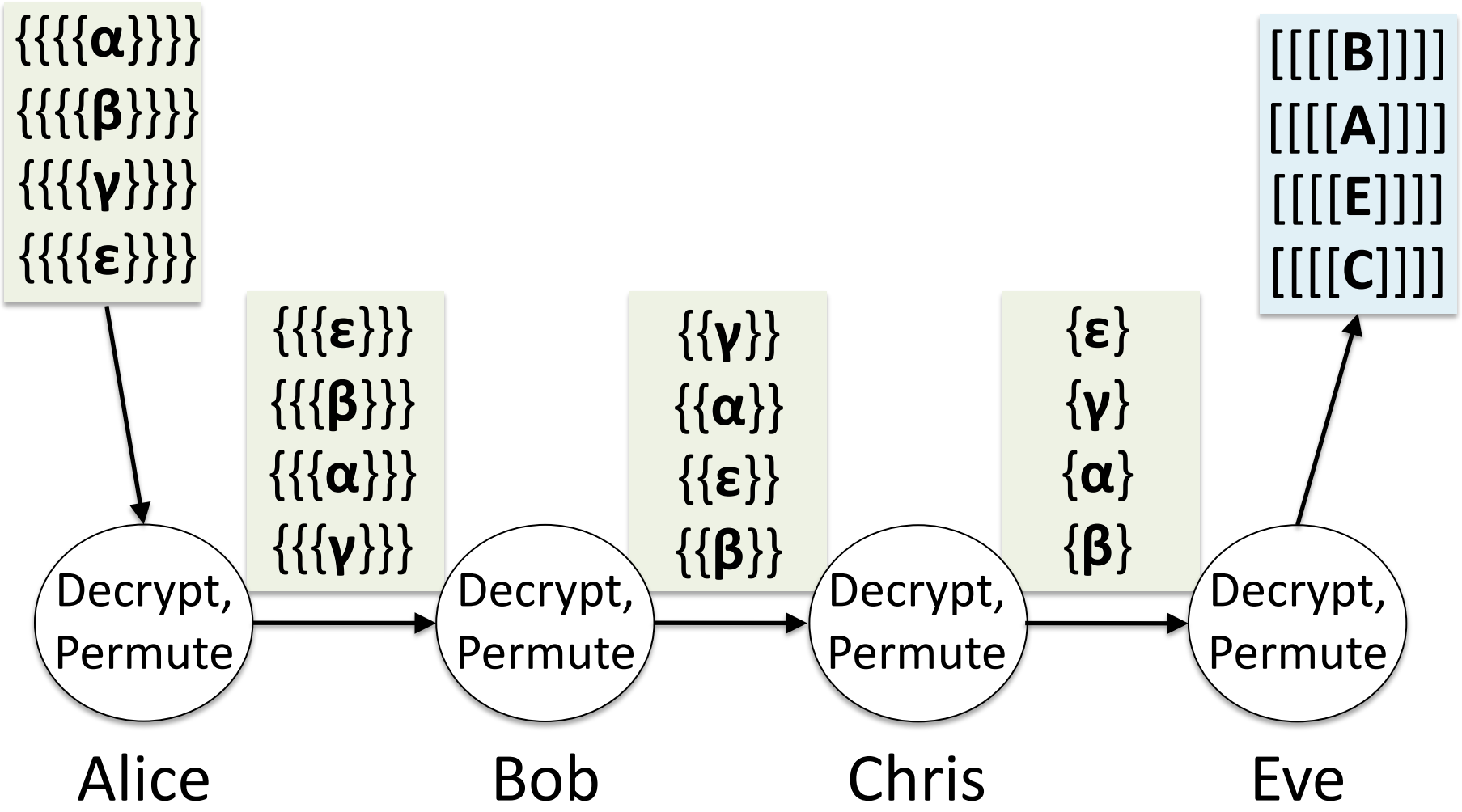
Message set under **primary** and **secondary onion encryption**

Permuted **ciphertext set** under secondary onion encryption

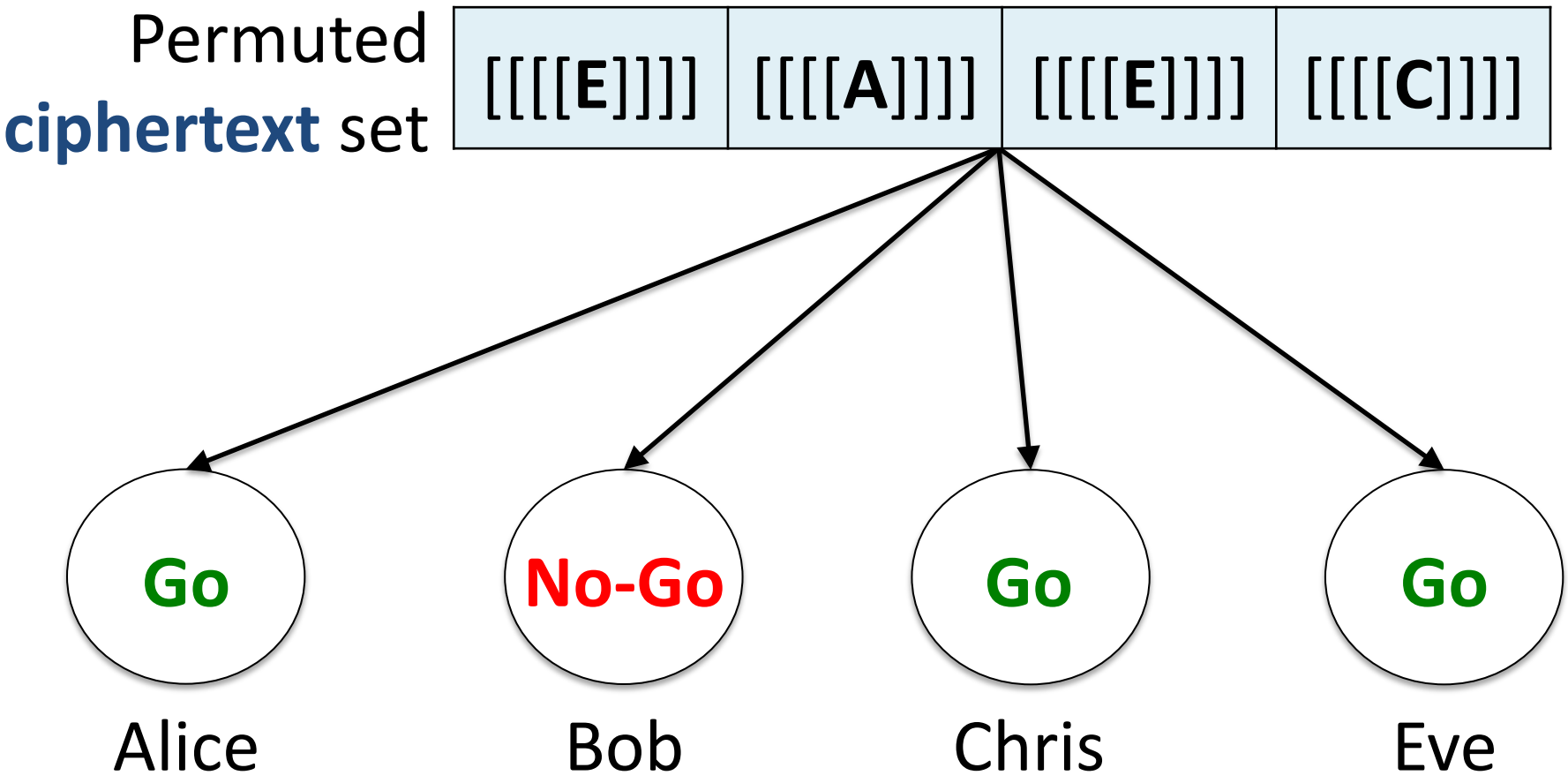


Message set under **primary** and **secondary onion encryption**

Permuted **ciphertext set** under secondary onion encryption



Issue 2: Integrity



Issue 2: Integrity

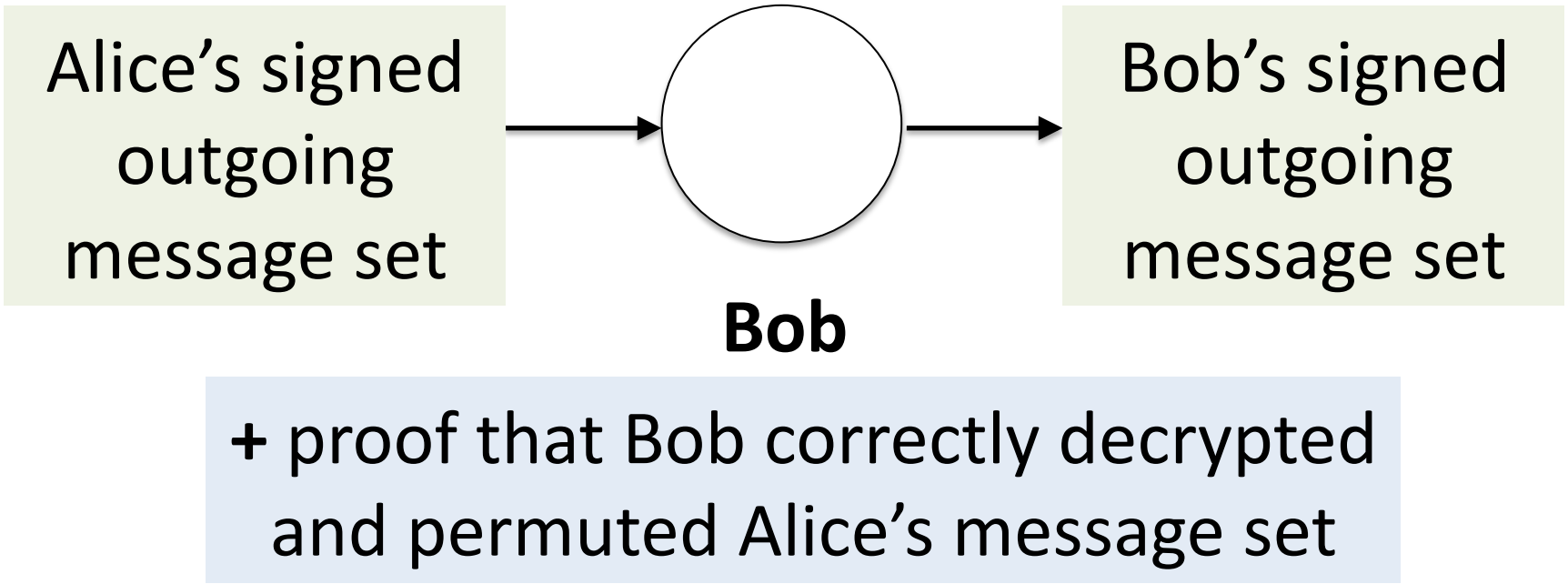
- If **all** group members report **Go**, then each member publishes her secondary private key
- All members can decrypt each ciphertext in the set, and the permutation is kept secret

Issue 3: Accountability

- How do group members expose an attacker?
- If **any** group member reports **No-Go**, then each member:
  1. Destroys her secondary private key, rendering the messages unrecoverable, and
  2. Publishes a proof of her correctness: signed intra-group transmissions and information necessary to replay anonymization process

Issue 3: Accountability

- Once group members destroy secondary private keys, they can safely **reveal the secret permutation**



Fixed-Length Shuffle Recap

1. Group members **encrypt** their message with two layers of onion encryption
2. Each group member **permutes** the set and **decrypts** a layer of primary key encryption
3. Members send a **Go** or **No-Go** message indicating whether or not to decrypt
4. Members publish either their secondary private keys **OR** proofs of their correctness

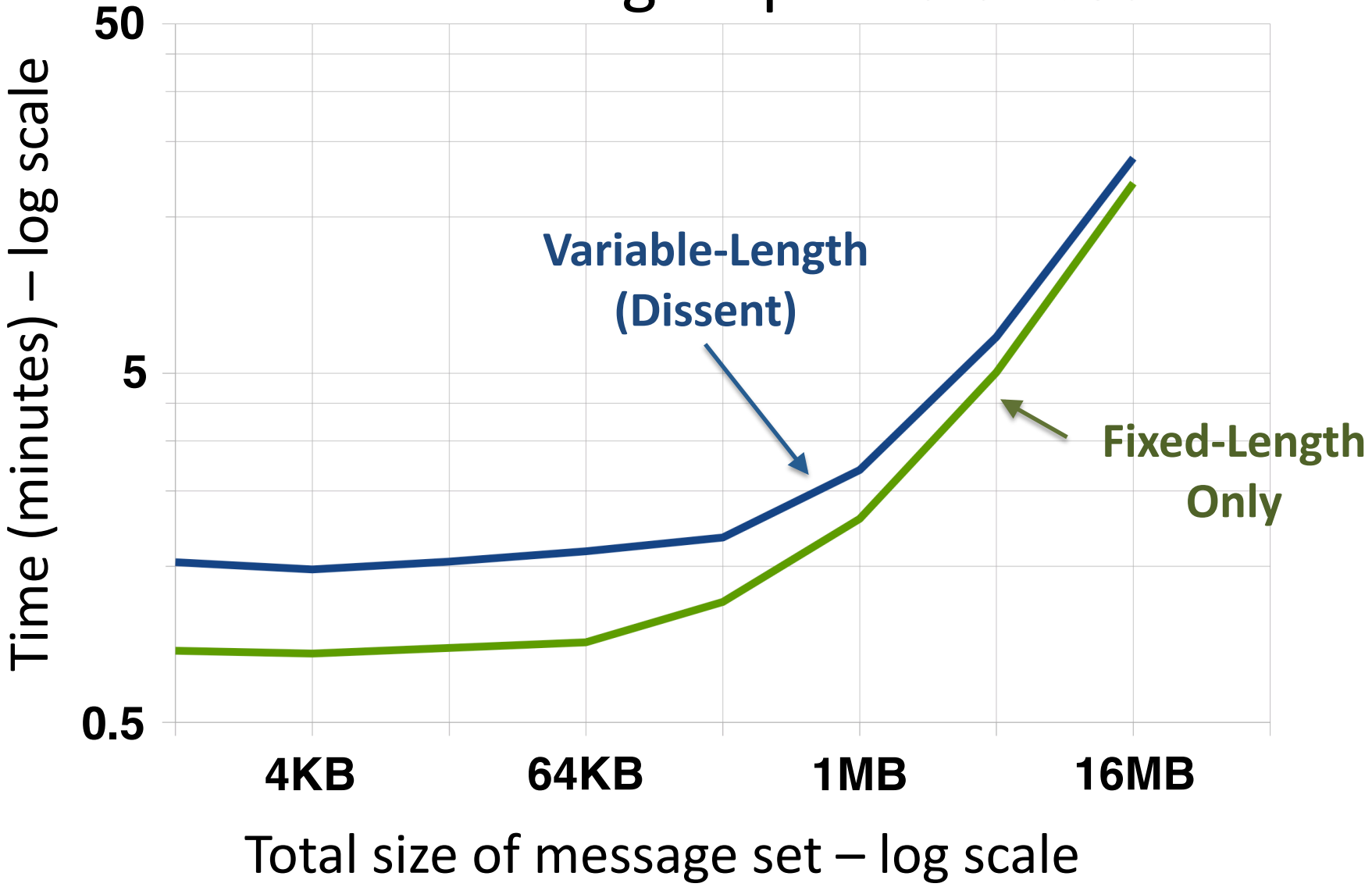
Outline

- Introduction to Dissent
- How Dissent works
  - Overall protocol: Variable-length shuffle
  - Key component: Fixed-length shuffle
- **Prototype and experimental results**
- Goals for future work

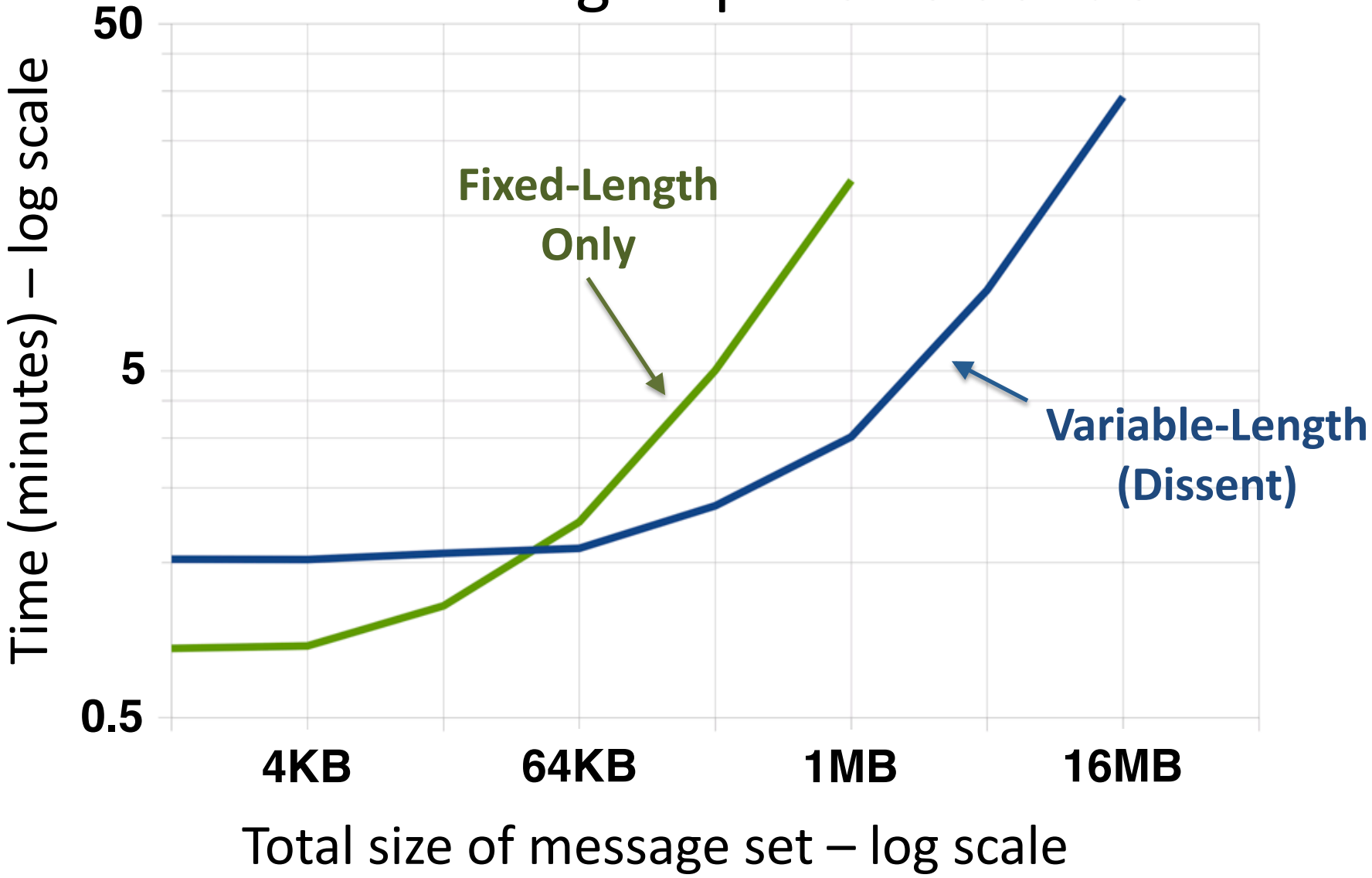
Prototype

- Implemented fixed-length and full Dissent (variable-length) protocols in Python
- Used Emulab, a network testbed, to run performance tests
  - 100ms node-to-node latency, 5Mbps link bandwidth
  - Up to 44 nodes
- Did not implement “blame” phases

Time required for transmission in a 16-node group – **Balanced**



Time required for transmission in a 16-node group – **One Sender**



Prototype

- Broadcasting a 16 MB file in a 16-node group took **3.6 times** longer using Dissent than broadcasting file without encryption or anonymization
- Refer to paper for full results
  - Round time over message set size
  - Round time broken down by component
  - Round time over group size

Outline

- Introduction to Dissent
- How Dissent works
  - Overall protocol: Variable-length shuffle
  - Key component: Fixed-length shuffle
- Prototype and experimental results
- **Goals for future work**

Future Work

- Use a more efficient fixed-length shuffle
  - Brickell-Shmatikov shuffle requires **serial interactivity**
- Reduce size complexity of assignment tables
- Reuse one fixed-length shuffle for many message transfers
- Implement protocol on large scale

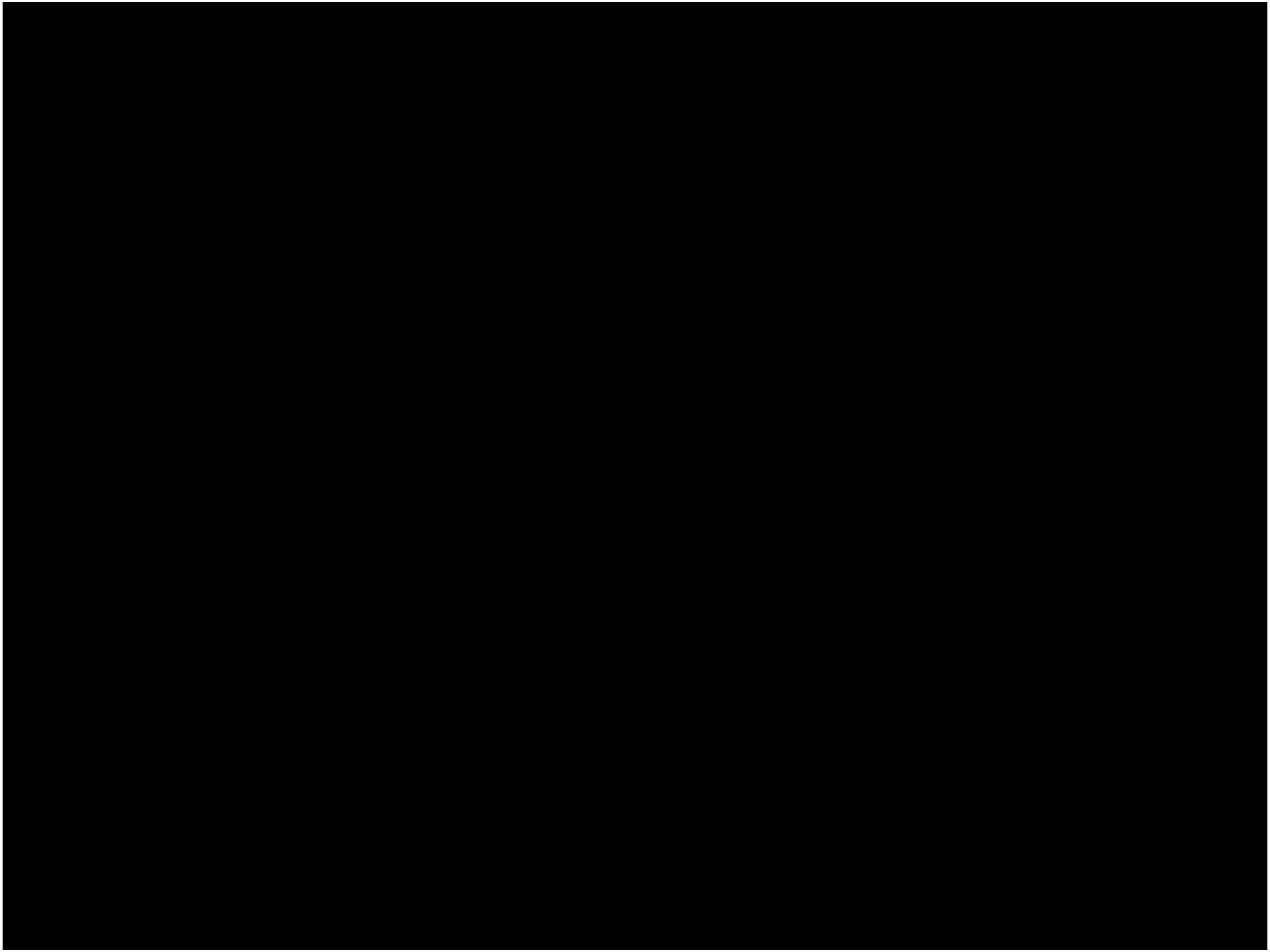
Wrap-Up

- Dissent is a sender-anonymous protocol for broadcast within a group
- Dissent contributes:
  - **Accountability**, the ability to identify group members who try to block message transmission
  - **Communication efficiency**, under variable message lengths
- Dissent has the potential to be a practical tool for anonymous *latency-tolerant* group messaging

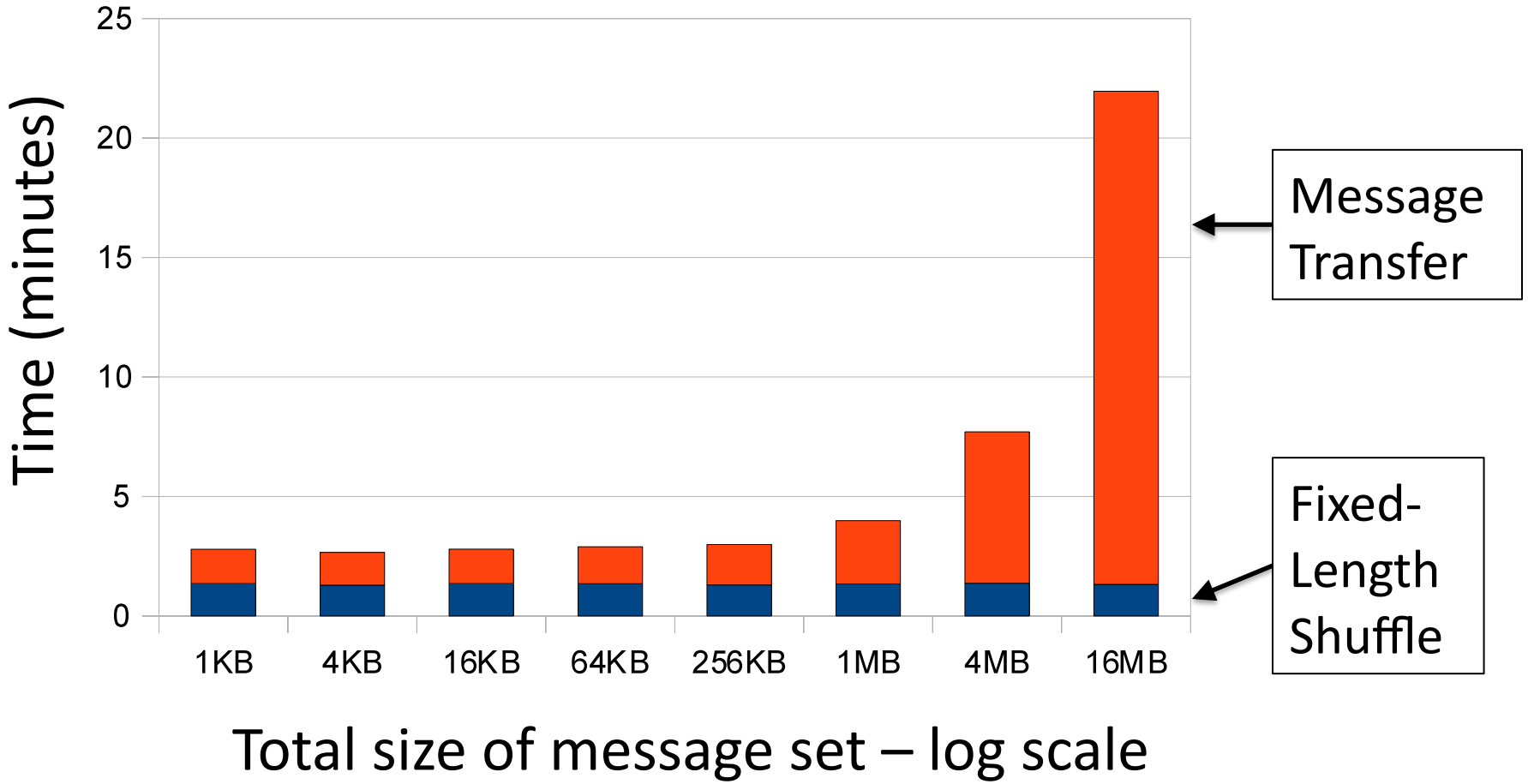
Acknowledgements

- The anonymous CCS reviewers
- Vitaly Shmatikov, Michael Fischer, Bimal Viswanath, Animesh Nandi, Justin Brickell, Jacob Strauss, Chris Lesniewski-Laas, Pedro Fonseca, Philip Levis
- All of you for listening

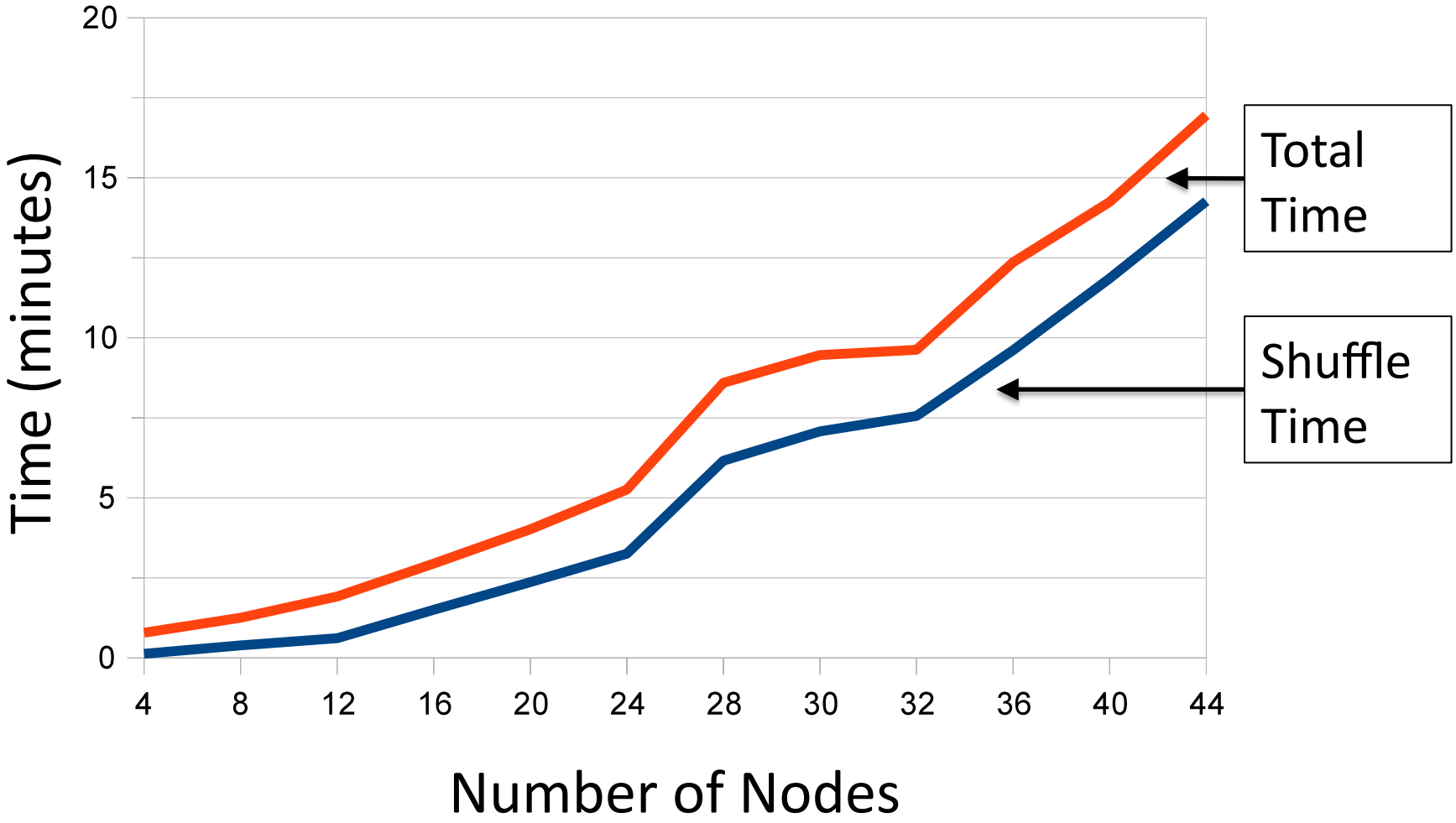
Questions?



Time required to send varying message sizes in a 16-node group



Time required to send 1MB of data (balanced) using Dissent



Definitions

- **Integrity:** All honest group members have the messages of all other honest diners, or know that the shuffle failed
- **Anonymity:** No subset of members of size $\leq N - 2$ can match another member to her message with probability better than random guessing
- **Accountability:** No honest group member has a proof that an honest member is faulty, and either (a) all honest members obtain the message of all other honest members, or (b) all honest members expose at least one faulty member

Protocol Components