Proactively Accountable Anonymous Messaging in Verdict

Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford

Department of Computer Science
Yale University

22nd USENIX Security Symposium
14 August 2013

Acct #35139387
Acct #09836271

On the eve of an election in country X…
activist learns that the prime minister is stashing stolen money in a secret bank account.

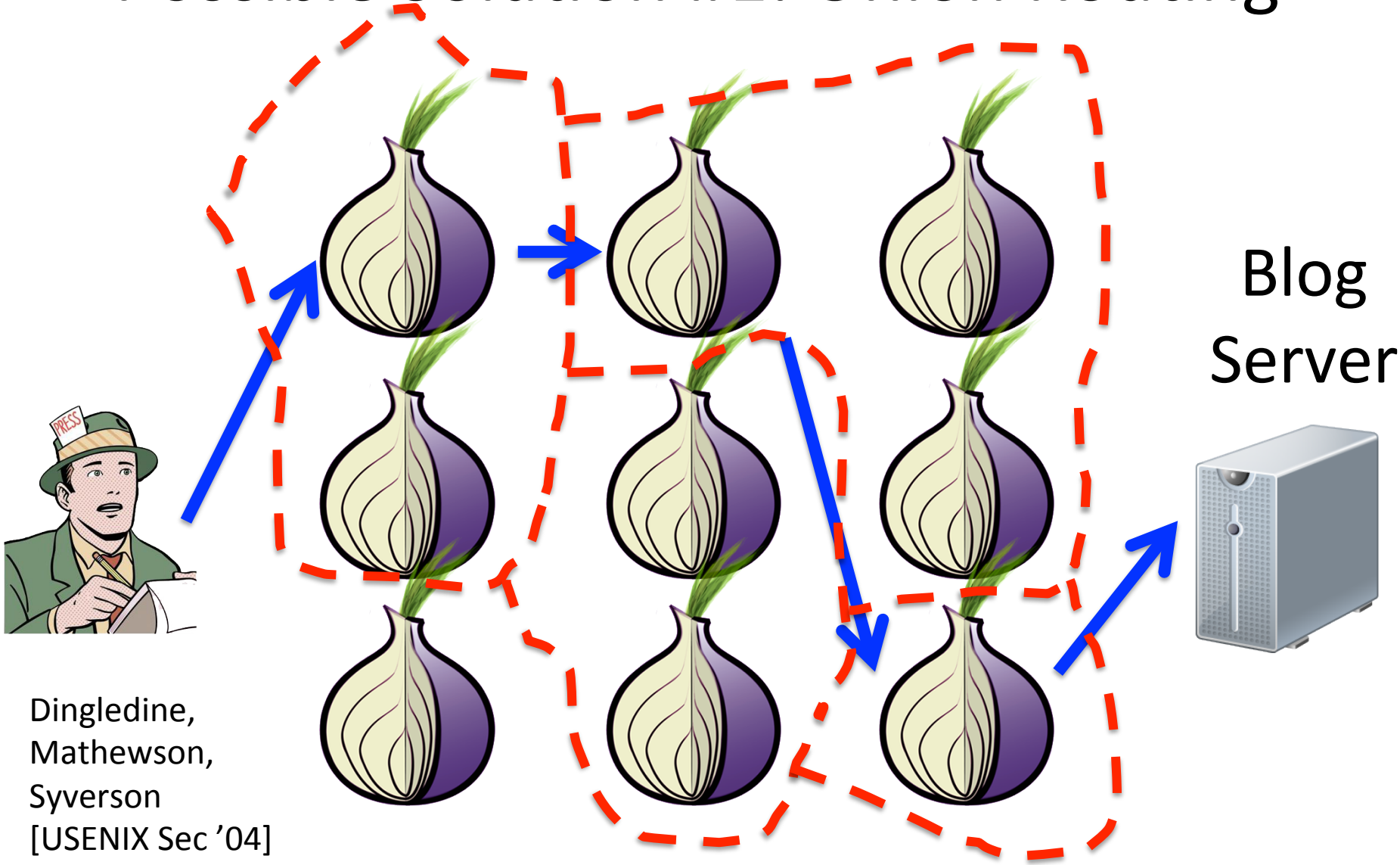
MUST PUBLISH this info **before the election**

Can I publish these anonymously?

Can I publish these anonymously?

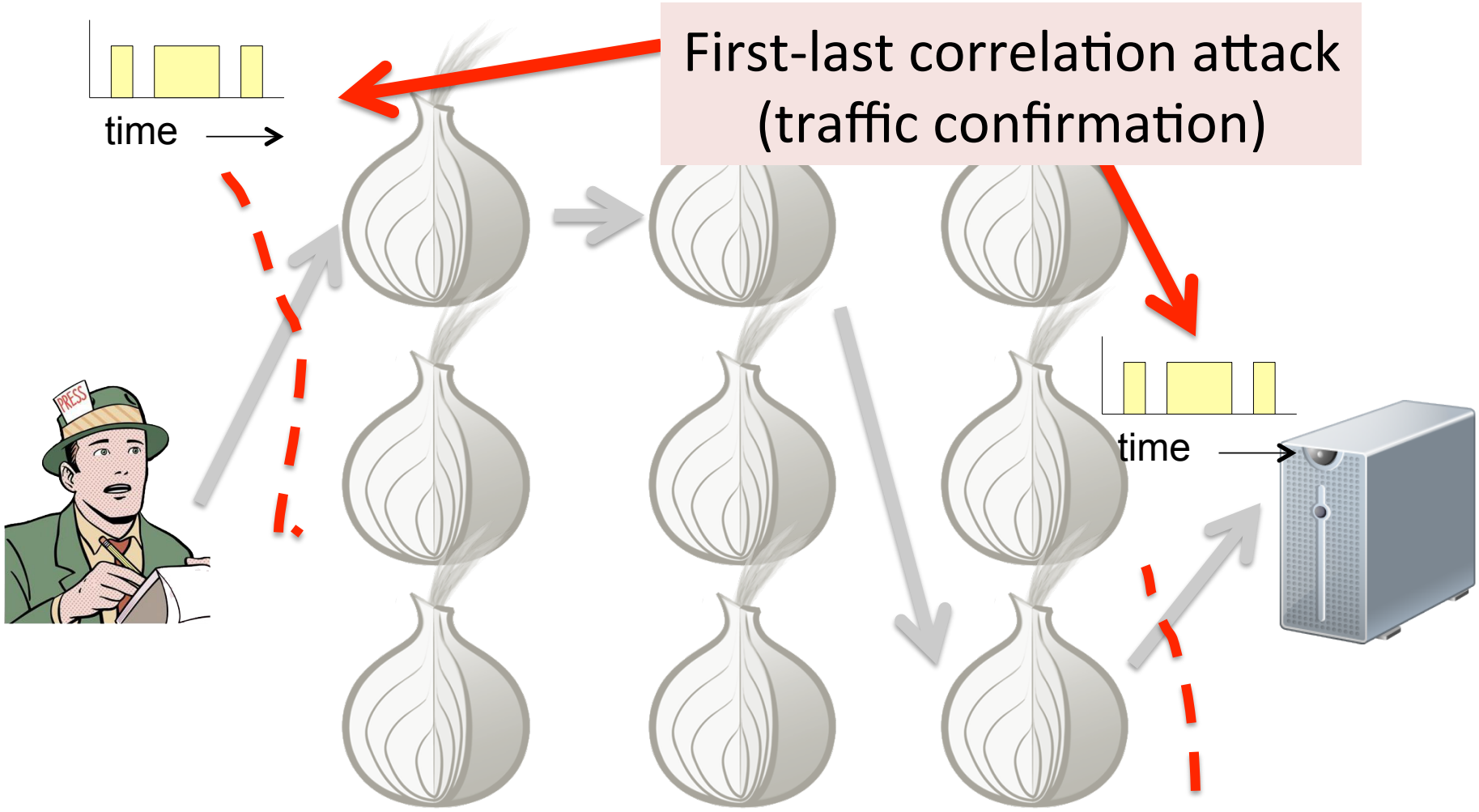
Blog Server

Possible Solution #1: Onion Routing



Dingledine, Mathewson, Syverson [USENIX Sec '04]

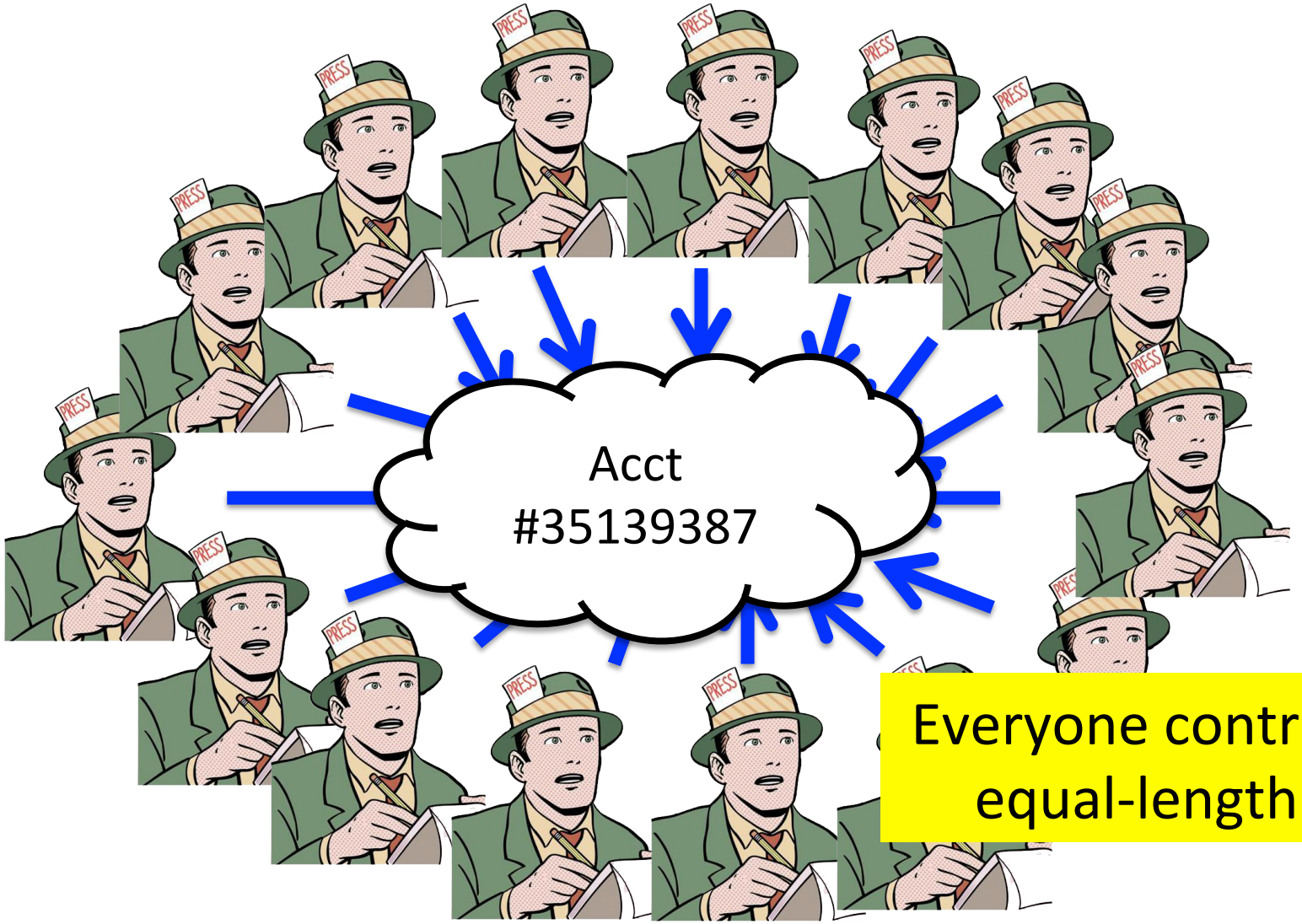
Possible Solution #1: Onion Routing



Dining Cryptographers networks (DC-nets) are resistant to traffic analysis attacks

Blog Server

Possible Solution #2: DC-nets

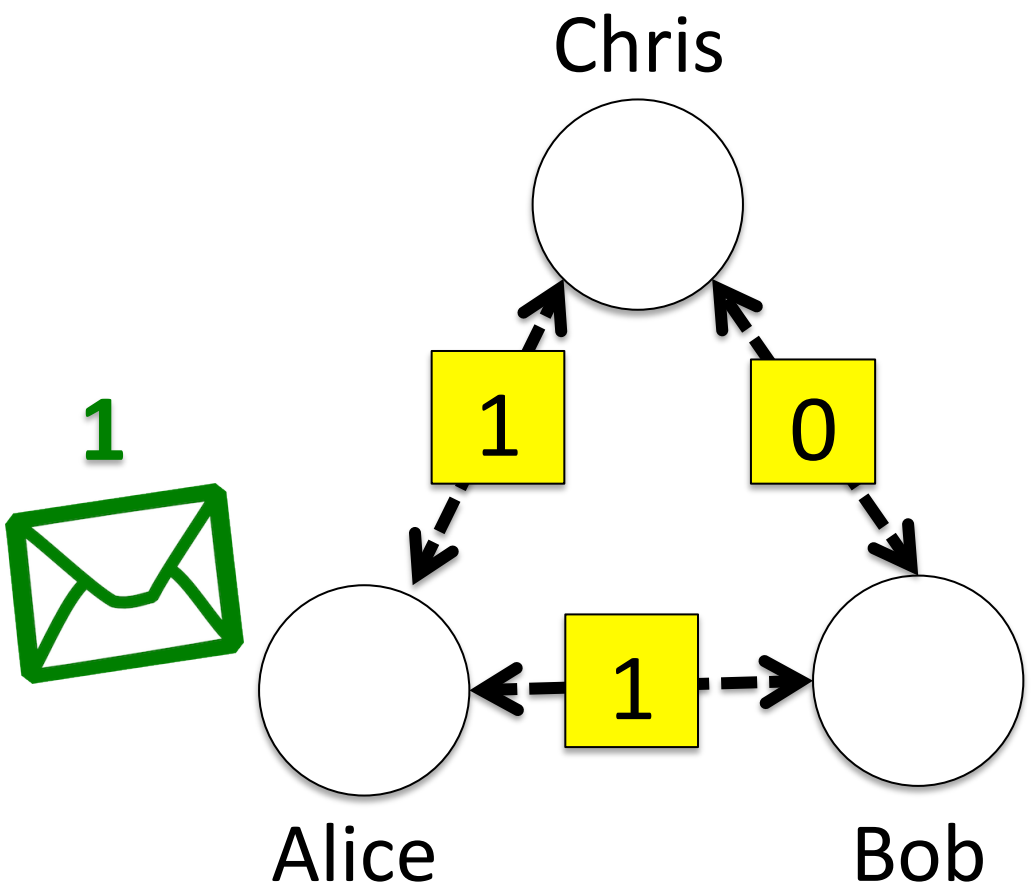


Blog Server

Everyone contributes an equal-length message

Possible Solution #2: DC-nets

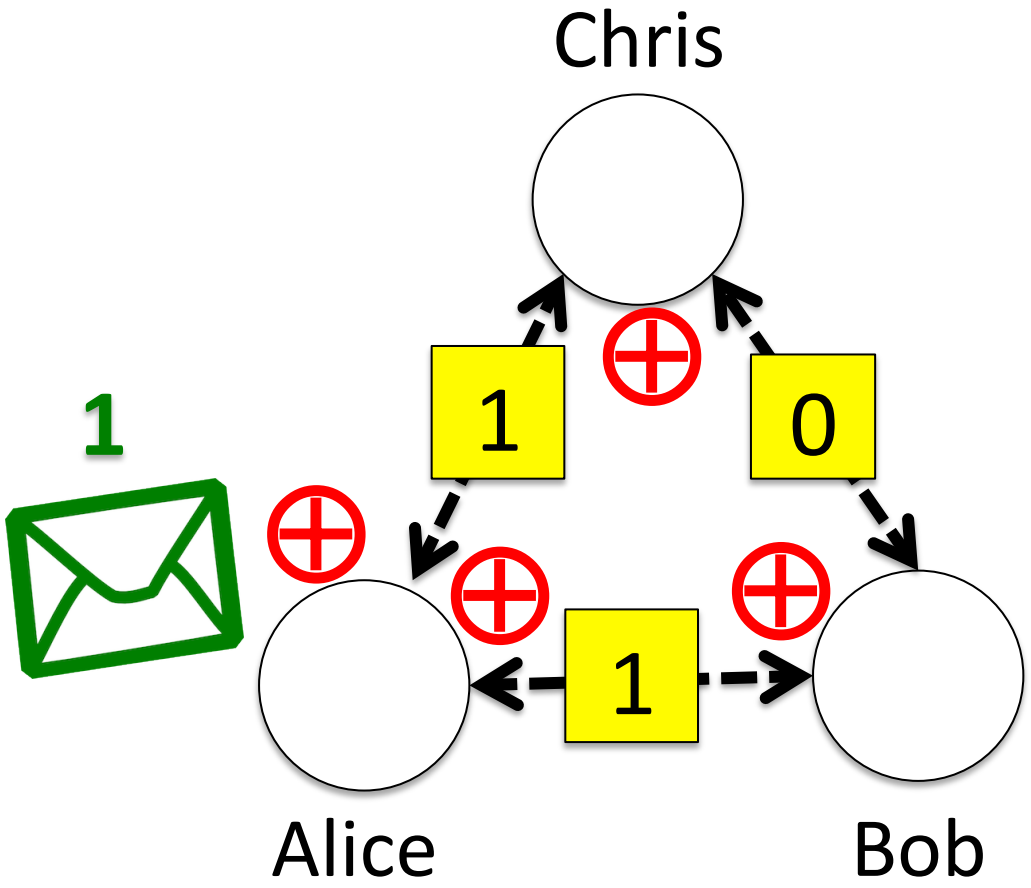
DC-nets in 30 Seconds



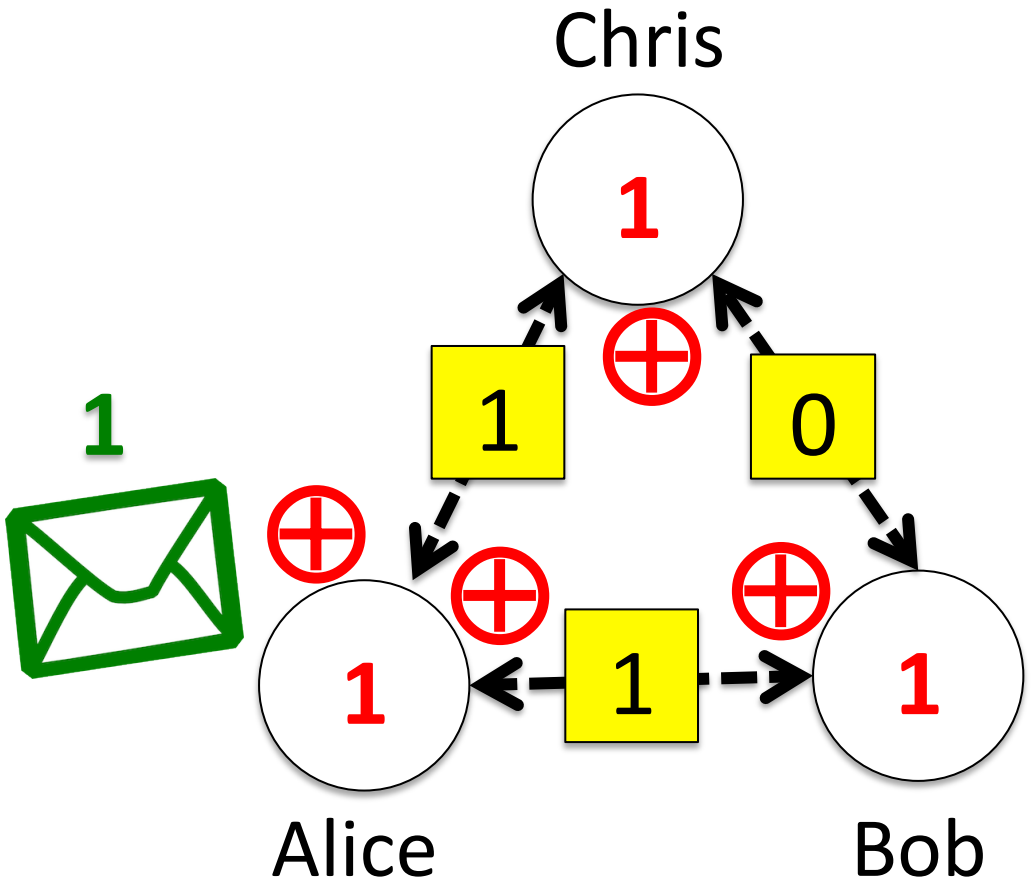
Implement an anonymous group broadcast primitive

David Chaum
"Dining Cryptographers Problem"
[J. Cryptography '88]

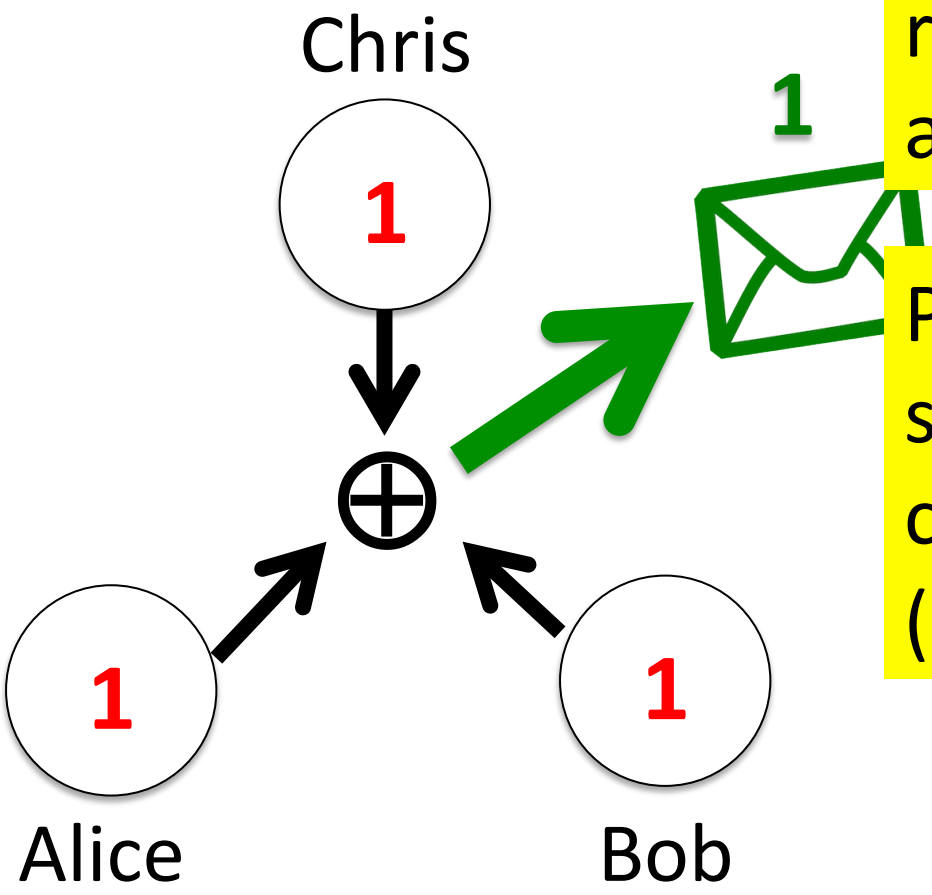
DC-nets in 30 Seconds



DC-nets in 30 Seconds



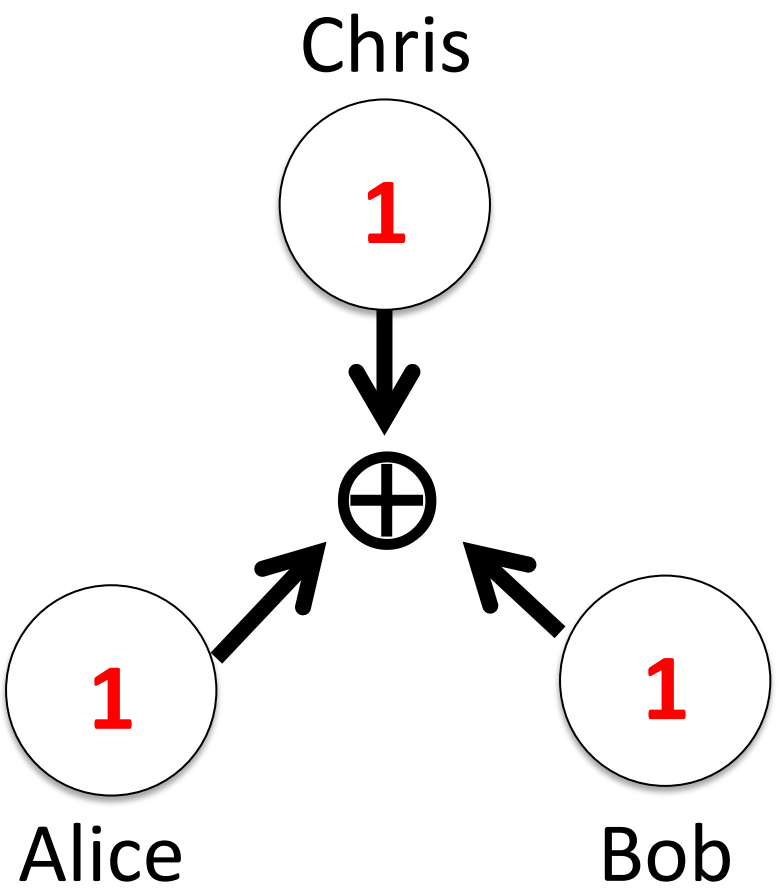
DC-nets in 30 Seconds



DC-nets are resistant to traffic analysis attacks

Primarily use fast symmetric-key crypto operations (PRNG, XOR)

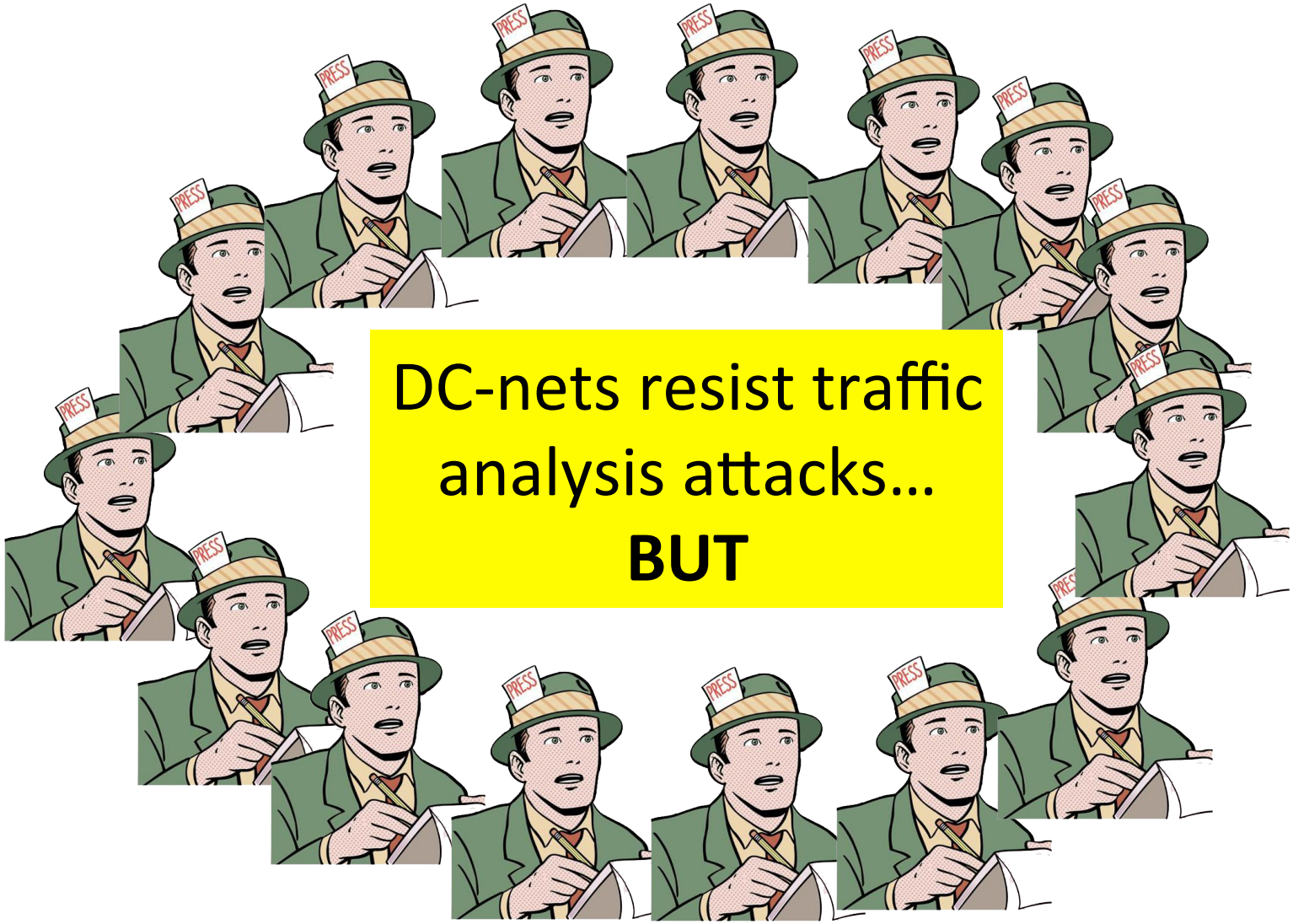
DC-nets in 30 Seconds



Dissent: DC-nets made practical

- Splits nodes into clients and servers
- Scales to 1000s of nodes
- Handles client churn
- Anonymity set size = set of honest nodes

Possible Solution #2: DC-nets

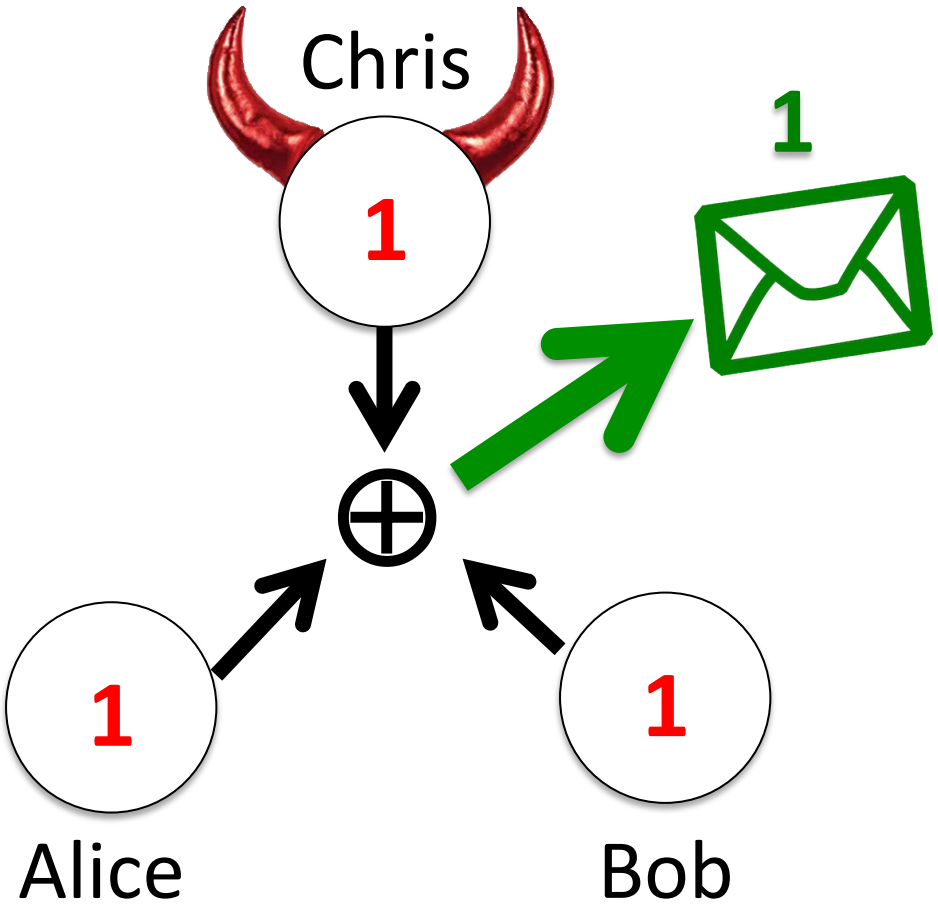


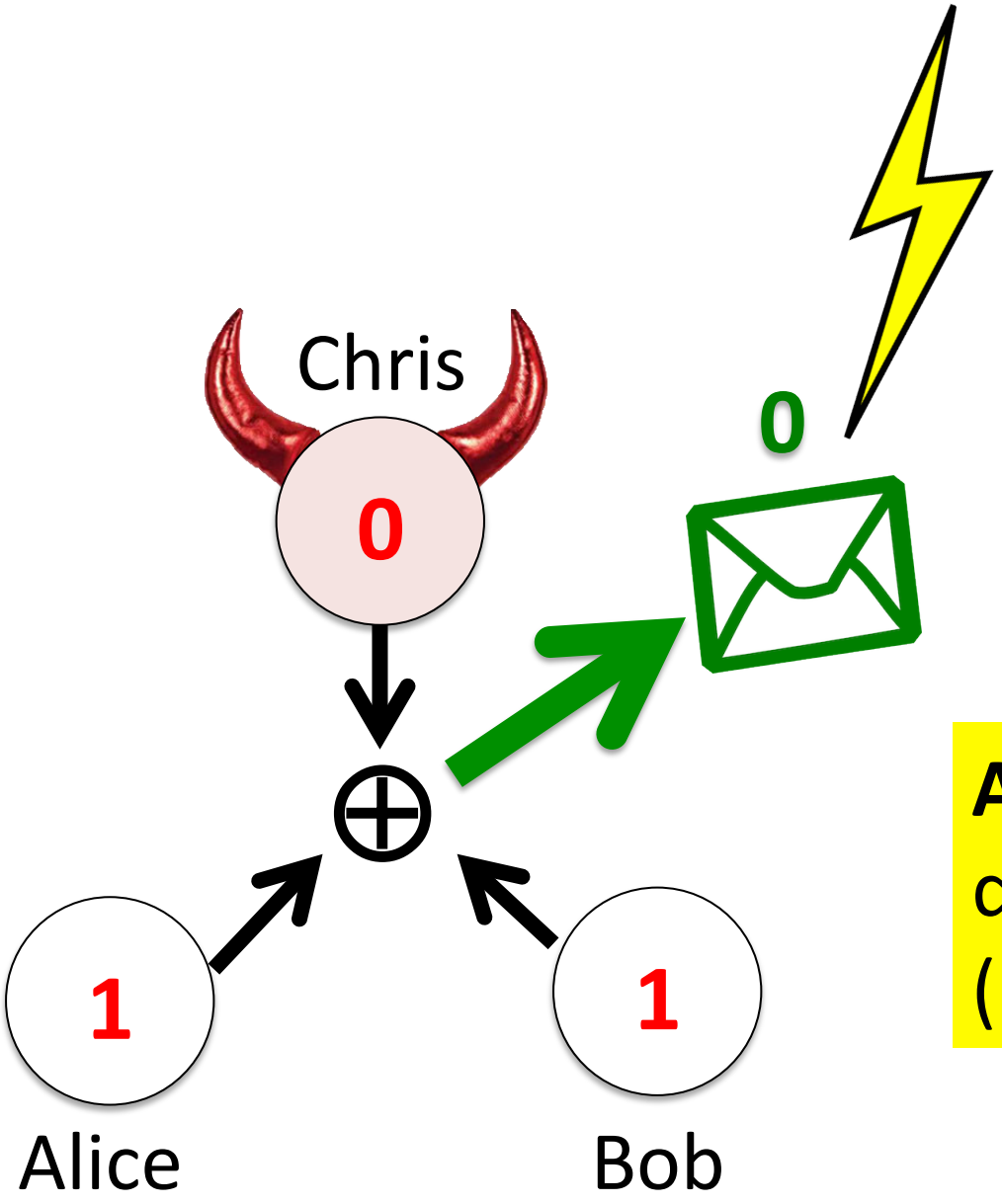
DC-nets resist traffic analysis attacks…
BUT

Blog Server

Possible Solution #2: DC-nets

Blog Server


Chris
1
1
1
1
Alice
Bob

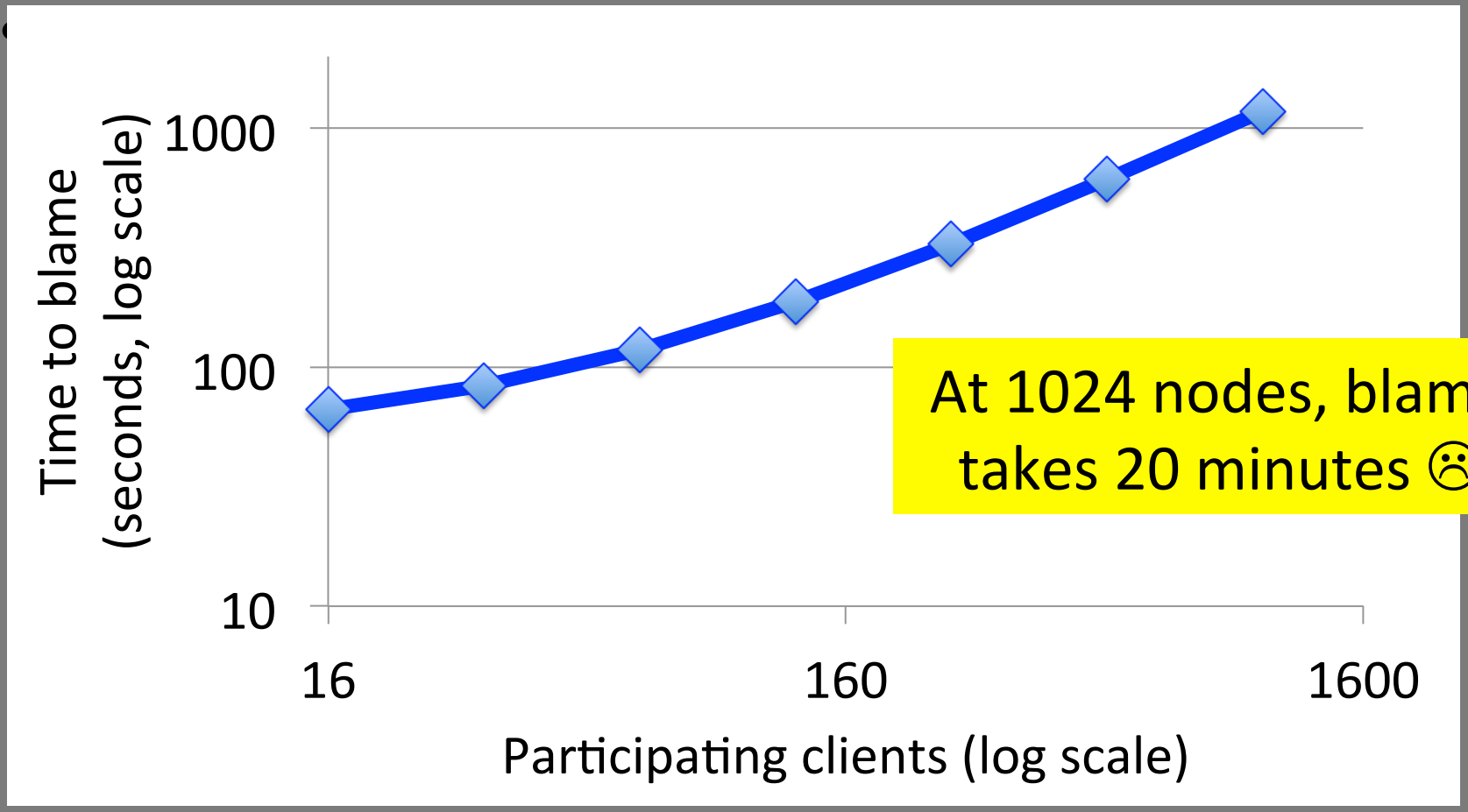


Anonymous disruption (DoS) attack

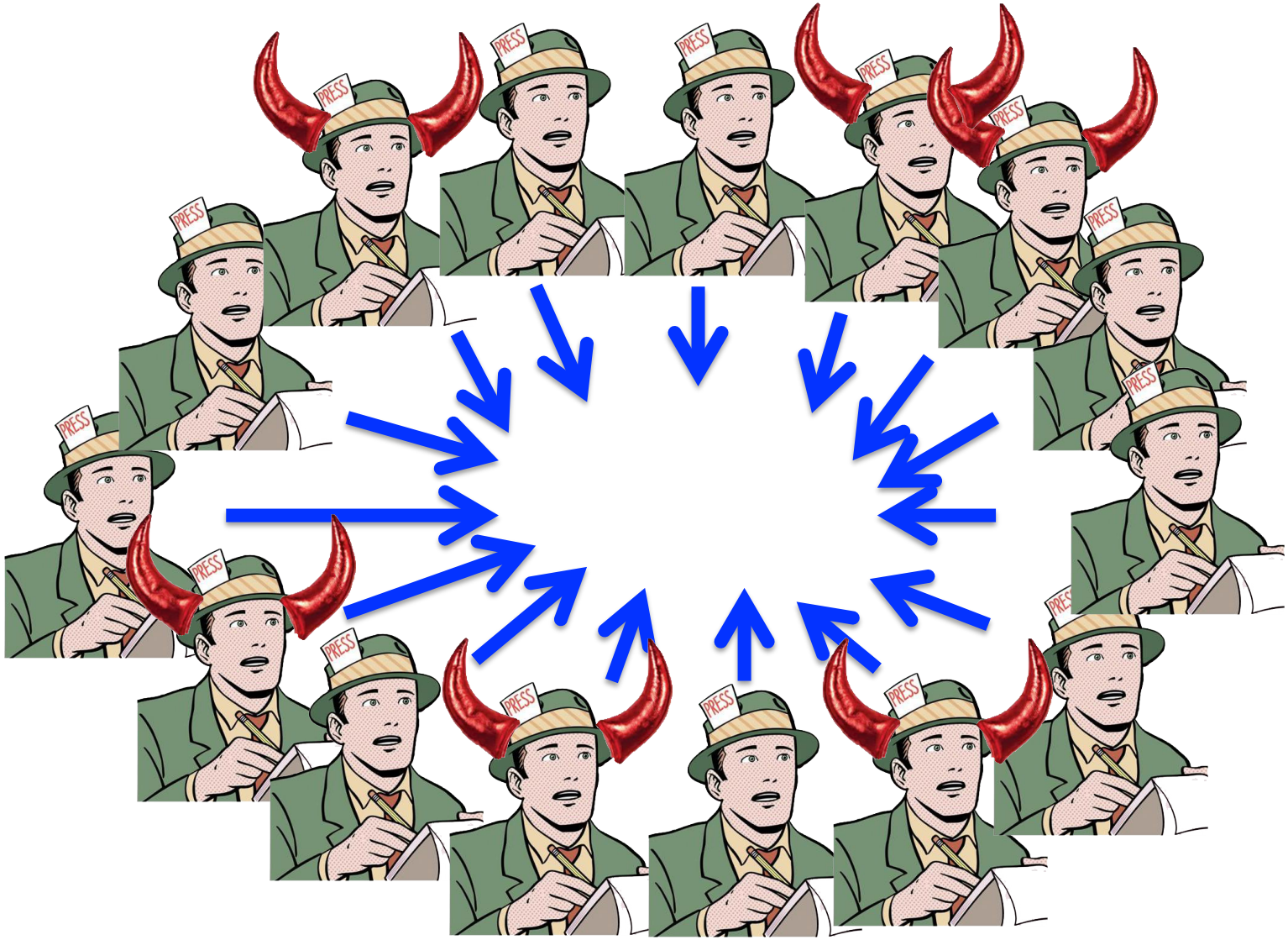
Possible Solution #3: Dissent

- Dissent can handle this sort of misbehavior
  - After a disruption occurs, participants run a **shuffle/e-voting protocol**
  - The anonymous sender sends an **accusation** through the shuffle
  - All nodes use the accusation to trace ("**blame**") the disruptor

Possible Solution #3: Dissent

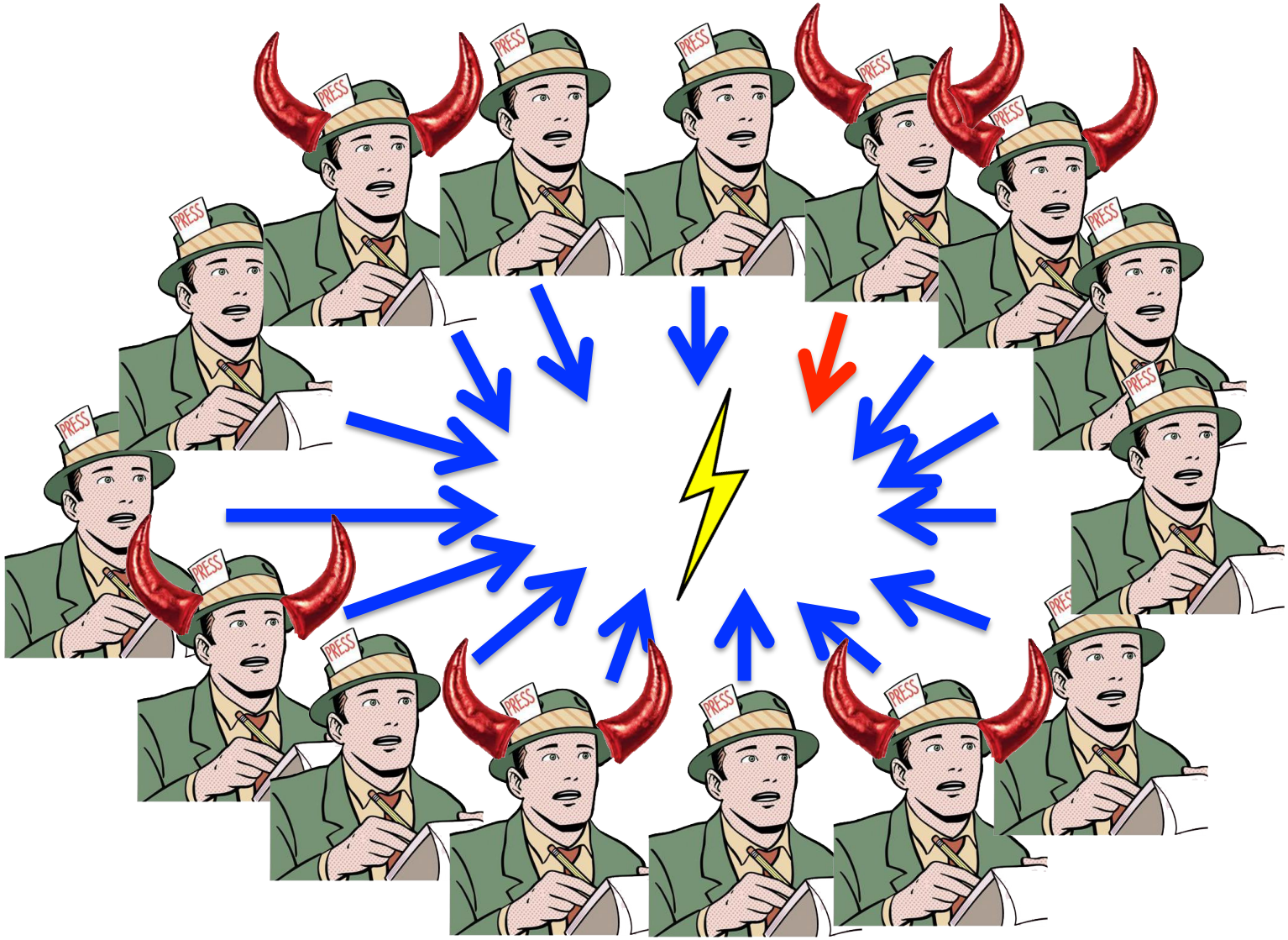


Possible Solution #3: Dissent



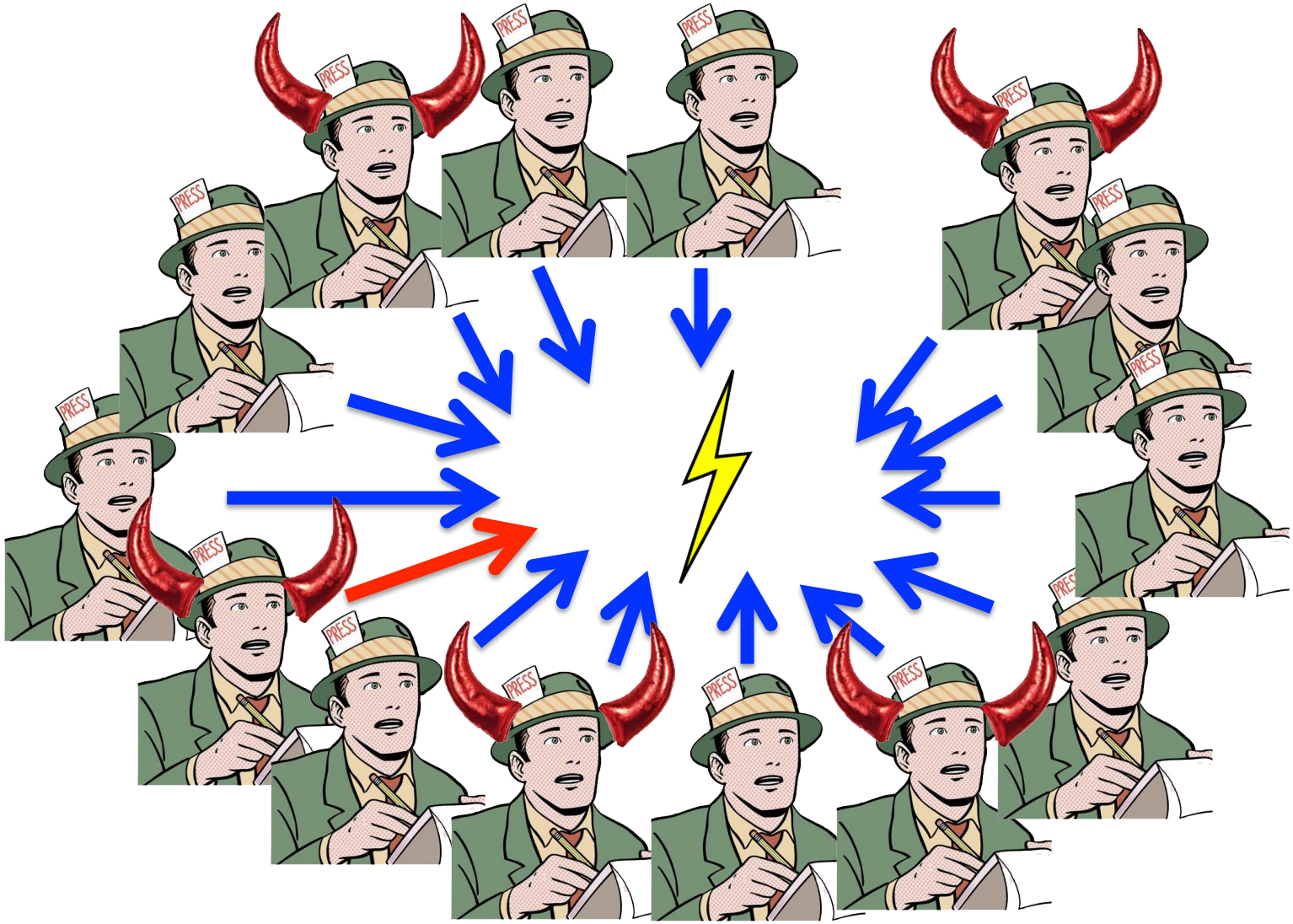
Blog Server

Possible Solution #3: Dissent



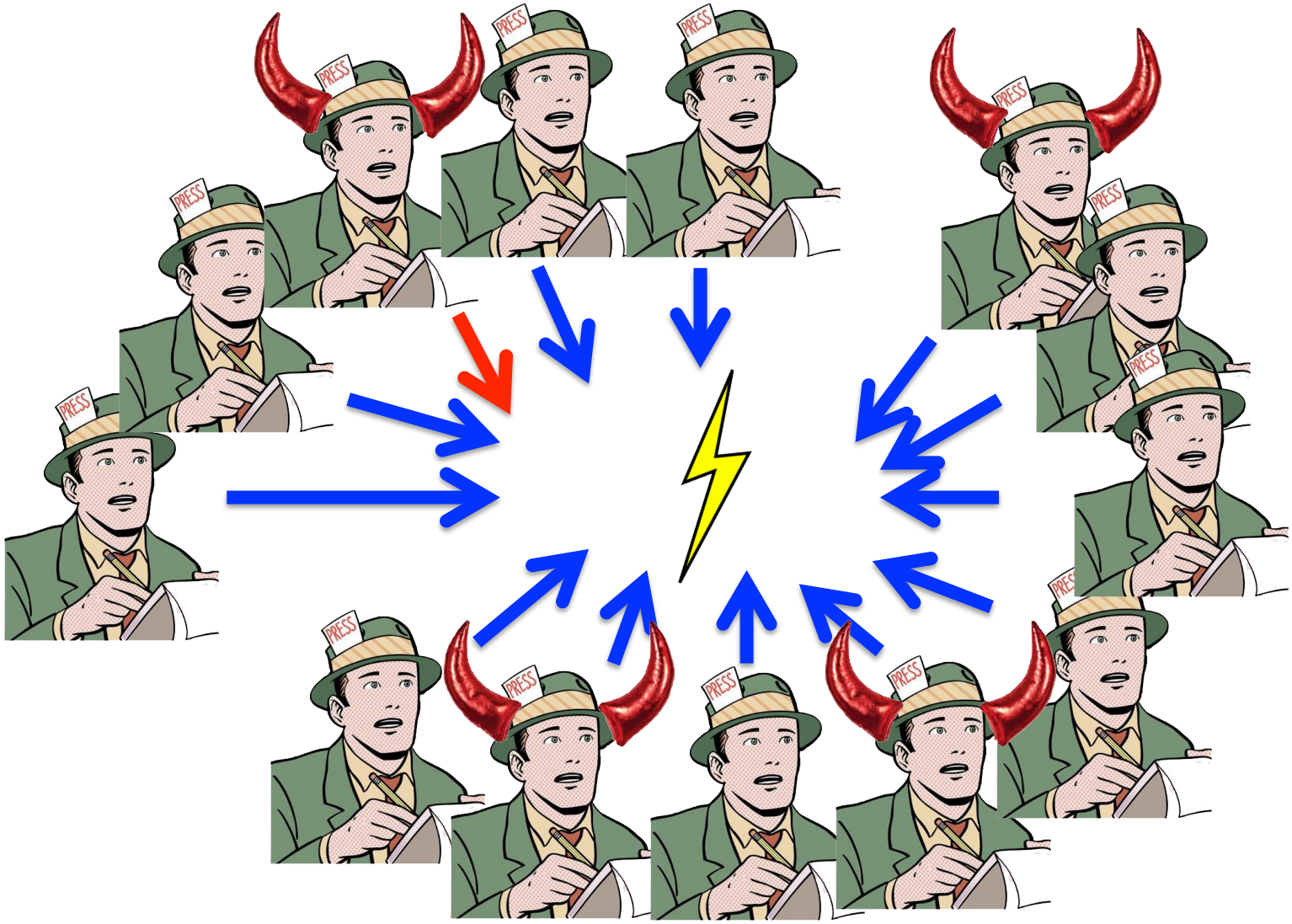
Blog Server

Possible Solution #3: Dissent



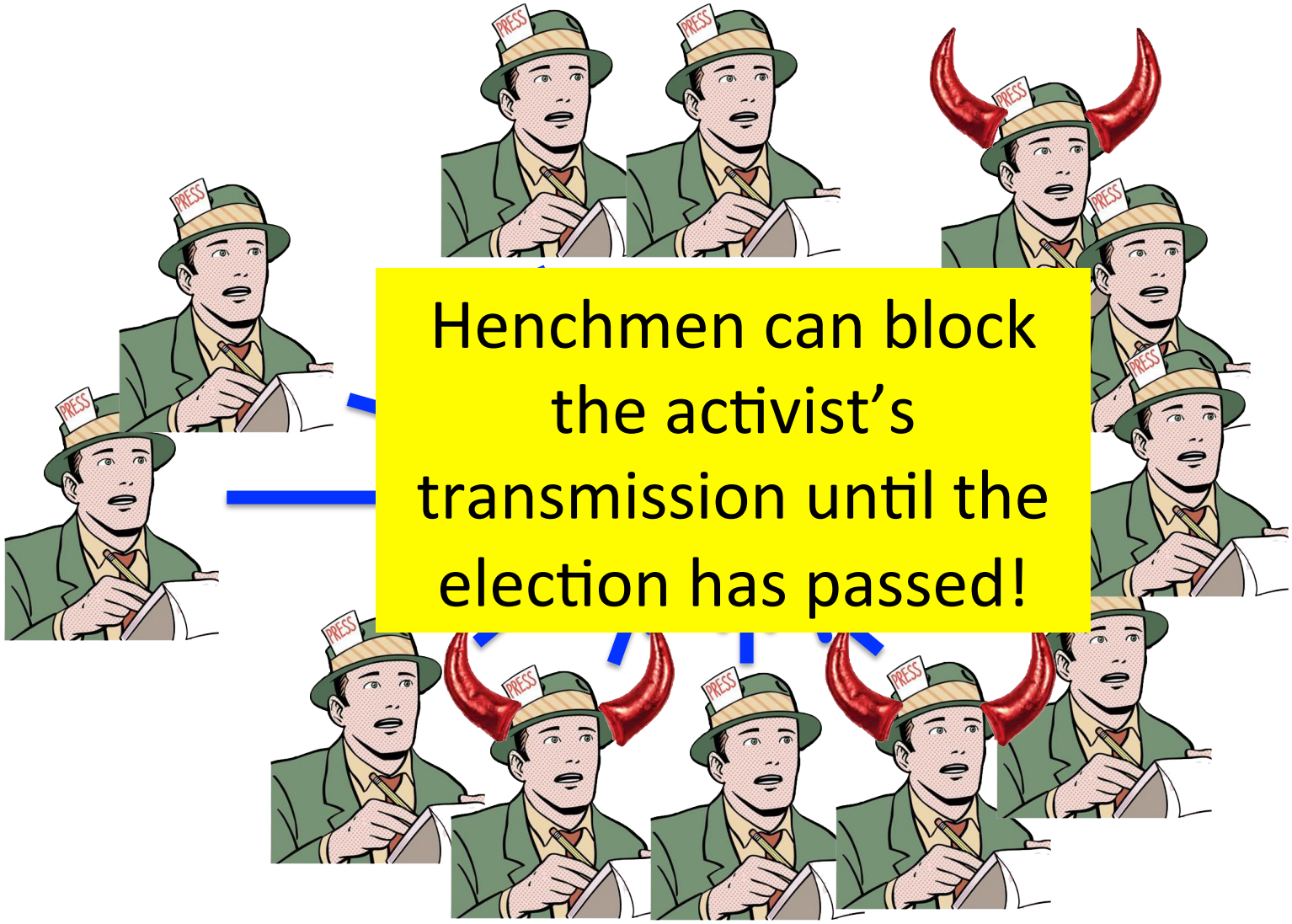
Blog Server

Possible Solution #3: Dissent



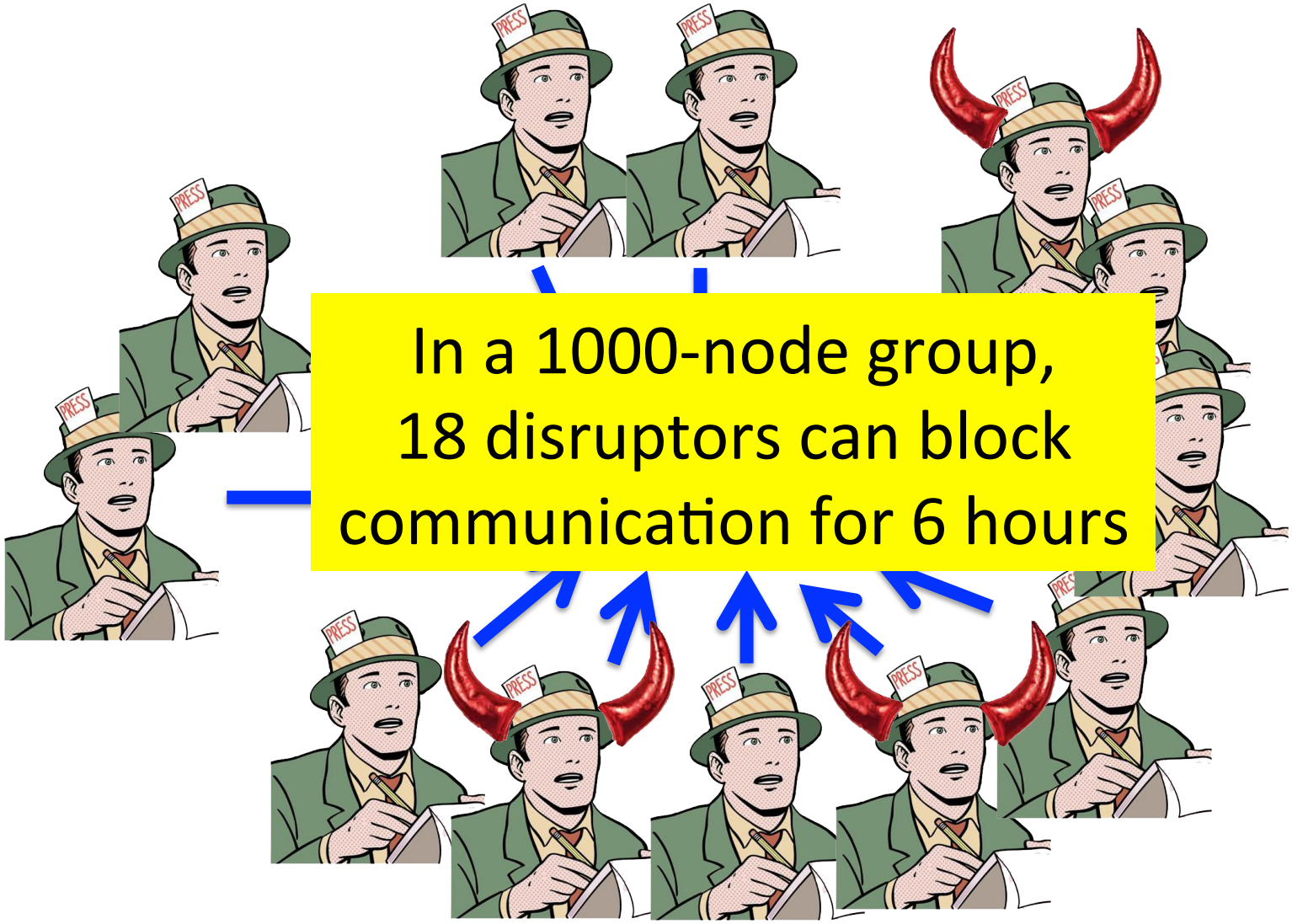
Blog Server

Possible Solution #3: Dissent



Blog Server

Possible Solution #3: Dissent



Blog Server

Verdict: Motivation

- Can we get
  - the traffic-analysis-resistance of DC-nets and
  - the scalability of Dissent

  with **lower blame cost**?
- Idea: Group members **prove** that their messages are sending are correctly formed.

  → Identify disruptors **before** they jam the anonymous communication channel

“Verifiable” DC-nets

- In 2004 Eurocrypt paper, Golle and Juels propose applying zero-knowledge proof (ZKP) techniques to DC-nets
- Participants **prove correctness** of messages
- Drawbacks of Golle-Juels work: computationally expensive, inefficient in communication cost, uses pairings, requires trusted setup, …
- **Never implemented**…

Verdict: Contributions

1. First (to our knowledge) implementation and evaluation of verifiable DC-nets
2. Two new verifiable DC-nets constructions which give 5.6x speedup over Golle-Juels approach
3. Optimizations to make verifiable DC-nets fast
   - for long messages,
   - when there are no *active* disruptors, and
   - by optimistically using XOR-based DC-nets when possible (**138x speedup**)

Outline

- Background and Motivation
- **Verdict**
  - **Design Challenges**
  - Optimizations
- Evaluation
- Conclusion

Design Challenges

1. Resist traffic analysis attacks
2. Make sender's transmission indistinguishable
3. Prove that transmissions are well-formed

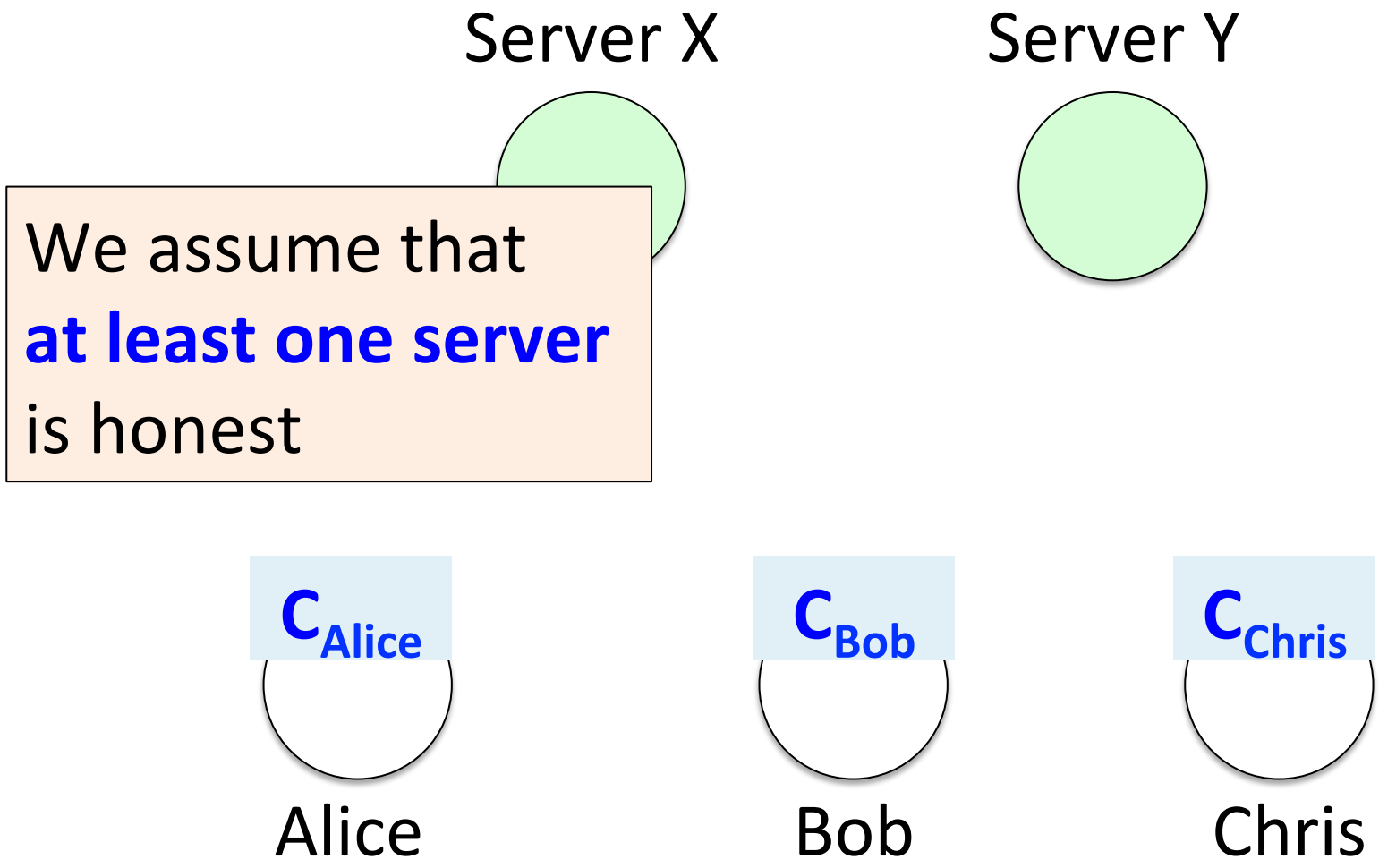
Design Challenges

1. **Resist traffic analysis attacks**
2. Make sender's transmission indistinguishable
3. Prove that transmissions are well-formed

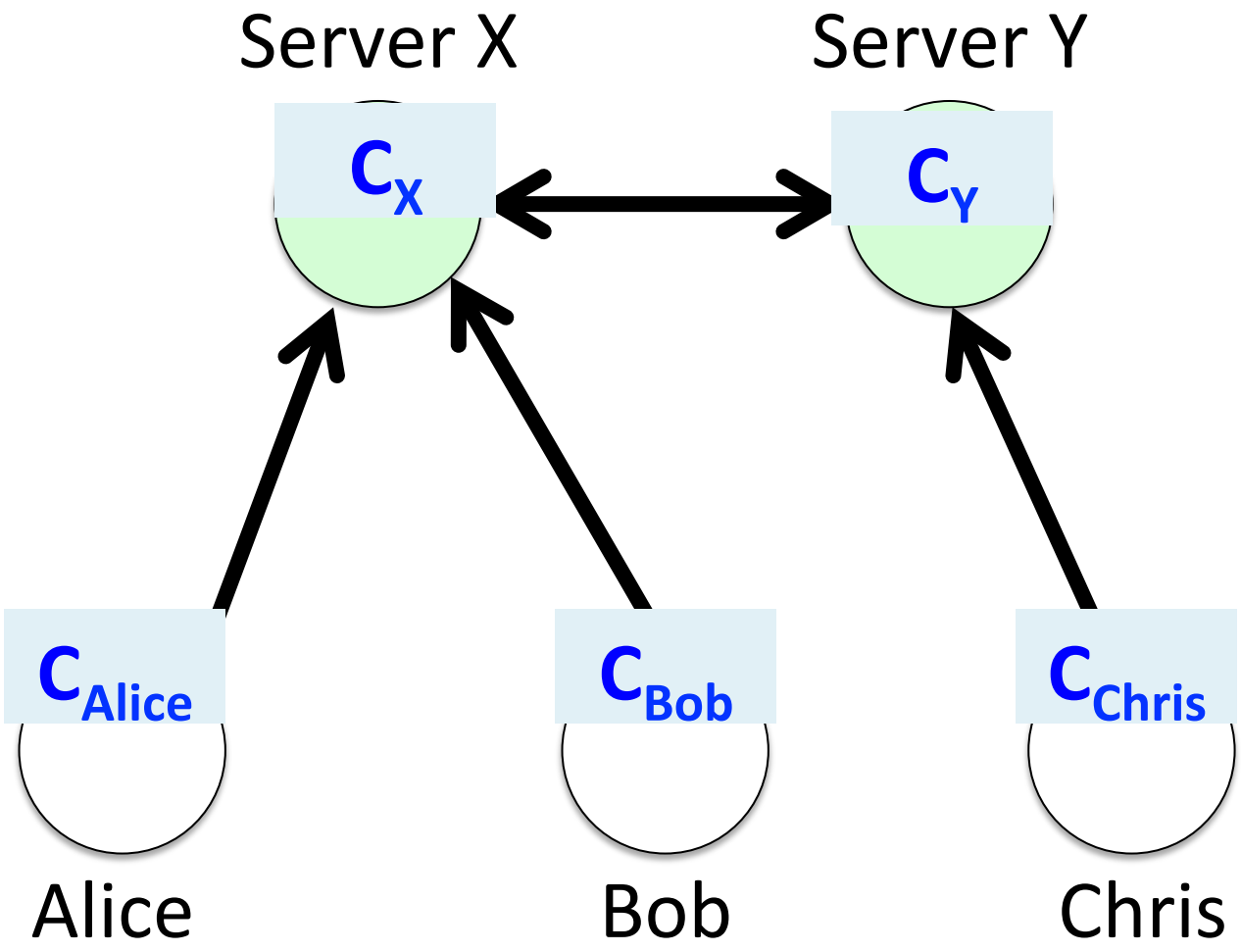
Challenge 1: Traffic Analysis Resistance

- Time is divided into **messaging rounds**
- One anonymous sender per messaging round
- Every client transmits the **same number of bits** in every messaging round
  - # of bits sent does not leak sender's identity
- Clients' ciphertexts are **cryptographically indistinguishable**
  - Content does not leak sender's identity

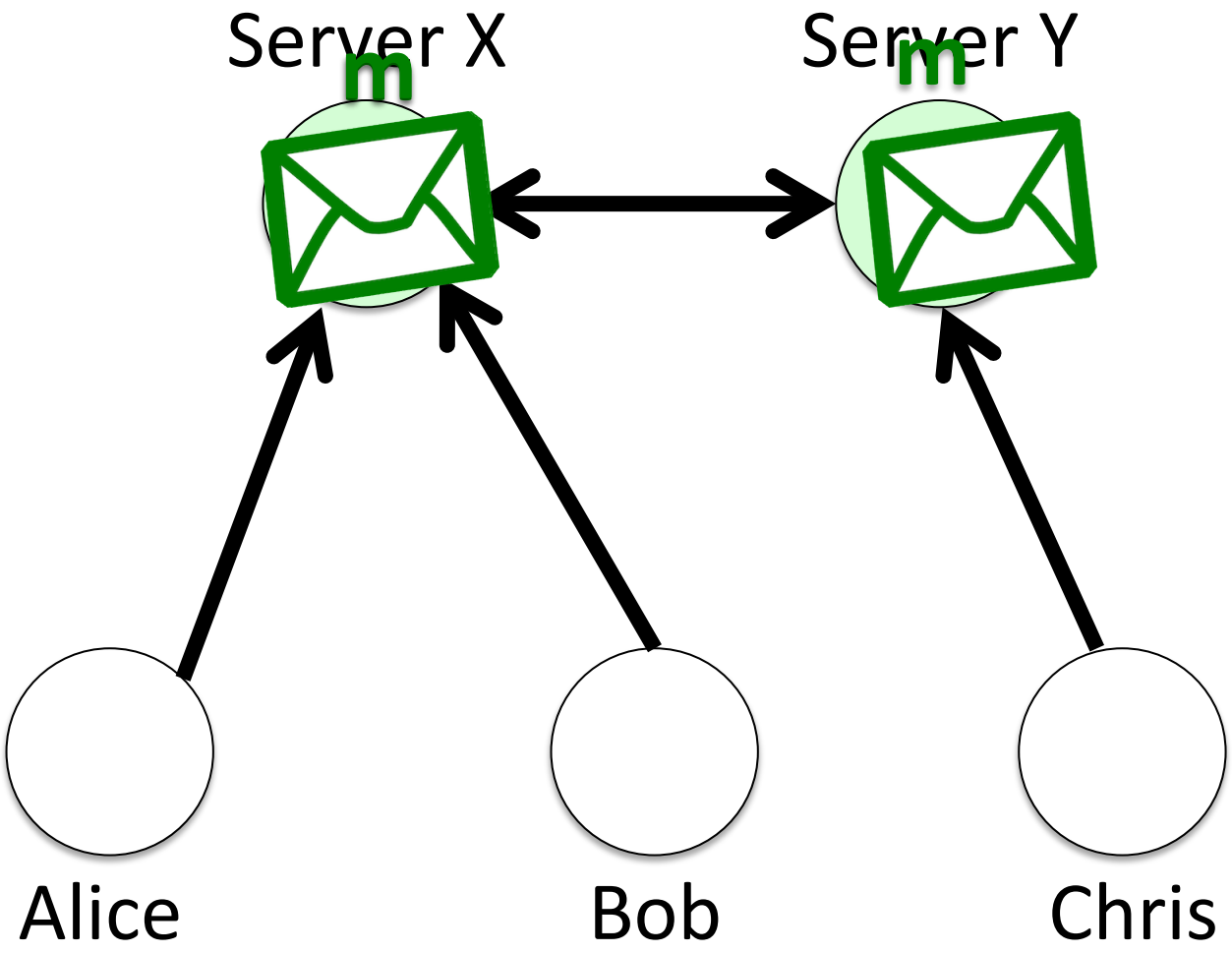
Challenge 1: Traffic Analysis Resistance



Challenge 1: Traffic Analysis Resistance



Challenge 1: Traffic Analysis Resistance



Design Challenges

1. Resist traffic analysis attacks
2. **Make sender's transmission indistinguishable**
3. Prove that transmissions are well-formed

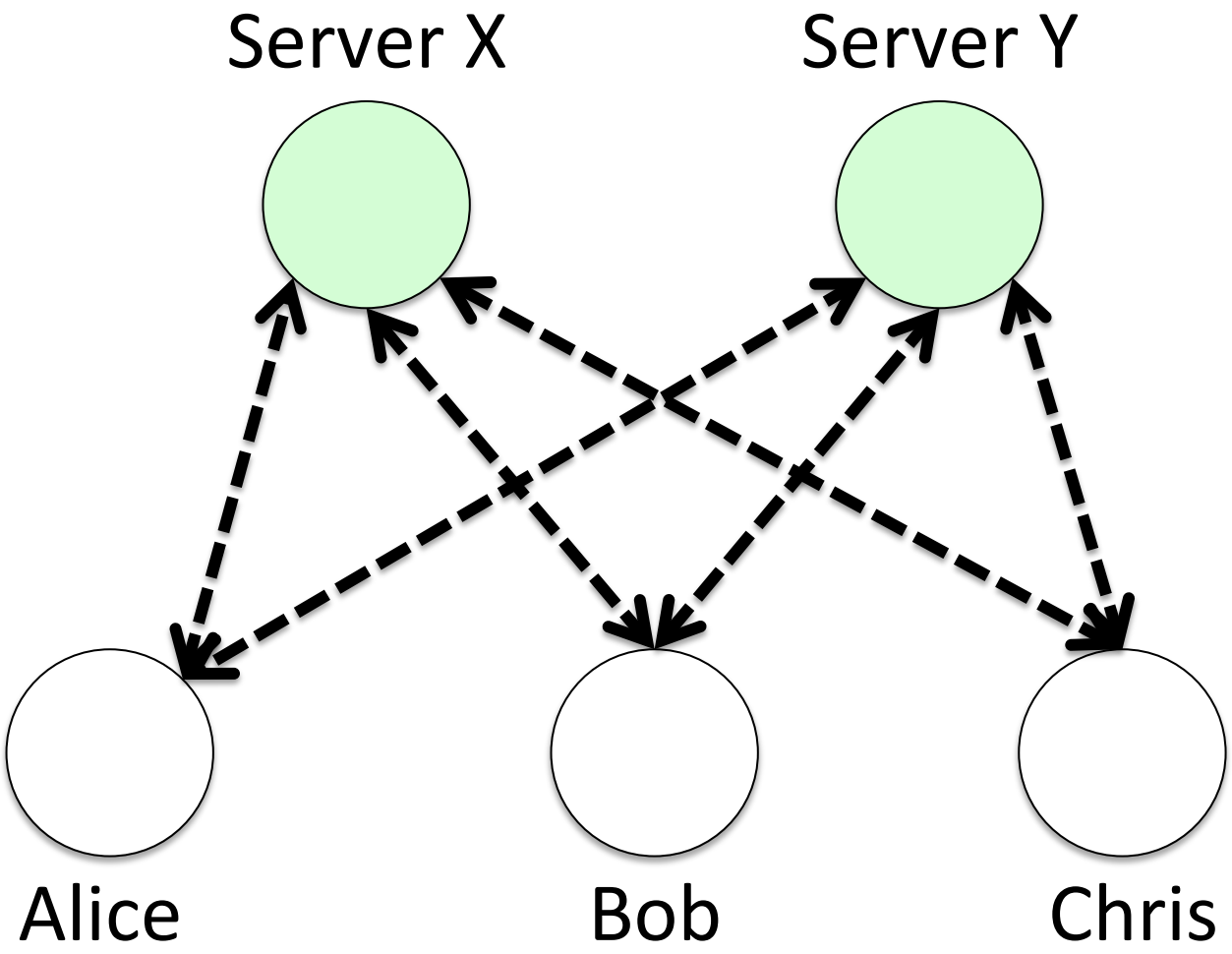
Challenge 2: Encoding Messages

- The transmitting client sends an encryption of arbitrary message: ***m***
- Non-transmitting clients set ***m* = 1**
  - An encryption of the identity element
- Use an ElGamal-like scheme to encrypt

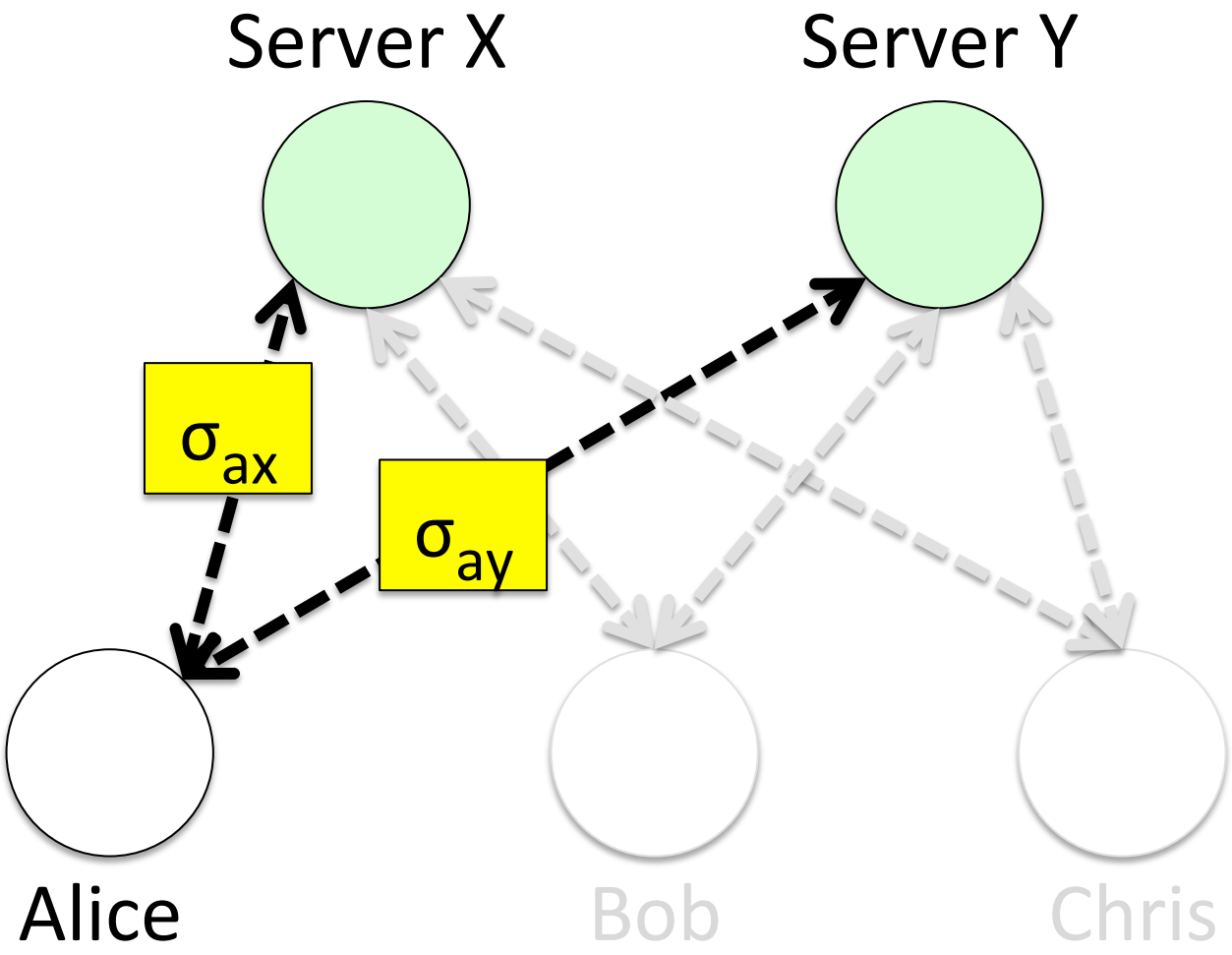
$$E(m, \sigma_1, \ldots, \sigma_N) = m g^{\sigma_1 + \ldots + \sigma_N}$$

… where the σs are secrets shared between clients and servers.

Challenge 2: Encoding Messages

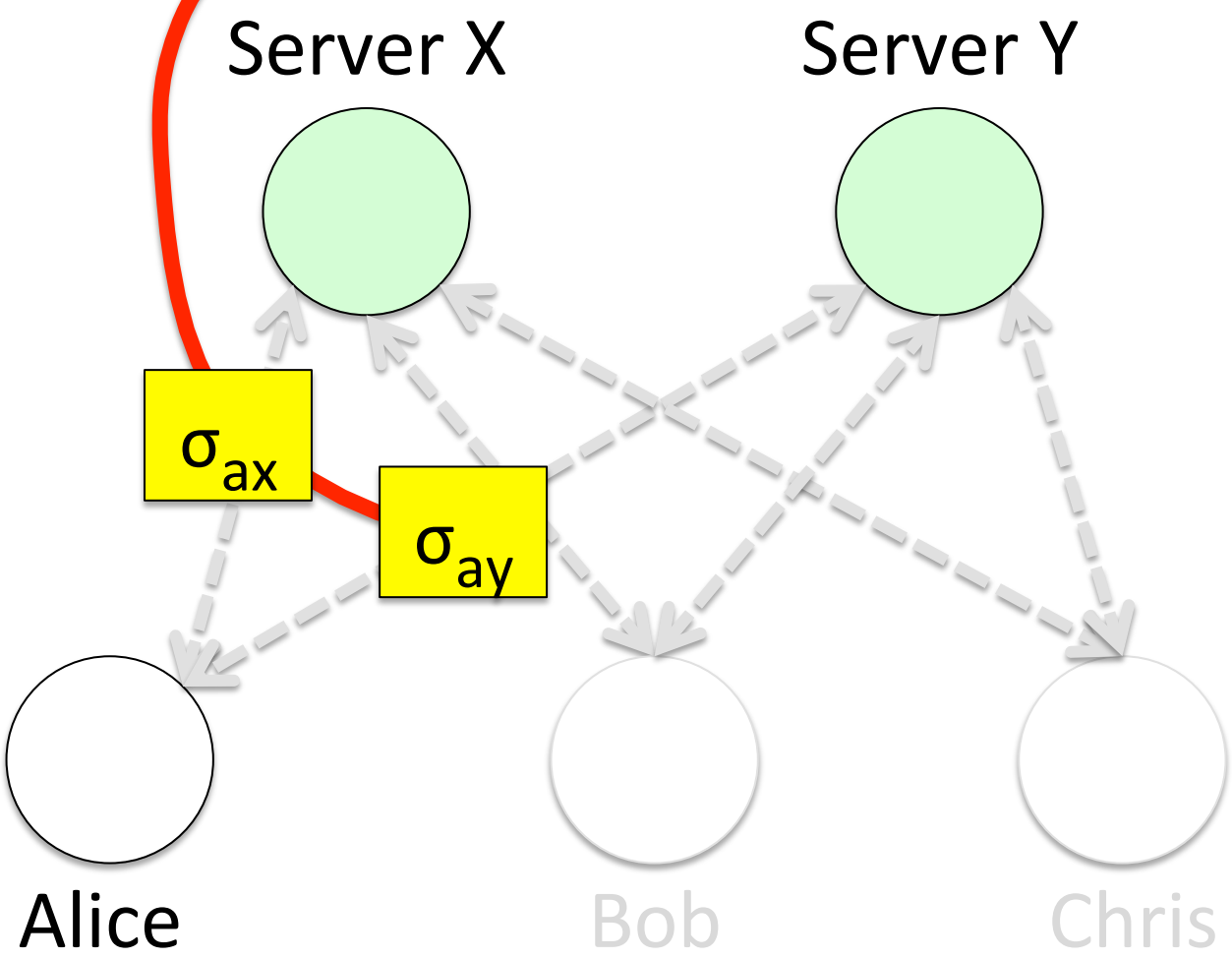


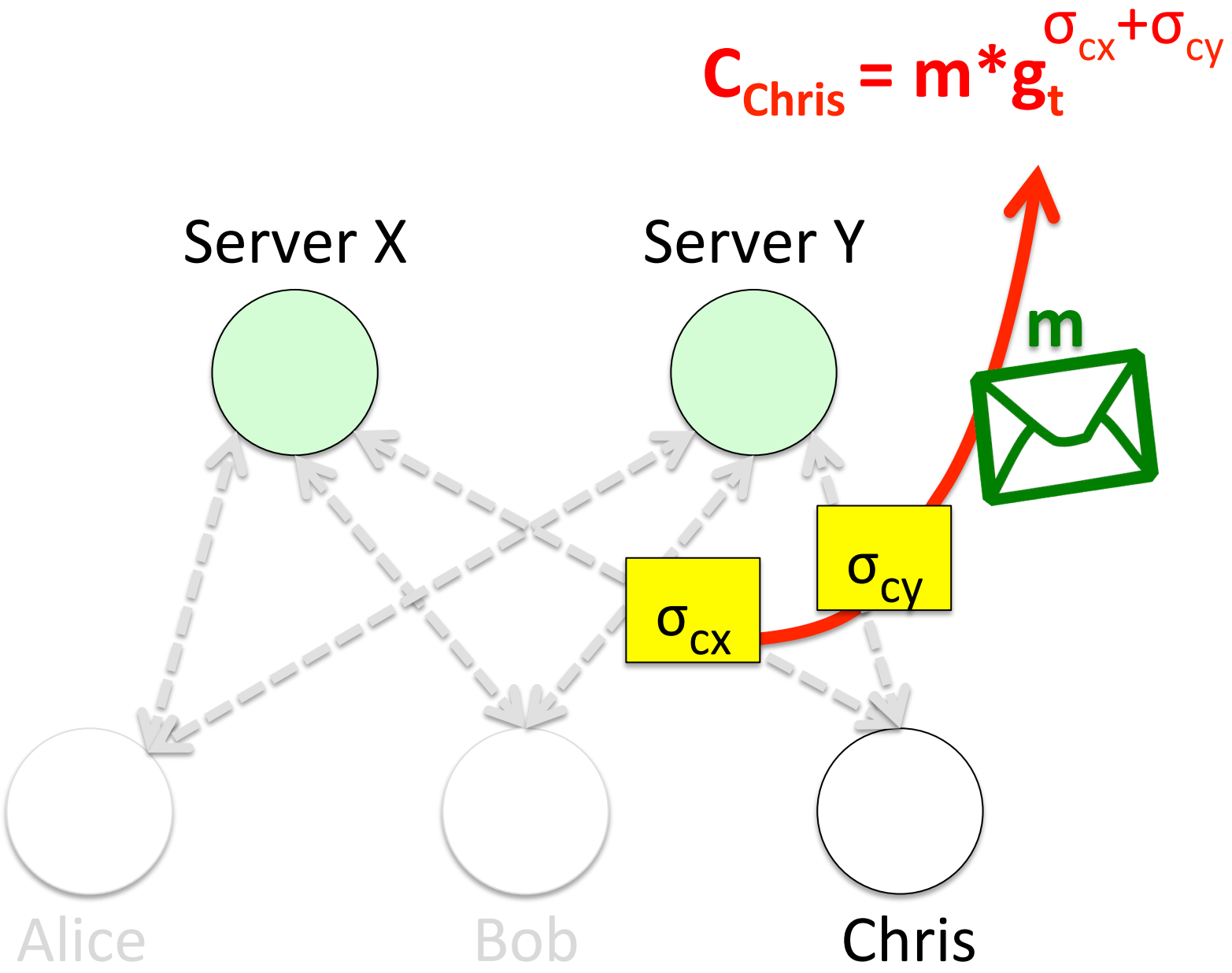
Clients and servers agree to *k*-bit shared secrets σ using DH exchange



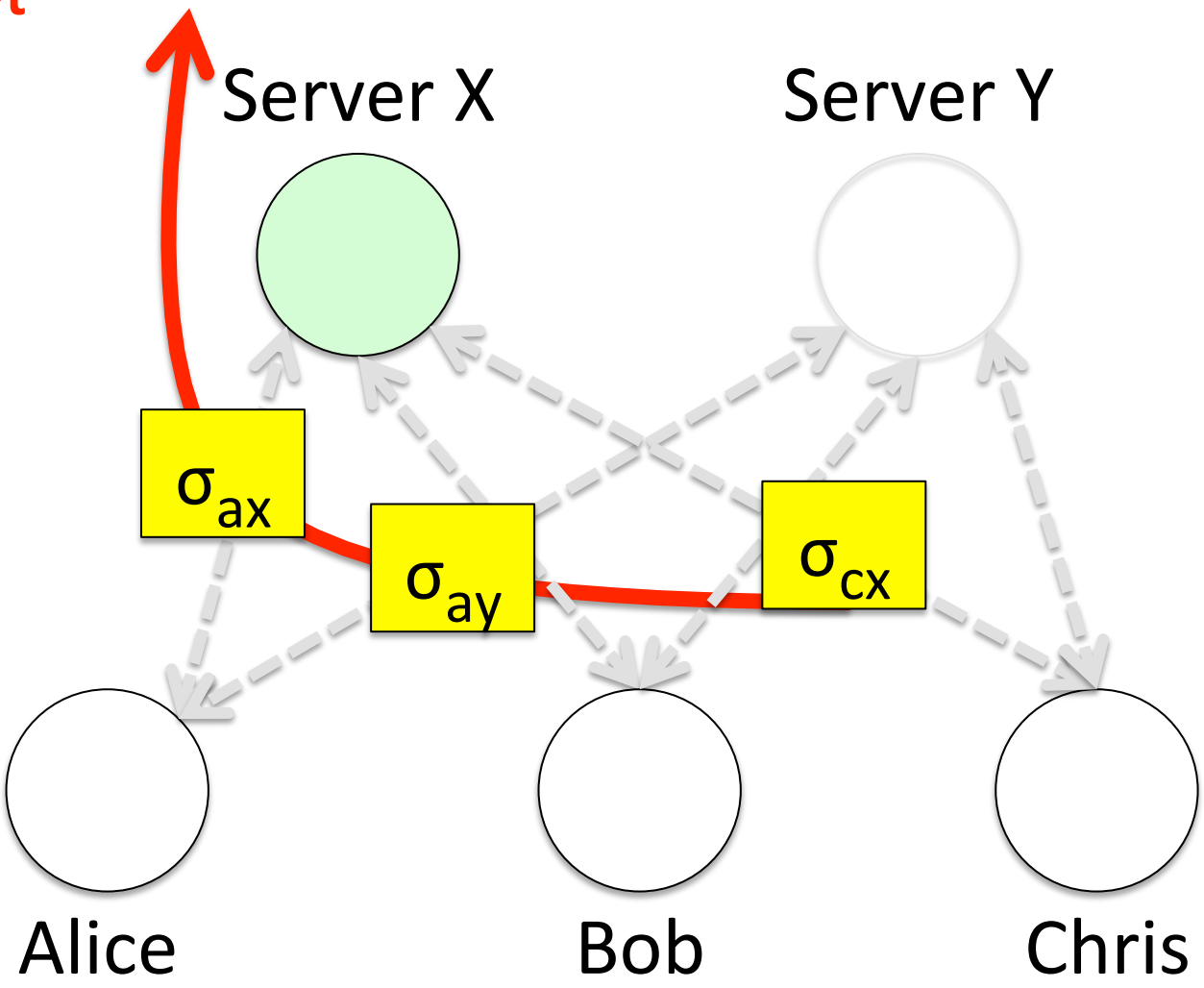
$C_{Alice} = 1 * g_t^{\sigma_{ax}+\sigma_{ay}}$

g_t generates some group in which DDH is hard




$C_{Chris} = m^{*}g_t^{\sigma_{cx}+\sigma_{cy}}$
Server X
Server Y
m
σ_{cy}
σ_{cx}
Alice
Bob
Chris

$$C_X = g_t^{-\sigma_{ax} - \sigma_{bx} - \sigma_{cx}}$$



Challenge 2: Encoding Messages

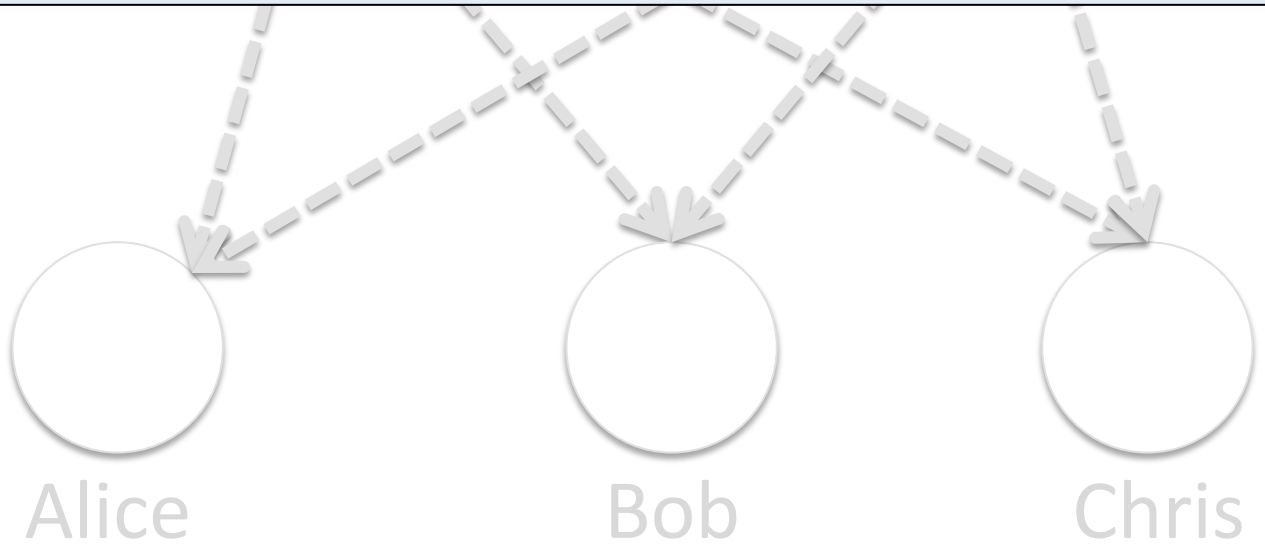
- In product of **C**s, every secret σ_{ij} is included as an exponent once with (+) sign and once with (-) sign:

$$C_{Alice} C_{Bob} C_{Chris} C_X C_Y = m$$

$C_{Alice} = 1 * g_t^{\sigma_{ax}+\sigma_{ay}}$

$C_{Chris} = m * g_t^{\sigma_{cx}+\sigma_{cy}}$

Without knowing the secrets σ, an attacker cannot tell whether Alice or Chris is the anonymous sender of *m* (by DDH assumption)



Design Challenges

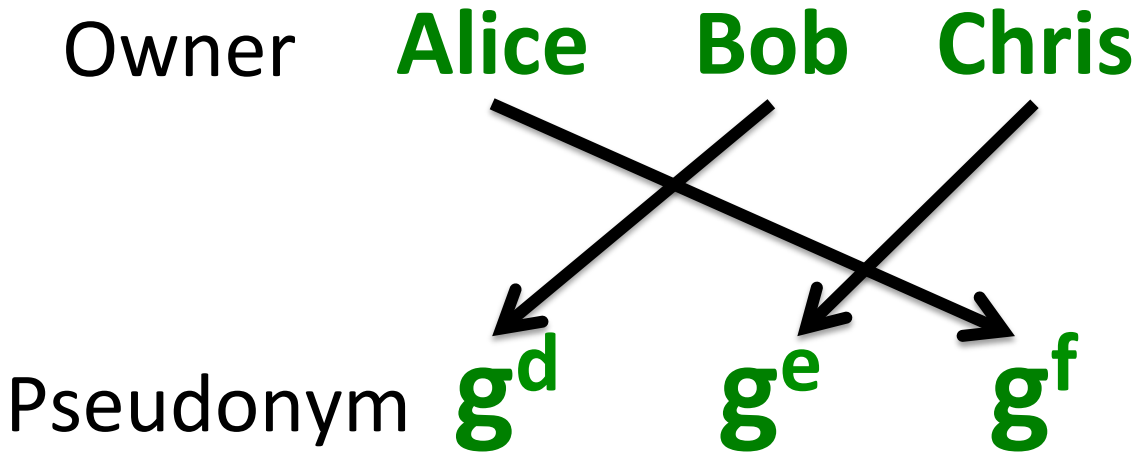
1. Resist traffic analysis attacks
2. Make sender's transmission indistinguishable
3. **Prove that transmissions are well-formed**

Challenge 3: Proving Correctness

- Clients attach non-interactive zero-knowledge proofs of knowledge to their ciphertexts
  - Use off-the-shelf ZKP techniques
    Camenisch-Stadler [ETH Zurich TR-260, '97]
  - Servers check proofs before accepting client ciphertexts
- Servers prove validity of their ciphertexts too

Challenge 3: Proving Correctness

- Recall: one client transmits in each messaging round
- As in Dissent, we use a **key shuffle** to assign pseudonymous "owners" to messaging rounds
  - Each client submits a pseudonym public key to shuffle
  - Shuffle hides owner-to-pseudonym mapping



"A Verifiable Secret Shuffle and its Application to E-Voting" – Neff [CCS '01]

Challenge 3: Proving Correctness

- When Bob submits a ciphertext in a messaging round owned by pseudonym $\mathbf{g^d}$, Bob attaches a proof that:

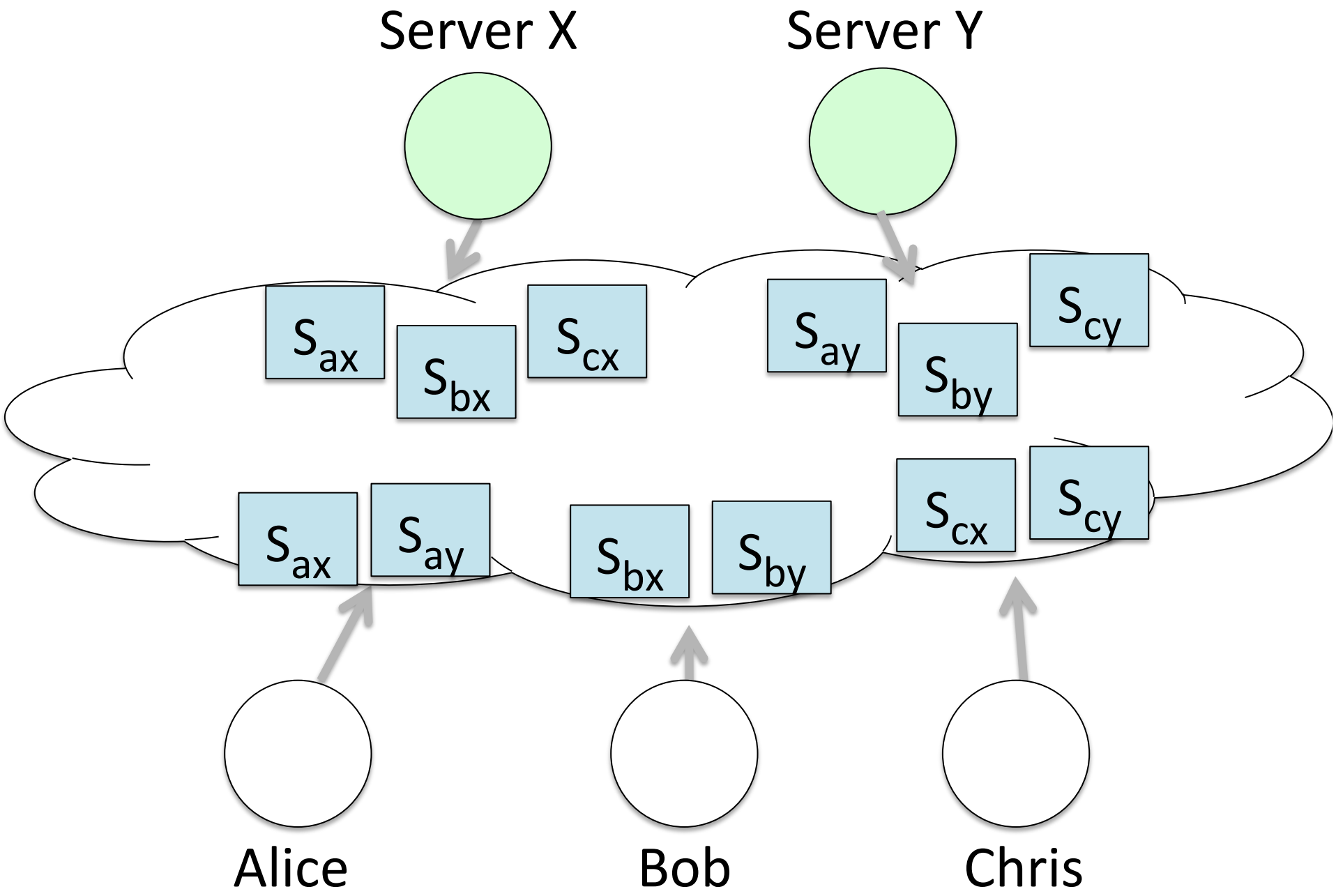
PoK { $\mathbf{C_{Bob}}$ is a correct encryption of **m=1**
OR
I know the pseudonym secret **d** }

Challenge 3: Proving Correctness

- Clients and servers publish **commitments** to their shared secrets $\boldsymbol{\sigma_{ij}}$

$$S_{ax} = \text{Commit}(\sigma_{ax}) = h^{\sigma_{ax}}$$

…using some generator h of group G for which no one knows $\log_{g\_t}(h)$.


Server X
Server Y
S_{ax}
S_{bx}
S_{cx}
S_{ay}
S_{by}
S_{cy}
S_{ax}
S_{ay}
S_{bx}
S_{by}
S_{cx}
S_{cy}
Alice
Bob
Chris

Challenge 3: Proving Correctness

- When Bob submits a ciphertext in a messaging round owned by pseudonym $\mathbf{g^d}$, Bob attaches a proof that:

PoK { $\mathbf{C_{Bob}}$ is a correct encryption of **m=1**
OR
I know the pseudonym secret **d** }

Challenge 3: Proving Correctness

- When Bob submits a ciphertext in a messaging round owned by pseudonym $\mathbf{g^d}$, Bob attaches a proof that:

$$\mathbf{PoK}\left\{ \begin{matrix} \sigma_{bx} \\ \sigma_{by} \\ d \end{matrix} : \left(\mathbf{AND} \begin{matrix} C_{Bob} = g_t^{\sigma_{bx}+\sigma_{by}} \\ S_{bx}S_{by} = h^{\sigma_{bx}+\sigma_{by}} \end{matrix} \right) \mathbf{OR} \left(D = g^d \right) \right\}$$

Challenge 3: Proving Correctness

- When Bob submits a ciphertext in a messaging round owned by pseudonym $\mathbf{g^d}$, Bob attaches a proof that:

$$\mathbf{PoK}\left\{ \sigma_{bx}, \sigma_{by}, d : \left(\mathbf{AND} \begin{array}{l} C_{Bob} = g_t^{\sigma_{bx}+\sigma_{by}} \\ S_{bx}S_{by} = h^{\sigma_{bx}+\sigma_{by}} \end{array} \right) \mathbf{OR} \left(D = g^d \right) \right\}$$

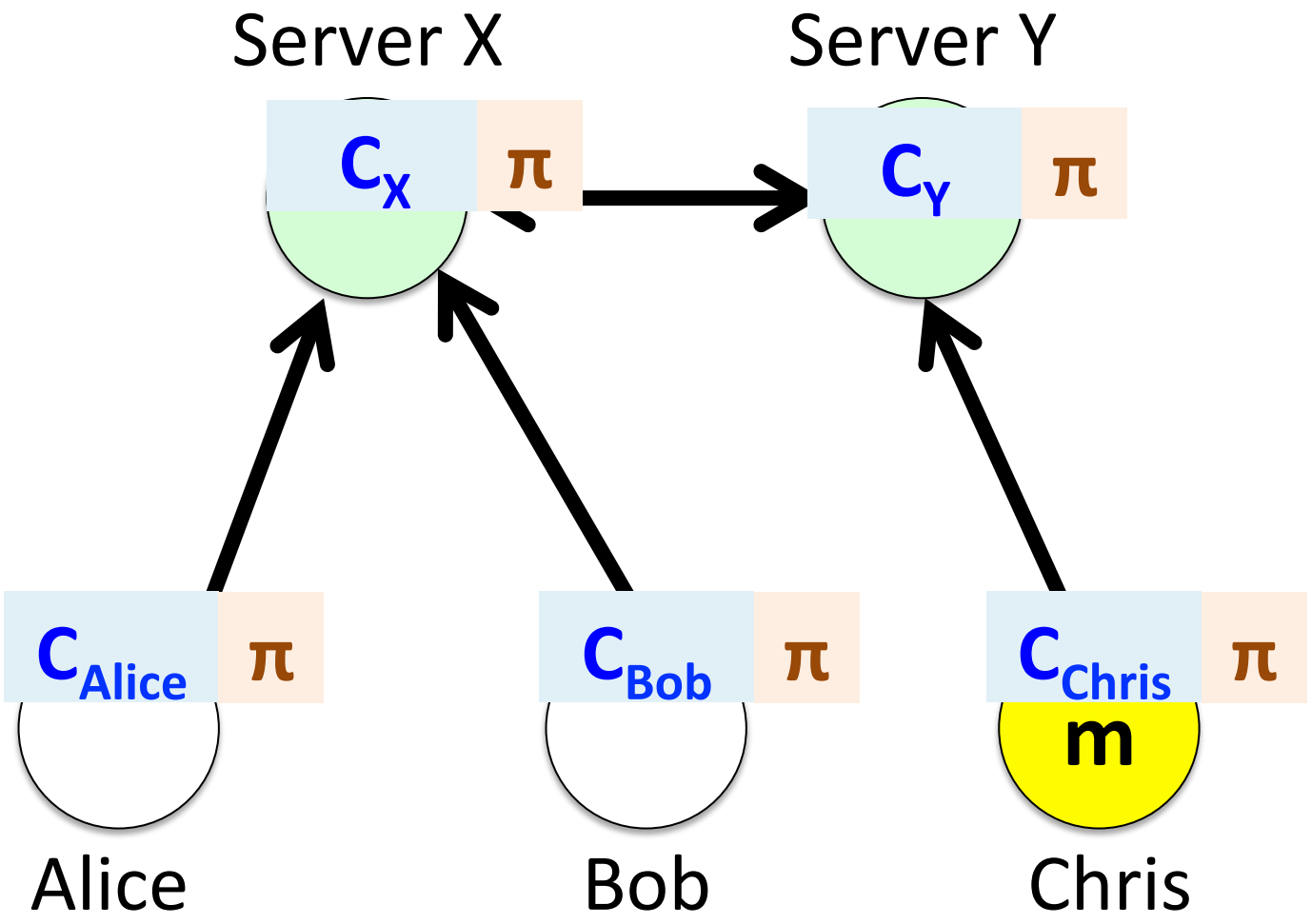
Challenge 3: Proving Correctness

- When Bob submits a ciphertext in a messaging round owned by pseudonym **g** a proof that:

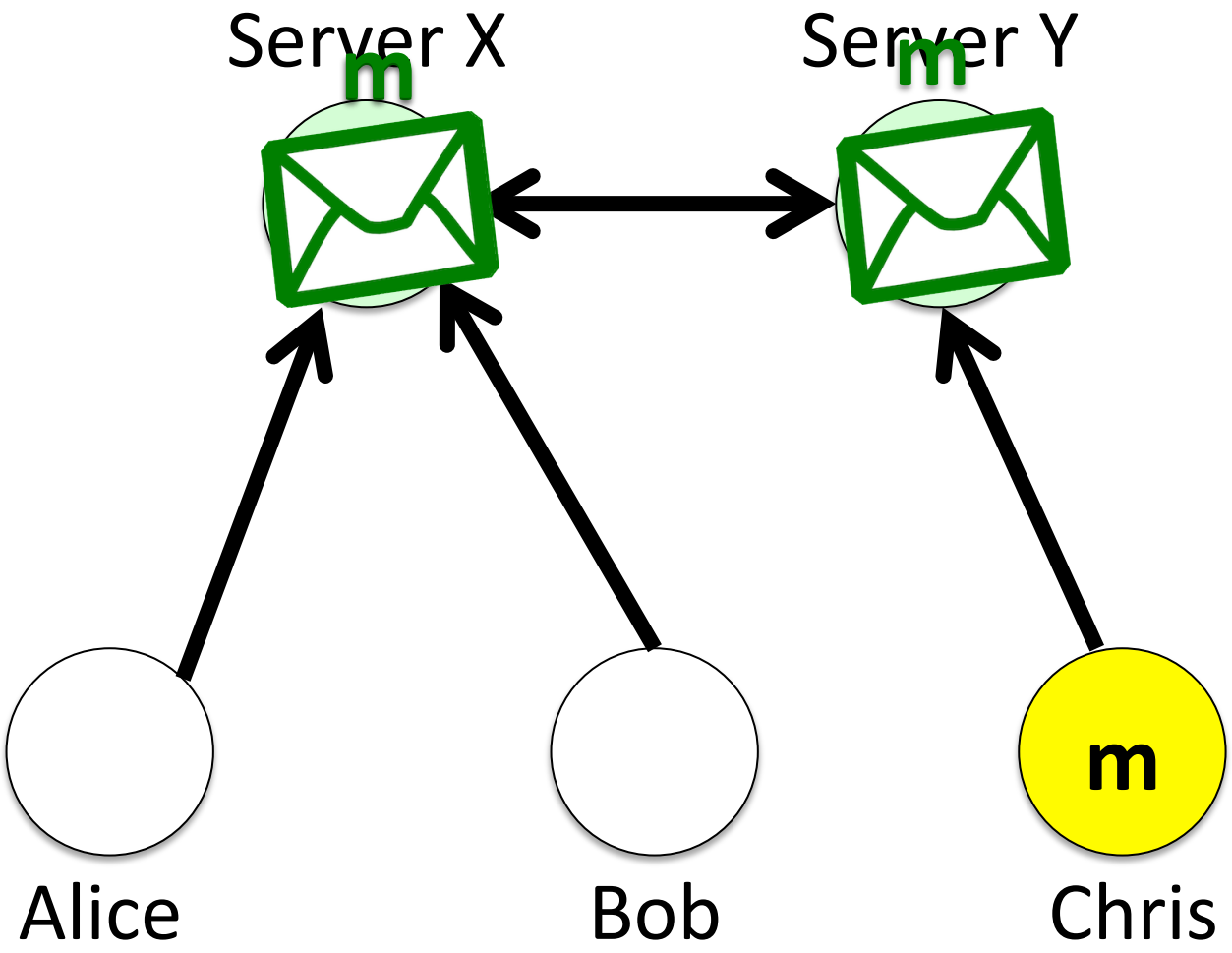
Proof is just a bit-string —can send it over the network, etc

$$\mathbf{PoK}\left\{ \begin{array}{l} \sigma_{bx} \\ \sigma_{by} \\ d \end{array} : \left(\mathbf{AND} \begin{array}{l} C_{Bob} = g_t^{\sigma_{bx}+\sigma_{by}} \\ S_{bx}S_{by} = h^{\sigma_{bx}+\sigma_{by}} \end{array} \right) \mathbf{OR} \; D = g^d \right\}$$

Challenge 3: Proving Correctness



Challenge 3: Proving Correctness



Outline

- Background and Motivation
- **Verdict**
  - Design Challenges
  - **Optimizations**
- Evaluation
- Conclusion

Optimizations

1. Long messages
2. “Lazy” proof verification
3. Hybrid Dissent+Verdict DC-net

Optimizations

1. **Long messages**
2. "Lazy" proof verification
3. Hybrid Dissent+Verdict DC-net

Long Messages

- ssages

We use a public hash function to pick these generators

Golle-Juels scheme required a pairing to achieve a similar effect

$C_{A,1} = g_1^{\sigma_{ax}+\sigma_{ay}}$

$C_{A,2} = g_2^{\sigma_{ax}+\sigma_{ay}}$

... = ...

$C_{A,L} = g_L^{\sigma_{ax}+\sigma_{ay}}$

Po

d

$S_{ax}S_{ay} = h^{\sigma_{ax}+\sigma_{ay}}$

OR $D = g^d$

Optimization 1: Long Messages

Proof length is linear in the number of variables, so we get O(1)-length proofs

$$\mathbf{PoK}\left\{ \sigma_{ax}, \sigma_{ay}, d : \left(\text{AND} \quad \ldots = \ldots\right) \right\}$$

$$C_{A,L} = g_L^{\sigma_{ax}+\sigma_{ay}}$$

$$S_{ax}S_{ay} = h^{\sigma_{ax}+\sigma_{ay}}$$

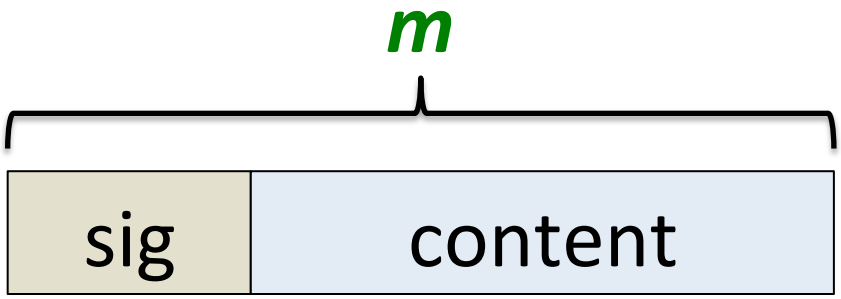
$$\text{OR} \quad D = g^d$$

Optimizations

1. Long messages
2. **"Lazy" proof verification**
3. Hybrid Dissent+Verdict DC-net

Optimization 2: "Lazy" Verification

- Checking proofs is expensive
- Servers defer checking proofs until **after a disruption occurs**



- Anonymous sender signs content with **pseudonym secret key**
- If sig check fails, servers know that disruption has occurred—then they check proofs

Optimizations

1. Long messages
2. “Lazy” proof verification
3. **Hybrid Dissent+Verdict DC-net**

Optimization 3: Hybrid DC-net

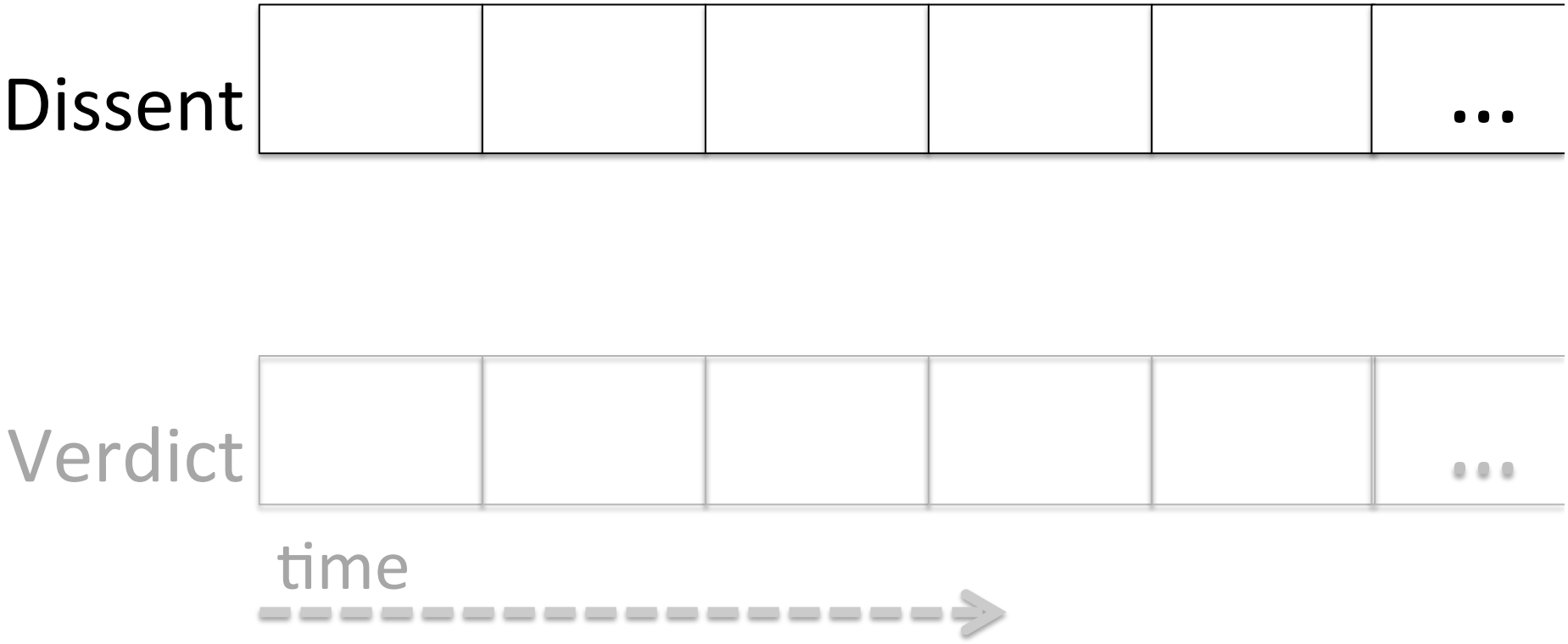
Verdict: heavy pub key crypto

Dissent/DC-nets: AES + XORs

- Recall: After a disruption in Dissent, the anonymous sender broadcasts an "accusation" using a **verifiable shuffle protocol**
  - Participants use the accusation to trace the disruptor
  - Over **99% of the "blame" process is spent in shuffle**
- **Idea**: Use Verdict to broadcast Dissent's anonymous accusations → *hybrid* DC-net

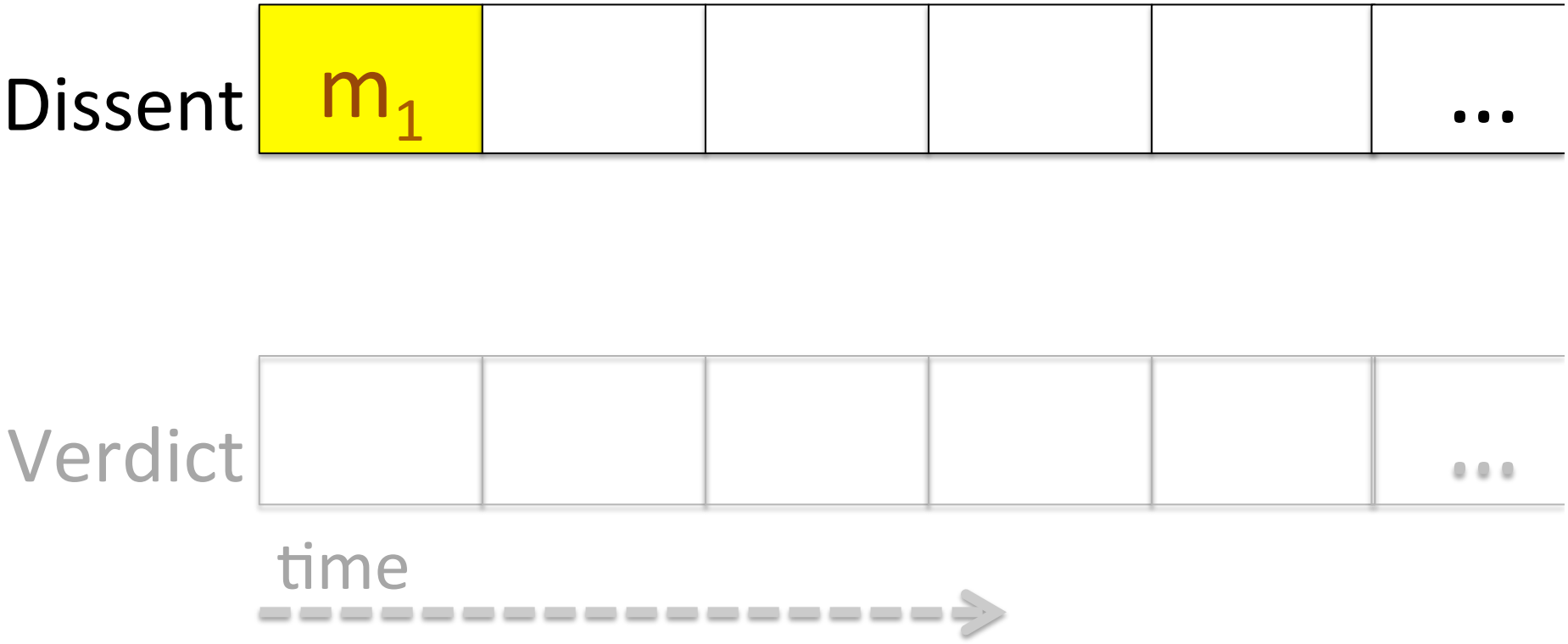
Optimization 3: Hybrid DC-net

- Participants set up parallel Dissent and Verdict communication sessions



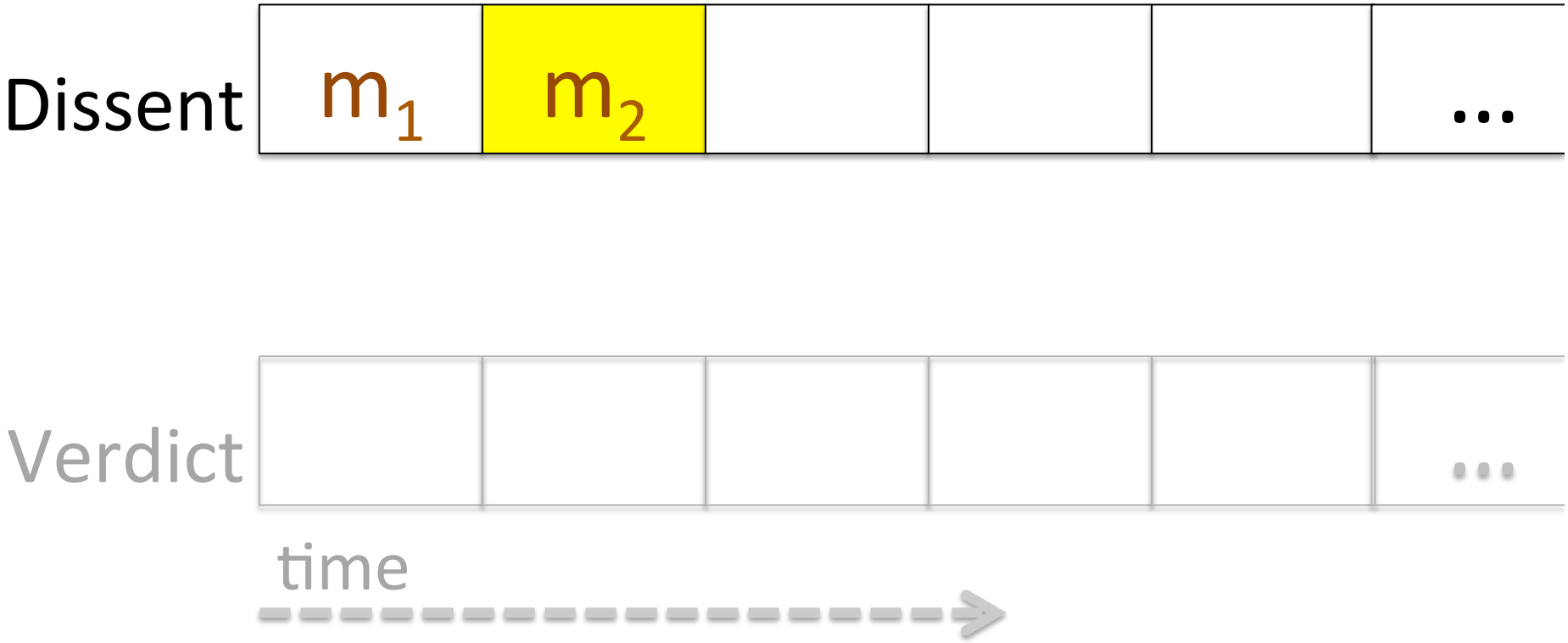
Optimization 3: Hybrid DC-net

- Participants set up parallel Dissent and Verdict communication sessions



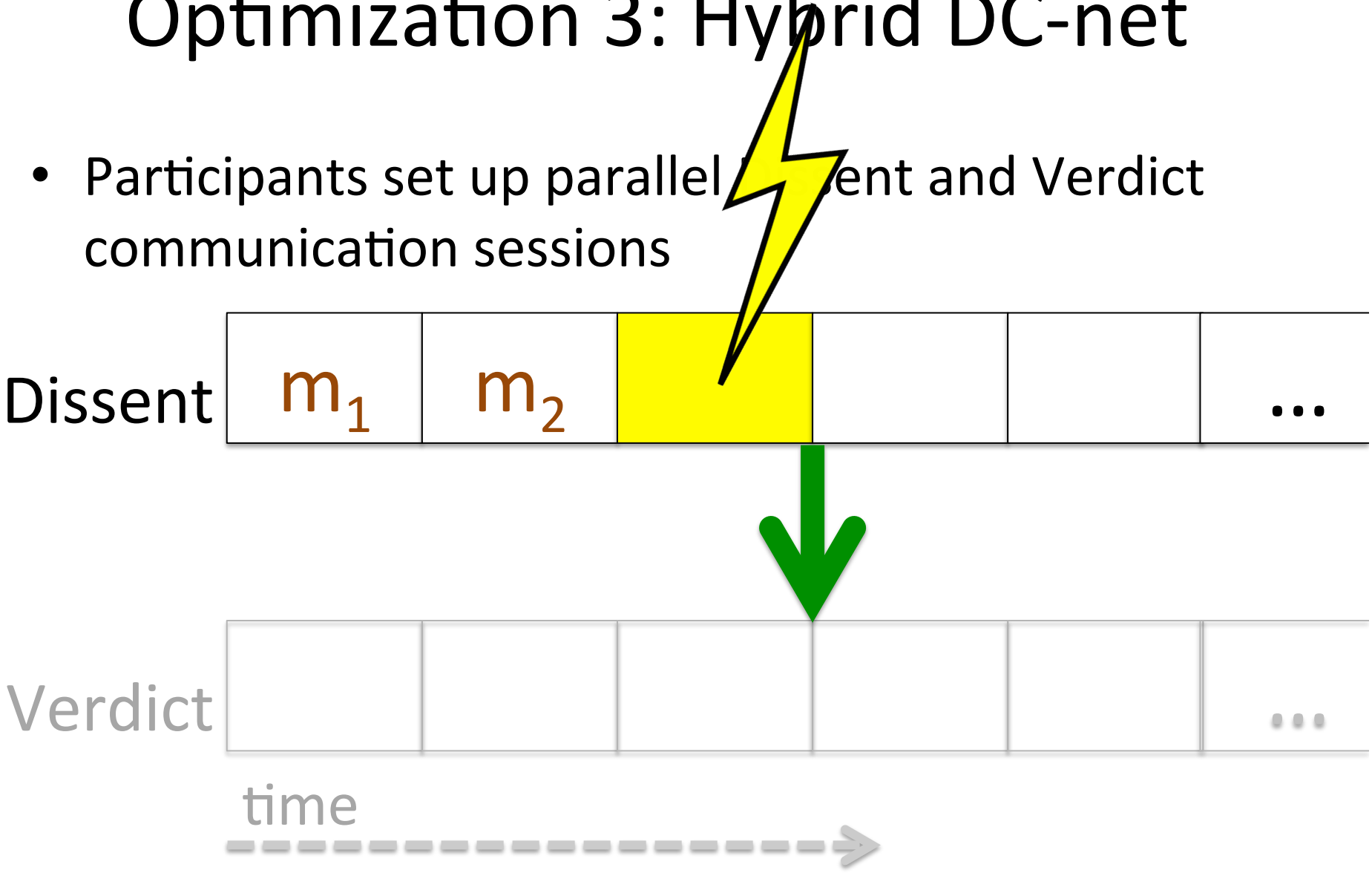
Optimization 3: Hybrid DC-net

- Participants set up parallel Dissent and Verdict communication sessions



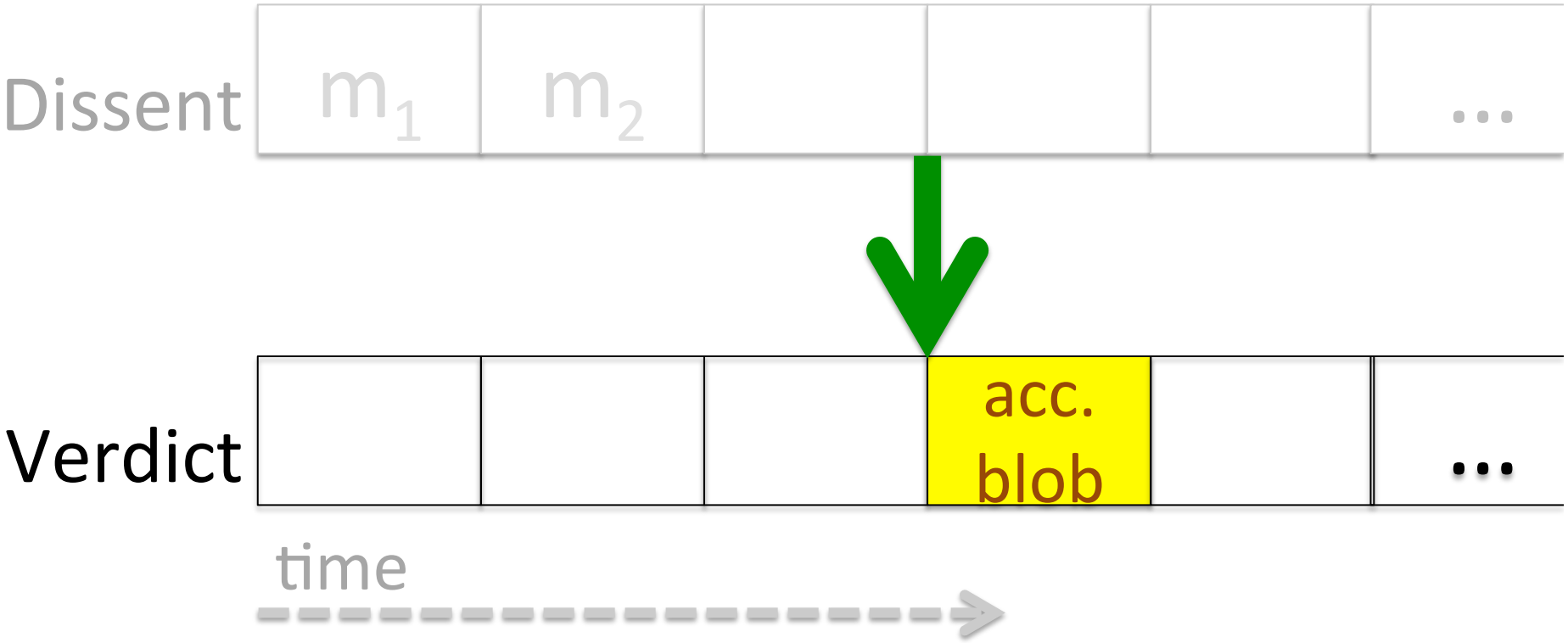
Optimization 3: Hybrid DC-net

- Participants set up parallel Dissent and Verdict communication sessions



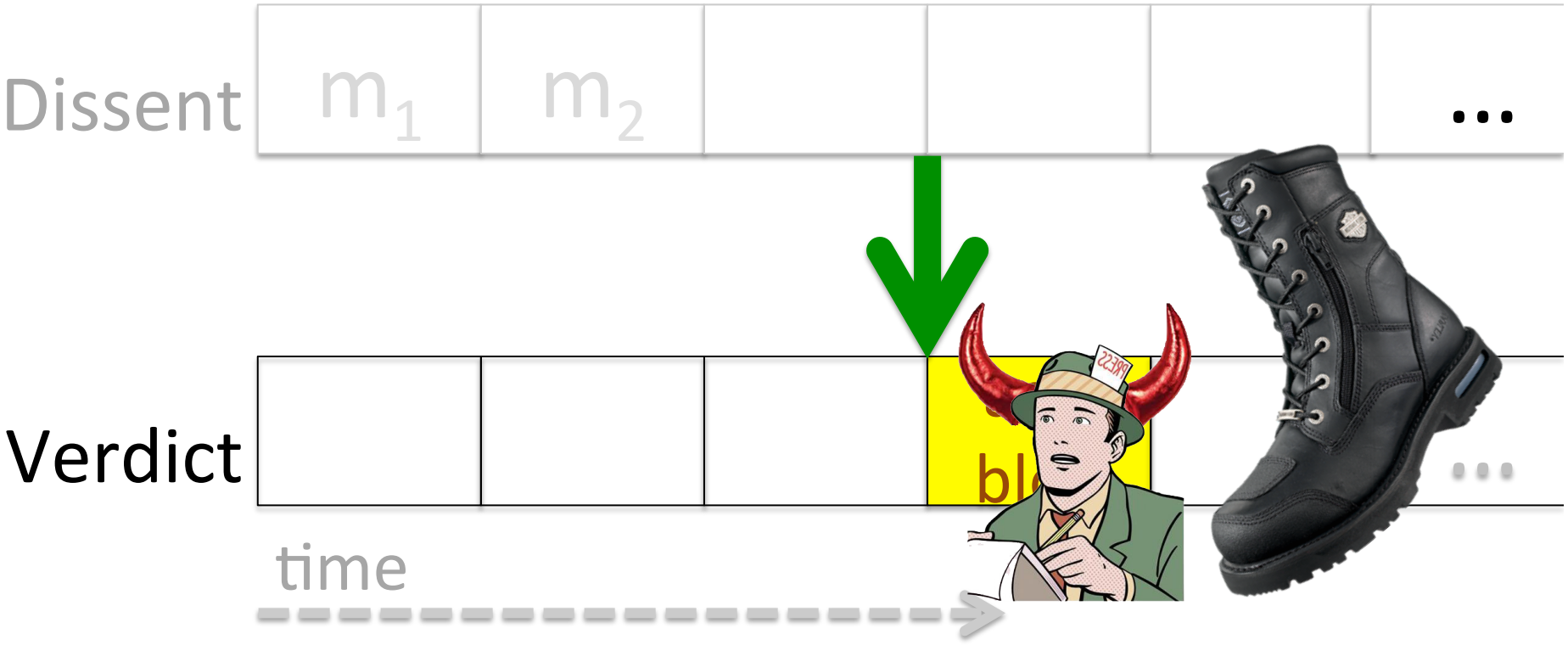
Optimization 3: Hybrid DC-net

- Participants set up parallel Dissent and Verdict communication sessions



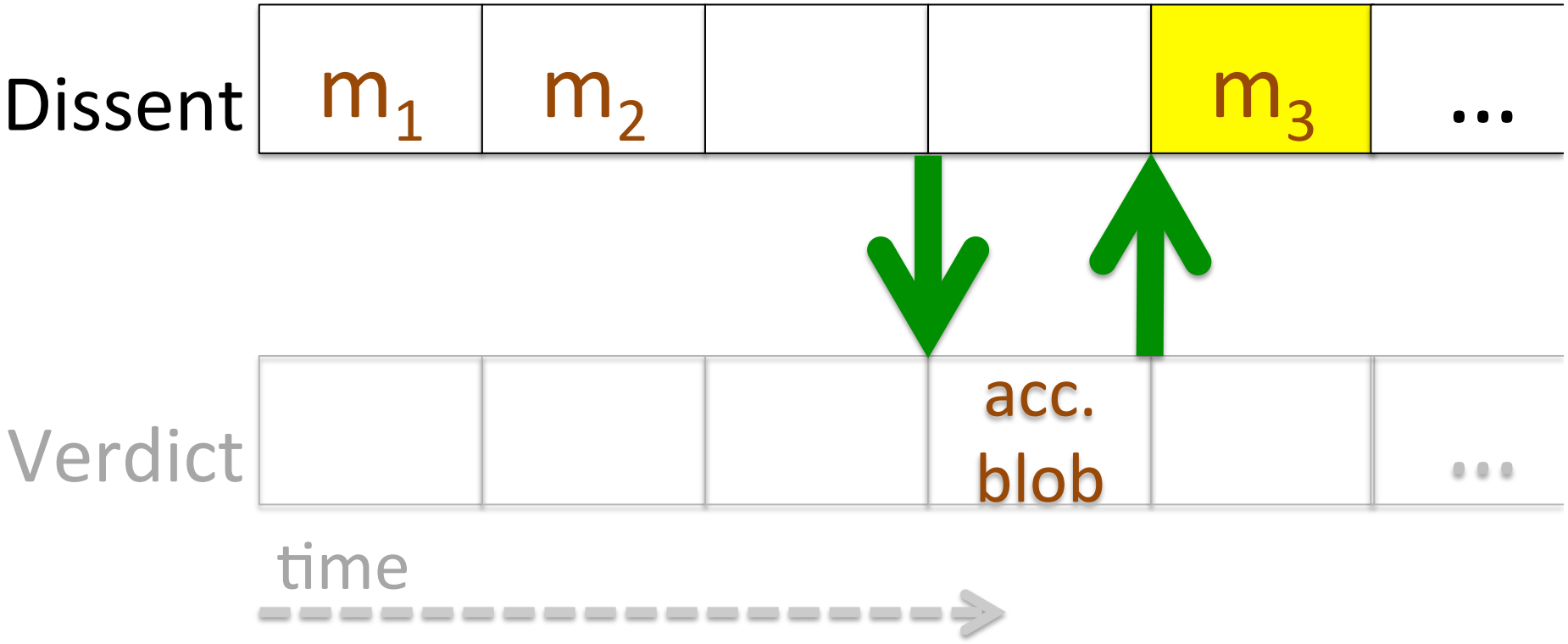
Optimization 3: Hybrid DC-net

- Participants set up parallel Dissent and Verdict communication sessions



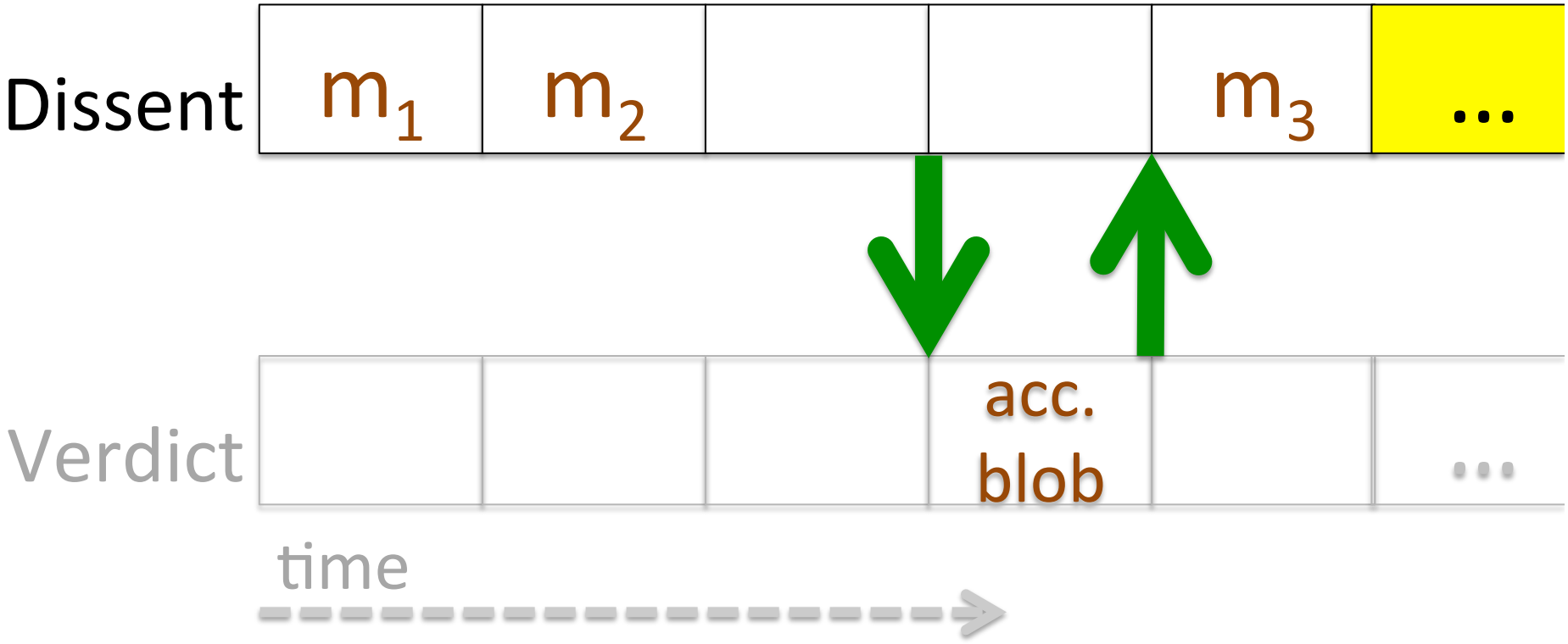
Optimization 3: Hybrid DC-net

- Participants set up parallel Dissent and Verdict communication sessions



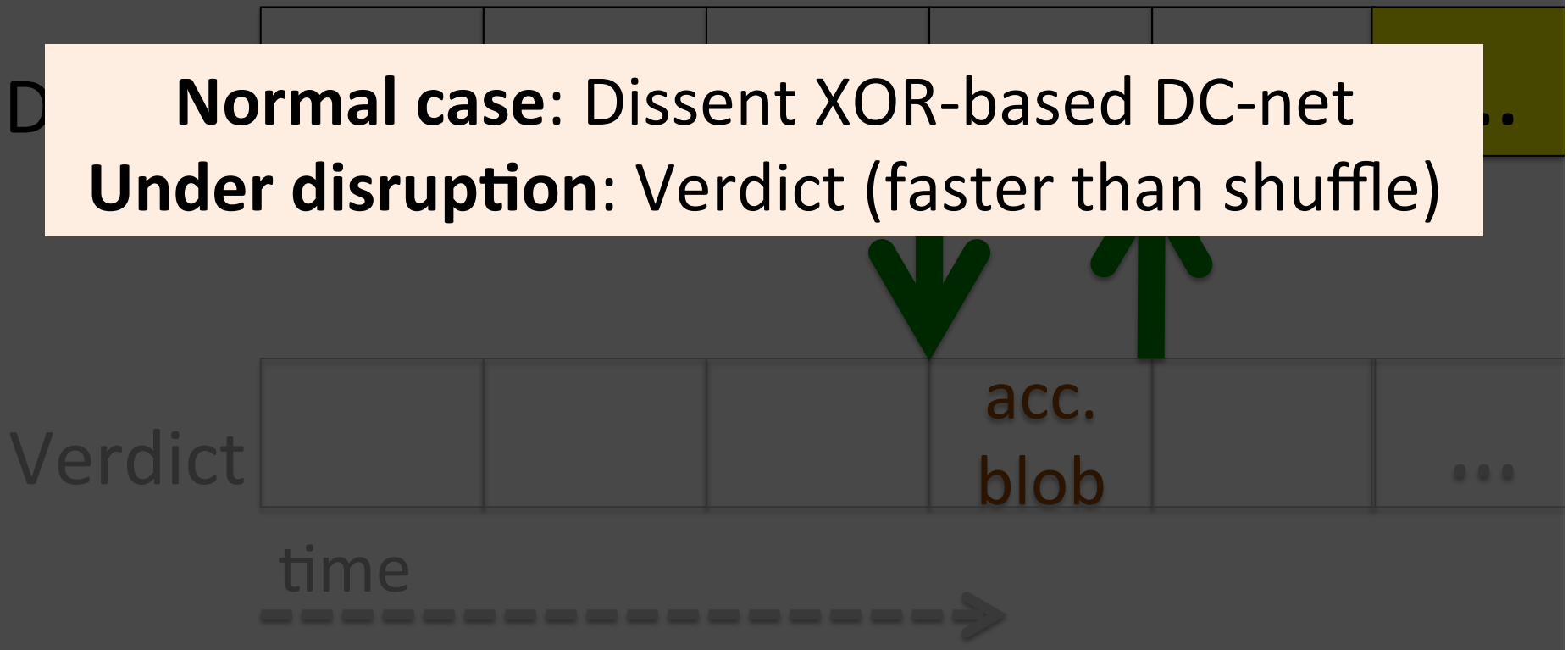
Optimization 3: Hybrid DC-net

- Participants set up parallel Dissent and Verdict communication sessions



Optimization 3: Hybrid DC-net

- Participants set up parallel Dissent and Verdict communication sessions



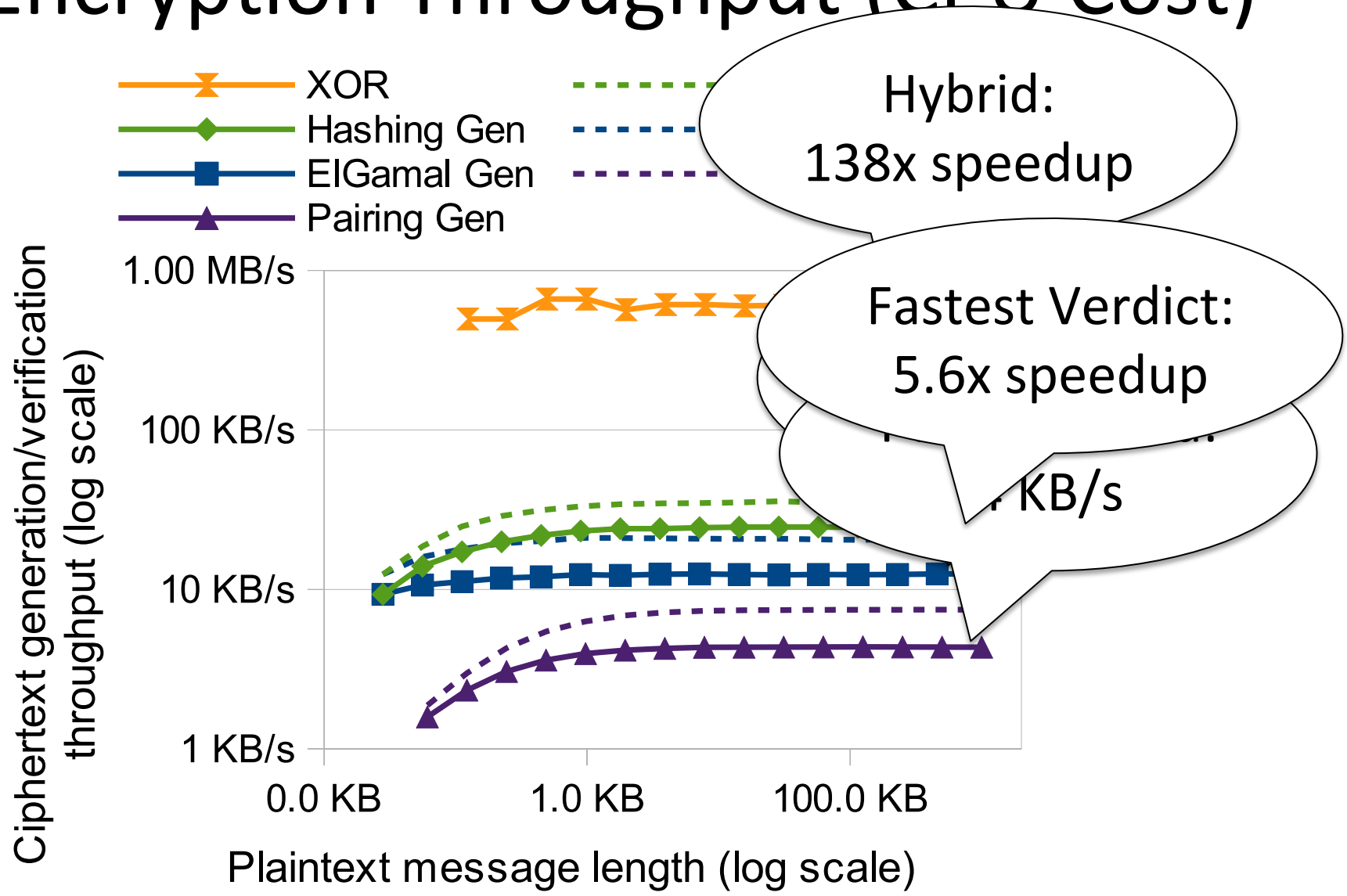
Outline

- Background and Motivation
- Verdict
  - Design Challenges
  - Optimizations
- **Evaluation**
- Conclusion

Implementation

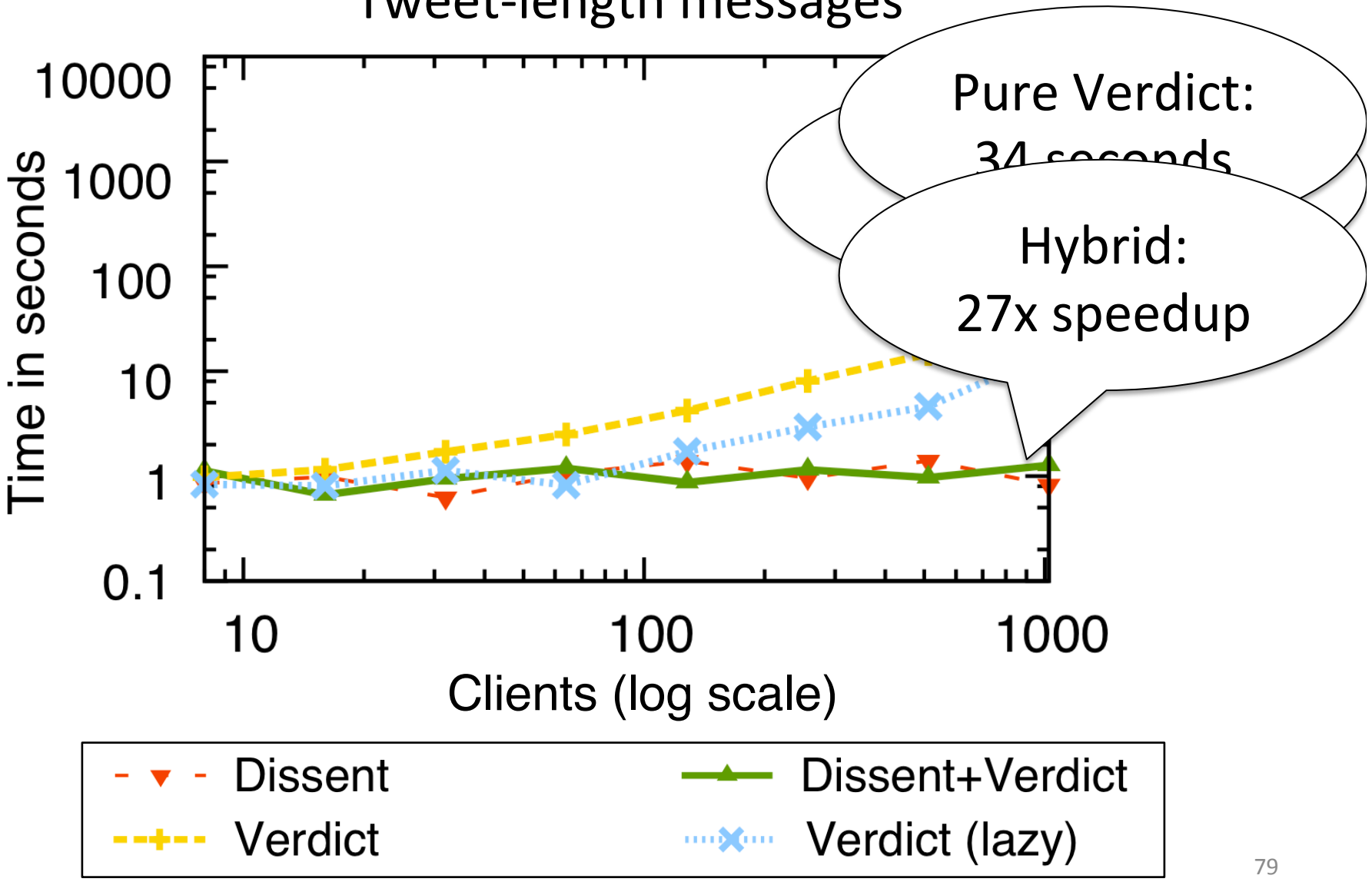
- Implemented in C++ as an extension to Dissent
- Cryptographic primitives
  - OpenSSL, Crypto++, and Botan libraries
  - 256-bit NIST elliptic curve group
- Used the DeterLab testbed
  - Physical nodes: 8 servers, 128 clients
  - Ran many client processes per machine to simulate up to 1024 clients
- Source code: https://github.com/DeDis/Dissent

Encryption Throughput (CPU Cost)

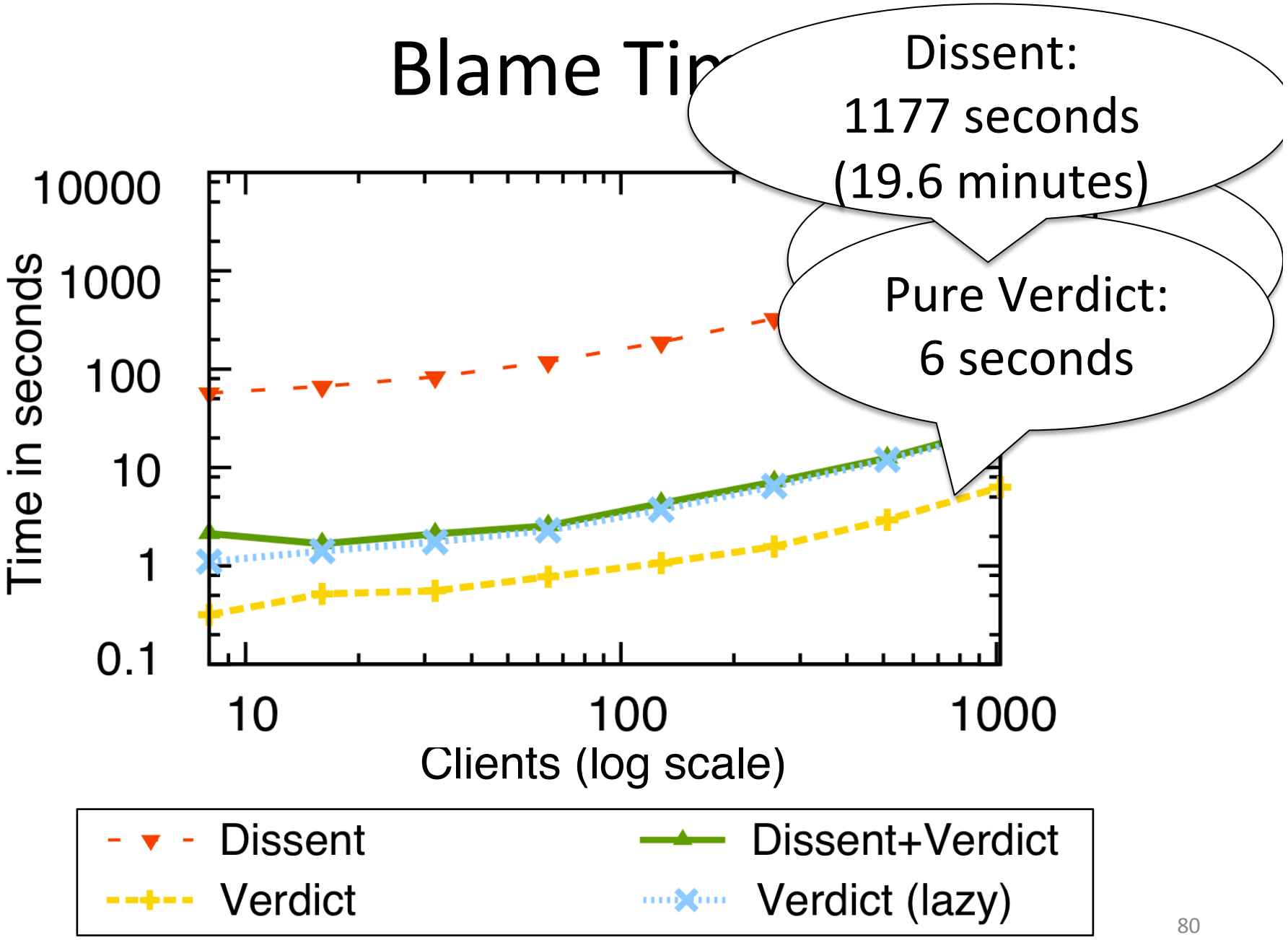


Messaging Latency

Tweet-length messages



Blame Tir


Dissent:
1177 seconds
(19.6 minutes)
Pure Verdict:
6 seconds
Time in seconds
10000
1000
100
10
1
0.1
10
100
1000
Clients (log scale)
Dissent
Dissent+Verdict
Verdict
Verdict (lazy)

Outline

- Background and Motivation
- Verdict
  - Design Challenges
  - Optimizations
- Evaluation
- **Conclusion**

Details in the Paper

- Messaging protocol
  - Handling equivocation, dropped messages, etc.
- Proof constructions
  - The paper describes three variants
  - Implementation details
- Handling server failure
- Handling client churn

Conclusion

First practical verifiable DC-nets scheme

- Introduces two new verifiable DC-nets constructions
- Reduces the cost of finding DC-net disruptors by **two orders of magnitude**
- By reducing the cost of disruption, Verdict brings strong **traffic-analysis-resistant anonymity** closer to practicality

Acknowledgements

Thanks to:

- the anonymous reviewers,
- our shepherd, Micah Sherr,
- the DeterLab staff,
- Aaron Johnson, Ewa Syta, Michael J. Fischer, Michael Z. Lee, Michael “Fitz” Nowlan, Ramki Gummadi, and
- all of you for listening.

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

Shameless plug: **The Dissent project is hiring!**