CAFE: Causal Algorithms for Fault-tolerant Environments

Sinchan Sengupta

Gestion de Données Distribuées (GDD), LS2N.
Nantes Université, France.

PILOT Online Seminar
PEPR eNSEMBLE
16 October 2025

Table of Contents

CAFE : Causal Algorithms for Fault-tolerant Environments

Making Collaborative Software More Secure

Secure Collaborative Editor ?

Multi-User Text Editor (MUTE)

- MUTE [1] is a P2P collaborative editor.

MUTE's most important properties

- **Decentralization :** No central authority.
  - Uses a modern P2P Network layer between browser on WebRTC.
- **Eventual consistency :** All replicas converge towards an identical state.
  - Based on LogootSplit [2] CRDT algorithm.
- **Security :** Protect from unwarranted accesses.
  - Deploys an *n*-party Diffie-Hellman key exchange agreement.

[1] *MUTE : A peer-to-peer web-based real-time collaborative editor.* Nicolas, M., Elvinger, V., Oster, G., Ignat, C. L., & Charoy, F. : ECSCW 2017 (Vol. 1, No. 3, pp. 1-4).

[2] *Supporting adaptable granularity of changes for massive-scale collaborative editing.* André, L., Martin, S., Oster, G., & Ignat, C. L. : CollaborateCom 2013 (pp. 50-59).

Use Case : Ranking Candidates

Jack

Carol
Bob

John

Carol
Bob

Pierre

Carol
Bob

Let us rank Alice...

Use Case : Ranking Candidates

Jack

Alice
Carol
Bob

John

Alice
Carol
Bob

Alice is better than Carol

messages :

Insert(“Alice”, 1)

Pierre

Alice
Carol
Bob

Use Case : Ranking Candidates

Jack

Carol
Bob

John

Carol
Bob

No she is not!

messages :
Insert(“Alice”, 1)
Delete(1)

Pierre

Carol
Bob

Use Case : Ranking Candidates

Jack

Carol
Bob

John

Carol
Bob

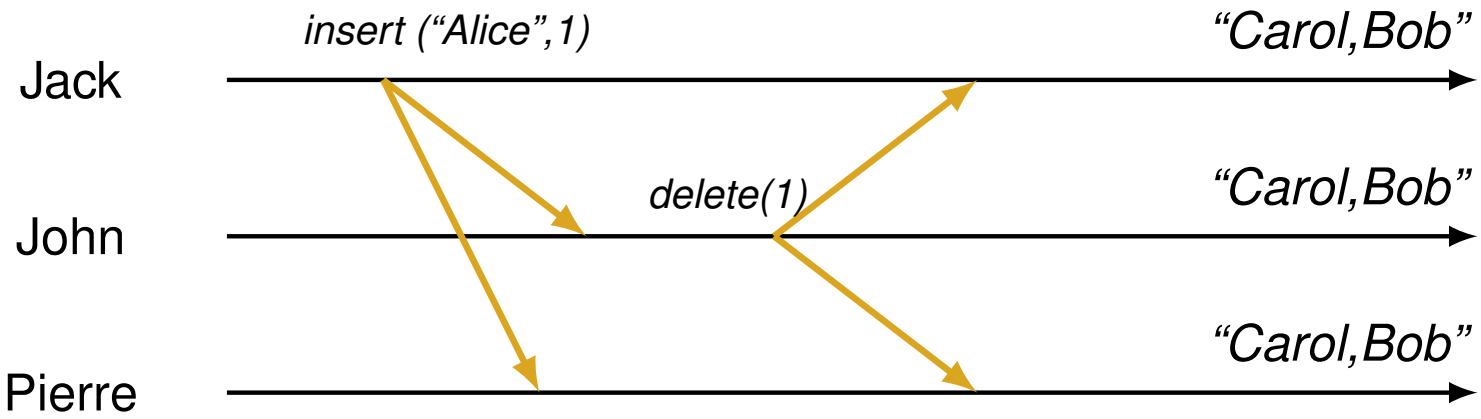
messages :
Insert(“Alice”, 1)
Delete(1)

Pierre

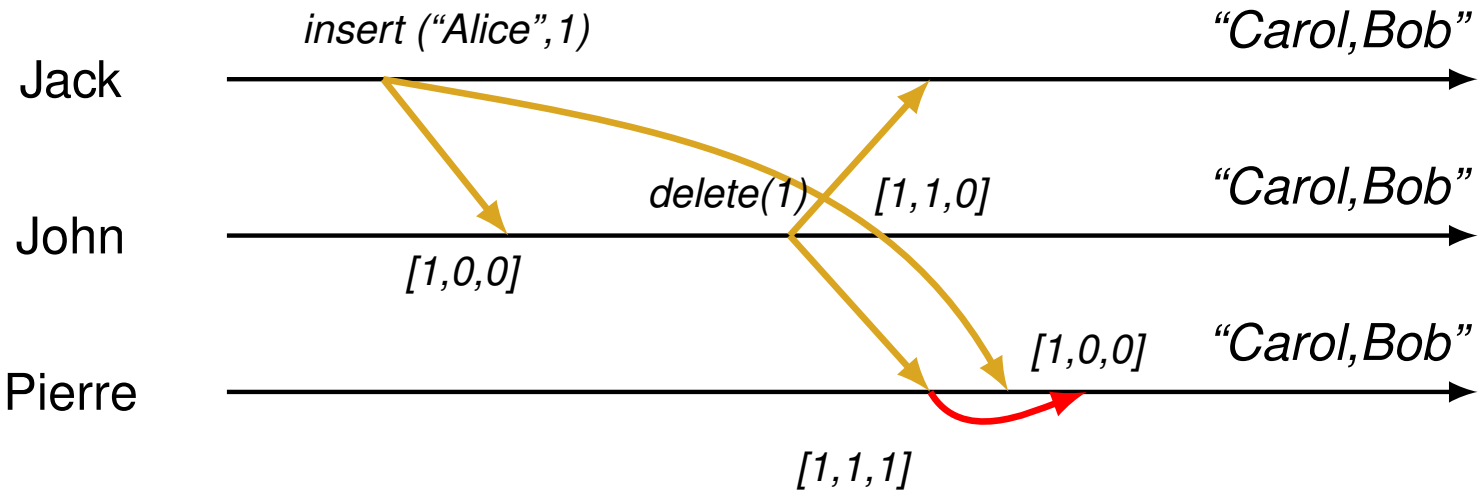
Carol
Bob

So Carol is the best!

Causality in MUTE



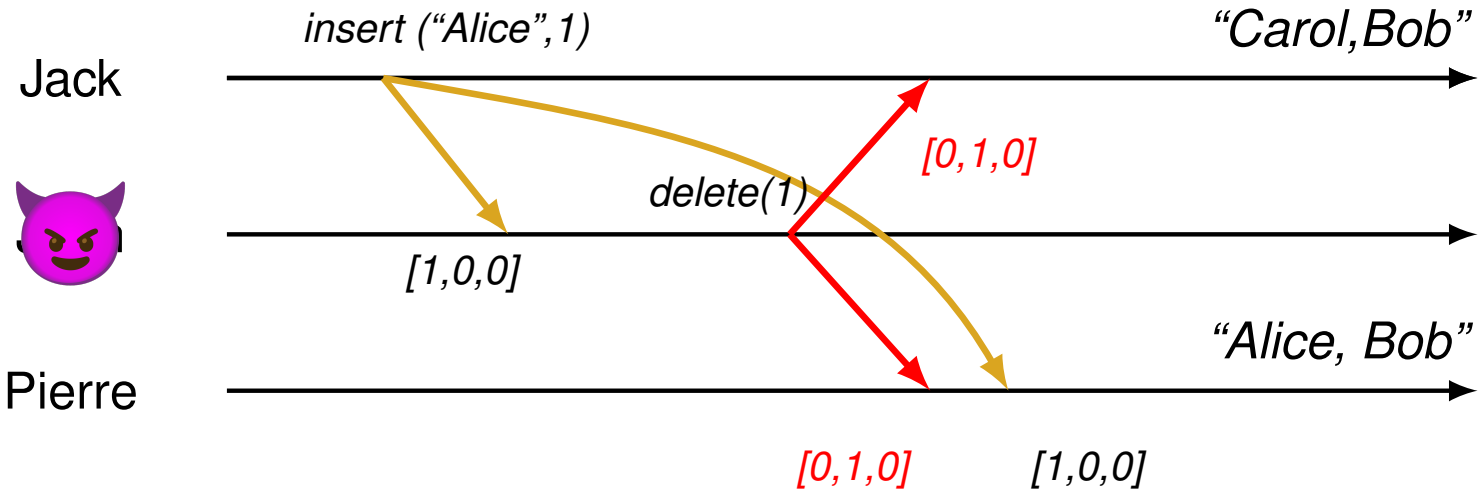
Causality in MUTE



- LogootSplit assumes a **Causal Broadcast** mechanism in the optimistic CRDT implementation.
- The causal dissemination is performed through *vector clocks* [1].

[1] *Logoot : A scalable optimistic replication algorithm for collaborative editing on p2p networks.* Weiss, S., Urso, P. and Molli, P. : ICDCS 2009 (pp. 404-412).

Attack on MUTE



- Nothing in the state-of-the-art to handle this case.
- Only the correct peers are assumed to transmit causality [1].

[1] *Byzantine-tolerant causal broadcast. Auvolat A, Frey D, Raynal M, Taïani F. : Theoretical Computer Science 885 (2021) : 55-68.*

Problem Statement

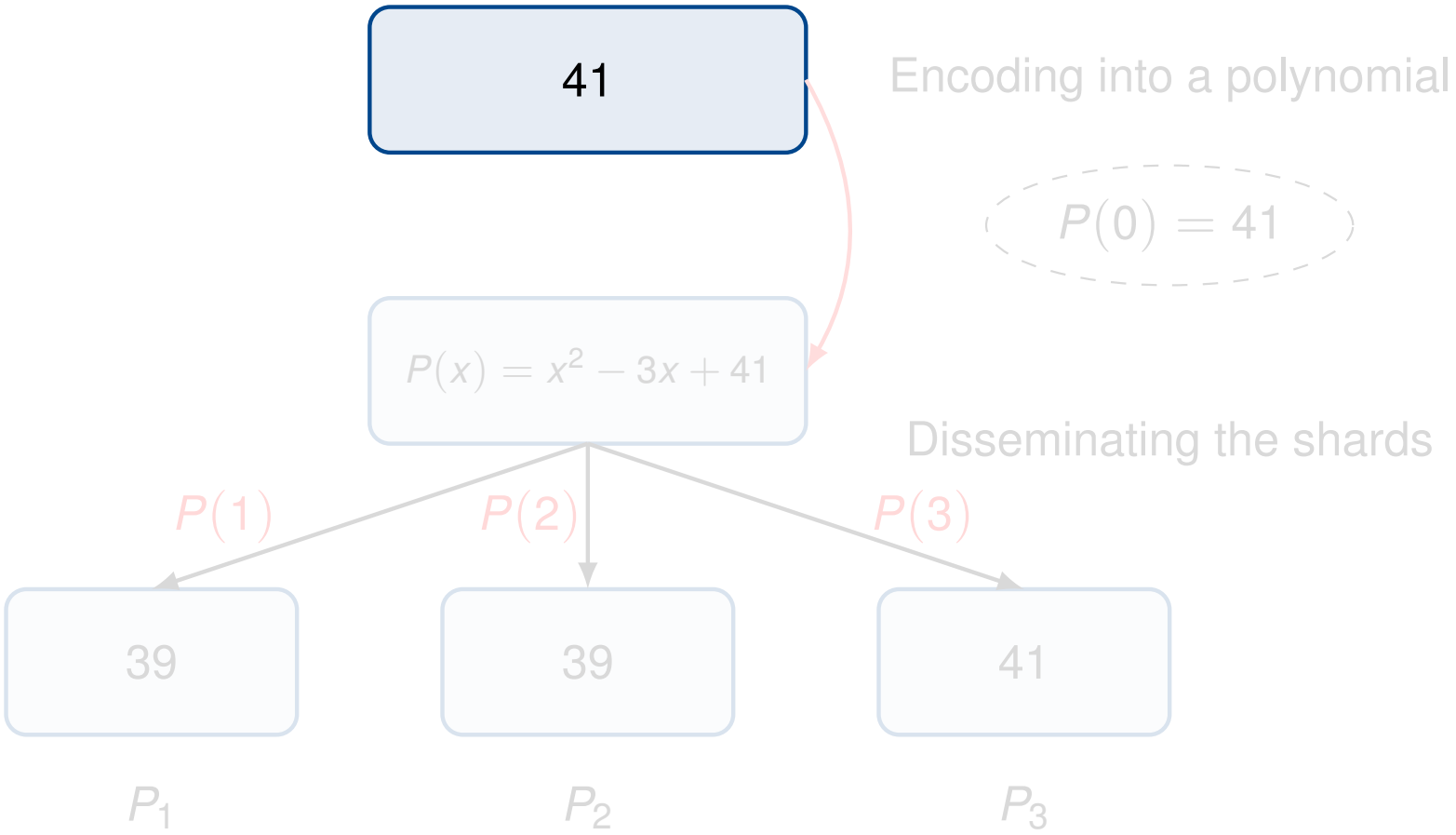
Project Objectives

- Propose a **Byzantine-tolerant Causal Broadcast** algorithm.
- Show that the integration of the above algorithm makes MUTE Byzantine fault-tolerant.

Related Problems : Insider Trading Reiter, M. K., & Birman, K. P. (1994). *How to securely replicate services*. ACM Transactions on Programming Languages and Systems (TOPLAS), 16(3), 986-1009.

Essentials for Broadcasting : Shamir's Secret Sharing

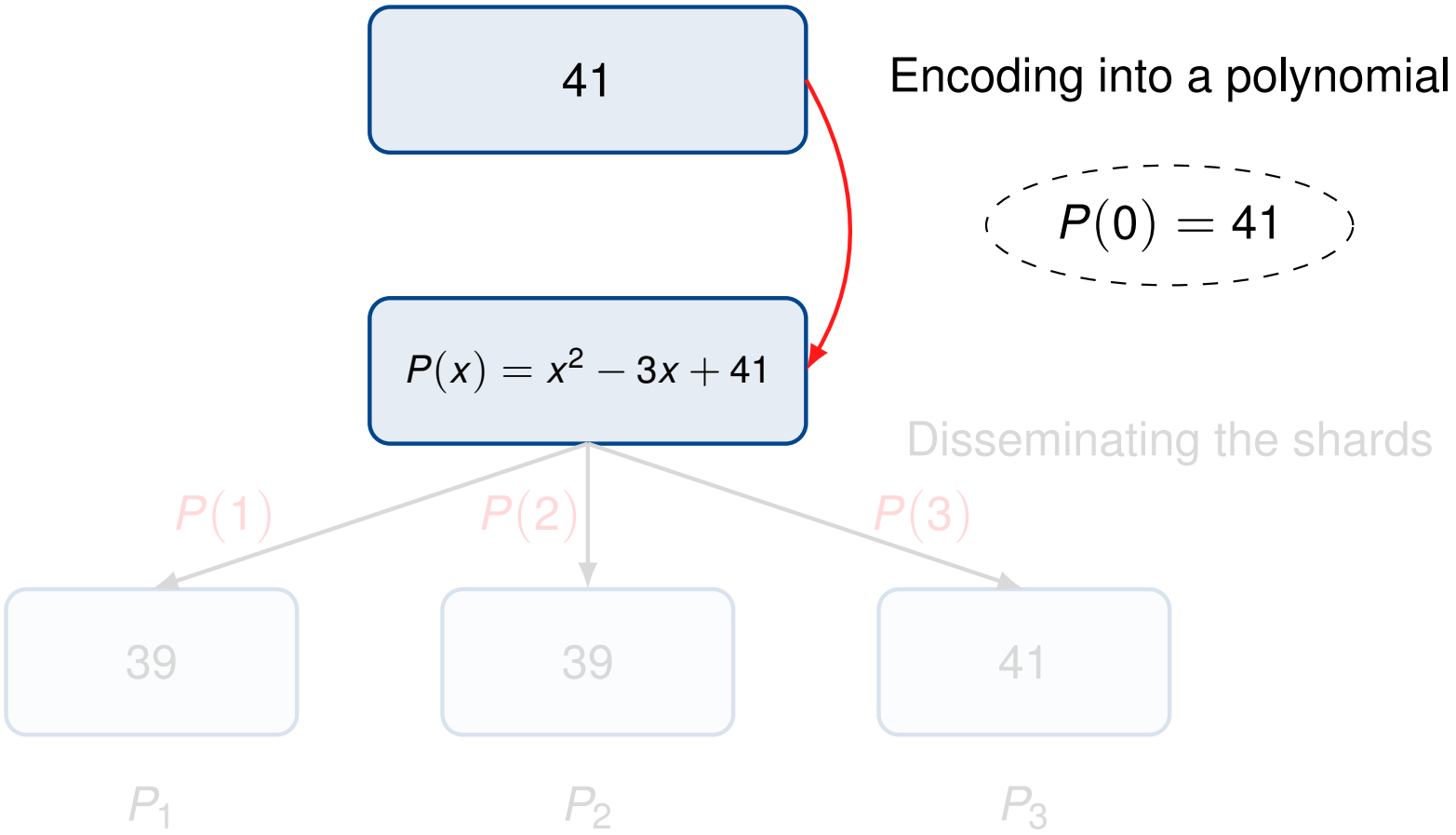
How to do a broadcast ?



Shamir79 Adi Shamir. *How to share a secret.* Communications of the ACM 22, no. 11 (1979) :

Essentials for Broadcasting : Shamir's Secret Sharing

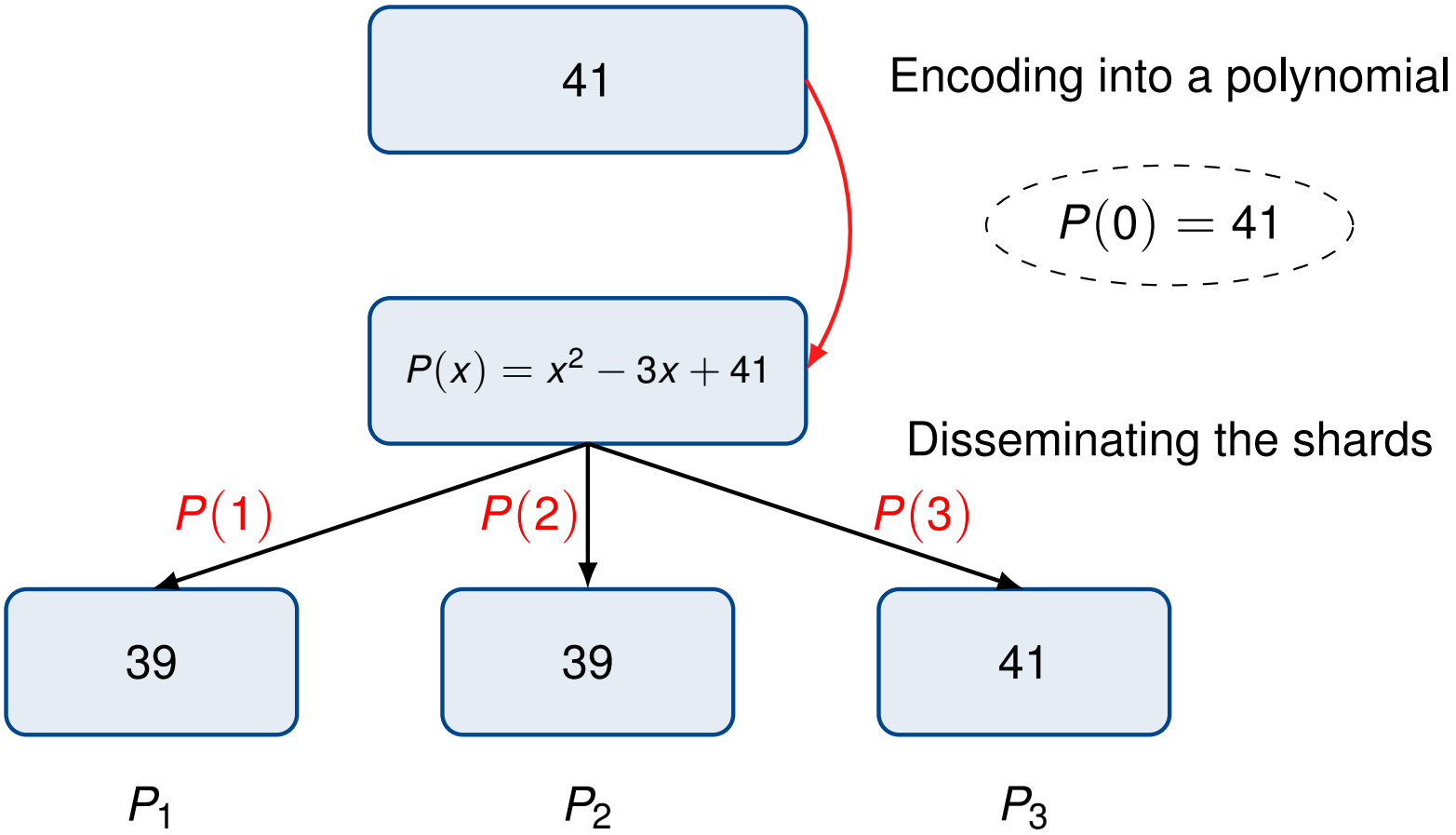
How to do a broadcast ?



Shamir79 Adi Shamir. *How to share a secret.* Communications of the ACM 22, no. 11 (1979) :

Essentials for Broadcasting : Shamir's Secret Sharing

How to do a broadcast ?



Shamir79 Adi Shamir. *How to share a secret.* Communications of the ACM 22, no. 11 (1979) :

Shamir's Secret Sharing : Practical Usage

- Many open source implementations are available for deploying Shamir's Secret Sharing.
- Transform a string using an encoding scheme like UTF-8.

Distributing a Secret

Command :

```
shamir distribute string "<secret string>" -n <number of shares> -k <minimum shares>
```

Example :

```
shamir distribute string "This is a secret." -n=5 -k=3
```

Shamir49pctber https://github.com/49pctber/shamir

Shamir's Secret Sharing : Practical Usage

Example Output (5 Shares)

```
shamir-ZDUIQPAX-1-5dxDU36YEiGQmcX3tCfJlHg
shamir-ZDUIQPAX-2-wzJianU+jsC3xr1O5qPj6W8
shamir-ZDUIQPAX-3-coZISivP78FGfwvcMfZPCTk
shamir-ZDUIQPAX-4-4GAeOAldaFJ07zR7rDxfTj8
shamir-ZDUIQPAX-5-UdQOGFesCVOFVoLpe2nzrmk
```

Explanation : Each share encodes a unique point on a random polynomial. The prefix `ZDUIQPAX` identifies the same secret across all shares.

Shamir49pctber https://github.com/49pctber/shamir

Shamir's Secret Sharing : Practical Usage

Reconstructing the Secret

Command :

```
shamir reconstruct string "<share 1>" "<share 2>" "<share 3>"
```

Example :

```
shamir reconstruct string "shamir-ZDUIQPAX-4-4GAeOAldaFJ07zR7rDxfTj8"
"shamir-ZDUIQPAX-5-UdQ0GFesCVOFVoLpe2nzrmk"
"shamir-ZDUIQPAX-1-5dxDU36YEiGQmcX3tCfJ1Hg"
```

Output :

```
ZDUIQPAX: This is a secret.
```

The order of shares does not matter.

Shamir49pctber https://github.com/49pctber/shamir

Shamir's Secret Sharing : Practical Usage

When Shares Are Insufficient

Using fewer than k shares (e.g., only 2 of 5) :

```
ZDUIQPAX: @*/V@/1uz
```

Results in **garbage output** — no partial information can be retrieved.

Shamir49pctber https://github.com/49pctber/shamir

Proposed Algorithmic Approach

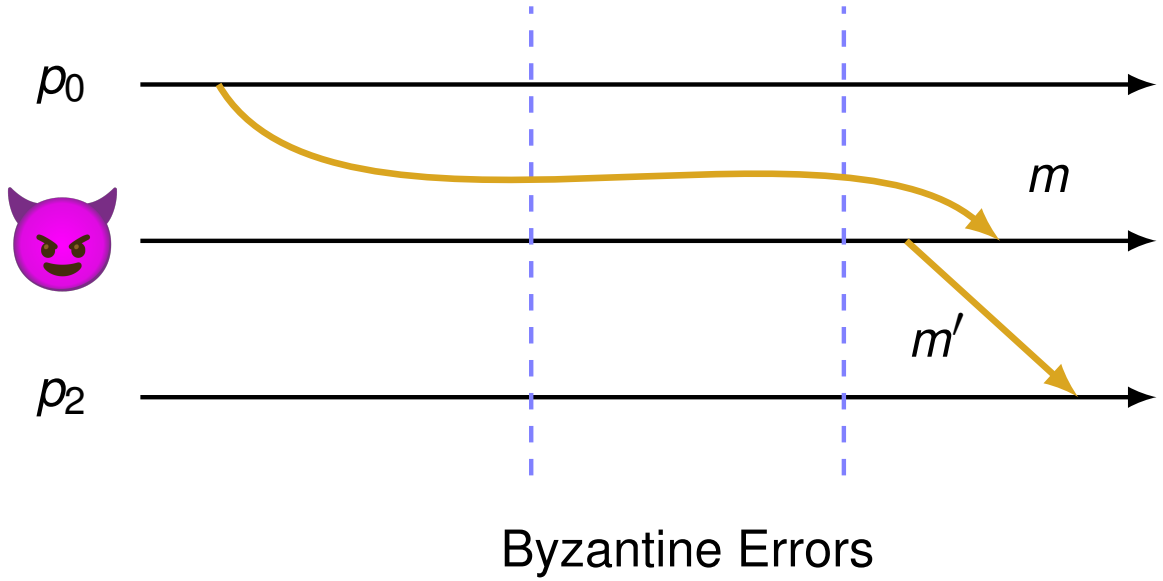
How do we achieve Byzantine Causality ?

- A piece of text is sent over the network is **Causal Broadcast** by a peer.
- Consider the following cases :

Proposed Algorithmic Approach

How do we achieve Byzantine Causality ?

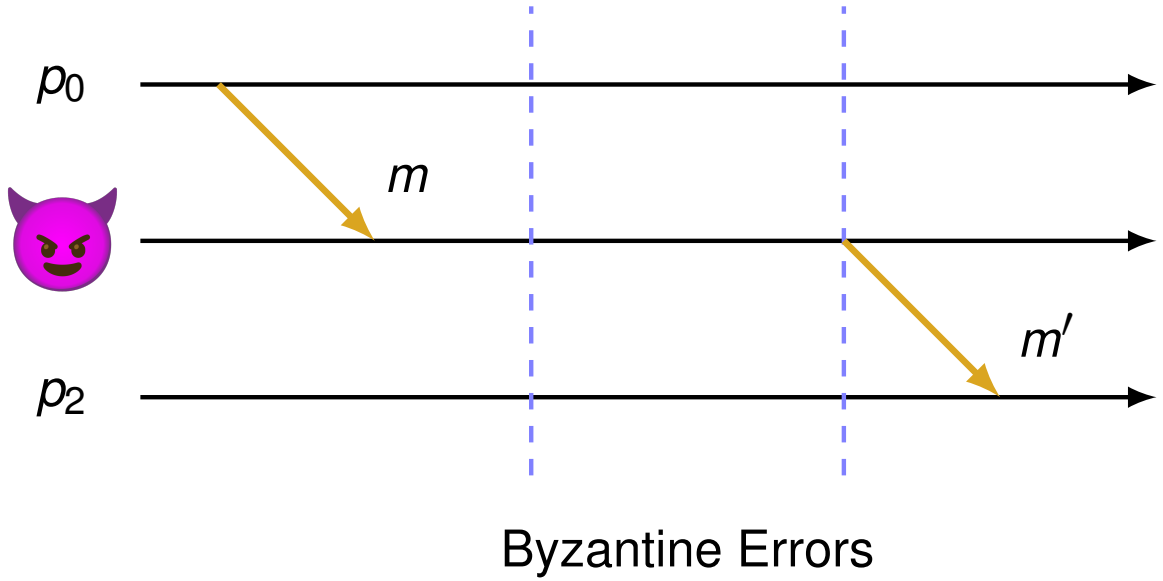
CASE - I : No Causality.



Proposed Algorithmic Approach

How do we achieve Byzantine Causality ?

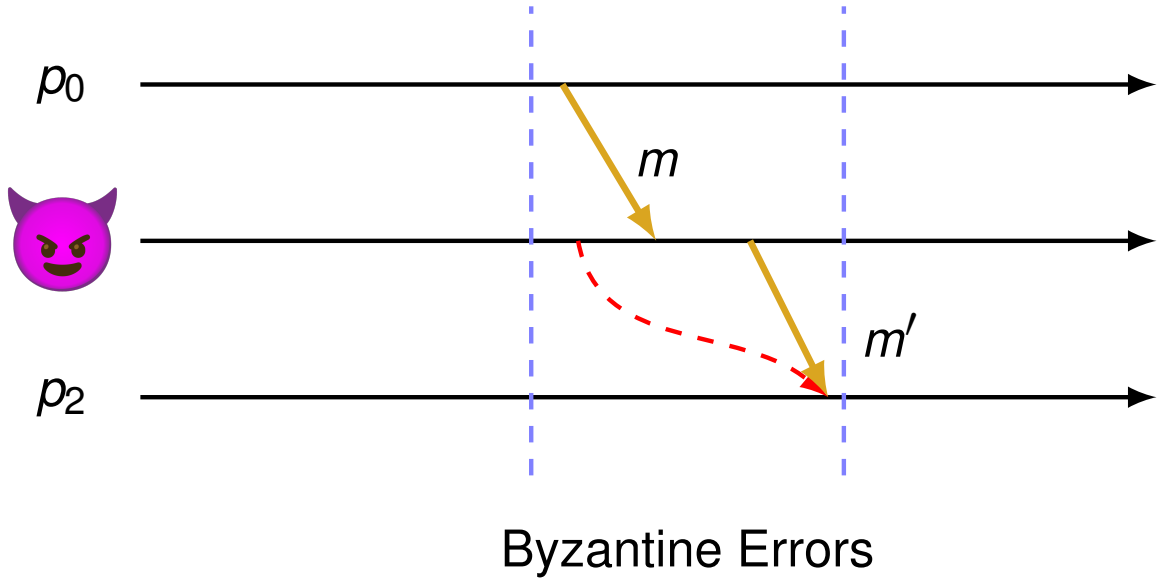
CASE - II : Causality Enforced.



Proposed Algorithmic Approach

How do we achieve Byzantine Causality ?

CASE - III : Byzantine Tampering.



Byzantine Errors

- Corrects must order m' after m even when the Byzantines send the red message.
- Correct peers must be aware of m before Byzantines can modify it.
- They must order Byzantine messages consistently after m.

Experiments

CAFE aims to answer the following questions in relation to the usage of collaborative editors in the presence of Byzantine participants.

Experimental Goals

- **Q1** : How long does it take for users to detect convergence failure due the presence of Byzantine participants manipulating causality in a collaborative editor ?
- **Q2** : How does the Byzantine-tolerant causal broadcast mechanism affect the delay in real-time collaborative editing ?

Experimental Details

Q1 : Can users detect convergence failure in presence of Byzantines ?

Proposed Setup

- Consider a group of n participants (peers).
- Setup a dummy server S. All peers communicate through S, as in a client-server architecture.
  - A sensible assumption, since MUTE needs a server to setup peer clients [1].
- Designate a Byzantine b. Simulate its byzantine behavior by making S re-order (delay or drop) messages from b to one or more peers.
- Observe if the correct peers report convergence or not.

[1] https://gitlab.inria.fr/coast-team/mute/

Experimental Details

Q2 : How does the Byzantine-tolerant causal broadcast mechanism affect the delay in real-time collaborative editing ?

Setup : (from [1])

- Integrate the proposed Byzantine causal broadcast algorithm into MUTE.
- Proceed to measure the delay using a collaborative note-taking task as shown in [1].

[1] *Measurement of Remote Response Delay in Multi-Synchronous Collaborative Editing (Doctoral dissertation, INRIA), 2013* Ignat, C. L., Oster, G., Newman, M., Shalin, V., Charoy, F.

Thank You

Appendix

Measuring Delay : MUTE + Byzantines

Goals :

- Evaluate the performance of MUTE with and without our proposed Byzantine Causal Broadcast (BCB).
- Measure different delay types affecting real-time collaboration.

Measuring Delay : MUTE + Byzantines

Setup :

Participants : 4-6 users collaborating in MUTE.
Test Scenarios :

Scenario	BCB Enabled ?	Artificial Delay
Baseline (No Delay)	No	0 ms
BCB Only	Yes	0 ms
Network Delay	No	200ms, 500ms, 1s
BCB + Network Delay	Yes	200ms, 500ms, 1s

[1] *Measurement of Remote Response Delay in Multi-Synchronous Collaborative Editing (Doctoral dissertation, INRIA), 2013* Ignat, C. L., Oster, G., Newman, M., Shalin, V., Charoy, F.

Measuring Delay : MUTE + Byzantines

Measuring Delay Metrics :

We measure four types of delays :

- **Local Typing Lag** (Keypress $\rightarrow$ Character Appears)
- **End-to-End Delay** (User A Types $\rightarrow$ User B Sees)
- **Update Propagation Delay** (Message Reaches All Peers)
- **Human-Perceived Delay** (User Perception of Lag)

Measuring Delay : MUTE + Byzantines

Measuring Local Typing Lag :

Definition :

- Time between a keypress and when the character appears on the screen.

Method :

- Log timestamps for keypress event (T_1) and render event (T_2).
- Compute :

$$\text{Typing Lag} = T_2 - T_1$$

- Tools : JavaScript performance API / screen recording.

Measuring Delay : MUTE + Byzantines

Measuring End-to-End Delay :

Definition :

- Time between when User A types an edit and when User B sees it.

Method :

- Log timestamps :
  - T_1 : User A sends edit.
  - T_2 : User B receives edit.
- Compute :

$$\text{End-to-End Delay} = T_2 - T_1$$

- Tools : Modify MUTE logging system / console logs.

Measuring Delay : MUTE + Byzantines

Measuring Update Propagation Delay :

Definition :

- Time for an edit to reach all peers in the system.

Method :

- Log timestamps :
  - T_1 : User A sends message.
  - T_2, T_3, T_4... : Each peer receives it.
- Compute :

$$\text{Propagation Delay} = \max(T_2, T_3, T_4, ...) - T_1$$

- Tools : Network monitoring (Wireshark, Chrome DevTools).

Measuring Delay : MUTE + Byzantines

Measuring Human-Perceived Delay :

Definition :

- How users perceive lag in collaboration.

Method :

- User survey after each test :
  - *Did you notice any delay ?*
  - *How frustrating was it ? (Scale 1-5)*
  - *Did the delay affect collaboration ? (Yes/No)*
- Tools : Google Forms, Excel for analysis.

Measuring Delay : MUTE + Byzantines

Running the Experiment :

Procedure :

1. Set up MUTE with BCB on participant devices.
2. Run each scenario and collect timestamps.
3. Measure delays using logs and network tools.
4. Conduct user survey after each test.
5. Analyze results and compare BCB vs. No BCB.

Analyzing Results

Expected Trends :

- **Typing Lag :** No significant difference.
- **End-to-End Delay :** BCB introduces slight increase.
- **Propagation Delay :** Higher with BCB due to causality enforcement.
- **Human Perception :** Users may find BCB more stable but slightly slower.

Tools for Analysis :

- **Python (Pandas, Matplotlib)** for delay trend graphs.
- **Wireshark** / **Chrome DevTools** for network delay analysis.

Final Deliverables

At the end of the experiments, we aim to produce :

- **Experimental Data :**
  - Raw logs of keypress timestamps, update propagation, and network delays.
  - Recorded network packets (Wireshark traces).
  - User survey responses on perceived delay.
- **Performance Analysis :**
  - Comparative graphs of BCB vs. Non-BCB delays.
  - Tables summarizing average latencies under different conditions.
  - Statistical analysis of human-perceived delay.
- **Technical Report :**
  - Description of the experimental setup.
  - Explanation of observed delay trends.
  - Recommendations for optimizing BCB in MUTE.
- **Code and Scripts :**
  - Custom logging and delay measurement scripts.
  - Experimental setup configuration for MUTE.