BYZANTINE

CONSENSUS

WHERE WE ARE

STATE MACHINE REPLICATION
(RELIABILITY FOR DET.
PROGRAMS)

FAIL STOP FAILURES

CONSENSUS PROTOCOL
(AGREE ON COMMAND
AT EACH SLOT)

PROVIDED BY

RAFT

PAXOS

ZAB
What About Byzantine Failures

- Why & Where?
  - Machines owned by ppl you don't trust

The Problems Caused By Byzantine Processes

A Byzantine Process Can:

- Pretend To Be Multiple Processes (Sybil)
- Use A Different Identity When Sending Messages
- Change Message Content Depending On Recipient
- Not Follow The Protocol
  - Send Nonsensical Messages
  - Send Messages Out Of Order
A Byzantine Process Can:

- Pretend to be multiple processes (Sybil)
- Use a different identity when sending messages
- Change message content depending on recipient
- Not follow the protocol
  - Send nonsensical messages
  - Send messages out of order
1. Have a trusted party attest to identity
   \[ \text{Public Key Cryptography} \quad \text{Root Of Trust} \]

2. Make it **Expensive** to participate
   \[ \text{Proof Of Work} \quad \text{Require Computation} \]
   \[ \text{Proof Of Stakes} \quad \text{Require Holdings} \]

   \[ \text{Does This Address Sybil Attacks?} \]
   \[ \text{Does This Address Participants Masquerading As Each Other?} \]

---

A tiny bit of cryptography

**Ethics Note:** The following description of cryptographic primitives is incorrect and incomplete. Do not implement anything based on this description. Here lie dragons.

- **Keys:** Fixed length numbers. For public key cryptography
  - Two keys: Pub, Priv
  - Two functions: Enc, Dec

\[ \text{Enc: Key} \times \text{Byte Buffer} \rightarrow \text{Byte Buffer} \]
\[ \text{Dec: Key} \times \text{Byte Buffer} \rightarrow \text{Byte Buffer} \]

S.t. \[ \text{Dec} (\text{Priv, Enc(\text{Pub, Buf})}) = \text{Buf} \]
Equiv. \( \text{Dec}_{\text{priv}} = (\text{Enc}_{\text{pub}})^{-1} \)

- \( \text{Enc}(\text{pub}, \text{buf}) \sim \text{Hard (impossible)} \) to invert without \text{priv}.

**Two Uses:**

**HMAC** Performance Enhancement:

\[ \text{HMAC}(k, m) \]

- \( \text{Cryptographic Hash Function } H \)
- \( H(\text{byte buffer}) \rightarrow \text{Fixed length hash} \)
  - Inverting \( H \) should be hard/impossible given current assumptions.
  - Usually faster to compute than \( \text{Enc} \).
- Challenge: Not keyed. Why problematic?
HMAC: TURN H INTO KEYED FUNCTION.

\[ \text{HMAC}(k, m) = H(k \oplus o) + H(k \oplus i) + m \] (PUBLICLY KNOWN)

- WHAT DOES BOB NEED TO CHECK HMAC(k, m)?

- GAP FROM USING PUB KEY CRYPTOGRAPHY?

---

USING CRYPTOGRAPHY TO ADDRESS IDENTITY ISSUES
A Byzantine Process Can:

- Pretend to be multiple processes (Sybil)  
- Use a different identity when sending messages
- Change message content depending on recipient
- Not follow the protocol
  - Send nonsensical messages
  - Send messages out of order

Protocol Requirements (RSM on BFT):

- Agreement: All correct processes agree on the log.
- Validity: All committed commands were proposed by a client.
- Liveness: If a client proposes a command then some command is eventually committed to the log.
LIVENESS: Does not violate FLP, why?

How the protocols we have looked at achieve agreement?
1. **Observation:**
   - In this case (not all), one can rely on set of messages received by a node to decide whether correct.

2. **Problem:**
   - Want to collect messages sent to each process. Use this to detect faulty processes.
Might think this is not a real problem (why?)

Impossibility result
For \( f \) faulty processes need \( 3f+1 \) participants.


- \( p_0 \) broadcasts \( m \) \( I \) \((OM(f))\)
  - Any process \( p_i \) receiving \( m \) broadcasts
    - "\( p_0 \) sent \( m \)" to all processes \( OM(f-1) \)
  - ...  
    - \( p_j \) broadcasts
      - "\( p_i \) sent \( p_0 \) sent \( m \)" to all processes \( OM(f-2) \)
  - ...

Pick \( m \) that shows up in majority

**Claim:** If \( p_0 \) correct, all processes agree on \( m \)
**Induction**

\[ f = 0 \Rightarrow \text{Trivially True} \]

Assume holds for \( f = k - 1 \)

\[ f = k^0 \]

1. \( P_0 \) sends \( m \) to \( n-1 \) processes
2. All correct processes invoke \( OM(k-1) \) to send \( m \)

We know

\[ n \geq 3f+1 \]
\[ \geq f + 2k + 1 \]
\[ \geq 2k + f \]
\[ n-1 \geq 2k + (f+1) \]
\[ > 2k \]
\[ > 2f \]

\[ \Rightarrow \text{Each correct process receives (up to) } \]
\[ n-1 \text{ copies of } m' \text{ of which a majority come from correct processes. Correct process } \]
\[ m' = m \Rightarrow \text{agree on } m \]

2. \( P_0 \) faulty \( \Rightarrow \) Induction shows agreement on message (but no \( m \)).
BASIC IDEA: COMMIT TO MESSAGES BEFORE PROCESSING THEM

TRANSFORMING TO PRACTICE: PBFT

$$\sigma_A : \text{Message Signed By } A \left[ (m, \text{Enc}(H(m), \text{Pub}_A)) \right]$$

Alice

Useful for checking validity

1

2

3

4

P_0

P_1

P_2

P_3

P_4
Each process waits for identical PREPARE messages from 2f processes before proceeding.

⇒ 2f+1 processes saw PRE-PREPARE
⇒ ≥ f+1 correct processes saw PRE-PREPARE
\[ \sigma_{\text{Alice}}(C) \]

\[ \sigma_{p_2}(\text{COMMIT}, v, i, d, 2) \]

**Observe:** All processes send out commit message. Why?

---

**Wait for** \(2f+1\) commit messages. Why?

---

\[ \sigma_{\text{Alice}}(C) \]

\[ \sigma_{p_3}(\text{RESPONSE}) \]
Liveness

1. What if Leader Is Byzantine?

What Should Other Process Do

- If process has seen Command
  $\Rightarrow$ Nothing for now, leader does not seem to be at fault

- If process has not seen Command
  $\Rightarrow$ Force leader election.

Leader Election + Liveness

- Want to ensure faulty nodes do not always respond.
BECOME LEADER

- How? Rotating Leadership

\[ \text{Process } i \text{ is leader when } i \equiv t \mod n \text{ \text{Term} } \]

Still need to prevent frequent leader election

How?