DeathNode

An anonymous reporting platform.

Network and Computer Security
Computer Science and Engineering
08/01/2026

Group: 53

Guilherme Pais	**N°** 116496
Pedro Duarte	**N°** 116390
Tomás Fernandes	**N°** 116122

Table of Contents

01

Project Scope

Project Scope

Main Assumptions

Static, fixed membership.

Preconfigured cryptographic trust.

Immutable report history.

No failures.

Fixed vigilant monitor server.

CA **fully trusted.**

Client & monitor server **partially trusted.**

Network, central server & database server **untrusted.**

Application-level anonymity.

Security Goals

SR1 Confidentiality

End-to-end encryption.

SR2 & SR3 Integrity

Individual (tamper-evident).

Batch (no missing/out of order/duplicated reports).

SR4 Consistency

Global history verification Prevent conflicting views.

Availability (Challenge B)

Resist flooding attacks.

02

Secure Document Format

Secure Document Format

Envelope Design

```
{
  "metadata": {
    "report_id": "abc123",
    "metadata_timestamp": "2025-10-28T12:00:00Z",
    "report_creation_timestamp": "2025-10-28T10:00:00Z",
    "node_sequence_number": 42,
    "prev_envelope_hash": "hex(SHA256(previous_envelope))",
    "signer": { "node_id": "nodeA", "alg": "Ed25519" }
  },

  "key_encrypted": {
    "encryption_algorithm": "RSA-OAEP-SHA256",
    "key_per_node": [
        {"node": "self", "encrypted_key": "b64url(...)"},
        {"node": "nodeB", "encrypted_key": "b64url(...)"},
        {"node": "nodeC", "encrypted_key": "b64url(...)"}
    ]
  },

  "report_encrypted": {
    "encryption_algorithm": "AES-256-GCM",
    "nonce": "b64url(encNonce)",
    "ciphertext": "b64url(...)",
    "tag": "b64url(...)"
  }
}
```

Lightweight security library

protect · check · unprotect

Confidentiality

Report encrypted with *AES-256-GCM* with metadata as *AAD*;
DEK encrypted per recipient using *RSA-OAEP-SHA256*.

Integrity & Authenticity

Ed25519 signature over (report || metadata), stored *encrypted*.

Consistency & Ordering

Per-sender sequence number & previous hash - detects duplicate, reordered, forked or missing (partially) reports.

Design Choices

Standard, well-supported crypto (AES-GCM, RSA-OAEP, Ed25519);
Separate keys for encryption and signing (cross-protocol attacks).

Why It Works

AEAD binds content + metadata - tamper detection;
Encrypted signatures prevent impersonation and signature swapping;
Hash chaining gives strong integrity/consistency without heavy consensus.

03

Security Protocol

Security Protocol

1

Report Submission

Nodes store encrypted reports locally, maintaining an append-only sequence with cryptographic links.

2

Merkle Commitments

Nodes compute and sign Merkle roots over buffered, unsynced reports, committing to their exact set and order.

3

Synchronization

Server collects, performs a **multi-stage verification** ***(slide 9)***, and globally sorts all buffers, creating a new signed block linked to the previous one.

4

Client Verification

Multi-stage pipeline verifies ***(slide 10)*** server block, block chain, Merkle root, per-node signatures, and hash chains.

Server Verification Pipeline

For each **buffer** of envelopes (with encrypted reports and metadata) received **from clients**:

1. **Client Buffer Root Signature**
   Verify each node's signature on its Merkle root to verify its authenticity.

2. **Buffer Merkle Root**
   Recompute the root from received envelopes to ensure the buffer is complete and untampered.

3. **Client Envelope Chain Continuity**
   Ensure the first report links correctly to the last known synced report from the sender node to detect diverging histories.

4. **Client Envelope Chain**
   Ensure all reports link correctly to their predecessors to detect missing, duplicated or out of order reports.

Client Verification Pipeline

For each **block** of ordered envelopes (with encrypted reports and metadata) received **from the server**:

1. **Server Block Signature**
   Verifies server's identity and sync result authenticity.

2. **Block Chain Continuity**
   Ensures history continuity with the previous block root and sequence number and detects server equivocation.

3. **Block Merkle Root**
   Recompute Merkle root from received ordered envelopes to confirm server has not reordered or omitted envelopes.

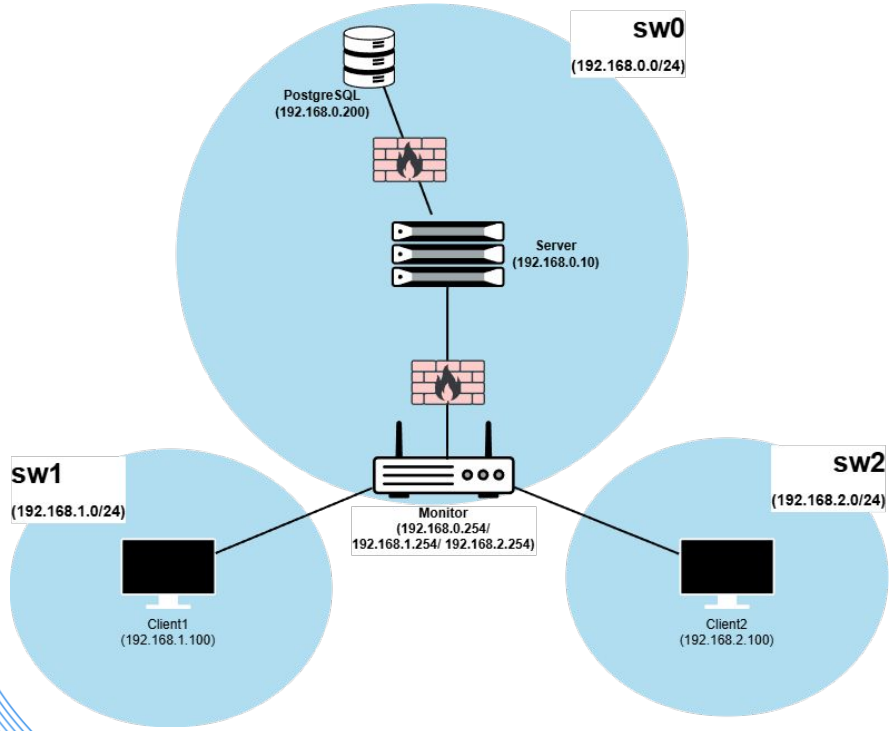
4. **Per-Node Buffer Signatures**
   Verifies each buffer signature, preventing the server from fabricating buffers on behalf of nodes.

5. **Per-Sender Hash Chains & Continuity**
   Using the previous envelope hash and sequence number, detects missing, duplicated or reordered reports per sender.

04

Infrastructure, Secure Channels & PKI

Infrastructure



Communication Flow

Client Nodes → Monitor (Gateway) → Server → Database
Database → Server → Monitor (Gateway) → Client Nodes
Clear separation of roles and network segments.

Monitor as gateway

All node traffic passes through the monitor, which performs NAT and forwarding, while supervising the network and enforcing policies.

Minimal attack surface

Server exposes only port 9090, reachable only from the monitor.
Database reachable only from the server on 5432.

Strong isolation

Server, DB, monitor, and nodes are on separate subnets.
Nodes cannot talk to each other or the DB.

Strict firewalling (*iptables*)

Default DROP policies everywhere.
Explicit allow rules only for required flows.

Communication Security: TLS & PKI

Mutual TLS (mTLS)

All gRPC (Client ↔ Server) and JDBC (Server ↔ Database) communications are secured with mTLS.

Two-Level Key Architecture

User-level keys: RSA-2048 for report encryption, Ed25519 for digital signatures.

Transport-level keys: RSA-2048 key pairs for TLS server/client authentication, X.509 certificates.

Public Key Infrastructure (PKI)

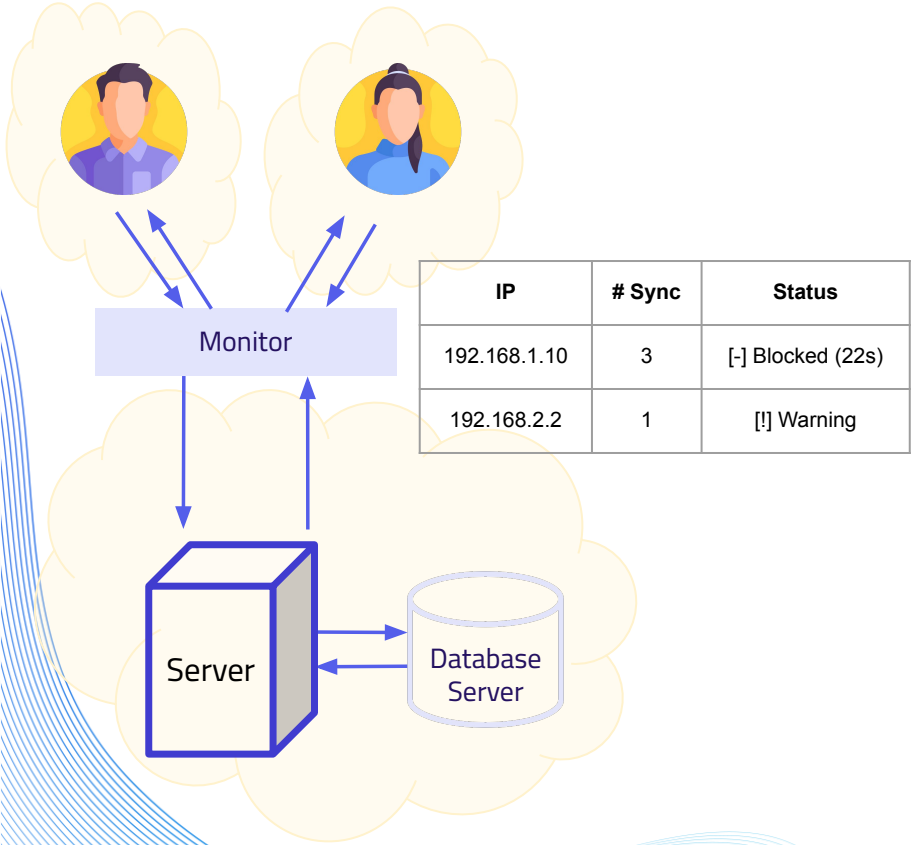
CA: Manages trust relationships with a self-signed root certificate.

Entity Certificates: Unique certificates for each node, server and database, signed by the CA, with SANs.

05

Security Challenge

Security Challenge: B



IP	# Sync	Status
192.168.1.10	3	[-] Blocked (22s)
192.168.2.2	1	[!] Warning

Monitor Server Role

A dedicated Python-based monitor server observes network activity to detect flooding attacks from misbehaving nodes.

Positioned as a gateway between clients and the central server, it intercepts all client-server communications.

Flooding Detection

Monitors client synchronisation requests. If a client sends too many requests within a 30-second window, the monitor inserts a dynamic ***iptables*** rule to drop that client's packets temporarily.

Blocking timeout doubles with each subsequent infraction.

06

Results & Conclusions

Results & Conclusions

Requirement	Mechanism	Enforcement Point
SR1: Confidentiality	End-to-end AES-GCM encryption	Only authorized nodes can decrypt
SR2: Integrity (individual)	End-to-end AES-GCM and Ed25519 signatures inside ciphertext	Client verification upon decryption
SR3: Integrity (batch)	Per-sender hash chains + Merkle trees	Server and client verification
SR4: Consistency	Signed block roots + block chaining	Client multi-stage verification pipeline
Availability	Gateway Monitor Server that supervises client-server traffic	Detects and blocks clients that exceed the sync requests threshold
Communications	mTLS and network isolation	gRPC & JDBC are implemented using mTLS

We believe our system meets all project requirements under the defined threat model and assumptions.

07

Demo

Thank you!