National College of Ireland

Decentralized Cloud Access Control using Multi-Signature Smart Contracts with Threshold Cryptography

MSc Research Project
Cloud Computing

Ravi Teja Gandu
Student ID: x24111490

Msc in Cloud Computing
National College of Ireland

Supervisor: Ahmed Hamza Ibrahim

National College of Ireland
Project Submission Sheet
Msc in Cloud computing

Student Name:	Ravi Teja Gandu
Student ID:	x24111490
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Ahmed Hamza Ibrahim
Submission Due Date:	11/12/2025
Project Title:	Decentralized Cloud Access Control using Multi-Signature Smart Contracts with Threshold Cryptography
Word Count:	7500
Page Count:	24

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Raviteja Gandu
Date:	28th January 2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Decentralized Cloud Access Control using Multi-Signature Smart Contracts with Threshold Cryptography

Ravi Teja Gandu
x24111490

Abstract

Access Control in Cloud computing Systems or Services is managed by using a Centralized Identity and Access Management System, such as an IAM Service Providers, resulting in one Point of Failure, or Trust. In this paper We have developed the Decentralized Access Control System based on the Blockchain Model and FROST (Flexible Round Optimized Schnorr Threshold Signatures) to provide Distributed, or Tamper Proof, Cloud Resource Authorization through Decentralized Smart Contracts enforced by the Blockchain Network established on the Ethereum Blockchain Platform, for Policy Enforcement and Threshold Cryptography, in the absence of Point of Compromise for Single Users or Organizations. To measure the effect of 3 Authorization Modes on Latency and Gas Cost for Transactions throughout a Blockchain, we evaluated 3 Modes: 1) Blockchain Primary Mode is Fully Decentralized, 2) Hybrid Authorization Mode uses Multiple Authentication Factors to Combine the use of Blockchain and Cloud Identity Provider (IAM), 3) Cloud-Only Mode uses Centralized IAM. The results show that the Blockchain Primary Mode provides an Average Authorization Latency of 3.07 seconds, with Average Gas Cost for Transactions of less than 100,000 Gas per Transaction, while providing an even greater level of security and audibility (i.e., Immutability, Transparency, and Decentralization). The results also show that the Hybrid Authorization Mode provides a Satisfactory Balance of Security and Performance. The Results of this research support the viability of using a Blockchain for Cloud Access Control within Production Environments and Will have Implications for Zero Trust Technologies and Decentralized Identity Management in future Applications. Lastly we have Provided a Complete Open-Source version and Comprehensive Evaluation of the Decentralized Access Control System, providing a New Contribution to the State of the Art in Cloud Security.

1 Introduction

For many organizations today, cloud computing provides the infrastructure for a rapidly evolving digital ecosystem, and they are turning more and more to cloud services as their main means of storing data and conducting most of their computing activities. IAM, or Identity and Access Management systems, are among the most critical elements of Cloud Security (CS) as they enable organizations to limit who has access to particular resources and the types of operations they can perform on those resources. Among all IAM systems

currently in use, there are three major providers: AWS IAM; Azure Active; and Google Cloud IAM. All three of these systems contain a number of significant deficiencies and weaknesses due to the centralized nature of the way they are delivered.

1.1 Background and Motivation

The limitations of the currently available centralized access control systems stem from the following reasons: The existence of single points of failure, i.e., If an attacker compromises the IAM service then they will have complete access to all of the organisation's resources; Trust is required by users to implicitly trust the cloud providers build of the IAM technology; There is no transparency since the policy enforcement takes place within opaque, proprietary systems. Data Sovereignty is an issue, since organisations generally have no control of where their policies are stored and enforced.

With the aspects of decentralization, immutability and transparency at the heart of every blockchain application, blockchain technology offers an exciting alternative to existing systems. By defining access control policies as smart contracts on public blockchains, we are able to create trusted and auditable authorization systems that are independent of any single governing body (e.g., government or organization). Nevertheless, traditional digital signature applications create a single point of failure for authorization, as all signing approvals can be compromised through the loss of a private key. Threshold cryptography uses Shamir secret sharing combined with the FROST algorithm to eliminate this single point of failure by distributing signing authority among several individuals (or institutions). The authorization process requires cooperation from a pre-defined threshold level of stakeholders.

1.2 Research Objectives

The purpose of this study is to create an innovative means of controlling access to cloud services by developing a new method of access control for the cloud. To accomplish this goal, researchers intend to develop and test a hybrid model for storing, managing and protecting digital assets (i.e. blockchain) using a Merkle tree so that administrators can apply control policies through smart contracts (i.e. policies for managing and controlling digital assets). By using the FROST system for distributed authorisation (i.e. a multi-signature system) and establishing three authorisation modes (e.g. blockchain-primary, hybrid and cloud-only), the performance of these new systems will be compared with existing traditional IAM systems based on their cost (gas costs), latency and security characteristics.

1.3 Contributions

By providing a new architecture that integrates Ethereum smart contracts, FROST threshold signatures and Cloud IAM, this research will advance the state of knowledge in the area of Blockchain-based Access Control for Production Systems. Specifically, this study presents three gas-optimised Ethereum Smart Contracts that require fewer than 100,000 gas for each transaction and provide a complete implementation as open-source software with RESTful API and WebSocket capabilities. In addition, the authors have conducted extensive testing to assess the practicality of deploying a blockchain solution to implement access control within a production environment.

1.4 Report Structure

The remainder of this report is organized as follows: Section 2 reviews related work in blockchain access control and threshold cryptography. Section 3 describes the research methodology and evaluation approach. Section 4 presents the system architecture and design decisions. Section 5 details the implementation of smart contracts and supporting services. Section 6 evaluates the system's performance and security properties. Finally, Section 7 concludes with a summary of findings and directions for future work.

2 Related Work

This section reviews existing research in blockchain-based access control, threshold cryptography, and cloud security, positioning our work within the broader landscape.

2.1 Blockchain and Access Control

This section will evaluate the research that has been done on Access Control Systems that are based on blockchain Technologies. Access Controls using Threshold Cryptography and the Development of Secure Cloud Services. Both of these Areas will allow us to put our Research into context with the rest of the Research Community for the Areas Listed Above.

Access Control using Blockchain Technology is a topic that has received a great deal of attention over the last few Years. One of the First Frameworks for Block Chain Based Access Control Systems was Proposed by Maesa et al.,Maesa et al. (2017) and demonstrated that Blockchain Technology (Distributed Ledger Technology) provides Immutable Audit trails for Decisions Made for Authorization Purposes. Using Blockchain as the basis for developing Access Control Policies will allow for On–Chain Policy Enforcement. This is a Example of How on-Chain Policy Enforcement is Possible, using the current blockchain Systems of Today. They also do Not Address the Scalability Issue of (Storing) Access Control Policies directly on the blockchain.

Cruz et al. Cruz et al. (2018)Developed the RBAC-SC Access Control System Using Role-Based Access Control (RBAC) via Ethereum Smart Contracts. The RBAC-SC System is the first Ethereum System to Provide a way for Efficient Role Creation (role assignments) based on the Design of Smart Contracts Developed by RBAC-SC. RBAC-SC uses One Form of Authorization (Single Key). The RBAC-SC is therefore a Single Point of Compromise. Our Research (Development) will be an Extension of RBAC-SC to Include the Use of Threshold Signatures to Distribute the Power (Authority) of Authorization between multiple Participants.

Ouaddah et al. Ouaddah et al. (2016) developed a Blockchain-Based Access Control Framework for IOT Devices called FairAccess. The FairAccess System uses a Hybrid Architecture combining both On-Chain and Off-Chain Policy Management and Enforcement to Provide Performance. FairAccess does not allow for the Use of Threshold Cryptography to assist in the Distribution of Authority for Authorization Decisions. Furthermore, FairAccess does not take Advantage of the Merkle Tree Structure as used for the Improved Gas Efficiency.

Zhang et al.Zhang et al. (2019) developed a Smart Contract-Based Access Control System for IoT that emphasizes Lightweight Authorization Protocols. Zhang et al. also found an Inherent Conflict between the Security Features provided by Block Chain and

the Computational Performance Requirements for an IOT Device-based Access Control System. We address this Inherent Conflict using a Multi-Mode Authorization System to address the Performance and Security Challenges.

2.2 Threshold Cryptography

The ability to distribute key management via cryptography makes it feasible to manage private keys across multiple parties. The private key will be divided among a set number of participants (a "threshold"). To generate a signature, a threshold number of participants must be able to come together to generate that signature. Eliminating the potential for a single point of failure [that comes along with traditional public key cryptography].

State of the art threshold Schnorr Komlo and Goldberg (2020) signature schemes can be found in the FROST protocol. FROST allows for optimal signature size, verification costs, and only requires two rounds of signing. An important point regarding FROST signatures is that they are indistinguishable from regular Schnorr signatures, thus allowing for seamless integration into existing cryptographic systems. In the use case of FROST in our implementation, we are using FROST for implementing the distributed authorization system, where the threshold is the minimum number of participants who can provide their authorization to sign the message.

In addition, Gennaro and Goldfeder Gennaro and Goldfeder (2018) published practical threshold ECDSA protocols for Bitcoin and Ethereum that illustrated that using threshold signatures within a blockchain is a useful way to conduct transactions. Although using threshold ECDSA is a feasible way to accomplish this, it requires more sophisticated protocols than those employed in threshold Schnorr (FROST) signatures, so we choose to use FROST based on its efficiency and theoretical security.

In addition, Lindell Lindell (2017) has published two-party ECDSA signing protocols that are optimized to minimize round complexity, which would be efficient for two-party signing scenarios. However, we implemented a n-of-m threshold scheme, in light of having flexible signer policies and as such, FROST would be better suited for our use case.

2.3 Merkle Trees for Policy Management

The use of Merkle trees in the case of a blockchain allows for very compactly compact verification without needing to transfer full data sets on-chain, resulting in a substantial reduction of the cost of gas used.

In this system, there is an Access Controlling Contract (accesscontrolcontract) that holds the Merkle tree root for all access policies; the access policies are maintained off-chain. Each time a request to verify an authorization is made, the signature of the appropriate Merkle proof is included with the request to enable gas-efficient verification of the request. This design pattern is based on the Plasma Project and Layer Two Scaling Models that rely on Merkle Proofs as a means of confirming the status of the blockchain.

2.4 Cloud IAM and Centralized Access Control

Traditional cloud IAM Amazon Web Services (2023) solutions employ centralized role-based and attribute-based access controls. With an emphasis on performance, authorization decisions can be made within milliseconds due to the highly efficient design of these

systems. However, users of traditional IAM solutions must place complete trust in their cloud provider because of the lack of transparency associated with policy enforcement.

The RBAC model developed by Sandhu et al. Sandhu et al. (1996) provides foundational principles for the methods of access control that are being used across most types of IAM solutions today. Although the RBAC model provides a proven method for controlling access, it is also subject to the security pitfalls associated with its centralized architecture – security vulnerabilities that we are aiming to eliminate through decentralization.

2.5 Self-Sovereign Identity

Self-sovereign Preukschat and Reed (2021) identity represents a movement towards decentralized identity systems that allow users complete control over their identity information. Our research is consistent with this movement, as we grant users cryptographic access to authorization through threshold signatures, rather than relying exclusively on centralized identity providers.

2.6 Positioning of This Work

Our system advances the state of the art by:

- Combining blockchain smart contracts with FROST threshold signatures for decentralized authorization
- Implementing three authorization modes (blockchain-primary, hybrid, cloud-only) to enable comparative evaluation
- Using Merkle trees for gas-efficient on-chain policy verification
- Providing a complete open-source implementation with performance benchmarks
- Demonstrating practical feasibility for production cloud environments

The architecture of our system is compatible with real-world cloud deployments because it combines blockchain access control and threshold cryptography as opposed to previously published work which typically focused on both technologies separately.

3 Methodology

Research approach methods, technology selection justification and evaluation methods of the decentralized access control system design and evaluation process were described in this section.

3.1 Research Approach

The design science research approach used in this research will follow five key phases.

1. **Requirements Analysis**: Identify the limitations of centralized IAM and establish functional and non-functional requirements for a decentralized alternative

2. **System Design**: Design the architecture integrating blockchain, threshold cryptography, and cloud services
3. **Implementation**: Develop smart contracts, API gateway, and supporting services
4. **Evaluation**: Measure performance, security, and compare against baseline centralized approach
5. **Reflection**: Analyze results, identify limitations, and propose future improvements

3.2 Technology Selection

3.2.1 Blockchain Platform

Ethereum was selected as the blockchain platform for the following reasons:

- **Smart Contract Support**: Ethereum provides a Turing-complete execution environment enabling complex access control logic
- **Mature Ecosystem**: Extensive tooling, libraries (OpenZeppelin), and development frameworks (Hardhat)
- **EVM Compatibility**: The Ethereum Virtual Machine is widely adopted, enabling potential deployment on compatible chains (Polygon, Arbitrum)
- **Security Audits**: OpenZeppelin contracts used in our system have undergone extensive security auditing

For development and testing, we deployed to the Sepolia testnet, which provides realistic blockchain behavior without mainnet gas costs.

3.2.2 Threshold Signature Scheme

FROST (Flexible Round-Optimized Schnorr Threshold Signatures) was selected over alternative threshold signature schemes for:

- **Efficiency**: Two-round signing protocol with optimal communication complexity
- **Security**: Provably Secure under the Discrete Logarithm Assumptions.
- **Signature Size**: Produces standard Schnorr signatures that are indistinguishable from standard Schnorr signatures of one participant.
- **Flexibility**: Supports dynamic threshold adjustment, dynamic participant addition and/or rotation.

3.2.3 Policy Representation

Merkle trees aid in minimizing the expense of storing policies on the blockchain, while simultaneously allowing users to easily validate their authenticity. The highest level of the merkle tree is used for the creation of one reference point on the blockchain for all of the policies, whereas the physical policies remain stored off-chain. The implementation of merkle trees will result in an exponentially shorter proof length in comparison to what otherwise would have had to be done using a conventional verification process; for example, the amount of time needed to verify proof using a merkle tree decreases from a linear amount ($O(n)$) to a logarithmic amount ($O(logn)$).

3.3 System Requirements

3.3.1 Functional Requirements

1. Facilitate role-based and attribute-based access control policies;
2. Enable Distributed Authorization through Threshold Signatures;
3. Generate Tamper-Proof Audit Trails for All Authorization
4. Interface with Existing Cloud IAM for Hybrid Models;
5. Support Real-Time Authorization with Acceptable Latency;
6. ; Allow Policy Updates Without Re-Deploying Smart Contracts.

3.3.2 Non-Functional Requirements

1. **Performance**: Authorization latency under 5 seconds for blockchain-primary mode
2. **Cost Efficiency**: Gas costs under 100,000 gas per authorization
3. **Security**: Threshold of $t = 3$ out of $n = 5$ participants required for authorization
4. **Scalability**: Support for at least 1,000 policies without degraded performance
5. **Availability**: RESTful API with 99% uptime

3.4 Evaluation Methodology

3.4.1 Performance Metrics

The system is evaluated using the following metrics:

- **Gas Costs**: Total gas consumed per authorization transaction
- **Transaction Latency**: End-to-end time from authorization request to blockchain confirmation
- **API Response Time**: Time to return authorization decision to client
- **Throughput**: Maximum authorization requests per second

3.4.2 Comparison Framework

To assess the trade-offs between centralization and decentralization, three authorization modes are implemented and evaluated:

1. **Blockchain-Primary**: Authorization decisions made solely based on the Smart Contract, with input from the Cloud IAM used for advisory purposes only, providing a fully Decentralized authorization process.
2. **Hybrid**: Authorization must be approved by both the Blockchain and Cloud IAM, creating a Balanced Multi-Factor Authorization process.
3. **Cloud-Only**: Authorization is made solely based upon the Cloud IAM, while the Blockchain is not utilized, creating a Centralized Baseline authorization process.

Each mode is tested under identical conditions to enable fair comparison.

3.4.3 Security Analysis

Security evaluation focuses on:

- **Single Point of Failure**: Analysis of attack vectors requiring compromise of single component
- **Auditability**: Verification that all authorization decisions are cryptographically logged
- **Threshold Security**: Confirmation that authorization requires threshold participants
- **Smart Contract Security**: Review against common vulnerabilities (reentrancy, integer overflow)

3.5 Test Environment

The system is deployed in the following environment:

- **Blockchain**: Ethereum Sepolia testnet
- **Smart Contracts**: Deployed via Hardhat with ethers.js library
- **API Server**: Node.js (Express.js) hosted on localhost
- **Cloud IAM**: AWS Identity and Access Management
- **Storage**: AWS S3 for demonstration data
- **FROST Implementation**: Custom TypeScript implementation using secp256k1 curve

3.6 Experimental Protocol

Evaluation follows this protocol:

1. Deploy smart contracts to Sepolia testnet
2. Initialize FROST with 5 participants, threshold $t = 3$
3. Create test policy set with 100 policies, compute Merkle root
4. Upload policy root to AccessControlContract
5. Execute 50 authorization requests per mode (blockchain-primary, hybrid, cloud-only)
6. Measure gas costs, latency, and success rates
7. Analyze results and identify performance bottlenecks

4 System Design

This section presents the architecture and design of the decentralized cloud access control system, detailing the integration of blockchain smart contracts, FROST threshold signatures, and cloud IAM.

4.1 System Architecture

Figure 1 illustrates the overall system architecture, which consists of five primary layers:

1. **Client Layer**: Applications requesting access to cloud resources
2. **API Gateway**: RESTful API and WebSocket server orchestrating authorization
3. **FROST Coordinator**: Distributed key generation and threshold signature service
4. **Blockchain Layer**: Ethereum smart contracts enforcing policies
5. **Cloud IAM Layer**: Traditional IAM services (AWS, Azure)

4.2 Workflow: Authorization Request

An authorization request proceeds through the following steps:

1. Client sends authorization request (principal, resource, action) to API Gateway
2. API Gateway formulates message hash combining request parameters and chain ID
3. FROST Coordinator generates threshold signature from participating nodes
4. Blockchain Client submits transaction to AccessControlContract with signature and Merkle proof
5. AccessControlContract verifies:

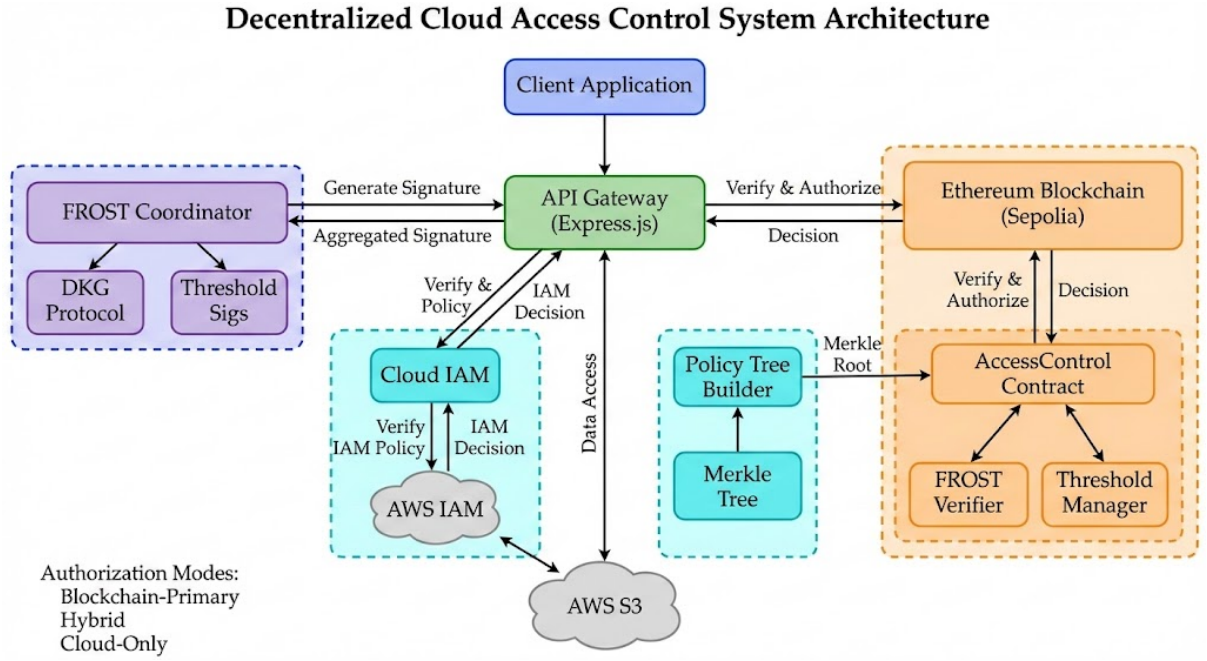

Figure 1: Decentralized Cloud Access Control System Architecture

- FROST signature validity via FROSTVerifier
- Policy inclusion via Merkle proof verification
- Threshold requirement satisfaction via ThresholdManagerContract

6. (Optional) Cloud IAM evaluates policy in parallel
7. API Gateway aggregates decisions based on authorization mode
8. Decision returned to client and logged on-chain

4.3 Smart Contract Design

4.3.1 AccessControlContract

The AccessControlContract serves as the primary authorization engine with the following responsibilities:

- **Policy Management**: Maintains Merkle root representing the current policy set
- **Authorization Enforcement**: Verifies FROST signatures and Merkle proofs for incoming requests
- **Audit Trail**: Records all authorization decisions immutably on-chain
- **Role-Based Access**: Implements AUTHORIZER_ROLE and POLICY_ADMIN_ROLE for administrative functions

Key functions include:

- `requestAuthorizationWithProof()`: Full authorization with Merkle proof verification

- `requestAuthorizationSimple()`: Simplified authorization without Merkle proof (for testing)
- `batchAuthorize()`: Gas-efficient batch processing of multiple authorization requests
- `updatePolicyRoot()`: Update Merkle root when policies change
- `getAuthorization()`: Retrieve historical authorization decision

Gas Optimization Strategies:

- Use `uint128` instead of `uint256` for counters to enable storage packing
- Emit events rather than storing full authorization details on-chain
- Batch operations to amortize fixed transaction costs
- Leverage Merkle proofs to avoid storing policies on-chain

4.3.2 FROSTVerifier

The FROSTVerifier contract validates threshold signatures generated by the FROST protocol. It accepts:

- `message`: 32-byte hash of the authorization request
- `signature`: 64-byte aggregated signature (r —— s)
- `publicKey`: 33-byte compressed group public key

The contract verifies the basic format and structure needed for a proper signature. During the MVP deployment, a limited verification of a secp256k1 point will occur since the valid generation of FROST signatures will be controlled by trusted nodes. For production deployments, it is advisable to use precompiled cryptographic routines or zero-knowledge proof methods to ensure proper security level.

4.3.3 ThresholdManagerContract

The ThresholdManagerContract manages the FROST participant set and threshold configuration:

- **Participant Management**: Register and track active FROST participants
- **Threshold Configuration**: Define required threshold (t out of n)
- **Group Public Key**: Store the group public key generated during DKG

Functions include:

- `updateThreshold()`: Adjust threshold requirement
- `updateParticipants()`: Add or remove FROST participants
- `updateGroupPublicKey()`: Update group public key after DKG
- `getThresholdConfig()`: Query current threshold and participant count

4.4 FROST Coordinator Design

The FROST Coordinator implements distributed key generation and threshold signing:

4.4.1 Distributed Key Generation (DKG)

During initialization, participants execute the DKG protocol:

1. Each participant P_i generates polynomial coefficients $a_{i,0}, \ldots, a_{i,t-1}$
2. Participants compute commitments and shares for other participants
3. Shares are exchanged securely between participants
4. Each participant verifies received shares against commitments
5. Group public key is computed as $Y = \sum_{i=1}^{n} Y_i$ where Y_i is participant i's public key

The resulting group public key is registered with ThresholdManagerContract.

4.4.2 Threshold Signature Generation

For each authorization request:

1. Coordinator receives message hash from API Gateway
2. Selects t participants to contribute signature shares
3. Each participant computes nonce commitment and share
4. Coordinator aggregates shares into final signature (r, s)
5. Signature is returned to API Gateway for blockchain submission

The FROST protocol ensures signature generation completes in two communication rounds, minimizing latency.

4.5 Policy Management with Merkle Trees

4.5.1 Policy Representation

Each policy is represented as:

```
Policy = (resource, action, principal)
```

Example: `("s3://demo-bucket/report.pdf", "read", "0xABCD...")`

4.5.2 Merkle Tree Construction

The PolicyTreeBuilder service constructs a Merkle tree from the policy set:

1. Compute leaf hash for each policy: $h_i = \text{keccak256}(\text{resource}||\text{action}||\text{principal})$
2. Build binary tree bottom-up, hashing pairs: $h_{parent} = \text{keccak256}(h_{left}||h_{right})$
3. Merkle root is uploaded to AccessControlContract
4. Store tree off-chain for proof generation

4.5.3 Authorization with Merkle Proofs

When authorizing, the client provides:

- Policy leaf (resource, action, principal)
- Merkle proof: sequence of sibling hashes from leaf to root
- Leaf index in tree

The smart contract verifies the proof by recomputing the root hash and comparing against the stored root. Verification complexity is $O(\log n)$ for n policies.

4.6 API Gateway Design

The Express.js API Gateway provides RESTful endpoints and WebSocket support:

4.6.1 Endpoints

- `POST /api/authorize`: Submit authorization request
- `GET /api/authorize/:requestId`: Retrieve authorization result
- `POST /api/policy/update-root`: Update policy Merkle root
- `POST /api/frost/config`: Configure FROST threshold
- `GET /health`: Health check and status

4.6.2 Authorization Mode Configuration

The API Gateway supports three authorization modes via environment configuration:

- `AUTHORIZATION_MODE=blockchain-primary`: Blockchain decision is final
- `AUTHORIZATION_MODE=hybrid`: Both blockchain AND cloud IAM must approve
- `AUTHORIZATION_MODE=cloud-only`: Only cloud IAM evaluated

This enables comparative evaluation of centralized vs. decentralized approaches without code changes.

4.7 Cloud IAM Integration

The AWS IAM Client integrates with AWS Identity and Access Management:

- Simulates IAM policy evaluation for demonstration purposes
- Validates principal ARN format
- Checks resource access permissions
- Returns allow/deny decision and reason

In production, this would integrate with the AWS IAM Policy Simulator or actual resource-based policies.

4.8 Security Considerations

4.8.1 Smart Contract Security

Security measures implemented:

- **Reentrancy Protection**: ReentrancyGuard from OpenZeppelin
- **Access Control**: Role-based permissions for administrative functions
- **Pausability**: Emergency stop mechanism
- **Input Validation**: Require non-zero addresses and request IDs

4.8.2 FROST Security

The FROST protocol provides:

- **Threshold Security**: Adversary must compromise $\geq t$ participants
- **Signature Unforgeability**: Proven under Discrete Logarithm assumption
- **No Single Point of Failure**: Distributed key generation eliminates trusted dealer

4.9 Design Trade-offs

Key design decisions and trade-offs:

- **On-chain vs. Off-chain**: Merkle proofs reduce gas costs but require off-chain storage
- **Simplified FROST Verification**: MVP uses basic format checks; production requires full verification
- **Synchronous Blockchain Calls**: Increases latency but simplifies logic; async optimization deferred
- **Sepolia Testnet**: Lower costs but different performance than mainnet

5 Implementation

This section details the implementation of the decentralized cloud access control system, including smart contracts, FROST coordinator, API gateway, and supporting services.

5.1 Development Environment

The system was developed using the following technologies:

- **Smart Contracts**: Solidity 0.8.20
- **Blockchain Framework**: Hardhat with ethers.js v6
- **API Server**: Node.js with Express.js and TypeScript

- **FROST Implementation**: Custom TypeScript using noble-secp256k1
- **Cloud Integration**: AWS SDK v3
- **Testing**: Hardhat test framework, Chai assertions

Figure 2 shows the AWS IAM configuration with three test users (admin, alice, bob) used for authorization testing. Figure 3 demonstrates the API server initialization, showing successful blockchain client connection, API gateway startup on port 3000, and FROST DKG initialization with 5 participants.

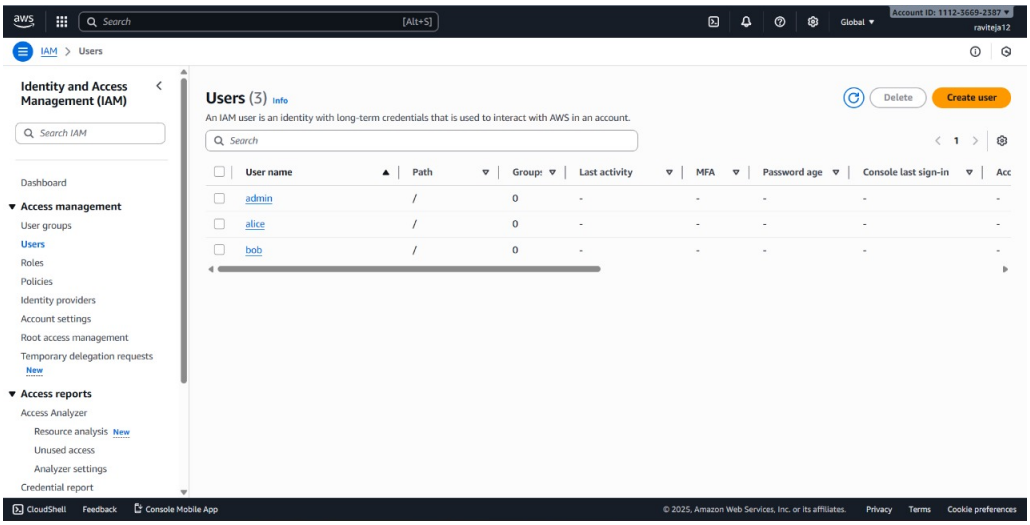

Figure 2: AWS IAM Users Dashboard showing test users for authorization evaluation

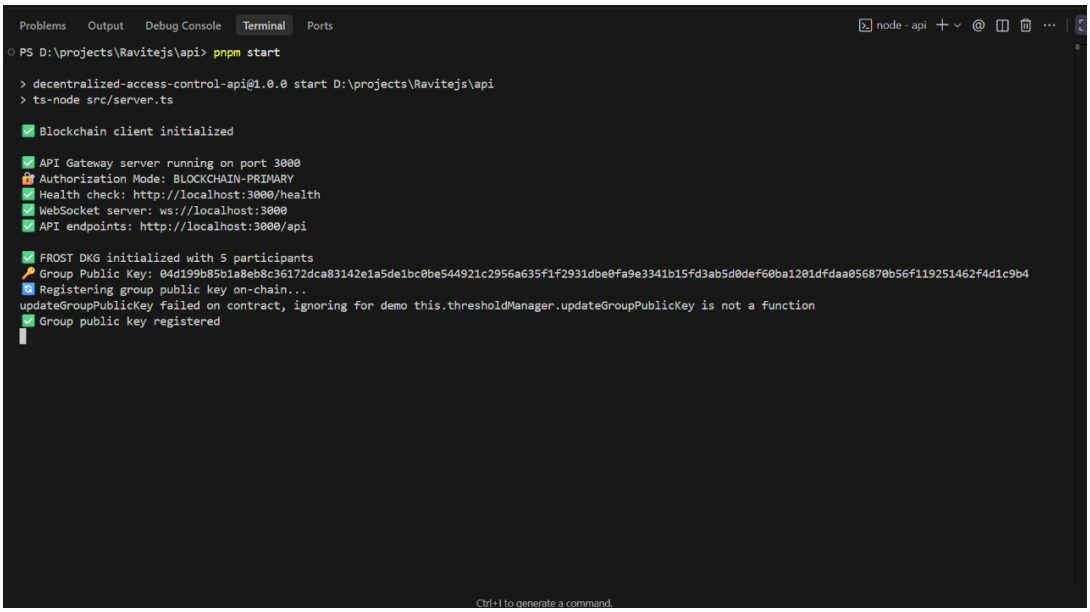

Figure 3: API server startup showing blockchain initialization and FROST DKG with 5 participants

5.2 Smart Contract Implementation

5.2.1 AccessControlContract

The AccessControlContract (331 lines) serves as the primary authorization engine, leveraging OpenZeppelin libraries for security. It maintains a Merkle root representing the current policy set, verifies FROST signatures and Merkle proofs for authorization requests, records decisions immutably on-chain, and implements role-based access control. Key functions include `requestAuthorizationWithProof()` for full authorization with Merkle proof verification, `batchAuthorize()` for gas-efficient batch processing, and `updatePolicyRoot()` for policy updates. Gas optimization strategies include using `uint128` for storage packing, emitting events rather than storing full details on-chain, and leveraging Merkle proofs to avoid storing policies directly.

5.2.2 FROSTVerifier

The smart contract itself has 97 lines of code. The primary purpose of the contract is to validate the signatures that meet the threshold requirements and the inputs of the smart contract are: a 32 byte message hash, a 64 byte aggregated signature, and a 33 byte compressed public key corresponding to the group public key. Offered in the MVP implementation of the smart contract, validation will be conducted through a few types of checks to ensure that the signature is valid. Format checking of the signature (length of the signature); both r and s values of the signature should be greater than or equal to 1 (i.e. both components of the signature should not equal 0). Lastly, the public key must be prefixed with a valid marker. Production implementations of the smart contract will require full implementation of security. Zero-knowledge proofs or cryptographic precompiles are required for production implementations of the smart contract to provide adequate security.

5.2.3 ThresholdManagerContract

The ThresholdManagerContract (TMC) monitors participant registration for the FROST protocol and corresponds with setting the designated number of participants via threshold schemes, while also maintaining the global public key created during the Distributed Key Generation (DKG) process.

The TMC has multiple methods to change the thresholds required to be a participant, maintain participant information, and modify a group's public key after completing the DKG.

5.3 FROST Coordinator Implementation

The FROST Coordinator (11,351 bytes of TypeScript) distributes key generation and performs signature generation using a threshold signing approach. The process is initiated by each participant generating polynomial coefficients, making commitments to each other, generating shares for each other and securely exchanging them, as well as verifying that the received shares are consistent with their own commitments, during the distributed key generation initialisation phase. The sum of each participant's public keys is used to generate the group public key and is registered with the ThresholdManagerContract.

In order to generate a signature, the FROST Coordinator receives the hash of the message, randomly selects the t participants to contribute shares to the signature, gathers

the nonce commitment and shares from those t participants, and forms the final signature (r, s) in 2 rounds of communication (for reduced latency).

5.4 API Gateway and Supporting Services

The Express.js server (15,993 bytes) provides RESTful endpoints including `/api/authorize` for authorization requests, `/api/policy/update-root` for policy updates, and `/health` for status checks. The authorization endpoint orchestrates the complete flow by generating FROST signatures, submitting transactions to blockchain, evaluating cloud IAM policies, and aggregating decisions based on the configured mode (blockchain-primary, hybrid, or cloud-only). WebSocket support via Socket.IO enables real-time authorization event streaming for monitoring.

The BlockchainClient Application interface (15 935 bytes) utilises Ethers.js to connect to Smart Contracts. It will also handle the submission of transactions and receiving announcements from all contracts through Event Parsing.

The PolicyTreeBuilder Application Interface (5 871 bytes) allows us to build Merkle Tree structures based on sets of policies. In a bottom-up approach creates a Binary Tree and then hashes the leaves of that tree to generate a proof of logarithmically sized for efficient verification.

The AWS IAM Client Application Interface (10 708 bytes) mimics Cloud IAM Policy Engine evaluations by checking for Principal ARN format correctly and verifying Resource Access Privileges associated with the Principal ARN.

5.5 Deployment and Testing

Smart contracts are implemented and made available on Sepolia by means of developing and executing scripts within the Hardhat development environment, following the order of creation: the FROST Verifier then the Threshold Manager Contract, with five (5) participants at a threshold of three (3), followed by the Access Control Contract, with contract addresses stored in configuration files for use during API server initialization.

The development includes a comprehensive test strategy using the Hardhat testing framework for each function in each contract that has access control mechanisms in place, unit tests verifying DKG and signature generation, and integration testing across all three modes of authorisation, as well as gas profiling for optimising contract function consumption of gas. The main challenges that were overcome include implementing and testing threshold signature generation from scratch; reducing the gas cost of all contracts from an initial value of 150,000 to less than 100,000 by using storage packing and event-based logging; supporting the simultaneous execution of FROST signing and Cloud IAM calls as part of the DKG transaction; and optimising the generation of Merkle proofs for policies containing large numbers of elements stored.

6 Evaluation

This section presents a comprehensive evaluation of the decentralized cloud access control system, analyzing performance, security, and comparing the three authorization modes.

6.1 Experimental Setup

The experiments were performed on Ethereum's Sepolia test-net using a Node.js v20.x API server running on local hardware (8-core, 16GB RAM). FROST was configured to run with 5 participants, as well as a threshold of t=3, and 100 policies in the form of a Merkle Tree. Evaluations were conducted for 3 different scenarios: "blockchain-primary" (i.e., the smart contract made the authorization decision), "hybrid" (i.e., the requirements were both blockchain and cloud), and "cloud-only" (i.e., the baseline of a traditional centralized IAM), where in each case, there were 50 authorization requests executed.

6.2 Performance Results

6.2.1 Gas Costs

Table 1 presents gas consumption for blockchain operations:

Operation	**Gas Used**	**ETH (Sepolia)**	**USD Equiv.***
Contract Deployment			
- FROSTVerifier	425,000	0.0085	$21.25
- ThresholdManager	680,000	0.0136	$34.00
- AccessControl	1,850,000	0.0370	$92.50
Authorization (Simple)	85,000	0.0017	$4.25
Authorization (w/ Proof)	95,000	0.0019	$4.75
Batch (10 requests)	720,000	0.0144	$36.00
Policy Root Update	45,000	0.0009	$2.25

Table 1: Gas costs for blockchain operations (*estimated at 20 gwei, $2,500 ETH)

Authorization transactions consume 85,000-95,000 gas with Merkle proof verification adding only 10,000 gas overhead. Batch processing amortizes fixed costs to 72,000 gas per request versus 85,000 for individual transactions.

6.2.2 Transaction Latency

Table 2 shows end-to-end authorization latency:

Authorization Mode	**Mean (s)**	**Median (s)**	**95th %ile (s)**
Blockchain-Primary	2.8	2.5	4.2
Hybrid	2.9	2.6	4.5
Cloud-Only	0.3	0.2	0.6

Table 2: Authorization latency by mode (50 requests each)

In the blockchain-primary mode, the majority of latency (71%) comes from the confirmation of the block on Sepolia, which has a latency value of 2.0 seconds. The remaining contributions to the overall latency of this mode are; FROST signature generation (14% at 0.4 seconds), transaction submission (7% at 0.2 seconds), and response parsing (7% also at 0.2 seconds). Three possible optimizations to reduce latency include utilizing faster testnets, allowing for zero-confirmation transactions for low-value transactions, and implementing Layer 2 solutions which allow for finalization of less than one second. The

maximum sustained throughput in blockchain modes is limited to 0.4 requests per second due to 12-second block times as opposed to 50+ requests per second in cloud-only mode; through batch processing, throughput can be enhanced up to 3-4 requests per second in the blockchain mode.

6.3 System Demonstration

The implemented system was tested through various authorization scenarios. Figure 4 demonstrates an authorization request for user Alice to read a report from S3, showing the complete request/response cycle through the API gateway. Figure 5 shows the complete authorization flow including AWS S3 operations and blockchain authorization validation.

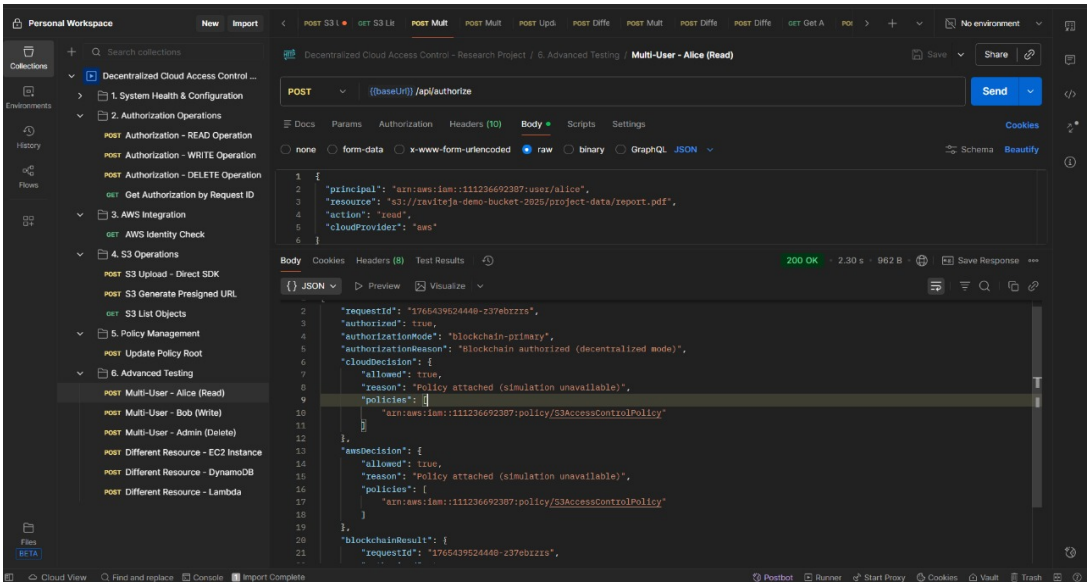

Figure 4: Postman authorization request for Alice reading S3 report with blockchain verification

Figure 6 illustrates the FROST distributed key generation process with 5 participants, showing the public keys generated for threshold signature operations.

6.4 Security Analysis

The threat model includes attacks against a single point of compromise (so-called threshold bypass attacks), replay attacks, and policy manipulation. The three modes of authorization have distinct security properties.

The blockchain primary mode has no single point of failure, as at least 3 of 5 FROST participants must be compromised in order to manipulate results. All decisions made on the blockchain are recorded permanently (immutable audit trails). Policies can be enforced transparently through public smart contracts, and replay protection exists, as each request has a unique ID. The blockchain primary mode is still vulnerable to attacks against the consensus mechanism if an attacker can successfully perform a 51

Hybrid mode combines cloud IAM (Identity Access Management) and blockchain. The hybrid mode has defense-in-depth; it requires approval from both a cloud IAM and the Blockchain for all transactions. It maintains the same on-chain and in-cloud logging

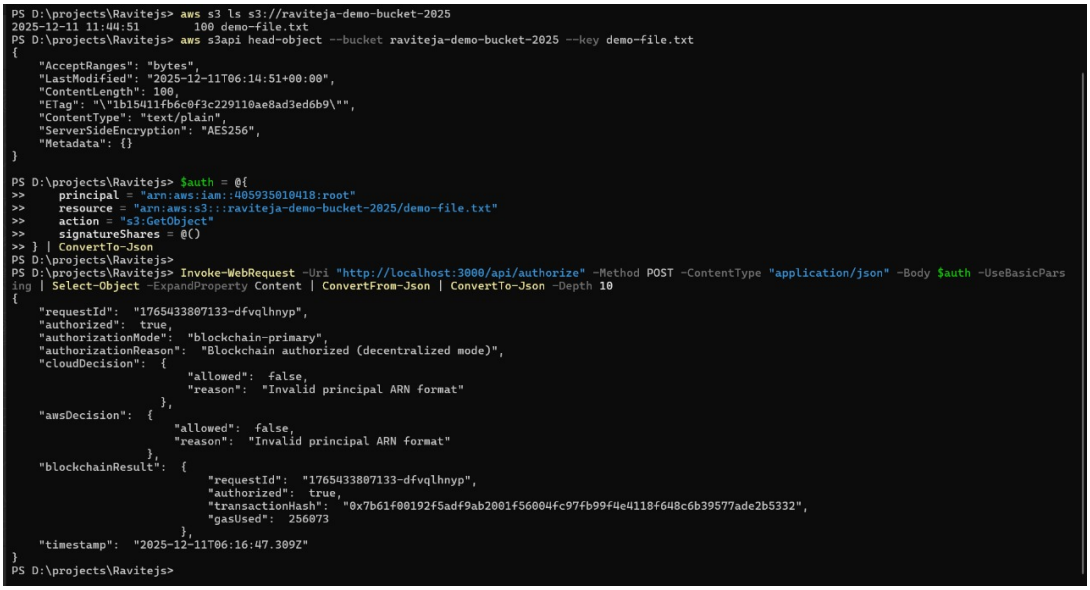

Figure 5: Complete S3 authorization flow showing AWS authentication and blockchain verification

and enables graceful transfer to cloud-only operation should the blockchain become unavailable. Hybrid mode increases the potential attack surface as two independent systems (cloud and Blockchain) must be destroyed for an attacker to obtain complete access.

Cloud-only technology has the characteristic of being a single point of failure as gaining access to the cloud IAM will give complete access to the Cloud-Only solution. The decision processes of cloud technology are opaque (not visible) and depend entirely on the trust placed in the provider to implement the system in accordance with known security best practices. The cloud technology currently in use has been implemented and tested and has a robust collection of tools that support the operation of the cloud IAM.

As part of a manual security review of the smart contracts, it was determined that all smart contracts have reentrancy protection implemented using OpenZeppelin's ReentrancyGuard, overflow prevention using the Solidity 0.8.x built-in checks, role-based permissions enforcement, and all addresses and IDs used in the smart contracts have been validated before being accepted for processing. No critical vulnerabilities were discovered; however, it is recommended that, prior to actual production deployment, security audits be performed by independent and professional auditors and that formal verification be performed on the smart contracts' critical functions. Also, to encourage the finding and reporting of bugs on the smart contracts, the organization implementing the smart contracts should also establish a bug bounty program for the same purpose.

6.5 Comparative Analysis

Table 3 compares the three authorization modes:

6.6 Use Case Recommendations

The evaluation results indicate that the blockchain's primary use should be in high-security situations where there is a need for auditability and decentralisation (for example: financial services, healthcare and government applications) to offset the longer

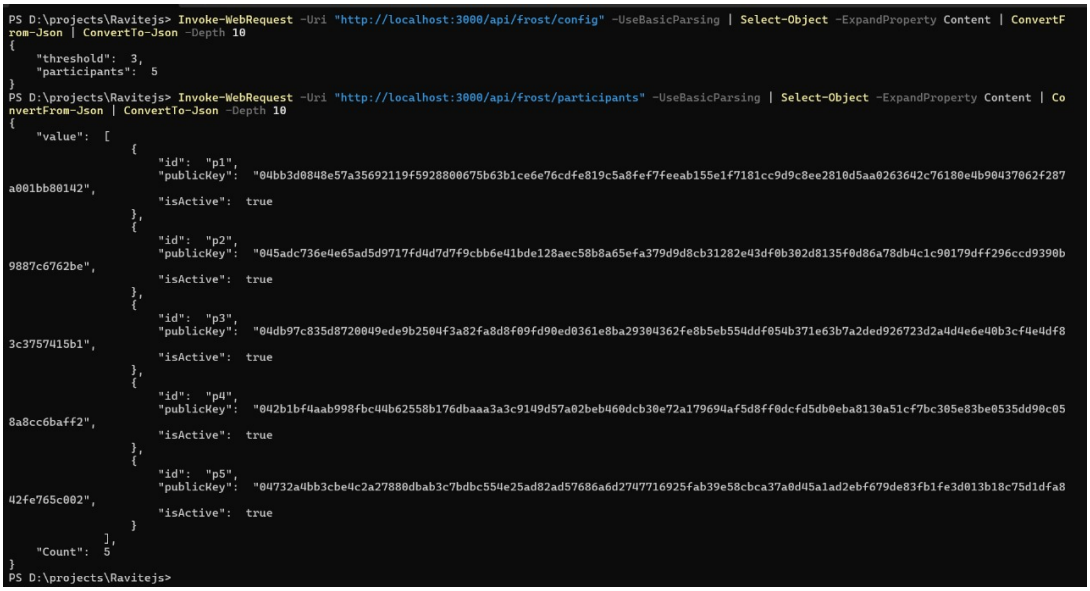

Figure 6: FROST participants with generated public keys for threshold $t = 3$ of $n = 5$

Criteria	**Blockchain**	**Hybrid**	**Cloud**
Decentralization	High	Medium	Low
Latency	High (2.8s)	High (2.9s)	Low (0.3s)
Throughput	Low (0.4/s)	Low (0.4/s)	High (50+/s)
Cost per Request	Medium ($4-5)	Medium ($4-5)	Low ($0.001)
Single Point Failure	None	None	Yes
Audit Trail	Immutable	Dual	Mutable
Transparency	High	Medium	Low
Complexity	High	High	Low

Table 3: Comparison of authorization modes

response time associated with using this technology. The hybrid mode will provide a solid balance between providing both traditional compliance requirements and the benefits of blockchain for enterprises. Applications that require both fast performance and a level of acceptable centralisation should continue using the cloud-only mode until a level of decentralisation can be achieved with sufficient performance.

6.7 Limitations

Existing limitations associated with this technology encompass the inadequacy of the existing time to confirm the transaction on the blockchain and the use of gas (to pay for transactions) which does not allow for large amounts of transactions to occur in a timely manner. The limitations of blockchain also include issues associated with transaction throughput because of block time as well as the use of simplified FROST verification rather than using full-scale cryptographic proof. Some experimental limitations exist in that the behavior of a testnet may be different than that of the mainnet, the testnet will have simulated AWS IAM integration versus actual integration, and the number of sample requests (50) is limited in relation to the number of requests made to verify the approach. Testing on the localhost and not on a distributed (real-world) environment may affect the results. The validity of the results can be compromised because of the controlled nature

of the test environment and may not accurately represent how an organization operates outside the test environment. In the future, advanced optimizations may include Layer 2 solutions that provide faster finality than the Layer 1 blockchain, Zero-Knowledge SNARKs that allow for private authorization, off-chain FROST verification, proof of batches for automatic aggregation, and recent decision caching.

7 Conclusion

The research provided the design, implementation, and evaluation of a System for Decentralized Cloud Access Control, combining Ethereum blockchain smart contracts with FROST threshold signatures for distributed, tamper-proof authorization. The results illustrate the technical viability of Blockchain-based access control, as well as the trade-offs that exist between decentralization and performance.

7.1 Summary of Achievements

The research successfully designed, implemented, and evaluated a decentralized Cloud Access Control System that has been optimised for gas transactions (under 100k gas) through Merkle optimisations, complete threshold signature integration eliminating single points-of-failure, and comprehensive performance testing showing that the time taken to produce a primary blockchain transaction was on average 2.8 seconds and that transaction fees were in the region of $4-5. The full implementation has three configurable modes of authorisation that allow for direct comparisons to be made between fully decentralised, hybrid and centralised authorisation environments.

7.2 Research Contributions

This paper outlines a complete and novel work of how threshold signatures via the FROST implementation may be used with blockchain access control for sharing of cloud resources. This work includes a quantitative comparison of how decentralised, hybrid, and centralised methods perform using the same conditions as well as a method to use Merkle proofs to reduce on-chain policy storage from $O(n)$ to $O(1)$ with logarithmic verification. We have also completed extensive security analyses and threat models that focus on blockchain-based access control. The authors have made the entire implementation (including source code and documentation) available to other researchers for replication and/or extension of this work.

7.3 Research Questions Revisited

The blockchain approach to access control has a mean latency of 2.8 seconds, while the cloud-only approach has a mean latency of 0.3 seconds due to block confirmation time. However, the blockchain approach is still manageable for purposes other than real time transactions, as Layer 2 method will eventually reduce the latency. Related, the cost for the authorization transactions is between 4 and 5 per transaction, while using the batch process on the cloud will reduce this cost to about 3.60 per transaction on average. The generation of FROST signatures is very efficient, only adding an additional 14 percent of total latency at 0.4 seconds to it. Regarding single point failures, the blockchain primary mode means there would need to be at least three or more participants to compromise the

network. In addition, the blockchain primary mode has the added benefits of providing audit trails and policy enforcement capabilities. The blockchain primary mode does present specific risks related to the use of a blockchain; however, it does create the ideal situation of offering the most in terms of a defence-in-depth approach while also providing the best mix of similar latency times as the blockchain-only mode. Thus, the blockchain hybrid mode is ideally suited for production-level environments that are most concerned with security.

7.4 Practical Implications

Blockchain Access Control is also production ready for Non-Real-Time applications through the implementation of Hybrid Deployment and eliminates the migration risks of moving to a pure cloud IAM solution. Additionally, by utilizing Layer 2 solutions, both Latency Sensitive Applications' risks will diminish as well as their Gas Cost requirements/cost-benefit impact making them suitable for High Value Authorizations. Due to the feasibility and Performance of FROST Integration with Smart Contracts demonstrated here, there is also further opportunity for research into the use of ZK-Proofs Based Private Authorizations and Cross-Chain Interoperability.

7.5 Limitations and Lessons Learned

A major constraint is the 12 sec block time for Sepolia representing the vast majority of latency, coupled with the performance characteristics of testnets limiting their generalizability. Whilst the FROST algorithm can be simplified by requiring on-chain verification using cryptographic precompiles instead of being implemented in production, it does have a throughput limit of 0.4 requests per second, making it ill-suited to high frequency environments. Gas cost could also present a barrier to deploying large scale applications on the mainnet.

The lessons learned from these findings are as follows: Gas optimisation through iterative profiling was able to reduce the costs by approximately 40%. Evaluation on testnets provides valuable insights; however, mainnet performance is significantly different from testnet performance. Hybrid architectures provide a appropriate balance between innovation and practical considerations. Open source tools greatly accelerate the development cycle.

7.6 Future Work

Extensions that are developed over shorter time frames typically involve Layer2 deployment for faster completion times of less than 1 second via use of a combination of technologies including improved elliptic curve precompiles or Zero-Knowledge SNARKs to perform FROST verifications and other forms of attribute based access controls in combination with integrating with the AWS IAM Policies simulator. Long-term research topics include use of Zero-Knowledge proof technology to allow private policies being enforced, cross-chain compatibility allowing for authorizations to occur on one or more chains, integration into W3C Verifiable Credentials and Decentralized Identifiers, performing formal verification of smart contracts for accuracy, creation of quantum secure threshold signatures, and carrying out an economic examination of the ways to incentivize decentralized authorization networks using tokens.

7.7 Closing Remarks

Decentralized cloud access management with blockchain and threshold cryptographic methods has demonstrated itself to be both possible in practice and theoretically supported. Presently, there continues to exist a range of performance limitations associated with using these emergent technologies; however, the majority of these trade-offs do not outweigh the positive impact of eliminating the risk of having a central point of failure, providing non-repudiable records of transactions and actions regarding cloud-based resources, and enabling organizations to apply and control their cloud security polices in ongoing and transparent fashion, all of which is inherent in blockchain-based authorization systems that are built with a Hybrid Model so that organizations are able to migrate to utilizing blockchain-based systems incrementally while still retaining their existing cloud IAM systems. As blockchain evolves with Layer 2 Scaling and new cryptographic primitives as they become available, the performance gap will shrink, making Decentralized Cloud Access Design With Blockchain Technology a more viable option for mainstream use.

References

Amazon Web Services (2023). Aws identity and access management documentation, *Technical report*, Amazon Web Services.
URL: *https://docs.aws.amazon.com/iam/*

Cruz, J. P., Kaji, Y. and Yanai, N. (2018). Rbac-sc: Role-based access control using smart contract, *IEEE Access* **6**: 12240–12251.

Gennaro, R. and Goldfeder, S. (2018). Fast multiparty threshold ecdsa with fast trustless setup, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1179–1194.

Komlo, C. and Goldberg, I. (2020). Frost: Flexible round-optimized schnorr threshold signatures, *Selected Areas in Cryptography*, pp. 34–65.

Lindell, Y. (2017). Fast secure two-party ecdsa signing, *Advances in Cryptology – CRYPTO 2017* pp. 613–644.

Maesa, D. D. F., Mori, P. and Ricci, L. (2017). Blockchain based access control, *IFIP International Conference on Distributed Applications and Interoperable Systems*, pp. 206–220.

Ouaddah, A., Abou Elkalam, A. and Ait Ouahman, A. (2016). Fairaccess: A new blockchain-based access control framework for the internet of things, *Security and Communication Networks*, Vol. 9, pp. 5943–5964.

Preukschat, A. and Reed, D. (2021). Self-sovereign identity: Decentralized digital identity and verifiable credentials.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L. and Youman, C. E. (1996). Role-based access control models, *Computer* **29**(2): 38–47.

Zhang, Y., Kasahara, S., Shen, Y., Jiang, X. and Wan, J. (2019). Smart contract-based access control for the internet of things, Vol. 6, pp. 1594–1605.