National College of Ireland

Decentralized Cloud Access Control System - Configuration Manual

MSc Research Project
Cloud Computing

Ravi Teja Gandu
Student ID: x24112490

Msc in Cloud computing
National College of Ireland

Supervisor: Ahmed Hamza Ibrahim

National College of Ireland
Project Submission Sheet
Msc cloud computing

Student Name:	Ravi Teja Gandu
Student ID:	x24112490
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 System - Configuration Manual
Word Count:	3500
Page Count:	3

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:	Ravi Teja Gandu
Date:	11th December 2025

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 System - Configuration Manual

Ravi Teja Gandu
x24112490

1 Introduction

The configuration guide includes all the necessary steps to enable or set up the Decentralised Cloud Access Control System on a blockchain and also with FROST Threshold Signatures. This allows companies to secure access to their Cloud-based resources through Blockchain-enabled authorisations via the following modes: Blockchain-Primary (fully decentralised) Hybrid (multi-factor) and Cloud-Only (traditional IAM).

1.1 System Overview

This system contains five constituents, including smart contracts running on Ethereum Sepolia Testnet, a FROST Coordinator Service, which is a DLT for distributing threshold signatures, an API Gateway that exposes RESTful API end-points, a Blockchain client for interfacing with the smart contracts, and AWS Identity Access Management for Hybrid Authorizations. This manual will guide you through the Steps for Setting Up, Configuring, Deploying, and Maintenance of the Complete System.

1.2 Prerequisites

Prior to installing, it is important to ensure that you have met the following requirements. Node.js version 20.x (or later), either npm or yarn package manager, an account on the Ethereum Sepolia testnet with test ETH (obtained from a faucet), an AWS account with IAM permissions (for hybrid mode) and Git installed for cloning the repository. In terms of hardware, you should have a minimum of 8 GB of RAM, at least a 4-core CPU and 10 GB of available disk space.

1.3 Document Structure

The organization of this manual is as follows: Setting Up Your Environment: Dependencies (Section 2), Deploying Smart Contracts (Section 3), Configuring API Servers (Section 4), Setting Up FROST Coordinators (Section 5), AWS IAM Integration (Section 6), Verifying & Testing (Section 7), Monitoring & Maintenance (Section 8), Troubleshooting & Advanced Configuration (Sections 9-10).

2 Environment Setup

2.1 Repository Clone and Installation

Clone the project repository and install dependencies:

```
git clone <repository-url>
cd blockchain
npm install
```

This installs all required packages including Hardhat, ethers.js, Express, and cryptographic libraries.

2.2 Environment Configuration

Create a `.env` file in the project root with the following variables:

```
# Blockchain Configuration
RPC_URL=https://sepolia.infura.io/v3/<your-infura-key>
PRIVATE_KEY=<your-sepolia-private-key>
CHAIN_ID=11155111

# Contract Addresses (populated after deployment)
FROST_VERIFIER_ADDRESS=
THRESHOLD_MANAGER_ADDRESS=
ACCESS_CONTROL_ADDRESS=

# API Configuration
PORT=3000
AUTHORIZATION_MODE=hybrid

# FROST Configuration
FROST_THRESHOLD=3
FROST_PARTICIPANTS=5

# AWS Configuration (for hybrid mode)
AWS_REGION=us-east-1
AWS_ACCESS_KEY_ID=<your-aws-key>
AWS_SECRET_ACCESS_KEY=<your-aws-secret>
```

Important: Never commit the `.env` file to version control. Use `.env.example` as a template.

2.3 Obtaining Sepolia Test ETH

Visit a Sepolia faucet (e.g., `sepoliafaucet.com`) and request test ETH using your wallet address. You will need approximately 0.05 ETH for smart contract deployment and testing.

3 Smart Contract Deployment

3.1 Compilation

Compile smart contracts using Hardhat:

```
npx hardhat compile
```

This generates contract artifacts in `artifacts/` directory and ABI files for client interaction.

3.2 Deployment Script

Deploy contracts to Sepolia testnet in the correct sequence:

```
npx hardhat run scripts/deploy.js --network sepolia
```

The deployment process consists of four separate stages as follows: (1) Establishment of the FROSTVerifier Contract, (2) Establishment of the ThresholdManager Contract by setting threshold=3 and participants=5, (3) Establishment of the AccessControl Contract with references to previously established contracts for storage of initial policy root, and (4) Storing contract addresses in the `.env` file.

3.3 Deployment Output

Upon successful deployment, you will see output similar to:

```
Deploying FROSTVerifier...
FROSTVerifier deployed to: 0xABC123...

Deploying ThresholdManagerContract...
ThresholdManager deployed to: 0xDEF456...

Deploying AccessControlContract...
AccessControl deployed to: 0x789GHI...

Deployment complete!
```

Copy these addresses to your `.env` file.

3.4 Contract Verification

Verify contracts on Etherscan for transparency:

```
npx hardhat verify --network sepolia <CONTRACT_ADDRESS>
```

Repeat for each deployed contract. Verified contracts allow public inspection of source code and enhance trust.