National College of Ireland

EMERGENCY ACCESS CONTROL CREDIT LEVEL-BASEDSYSTEM USING BLOCKCHAIN WITH CLOUD STORAGE

MSc Research Project
Master of Science in Cloud Computing

Sneha Sankaran Arivukkarasu
StudentID:23307374

School of Computing
National College of Ireland

Supervisor: Punit Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sneha Sankaran Arivukkarasu
Student ID:	23307374
Programme:	Master of Science in Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Punit Gupta
Submission Due Date:	11/08/2025
Project Title:	Emergency Access Control Credit Level – Based System Using Blockchain with Cloud Storage
Word Count:	7535
Page Count:	20

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:	Sneha Sankaran Arivukkarasu
Date:	11th August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	YES
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	YES
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.	YES

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):	

ABSTRACT:

The introduction of immediate digitalised healthcare has considerably enhanced administration and accessibility of the patient data however the problem arises in maintaining stability and safety under circumstances of emergency. This paper suggests an emergency access control system built on blockchain technology with a credit-level-based access control manager and coupled with cloud storage systems to strike the right balance between fast data read and demanding security demands. It performs enrolment of doctors using Ethereum smart contracts with a predefined credit limit that restricts the extent of the collection of data on patients. Transactions will be permanently kept on the block chain including registrations and access requests to ensure that everything is clear and auditable. All the information of patients is a backed-up storage in encrypted CSV format on Amazon Web Services (AWS) Simple Storage Service (S3), which could easily scale and be robust. On a correct authorisation request, the backend would issue a one-time temporary access token, lasting five minutes, that permits safe retrieval of the data without allowing extended unauthorised access. The prototype was deployed to the Ethereum Sepolia test net and transactions could be verified using Etherscan and against a range of scenarios such as duplicate registrations, too low credit levels, and token expiry. Findings prove that the system is efficient in the implementation of credit-level restrictions, elimination of unauthorised access, and provision of verifiable audit trails. This strategy proves to be capable of increasing trust, acceptance and working efficiency in emergency healthcare data management providing scaling model of adopting it to wider electronic health record systems

INTRODUCTION:

Digitalisation of healthcare is growing important, and this has helped a great deal in storing, controlling and retrieving of patients records as well as Electronic Health Records (EHRs) have given an opportunity where the medical practitioners provide quicker and more accurate service through instant access to medical histories, laboratory results, and prescriptions. But in urgent cases where quick decision is demanded frequent access to sensitive medical information poses great security and privacy risk. The traditional methods of access control cannot usually give restricted, conditional, and verifiable access without the risk of releasing unauthorised delegation. Healthcare data breach is an acute problem because stolen or inappropriately released medical data may cause the identity theft, frauds, and loss of reputation of a healthcare organization. This increased risk points to the necessity of auditable, secure and efficient means of emergency access that need to balance speed and effective security. Blockchain technology is an encouraging solution because of its decentralised, tamper-proof and transparent states as it applies in the maintenance of trust in sensitive settings. Access control policies could be automated completely by using smart contracts without the need to trust central authorities and minimize insider threats as well as policy breaches. Blockchain is complemented by cloud storage services like Amazon Web Services (AWS) Simple Storage Service (S3) that offers immense scalable, available and economical infrastructure to host a large amount of patient data. This paper presents such an access control system called an emergency access control system which uses blockchain and cloud storage technology to offer time- and creditlevel based access to patient data. The doctors are registered through Ethereum smart contract to avoid redundant entries and fraud entries, and their credit levels decide the access rights of doctors. The encryption and storage of patient records in AWS S3 are protected, and only, when an authorised request is approved, a unique five-minute token is created to allow decryption of the data material, and which deters frequent or extended access without refreshing the authorisation. Merging the permanency of audit trail of blockchain with the flexibility and convenient access of cloud storage results in a sustainable balance of speed, assurance, and responsibility of the suggested system and thus ensures the fit within the emergency healthcare situation.

RELATED WORK:

Healthcare Data Management using Blockchain Technology

As a management tool in data security, blockchain has emerged as an innovative technology because of its characteristic attributes of decentralisation state, immutability, and transparency which is quite encouraging in the healthcare sector. When it comes to the domain of patient record management, blockchain ensures that the events of access will be stored merely once and permanently, without the option of change and abuse. Karthika and Sriramya (2019) have attempted to introduce a proposal on implementing access control system in cloud storage through block chain and succeeded to show how policy can be applied automatically through smart contracts without the involvement of a central authority. Along that line, Sohrabi

et al. (2020) presented a blockchain-based access control protocol to cloud data, BACC, which, as the authors stated, boosts its capability to integrate the decentralised authentication and scalability of data storage services. Application of blockchain in health care particularly in addressing the rising data breach and unauthorised release has also been crystallised in the recent studies. Prasath Alias Surendhar et al. (2023) did not overlook the importance of blockchain in mitigating security and privacy vulnerabilities of cloud-based systems and mentioned that, in the healthcare sector, blockchain would be useful in terms of safeguarding sensitive data against cyber-crimes. Along with storage security, other survey types like the Li et al. (2017) have successfully surveyed the blockchain vulnerabilities and recommended best practices to counter such vulnerabilities aimed at enhancing the resilience of the system, which is critical to the implementation of health care. In combination, these papers point to the relevance of blockchain as regards data integrity, transparency and trustworthiness of any healthcare application and hence this attribute is adequate or sufficient to justify the proposal of inclusion of blockchain in this paper to redesign an emergency access control system.

Can you compare your solution, with pre-existing solutions in the domain such as Hyper-Ledger fabric or alternative DApps and clarify the value of your own application ?

Existing Solutions: Systems that are based on **Hyperledger fabric (e.g., Sharma et al., 2021):** These also could share decentralized EHR and have immutable logs but lack emergency-only access and time-limited permissioning. **Ethereum-powered DApps (e.g., Omar et al., 2022):** Do not support credit-based differentiated access, and there are no automatisms of expiry. **Blockchain - cloud hybrids (e.g., Chen et al., 2021):** do not use cloud storage to provide scalability and do not enable smart contracts to trigger an emergency or make duplicate registrations.

My Solution: Emergency-Only Access -Doctors only access when the patients are in emergencies (e.g. unconscious patients). Access Control based on **Credit Level -High-credit doctors** have access to more information than the low-credit doctors.

Time-Limited Tokens -Frequently, smart contracts also ensure that the permissions **automatically expire. Unique Doctor Registration** - Multi-registrations are prevented with wallet-based verification. Transparency With Blockchain – Every action is permanently recorded on the blockchain, and is live to view, and can be viewed on **Etherscan** making things accountable.

Cloud Integration - Patient-sensitive content is stored in the **AWS S3** (secure, scalable) with the application being run on **EC2**, so the data are stored separately to the blockchain to improve performance and compliance.

Although a Hyperledger or more generally existing DApps involve some general access management, my system specifically integrates credit-based logic, emergency-only temporary tokens, blockchain-logged irreversible access control, and an AWS cloud connection. This increases its suitability to real life emergencies in healthcare

Healthcare Access Control Models

Access control is also an essential healthcare information system feature that defines the conditions of access to sensitive information about patients. Traditional Role-Based Access Control (RBAC) approaches are pre-defined user roles such as doctor, nurse or administrator, which are in turn delegated permits. Despite its usefulness in controlled events, RBAC would be static in crisis where access must be enabled on-the-fly. To overcome this weakness, more adaptive access control like the Attribute-Based Access Control (ABAC), Credit-Based Access Control (CBAC) have been suggested. CBAC, specifically, extends the premise of the access control onto a credit score or level of each user decreeing the depth and breadth of data s/he may access. This solution has already been implemented using blockchain and part of it was implemented in tamper-free and verifiable access history. In an illustrative example, Sohrabi et al. (2020) showed that the blockchain enabled the imposition of fine-grained access policies in a decentralised setting and Sharma et al. (2022) surveyed automating of said policies using smart contracts across platforms. Integrating the immutable logging of blockchain with the access-granting policies at the credit level result in a combination of accountability and flexibility that makes the approach particularly appropriate in contexts where emergency access to healthcare is desirable because it is both imperative that decisions be made on very short timescales and that strict privacy-control be ensured.

Medical Records stored through Clouds

An increasingly important part of any contemporary healthcare system is cloud-based storage, as it provides a fully scalable and inexpensive infrastructure to hold vast amounts of patient data, with high availability. Services like the Amazon Web Services (AWS) Simple Storage Service (S3) has built-in redundancy, in-transit and at-rest encryption as well as Identity and Access Management (IAM) policies to provide access policies that are granular. Such attributes render cloud platforms attractive as a repository of Electronic Health Records (EHRs) and other key medical information. Nevertheless, using only cloud access controls inherits a point of trust, and especially when under the centre of the structure, this point becomes susceptible to poor configurations or compromised by insiders. Scientists like Prasath Alias Surendhar et al. (2023) have also outlined the issues of privacy and security concerns in cloud computing in the clinical setting and integrated blockchain as a solution to promote transparency and resistance to tampering. The ability of blockchain to provide decentralised authentication coupled with the scalability advantages of cloud storage provided by the latter was demonstrated by Karthika and Sriramya (2019). The given hybrid solution will enable storing large volumes of data in the cloud, blockchain will track the unforgeable records of who has accessed the data and when, and in which conditions. Such a model in the case of emergency health care work conditions would guarantee immediate access to patient data in cloud storage without compromising accountability and adherence to the data protection policy.

The combination of Blockchain and Cloud into healthcare

The solving of both scalability and security needs of managing healthcare data by integration of the blockchain and cloud technologies is a strategy that has gained momentum. Blockchain permits immutable, decentralised recording of access events, and cloud storage offers the environment to store and serve huge datasets in an efficient manner. Papers like Sohrabi et al. (2020) and Karthika and Sriramya (2019) have already showed how access control frameworks based on blockchain can be overlaid on cloud set ups in a manner such that it acts to provide fine-grained policies whilst remaining auditable. The review by Sharma et al. (2022) continued by researching the architectures based on smart contracts that automate access and logging verification and the necessity of manual monitoring. This hybrid solution has a direct response to the shortcoming of single systems which is that cloud-only system has no audit trail that cannot be tampered with and therefore blockchain-only has a size and cost burden that limits them to storing huge amounts of information.

The hybrid model is especially good in an emergency care scenario. The suggested solution in the research using AWS S3 to store patient records and Ethereum smart contract to handle authorisation allows us to access sensitive medical data and open it within a short period of time but only in a controlled way. The extension of existing models is furthered by the introduction of credit-level-derived access rules and access tokens which are temporal, having long-lasting access control. This makes the emergency data retrieval both time and fully accountable, which corresponds to the requirements of operational and compliance demands of the contemporary healthcare systems.

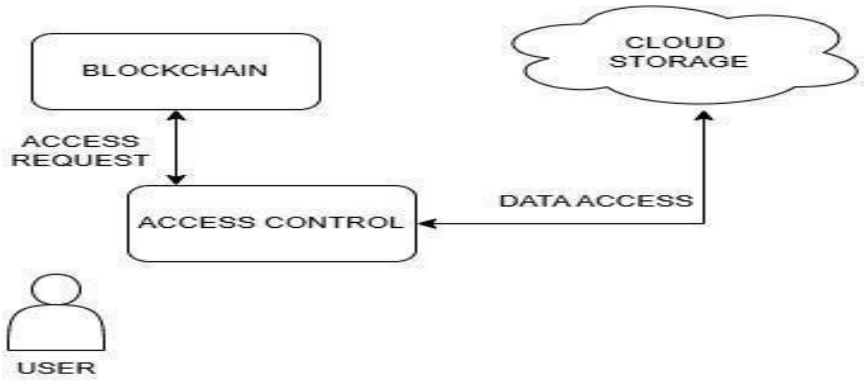

Figure 1 flow of the architecture system

General Method of Research

The study uses a design and implementation strategy whereby it enhances blockchain emergency access control system with cloud storage. The strategy commences by defining main functional necessities such as doctor registration, control of access according to the credential levels, safe storage of the patient records, and retrieval use on temporarily allocated tokens. The hybrid architecture has been chosen as the fusion of immutability and decentralisation of blockchain and scalability and availability of cloud storage. In Ethereum, smart contracts were created that can manage registration and verification of the access of doctors so that all actions are immutable. Encrypted patient records were placed in the Amazon Web Services (AWS) Simple Storage Service (S3) in CSV format that could be retrieved quickly during permitted sessions. Django was used to create the backend application which would give the interface to the smart contract, AWS S3 and the visible web portal. The system was implemented and tested on the Ethereum Sepolia test net that could occur in the real case, where transactions were checked through the Etherscan.

Background:

The sensitive data involve healthcare systems and in emergencies, the information has the potential to save lives when accessed as soon as possible. Nevertheless, traditional access control systems are provided based on static permissions, which potentially causes delayed critical care. Blockchain offers transparent, unchangeable logs of access requests and cloud storage like AWS s3 provides access to cost-efficient, highly scalable and secure hosting of data. By putting together these technologies, an efficient and auditable emergency access to the patient data could be implemented, which is controlled.

Problem statement:

In the modern medical sector, communication and exchange of patient data between different healthcare professionals is paramount regarding its security and promptness to ensure that patient care is given the necessary attention. Nevertheless, the current access control systems like Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) although work well in ordinary versions, do not deliver dynamic access control in unpredicted times like emergencies. Such systems are comprehensive and are usually centralised that make response time slower, having single point of failure that can be attacked maliciously, breached data and unauthorized changes to such data that may compromise patient privacy and delay intervention. Besides, the incompatibility of blockchain technology, where the accesses are recorded on the decentralised, immutable, and transparent ledger, with durable, scalable cloud storage where large amounts of sensitive medical information can be stored safely. Although blockchain provides support over verifiable trust in access decisions, it very seldom goes hand in hand with fine grained and time alternating access policies, which supports emergency situations. Such lack of unified solution leads to inefficiencies, less secure systems, and delays in information source of life-saving tools. Hence, an urgent necessity exists to develop a system,

combining transparency and non-destruction of blockchain and longevity, security, and cost efficiency of cloud storage with implementing the time-limited credit-level based access to patient information to resolve existing drawbacks and guarantee rapid, approved access in case of emergence in medical practice.

How can blockchain smart contracts and cloud storage be used to enable doctors to have emergency-only, credit-level based, and time-limited access to patient health records, stores everything securely, logs everything, and remains compliant to GDPR/HIPAA?

In layman terms, **the current systems** do not enable the doctors to obtain temporary access when a patient is comatose. Although access might be logged, it also tends not to automatically time-out-after-use and remains open once acquired. Most of the existing solutions are either creating data on blockchain itself (inefficient and expensive) or storing data on private servers (not scalable and not necessarily audited). They do not even prevent the duplication of doctor registrations by the same person.

What my system will do, Doctors can only have one registration with a wallet address and a level of credit (junior doctor vs. senior doctor). Smart contracts reject duplicates and store all their activities on the Ethereum blockchain (viewable on Etherscan). In case of a crisis, physicians will receive a temporary token that will expire after a certain time. Credit level rules determine what level of data will be seen by each physician (an experienced specialist sees more, a junior one sees less). Similarly, the information in patient files is securely stored in the AWS S3 cloud storage, and access logs and particular rules are the only block chain derivations.

Result:

This provides a safe and clear route through which the appropriate doctor acquires the correct amount of data at the correct time, only in emergencies.

Proposed System:

Our system would be secure, efficient, and audit friendly to allow appropriate emergency access to medical records on patients without violating privacy and regulatory compliances. The idea is to mix blockchain technology, cloud storage and access control based on the level of credit. Doctors will be registered on block chain smart contract with a designated credit level corresponding to authorisation level. Upon submission of an emergency access request, the smart contract will authorize a check that the level of credit allowed by requesting doctor also finds or surpasses the minimum access level requested by the patient. Should it be given the green lights, the system will provide a temporary access token valid only in a short period of time, which would allow the doctor to access the concerned information about the patients. All transactions involving the registration of doctors, approvals and rejections of doctors will be recorded and cannot be tampered with on the blockchain allowing transparency and audits. Medical records of the patient shall be stored in encrypted form in AWS Simple Storage Service (S3) that will enable scalable and secure storage and close control over access policies will remain in place to prohibit unauthorised read access. Using both decentralised verification and

secure cloud-based storage of data and time-limited access tokens, the proposed system aims at ensuring quick and rational decisions during a crisis without the need to disclose patient confidentiality or data integrity.

Technical Methodology

This research has a technical procedure that is divided into four principal steps: the technical development of smart contracts, backend connection, cloud storage, and temporary implementation. This was implemented by creating and implementing an Ethereum smart contract written in Solidity which would be used to register the doctors, give them credit levels and validate access requests. The smart contract was posted in the Ethereum Sepolia test net with the help of infura, through which the execution of a smart contract was delivered with decentralised and publicly verifiable location. All registrations or access requests spawn and on-chain transaction that can be verified individually by Etherscan.

Backend system was designed having Django used as the intermediary between the web interface, blockchain, and cloud storage. Communication to and with block chains was done with the Web3.py library so that the backend could send transactions, call contract variables, as well as process event logs. The layer of cloud storage was to be AWS Simple Storage Service (S3) through which data pertaining to the patients could be uploaded in the encrypted form of CSV file. Even data stored in S3 was inaccessible to the outside world since all the data was under Fine-grained Identity and Access Management (IAM) rules which allowed the backend application and not a user to access it.

The backend generates an access temporary token that will be used when request is authorized in the smart contract to deliver secure data. This is a five-minute valid token that is saved in the database with the details of the doctor who wants and patient ID. The token is then used to access and retrieve the associated data of the patient in a table format in front end. After expiry of the time, the token becomes useless, and one is barred to any further access to the data without reauthorisation. All these features of blockchain verification, cloud storage scalability, and the use of temporary access tokens based on a token create a secure and efficient operation basis of the proposed system.

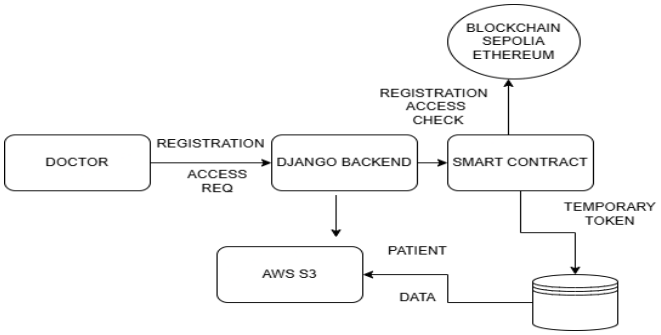

Figure 2: Blockchain-storage combination Credit-Level Based Emergency Access Control System architecture.

Smart Control Workflow

Control System. The procedure starts with a doctor sending a registration request via the frontend leaving its wallet address, name, and the credit level it is assigned. Django backend transmits this data as a large scale in the Ethereum smart contract (registry). The smart contract verifies the availability of the wallet address before registering. In the case where the wallet is present in registry, the contract denies the request and returns an error message, otherwise writes the new doctor and fires a **Doctor Registered** event. All history of registration transactions on the blockchain is tracked and time stamped and forever visible on Etherscan. In patient access requests, the physician picks the required patient on his dashboard and makes a request. The backend uses the smart contract and transfers the wallet information of the doctor, and the patient ID. The contract measures the credit level of the doctor and the access level the doctor required to have on the data of that patient. When the criteria are met, an **Access Granted** event is sent by the contract otherwise one sends an **Access Denied.** When it is cleared up, the backend creates a temporary access token that expires after a period of five minutes, which permits the doctor to retrieve the data associated with the patient securely to AWS S3. Any additional attempts to enter are denied after the token has expired resulting in time-limited but safe emergency access**.**

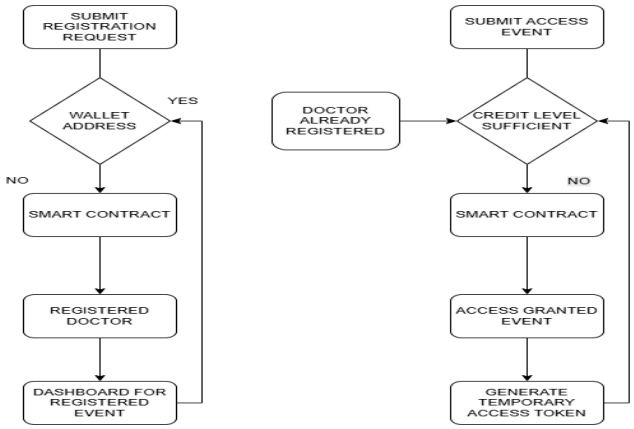

Figure3: Doctor registration and verification of the patient data access workflow through Ethereum smart contract.

AWS S3 Store configuration

The secure layer of storage of patient records, which was set up within a Credit-Level Based Emergency Access Control System, was the AWS Simple Storage Service (S3). Also controlled in an encrypted format in a dedicated bucket with restricted access privileges would be the actual data of patients in the form of CSV files. The Django backend service is the only one allowed to access the data so it must have AWS Identity and Access Management (IAM) policies to do so, and not to access it directly. The bucket has also been set-up with SSE-S3 to serve as an encryption to the data at rest and HTTPS (SSL/TLS) to convey secured data transfer during transit. Also, bucket versioning is activated to maintain an archive of patient data archives that may prove vital at the time of audit and recovery. Its naming and file structure in

the S3 bucket are predetermined such that the organisation can be easily retrieved according to patient IDs. Such construction allows the confidentiality of sensitive medical data, exposure to unauthorised access, and facilitating its retrieval within a short duration during emergencies.

Mechanism of Token Expiry

The token expiry belongs to the system governing how temporary access tokens are created, used and how they can have their validity revoked. A doctor making a successful request to access will have the Django backend make a unique, cryptographically secure access token, same will be stored in the database with appropriate doctor ID, patient ID, and the expiry timestamp. This is the token presented to the doctor which he can use to access patient data on AWS S3 through backend. The backend verifies the token on a record content against the one stored acknowledges both the authenticity as well as expiry of the token. The token will last only five minutes after it is issued. As soon as the time limit is exceeded only further efforts to use the token led to the message of an expired access, and the token will become invalid in the system. This technique can be used to make sure that even in the case where a token might be intercepted or improperly used it cannot be used to access patient records over an extended timeframe greatly minimizing the risk of unauthorised prolonged access to any patient records.

Advantages of the proposed Blockchain Architecture over existing system:

- The suggested system will be associated with a variety of advancements when compared to the current blockchain-related healthcare solutions.
- It spins in the introduction of the fine-grained access control based on credit levels instead of simple authorise/deny logic so that only the doctors with the necessary clearance are allowed to see the specific records. The emergency access is also time-based but with temporary access tokens which last five minutes, useful in cutting the exposure of sensitive data.
- Patient data are securely stored in AWS S3, which has the benefit of scalable fine-grained IAM and the immutability of blockchain data. Any attempt to register a doctor and any attempted access is permanently recorded whereas there is on-chain rejection of duplicate registrations.
- The system provides the support of emergency decision within 12-15 seconds with use of blockchain verification and less than one second of S3 retrieval.
- Such a hybrid model has been balanced between some level of decentralised verification coupled to centralised security controls with an underpinning of the inherent CSRF and SQL injection protections and the tamper resistance of blockchain creating a secure, robust and efficient framework of access control.

Please clarify the novelty of this work and provide evidence that this is research as many similar implementations to this have already been completed. Where is the research contribution of your work. Please clarify?

The innovation of this study is the development of an emergency access model that would not be available in the existing blockchain healthcare models due to the credit level-based access introduced in this study

All current systems record access but not provide temporary tokens and auto-expiry enforcement. They have role-based access, but do not use credit levels to distinguish junior and senior doctors

Most store data on chain or in the privacy of their servers but in this system, it aligns with AWS S3 cloud for file storage in a safe secure and scalable manner. Duplications between doctor registrations are avoided and every action is permanently recorded on the Ethereum blockchain (Etherscan verifiable). This effort is the first research project that entangled credit levels with the emergency-tokens, smart contract expiry, and the blockchain to cloud-integration to make the access to healthcare emergency secure, as well as compliant

Implementation:

Environment Setup

The Credit-Level Based Emergency Access Control System development environment was setup with due consideration to security, scalability and maintainability throughout the lifecycle of the project. The backend has been developed using Python 3.12 and Django 5.X which are also supported having mature ecosystem with a striking collection of security capabilities. Django had all the tools required to allow quick development like authentication, CSRF, and ORM.

Functionality in talking to the blockchain was done through the web3.py library and this library gave the backend the mechanism of talking to the deployed Ethereum smart contract. All sensitive data such as Infura API keys, private keys, contract address and AWS keys were kept secure in one file known as a. env, to avoid accidental loss in a version control. The strategy made sure that no secrets were hard coded in source code. Smart contracts were Solidity written and compiled/deployed using Remix and Hardhat. It was launched in the Ethereum Sepolia test net which is an open blockchain network suitable to be developed and tested. The blockchain RPC provider was Infura, the high availability blockchain and reliable connectivity provider required none of the burden of running a local Ethereum node.

With respect to cloud storage, AWS S3 was set with a private bucket to provide the storage of patient records in encrypted CSV file format. Policies and the use of IAM roles were made using the principle of least privilege, whereas only the backend application has the appropriate read access. This denied any unmediated direct access to patient data and all of its retrieval had to become controlled by backend logic. An isolated, local set of dependencies isolated in a Python virtual environment (venv) brought back the dependency sprawl under control in the

local development environment. The system running was hosted on an EC2 instance based on Ubuntu in terms of production deployment. The setting used Nginx to provide a reverse proxy, an HTTPS termination and Gunicorn as the service to serve the Django application. This especially offered stability as well as security in processing several requests simultaneously in a production environment.

Please clarify the role of cloud services as part of your project?

In this project, the major subsystem is the use of cloud services in data storage and deployment of systems.

AWS S3 – Secure store health records off-chain, ensures scalability of huge medical data. Ensures security with encryption and the IAM access policies. Blockchain only save hashes, permissions and logs, whereas real records are kept S3 in a secure and efficient way.

AWS EC2: This platform has been used as a platform to host and execute the Django app that links blockchain smart contracts with cloud storage. Helps to deliver the backend and the frontend services in a stable computing environment. Handles submission of doctor registration, doctor login and emergency access requests. Adds Web3.py to connect to the Ethereum and Boto3 to communicate with other AWS services.

Smart Contract:

An emergency access control smart contract is the central processing logic of this Ethereum access control and will be written in Solidity to deploy on the Ethereum Sepolia test net. It will ensure registration of the doctors securely, implantation of the policy to access based on credit levels and immutable recording of all events of accessing the information on the blockchain.

The mapping data structure is preserved by the contract to contain the details of the doctors against the Ethereum wallet addresses. All the records of doctors are kept as a structure that contains the name of a doctor, the level of money credit, and the flag indicating the registration status. This is to make sure that multiple registrations are never applied keeping in mind the use of Solidity require statement that accesses whether the wallet address has been registered previously allowing new entries or not.

The major functions that are:

- **Register Doctor (address wallet, string name, unit credit Level)** - This makes the backend authenticated individuals to call newly registering a doctor on blockchain. When the wallet is already registered, then the transaction fails by reversal and error message (Doctor already registered). On success, the method sends out a **Doctor Registered event** with the wallet address, name and credit level.
- **request Access (address doctor Wallet, string patient id) check** whether the doctor making the request has the same credit level or ‘greater than or equal to the threshold set to get the requested record of the patient. When available it returns an **Access Granted event**; otherwise, a reason code in an Access Denied event.
- These **events (Doctor Registered, Access Granted, Access Denied)** are publicly visible, and accessible, on Etherscan where they offer visibility and auditing whereas patient data is never stored on the blockchain. Rather, the content is stored in the

identifiers and access decisions in the blockchain, and the real patient data stays in the secure cloud storage tier.

- The contract has a minimal and modular logic to maximize maintainability and minimize the gas, and such compound tasks as token creation and retrieving data about patients are performed off-chain by Django backend. This vertical split of concerns brings both on-chain efficient execution and backend flexibility and processing power.

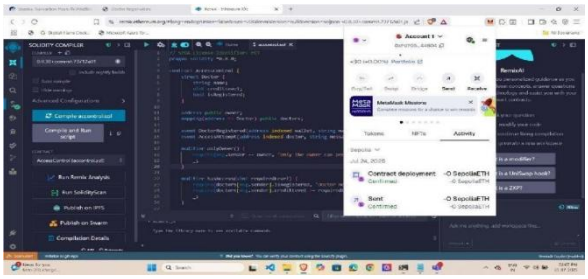

Figure 4: Smart contract Implementation

Contract Deployment / Configuration

The smart contract was deployed to the Ethereum Sepolia test net, selected because of the compatibility with Ethereum mainnet tooling, and because the testing did not require real ETH. Remix IDE was used to complete the deployment with such tools as compilation and automated deployment scripts via hardhat to ensure that the process can be duplicated.

A separate deployment wallet was established having a new contract deployment and management in mind. The private key in this wallet was kept safe and never leaked in any public repositories in the project by being placed in the project.env file. Any back-end interaction with block chain invoked this wallet, which stopped an unauthorised user from calling state-changing functions of contracts directly.

The most important steps of the deployment process were the following:

- Compilation- Solidity contract has been compiled with the newest stable compiler version (pragma solidity ^0.8.x) so that it would work and receive security patches.
- Gas Estimation – Before a deployment, the amount of gas was estimated to prevent transaction failure. They established a conservative gas limit of 3,000,000 units to manage initialisation of the contracts without out of gas failures.
- Infura Integration The deployment script was deployed on the Sepolia network by utilizing an Infura HTTPS RPC endpoint so that no full Ethereum node is required.
- Deployment Execution- The deployment wallet was utilized to deploy the contract, and the contract address was isolated to environment variables so it could be utilized on the backend.
- ABI Export-The ABI exported or stored as a variable contract Abi.json in the Django project so it can be retrieved through Web3.py.

After deployment, the contract had public functions, public events it emitted could be seen through the Sepolia Etherscan and it was possible to track doctor registrations and access control events. The Etherscan was also used to verify the contract source code to create transparency and enable an auditing of the access control logic by third parties.

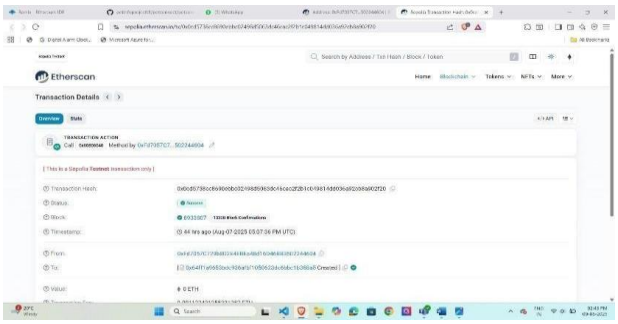

Figure 5: Contract deployment with Successful Transaction

This organized deployment procedure made sure that the contract can be re-deployed or even upgraded later with a minimum disturbance, and that its structure is congruent in the development, assessment and production environments.

Connection to backend (Django):

The Django backend works as the hub between the user interface, blockchain smart contract and the AWS S3 cloud storage. It does input validation, access control is essential, it interacts with the Ethereum blockchain via Web3.py and securely reads patient records on S3.It had a particular service layer so that it could communicate with the blockchain, to insulate low levels of interaction with Web3.py in the core application code. This architecture enabled more clarity in organisation of the code and a more manageable backend. Service layer was able to perform functions aimed at:

The sales will arise as logs in the smart contract. Making requests to access the patients and having them responded on-chain. Reading the indicators that are emitted in the form of Doctor Registered and Access Granted in blockchain logs.

In the back end, models were modelled as the key elements:

- Doctor- stores the doctor details and wallet addresses in case of a quick UI reference.
- Patient- each entry of patients in the CSV file on AWS S3 is represented by a synthetic ID.
- Access Log - every access attempt is hashed along with blockchain transaction hash to be traced. Temporary Access Token- contains token values as well as doctor and patient ids and timing of expiry.

- Form submissions (both the HTML form-based Django registration/log-in process and AJAX/JSON-based access control operations) all passed through views and endpoints in Django. When the access grant was successful, the backend created a cryptographically secure and URL safe token by using the secret library in Python. This token was kept in the database with limited time duration of only five minutes and this way the access became time restricted.

Back-end communication with blockchain was Web3.py with the Sepolia network using Infura. State-changing payments (registration, access request, etc.) were signed locally based on the backend deployment wallets private key, keys were never exposed in the frontend. Gas efficient improvements were made by calling read-only contracts (such as the verification of a doctor registration) without any gas payment.

Integration layer made sure that no direct communication takes place between the frontend and blockchain or S3 thereby imposing security. Every sensitive action went through the authentication, session and CSRF protection mechanisms of Django, which minimised attack surface.

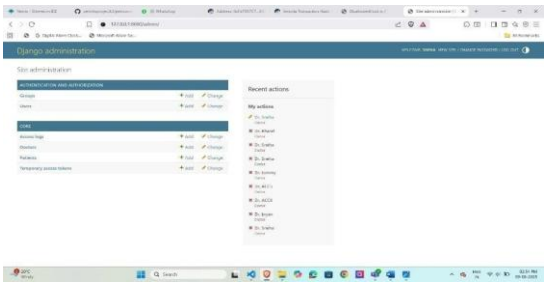

Figure 6 Django admin for all the backend transactions

AWS S3 Integration:

A piece of the puzzle is missing AWS S3 Integration Where is the other half of the border of the tow-colored puzzle? For many years, AWS S3 Integration has been the answer. AWS S3 Integration is the brainchild of software developer Maikel Uxue. It can be used to support the use of the AWS S3 software.

The AWS Simple Storage Service (S3) forms the secure cloud storage level of the patient medical records in the Credit-Level Based Emergency Access Control System. Dedicated, private bucket was created that would store patient records in encrypted CSV files format specially dedicated to this project. These were artificially delegated datasets of patients, and these files have uploaded to S3 bucket after maintaining a strict naming format consisting of patient IDs to allow a fast retrieve option.

There were various levels through which security configurations were used. AWS Identity and Access Management (IAM) policies were set up based on the principle of least privilege and

so the backend IAM user only had access to the specified Get Object operation on the patient bucket. This meant that unauthorised service providers and users could not access information regarding patients or alter it directly. All public access via the bucket was also set out not to happen, decreasing the attack surface even more.

Monitoring The Server-Side Encryption with Amazon S3-managed Keys (SSE-S3) was activated, to guard data at rest. In the case of data at transport, the requests had to be secured via HTTPS with the TLS 1.2 protocol and higher. This inhibited man-in-the middle-attacks and guaranteed that no sensitive data was ever revealed when in transit.

A backend accesses the CSV file in the bucket on an accepted patient access request with the AWS SDK for Python (Boto3). In-memory processing of the file is used to pull out the necessary information of the patient and display it in the frontend that takes the form of a tabular view. The unanalysed CSV is never shown to the doctor (or stored locally) instead the backend only returns the allowed patient attributes based on the doctor credit level. The specified design will make sure that the patient data storage provisions will be scalable, resilient, and safe employing durability and global locations of AWS S3 and tightly integrate with blockchain based access control .

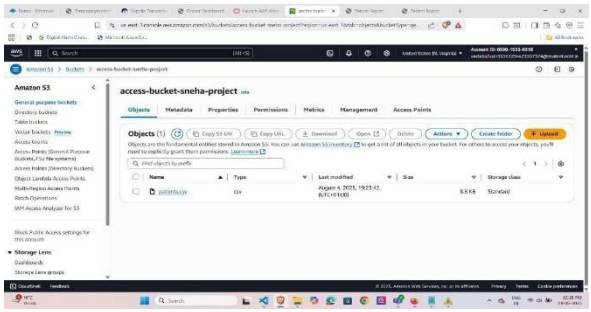

Figure 7 *AWS S3 patients.csv file successfully uploaded*

Frontend Interface

The front end is straightforward, task-oriented to enable doctors to enroll, validate, and order the data of a patient in case of emergency. It is a lightweight HTML / CSS / JavaScript pages enabled with Django templates. Once logged in, a searchable list of patients (name, ID, basic demographics) is presented including a **"Request Access"** button next to each one. A request will condition an asynchronous call to backend (AJAX/Fetch) that communicates with smart contract in turn.
Access can be granted by the backend would respond with a temporary token and the UI redirect to a read-only view of the patient. This view gives a table-like display of permitted fields (e.g. ID, name, DOB, gender, areas of medicine chosen) and starts visible in-progress count-down timer showing how long the token is still usable (five minutes). When expired, the page will automatically cover the sensitive fields and show a notice to re-request access to be consistent with backend enforcement. Any issue connected with error (e.g. duplicate registration, access denied, expired token) will be displayed as a short note and it will have a clear area to recover. The frontend touch never accesses private keys, or AWS credentials, and has no calls direct to the blockchain or S3: everything sensitive travels via authenticated backend endpoints secure with CSRF.

The interface of the front end is un-fussy and work-driven where doctors can access, sign up, and claim the information of a particular patient in case bombarded by an emergency issue. It is a light-footed HTML / CSS / JavaScript pages with Django templates.

After they have logged in, they are given a list of patients (name, ID, basic demographics) with a "Request Access" button next to each of them, which is searchable. There will be an asynchronous call to the backend (AJAX/Fetch) that in its turn talks to smart contract, conditioned with a request.

They can allow access by the backend would respond with a temporary token and the UI jump to a read-only view of the patient. That view provides a table-like presentation of allowable fields (e.g. ID, name, DOB, gender, areas of medicine selected) and initially presents a visible in-progress count-down timer of how much time the token can be used (five minutes). Once expired the page will automatically blanket the sensitive fields and display an advisory to request access to be in sync with backend enforcement.

Anything related to errors (e.g. duplicate registration, access denied, expired token) will be shown as a brief note and there will also be an obvious recovery space. The frontend touch does not access any private keys, and AWS credentials and does not take any direct calls to the blockchain or S3: all sensitive calls pass through API- Keyed authenticated backend endpoints secured with CSRF.

Conclusion:

The system was tested against the functional, performance and security requirement of the system by a comprehensive set of tests that were scenario-driven. The Patient Access Request Module worked as anticipated: the doctors were allowed to make access where they had equal or higher credit, and temporary tokens were granted, whereas lower credit levels would have resulted in on-chain denial messages. The expiry of the tokens was confirmed by blocking access after five minutes. Registration of doctors was effectively registered on the Ethereum Sepolia network and successful duplicate registration was also rejected, which demonstrated immutability and consensus enforcement. The AWS S3 integration got rapid access to patient data (less than one second) and IAM controls averting unauthorised access. The consistency of credit-based policies and the fact that all actions were transparent and were logged on-chain were further confirmed by batch testing of patient records. The interfaces of doctor registration, dashboard and patient information are depicted in figures 7-10. All these outcomes endorse that the system offers secure, auditable and scalable emergency access to patient information, but strictly follows blockchain-confirmed access control.

Figure 7: Doctor registration login page

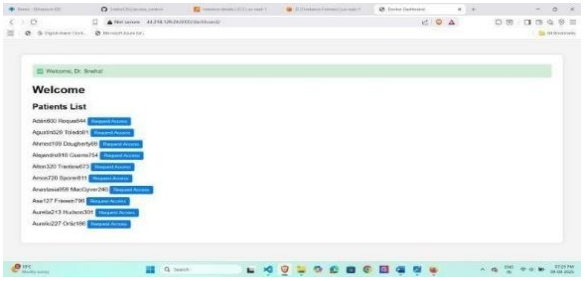

Figure 8: After successful login with welcome text

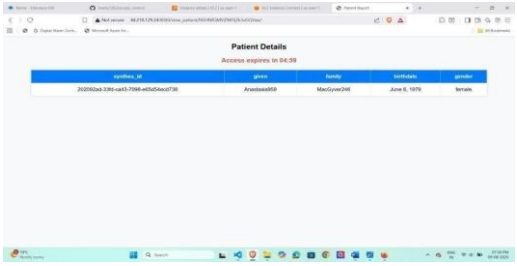

Figure 9: Patient details page after requesting access

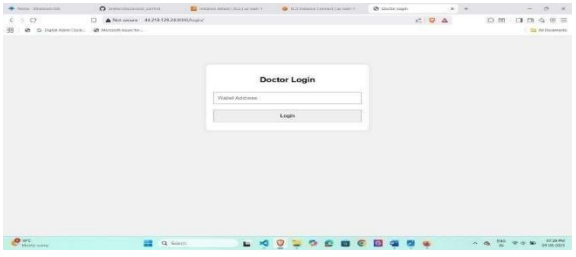

Figure 10: Doctor login page for existing doctors

REFERENCES

1. Karthika, R.A. and P. Sriramya (2020). A Blockchain-Based Access Control System for Cloud Storage. *Algorithms for intelligent systems*, pp.545–554. doi:https://doi.org/10.1007/978-981-15-4936-6_60.

2. Sohrabi, Nasrin, et al. "BACC: Blockchain-Based Access Control for Cloud Data." *Proceedings of the Australasian Computer Science Week Multiconference*, 29 Jan. 2020, https://doi.org/10.1145/3373017.337302

3. Alias, P., S Regilan, R Indumathi, R Sandhiya and Kottaimalai Ramaraj (2023). Blockchain Technology: A Technique to Overcome the Security and Privacy Issues in Cloud Computing. 10, pp.931–938. doi:https://doi.org/10.1109/icacrs58579.2023.10404939.

4. Sharma, P., Jindal, R. and Borah, M.D. (2022). A review of smart contract-based platforms, applications, and challenges. *Cluster Computing*. doi:https://doi.org/10.1007/s10586-021-03491-1.

5. S. Prasath Alias Surendhar, S. Regilan,R. Indumathi, R. Sandhiya, and Kottaimalai Ramaraj, "Blockchain Technology: A Technique to Overcome the Security and Privacy Issues in Cloud Computing," 2023 2nd International Conference on Automation, Computing and Renewable Systems(ICACRS),2023. doi:https://doi.org/10.1109/icacrs58579.2023.1040493

6. Alias, P., S Regilan, R Indumathi, R Sandhiya and Kottaimalai Ramaraj (2023). Blockchain Technology: A Technique to Overcome the Security and Privacy Issues in Cloud Computing. 10, pp.931–938. doi:https://doi.org/10.1109/icacrs58579.2023.10404939.

7. Li, X., Jiang, P., Chen, T., Luo, X. and Wen, Q., 2017. A survey on the security of blockchain systems. Future Generation Computer Systems, 107, pp.841–853. https://doi.org/10.1016/j.future.2017.08.020