

**SCHOOL OF INFORMATION, COMPUTER AND
COMMUNICATION
TECHNOLOGY (ICT), SIRINDHORN INTERNATIONAL
INSTITUTE OF TECHNOLOGY,
THAMMASAT UNIVERSITY**

**Blockchain Capstone Project
Decentralized Application Report: Certificate Authority on Blockchain**

DES484 Blockchain Development

By

Mr. Tipuvanad Youchmuangpri	6322770155
Mr. Chonnapat Leepagorn	6322772532
Mr. Wichapas Tangpremsri	6322772987
Mr. Natthaphat Utako	6322775072

Table of Contents

1. Project Overview	
1.1 Project Goals	1
Project Features:	1
Technical Details:	1
1.2 Problem Domain Understanding	2
1.2.1 Transparency and Trust	2
1.2.2. Removing Middleman Dependency	2
1.2.3. Single Point of Failure	2
1.2.4. Revocation of digital certificates	3
1.2.5.Immutable records	3
2. Case Study Selection and System Architecture	3
2.1 Smart Contract Architecture:	3
Components:	3
Interactions:	3
Technologies and Frameworks:	4
2.2 Decentralized App Architecture:	4
Components:	4
Interactions:	4
Blockchain Network and Framework:	5
3. Smart Contract Development	5
3.1 Design of the Smart Contract	5
3.1.1 Overview	5
3.1.2 Considerations	5
3.2 Functionality of the Smart Contract	5
3.2.1 Overview	5
3.2.2 Bug-Free Implementation	6
3.3 Use of Smart Contract Best Practices	6
3.3.1 Error Handling	6
3.3.2 Implement Access Control	7
3.3.4 Use Events	7
3.3.5 Use Low-level Functions Carefully:	7
3.3.6 Use External Libraries	7
4. Frontend Development	7
4.1 User Interface Design	7
4.1.1 Overview	7
4.1.2 Suitability for the Case Study	9
4.2 Integration with Smart Contract	9

4.2.1 Overview	9
4.2.2 Features	9
5. Testing and Deployment	10
5.1 Testing	10
5.1.1 Thorough Testing	10
5.1.2 Issue Resolution	11
5.2 Deployment	11
5.2.1 Blockchain Network	11
6. Conclusion	11
6.1 Future Work	11

1. Project Overview

This project presents the integration of Certificate Authorities (CAs) with blockchain technology, examining how blockchain's decentralized and tamper-resistant features can enhance the security and trustworthiness of digital certificates. We will focus on immutability, decentralized issuance through smart contracts, transparency, and the potential for decentralized identity systems and present a comprehensive overview of the synergies between CAs and blockchain in the realm of digital certificate management.

1.1 Project Goals

The capstone project aims to design and implement a decentralized application facilitating Certificate Authority by using decentralized application features to enhance its security and integrity. The project includes the development of a smart contract backend and a user-friendly front-end interface.

Project Features:

- Digital Certificate Generating: The application generates and signs the digital certificates on decentralized blockchain technology to improve on the traditional Certificate Authorities.
- Check Digital Certificate status: The application allows the user to request the digital certificate status.
- Revoke Digital Certificate: The applicant can revoke their own certificate. The owner can also do the same thing to all digital certificates.

Technical Details:

- Smart Contract: Developed in Remix, the smart contract ensures transparency and decentralization of contract creation, contract revocation, and certificate checking.
- Deployment: Utilize web3 architecture and JavaScript the smart contract can be used on websites. In this project, the smart contract will be deployed on Ganache which is an Ethereum network chain simulation.
- Testing: The testing process started in Remix IDE which is used for testing the smart contract functionality. The website also tested connectivity with smart contracts on a Python Flask server.

1.2 Problem Domain Understanding

Before the design, the Traditional Certificate Authority faced problems with transparency and single-point failure. We as a team believe that decentralization in Blockchain technology is a game changer to these problems. Blockchain from its terminology is defined as a distributed ledger, which means that it has a resistance to a single-point failure. The decentralization feature of blockchain also provided transparency in contract creation and revocation. The bonus from Blockchain technology is an immutable feature, which strengthens Certificate Authority transparency.

1.2.1 Transparency and Trust

Problem: Traditional Certificate Authority systems lack transparency, leading to concerns about Trust Dependency.

Blockchain Solution: Utilizing Blockchain technology we created transparency digital certificates creation and revocation. Which could elevate the transparency of a Certificate Authority to the next level. Users might have a trust issue with the Traditional Certificate Authority, but with the integration of blockchain technology users are able to verify the digital certificate's legitimacy by themselves.

1.2.2. Removing Middleman Dependency

Problem: Traditional Certificate Authority depended on Certificate Authority which centralized the creation and revocation of the digital certificate.

Blockchain Solution: Decentralization in Blockchain eliminates this problem, but in this case, the Certificate Authority is still necessary for the system to check the legitimacy of the website. Still, users can track the working process of the Certificate Authority.

1.2.3. Single Point of Failure

Problem: Traditional CAs are central points of trust. If a CA is compromised, it can lead to the compromise of a large number of certificates, affecting the security of the entire system.

Blockchain Solution: Blockchain is inherently decentralized, meaning there is no single point of control or failure. By distributing the authority across a network of nodes, the system becomes more resilient to attacks.

1.2.4. Revocation of digital certificates

Problem: Revocation in traditional Certification Authority has proved to be inefficient.

Blockchain Solution: transparency in Blockchain can help in verifying the actions of the CA and ensuring that certificates are issued and revoked correctly.

1.2.5. Immutable records

Problem: If the traditional Certification Authority if the CAs faced a cyber attack the record of digital certificates may have been tampered with.

Blockchain Solution: The immutable feature of Blockchain can address this problem. It makes it almost impossible to tamper with any record stored in the Blockchain. Hence, acquired immutable and trustworthy records.

While deploying a certificate authority on a blockchain has potential benefits, it is critical to evaluate the capabilities, scalability, and smart contract architecture of the specific blockchain platform. Furthermore, further implementation of blockchain-based CAs would be dependent on industry standards, interoperability, and regulatory concerns.

2. Case Study Selection and System Architecture

The chosen use case focuses on "Certificate Authority on Blockchain." The justification for this selection is:

- **Transparency and Trust:** Utilizing blockchain technology ensures transparency in certificate-generating process transactions, building trust by providing a verifiable and immutable record of Digital Certificates.

- **Decentralization:** By decentralizing the fundraising process, the application reduces reliance on centralized authorities, promoting a more inclusive and open fundraising environment.

2.1 Smart Contract Architecture:

Components:

- Transaction Handler: Manages in generating, and signing the certificates.
- Data Storage: Stores immutable records of digital certificates.
- Event Emitter: Triggers events for updating certificates, and registering the user.

Interactions:

- User Interaction: Users interact with the smart contract primarily through registering accounts, generating certificates, revoking certificates, and checking user domain names.
- Decentralized App Interface: The decentralized application interfaces with the smart contract to retrieve real-time certificate data for display to users.
- Blockchain Network: The smart contract interacts with the underlying blockchain network, recording transactions on the immutable ledger.

Technologies and Frameworks:

- Solidity: Utilized for writing the smart contract code, ensuring compatibility with the Ethereum blockchain.

- Web3.js: Facilitates communication between the decentralized app and the smart contract, enhancing user interaction.
- Python Flask: Flask is a web framework, a Python module that lets you develop web applications easily. It has a small and easy-to-extend core
- Ganache: Personal Ethereum chain network. Used for running tests, executing commands, and inspecting state for smart contracts testing.

2.2 Decentralized App Architecture:

Components:

- Front-End Logic: Oversees the communication between the smart contract and the user interface, making sure that everything works well.
- User Authentication: Secures user accounts and ensures the integrity of user information.

Interactions:

- Smart Contract Integration: The decentralized app interacts with the smart contract to register and check certificate status, ensuring real-time updates for users.
- User Interaction: Users navigate the UI to make registration, generate certificates, revoke certificates, and check user domain names.
- Blockchain Network: The decentralized app communicates with the Ethereum blockchain to ensure secure and transparent contract creation and revocation.

Blockchain Network and Framework:

- Consensus Mechanism: Enables secure, decentralized transaction validation by utilizing Ethereum's consensus mechanism, such as Proof-of-Stake.
- Smart Contract Deployment: By deploying the smart contract to the blockchain network, all participants are ensured accessibility and visibility.
- Network Nodes: Nodes propagate and validate transactions, preserving the blockchain's integrity.
- Ethereum Framework: Selected for offering a strong foundation for our CA on Blockchain with its well-established and secure decentralized platform.

This architecture is designed to mitigate the frustration of traditional Certificate Authority. Which lacks a decentralized and transparent working process. The design is aimed at creating a more proper version of Certificate Authorities in terms of transparency and eliminating single-point failure which can cause business dysfunctionality.

3. Smart Contract Development

3.1 Design of the Smart Contract

3.1.1 Overview

In this project, 2 individual smart contracts operate in different roles which are the user registration system and certificate verification system:

- User registration system: This blockchain-driven smart contract transforms user registration into a seamless and secure interaction. Users engage with the webserver to register their accounts and use it as an authority checker module that will be sent to the CA API in order to check if this account really exists.
- Certificate verification system: It takes the role of keeping the certificate and also acts like CRL allowing the user to check the certificate status.

3.1.2 Considerations

- Security: Implemented security measures to reduce the chances of successful attacks that occur against traditional CAs.
- Transparency: Allow anyone to check the transaction records and avoid data tampering.
- Reliability: Ensured the contract's reliability through rigorous testing.

3.2 Functionality of the Smart Contract

3.2.1 Overview

The procedure starts with the user register their accounts within the blockchain network via dedicated smart contract #1. Once registered, users then proceed to generate a cryptographic key pair and Certificate Signing Request (CSR), which are subsequently transferred to the Certificate Authority(CA) API. The Certificate Authority(CA) API has a role in pivotal, signing the certificate, and returning it to the web server. Furthermore, in instances where users already possess outdated certificates, the outdated certificate will be revoked by the Certificate Authority (CA) and stored in the Certificate Revocation List (CRL).

The Certificate Verification process is employed to ensure the integrity and security of digital certificates. Initially, users register their accounts through smart contract #1. Subsequently, when certificate validation is required, a request is made through another smart contract #2, to obtain a Certificate Revocation List (CRL). Finally, users can verify the status of their certificates by checking the CRL for revocation information, adding an essential layer of security and trust to the system.

Smart contract #2 serves a dual purpose by actively engaging in the detection of existing blockchain account addresses within a certificate status list. In the event that a match is identified, the contract takes decisive action by revoking the previous certificate associated with the implicated account and seamlessly initiates the process of generating a new certificate. This functionality ensures that the blockchain remains updated and reflective of the latest and most accurate certification status for each account, bolstering the system's integrity and responsiveness.

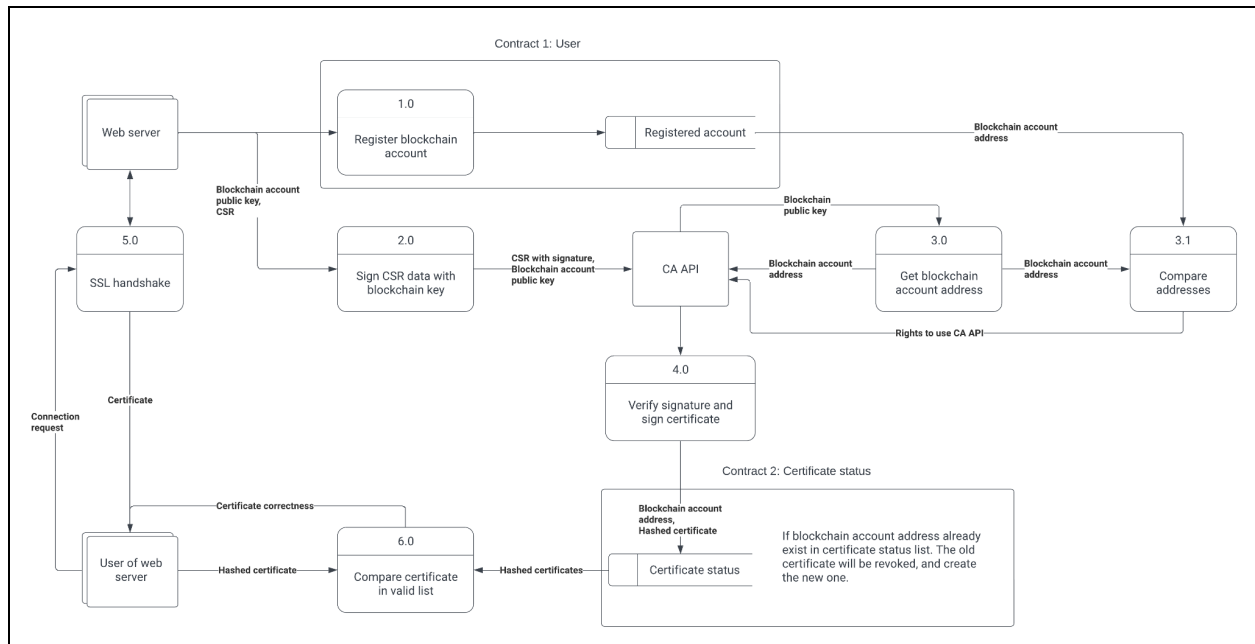


Figure 1: process of CA on blockchain

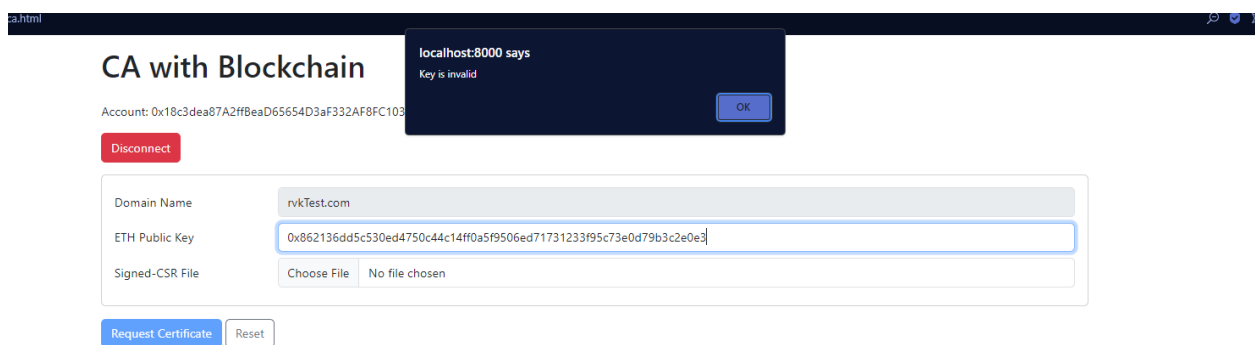
3.2.2 Bug-Free Implementation

A thorough testing process was conducted to identify and rectify any bugs, resulting in a robust and bug-free smart contract.

3.3 Use of Smart Contract Best Practices

3.3.1 Error Handling

Inputting the wrong key type will result in an error message.

Figure 2: **Error message**

3.3.2 Implement Access Control:

Access control ensures that only authorized parties can execute certain functions in the smart contract. It is important to implement access control to prevent unauthorized access to the smart contract.

3.3.3 Secure Coding Practices

Followed industry-standard secure coding practices to mitigate potential vulnerabilities.

3.3.4 Use Events:

Events provide a way for the smart contract to communicate with the outside world. They can be used to notify users of important events or to provide data for off-chain analysis.

3.3.5 Use Low-level Functions Carefully:

Low-level functions like `call()`, `send()`, and `delegatecall()` can be powerful but also risky. They should be used with caution and only after careful consideration of the security implications.

3.3.6 Use External Libraries:

Using external libraries can help reduce code complexity and improve code quality. It is important to vet any external libraries carefully and ensure they are secure and reliable.

4. Frontend Development

4.1 User Interface Design

4.1.1 Overview

The user interface is thoughtfully designed, offering an intuitive experience for users. It contains 5 main pages: Homepage, Register, Signing, CA, and Revoke.

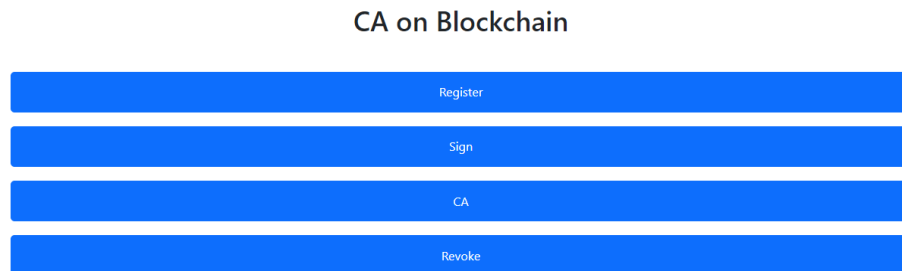
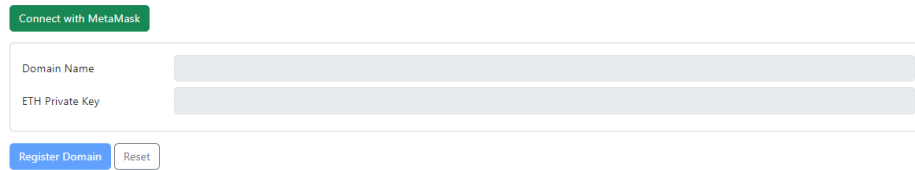


Figure 3: **Homepage**

The first page when you use our website is our home page. This page contains a button that will navigate you to other pages.

Register

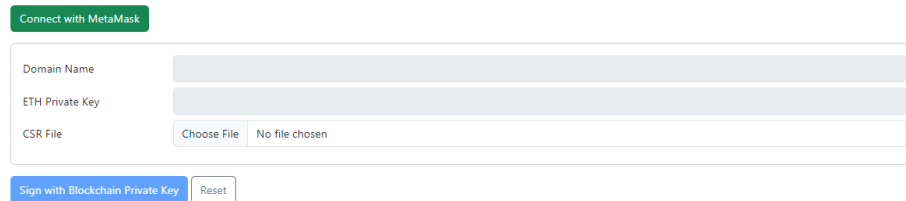


The registration form features a green 'Connect with MetaMask' button at the top. Below it, there are two input fields: 'Domain Name' and 'ETH Private Key'. At the bottom, there are two buttons: 'Register Domain' and 'Reset'.

Figure 4: **Registration Page**

Upon clicking on the register button on the home page, it will take you to our registration page. On this page, users need to connect their account with a digital wallet using MetaMask. Users need to input their domain name and their ETH account private key in order to register for an account in our system.

Ethereum CSR Signer

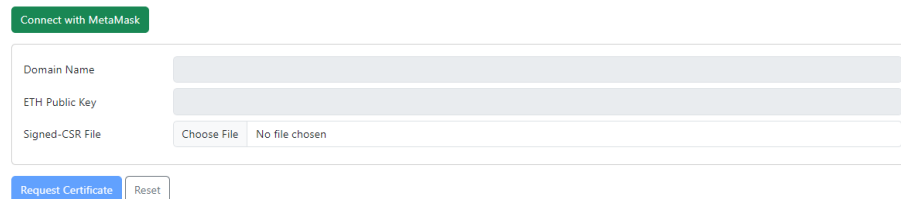


The Ethereum CSR Signer form includes a green 'Connect with MetaMask' button. It contains three input fields: 'Domain Name', 'ETH Private Key', and 'CSR File'. The 'CSR File' field has a 'Choose File' button and a 'No file chosen' status. At the bottom, there are two buttons: 'Sign with Blockchain Private Key' and 'Reset'.

Figure 5: **CSR Signing Page**

After registering, the user can go back to the home page in order to go to the sign page. To create their CSRS file. Users also need to connect with MetaMask to continue. Users need to provide their ETH Public key and CSR file for signing. The CSR file can be generated using a third-party website. There is no need in filling the domain name since it is bound to an account.

CA with Blockchain

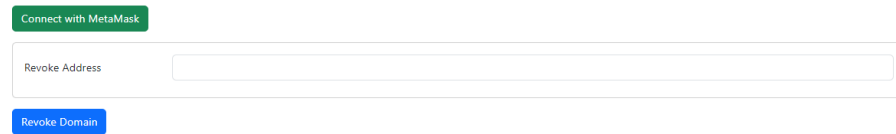


The CA with Blockchain form features a green 'Connect with MetaMask' button. It includes three input fields: 'Domain Name', 'ETH Public Key', and 'Signed-CSR File'. The 'Signed-CSR File' field has a 'Choose File' button and a 'No file chosen' status. At the bottom, there are two buttons: 'Request Certificate' and 'Reset'.

Figure 6: **SCSR Signing Page**

Users can then sign their CSR file using the CA button to go to this page. The owner needs to fill up their ETH private key and input their CSR file. After that, they will gain the certificate for their website in the form of domain_name.pem file.

Revoke Domain

The form is titled "Revoke Domain". It features a green button labeled "Connect with MetaMask" at the top. Below this is a text input field with the placeholder text "Revoke Address". At the bottom of the form is a blue button labeled "Revoke Domain".

Connect with MetaMask

Revoke Address

Revoke Domain

Figure 7: **Revoke Page**

This page is exclusively for the owner of the domain and the owner of the contract. This page will revoke any certificate. All the user needs to do is simply input their domain into the form and click revoke, it will immediately revoke the certificate.

4.1.2 Suitability for the Case Study

The design corresponds to the traditional Certificate Authority system, but with the implementation of blockchain technology, we are able to decentralize the Certificate Authority.

4.2 Integration with Smart Contract

4.2.1 Overview

The front end is integrated with the smart contract, allowing users to interact with the decentralized application. It allows users to register to the system and create or revoke their digital certificate.

4.2.2 Features

Digital Certificate Generating: The application generates, signs, and revokes the digital certificates on decentralized blockchain technology to improve on the traditional Certificate Authorities.

Check Digital Certificate status: The application allows the user to request for the digital certificate status.

5.1.2 Issue Resolution

Identified and resolved issues promptly to ensure the reliability and stability of the application.

5.2 Deployment

5.2.1 Blockchain Network

Successfully deployed the application to a blockchain network, ensuring operational efficiency and security. Deployed in the Ganache environment and Web 3.0 in Flask.

6. Conclusion

We have observed that when a user initiates a certificate request using the signed CSR, along with the Blockchain public key and the user's Blockchain account address, a certificate is generated through a CA API. Following signature verification and certificate signing, the finalized certificate is stored in a blockchain smart contract. This contract can assess the status of each certificate and facilitate its return to the user, the digital certificate can get attacked by tampering, and the traditional CAs are centralized. So, we created a system model to implement with Blockchain. It will prevent tampering attacks and make CAs to be decentralized.

6.1 Future work

We intend to improve our system to the next level in the following stage. Scalability, we will involve the development and adoption of scalable blockchain solutions that can accommodate the increasing volume of digital certificates securely and efficiently, Integration with Emerging Technologies by incorporating other emerging technologies, such as zero-knowledge proofs, homomorphic encryption, and secure multi-party computation, and improve features of our web server such as display certificate status. These technologies can enhance the efficiency, privacy, security, and overall robustness of certificate management systems.