Comprehensive Report on Blockchain-Based Credential Verification System

AI Technical Report Writer

November 20, 2025

Abstract

This report details the design and architecture of a digital credential verification system utilizing blockchain technology. The primary goal is to solve the real-world problem of credential fraud and difficult third-party verification by creating immutable, transparent, and decentralized records of academic achievements and competitive awards. The proposed system is centered on the ERC-721 Non-Fungible Token (NFT) standard to ensure the uniqueness and non-interchangeability of each issued credential.

Contents

1 Introduction and Problem Statement

The current process for verifying academic and competitive achievements is prone to failure, relying heavily on easily forged or damaged physical documents (PDFs, paper certificates) or necessitating manual contact with the issuing institution. This creates a significant bottleneck for employers and universities during the admissions or hiring process.

1.1 The Core Problem: Lack of Verifiability

Traditional credentials suffer from three main weaknesses:

1. **Forging**: PDF or paper certificates are simple to alter or replicate without detection.
2. **Loss/Damage**: Physical documents are susceptible to loss, requiring time-consuming re-issuance.
3. **Centralized Verification**: Verifying authenticity requires contacting a centralized institution, which introduces delays and friction.

The **Blockchain-based Credential Verification Project** fundamentally addresses this by replacing trust in a physical document with cryptographic proof stored on a distributed ledger.

1.2 Why Blockchain is Essential

Blockchain technology is the foundational layer for this project because it provides critical, non-negotiable features:

- **Immutability**: Once a record is permanently added to the ledger, it cannot be altered or deleted, preventing credential faking.
- **Transparency**: The record's existence and authenticity can be instantly verified by anyone without needing to trust a single authority.
- **Decentralization**: The records are stored across a network of computers, eliminating any single point of failure (SPOF) that could lead to data loss or manipulation.

2 Credential and Award Verification Mechanism

The system treats both course completions and competitive awards as unique, verifiable digital assets. The core mechanism involves an issuing institution creating a permanent record on the blockchain upon the achievement of a milestone.

2.1 Awards in the System

Awards are recognized similarly to certificates but represent competitive or merit-based achievements. This system provides a verifiable proof-of-achievement for:

- **Academic Competitions**: Hackathons, Math Olympiads, Coding contests.
- **Recognition**: Leadership, Innovation, or Best Project awards.

The traditional problem with such awards is that a physical trophy or social media announcement lacks cryptographic proof. The blockchain solution ensures that the award record, including the specific achievement (e.g., "1st Place - AI Challenge 2024"), date, and an official digital signature from the organizer, is publicly verifiable and cannot be fabricated.

2.2 Practical Verification Flow

A verifier (e.g., an employer) can instantly validate a credential by using a unique credential ID or scanning a QR code.

1. The student provides a QR code (or ID) linked to the blockchain record.
2. The verifier's system queries the blockchain network.
3. The system confirms the credential is: **Verified by** [Issuer Name], **not Revoked**, and represents the **Immutable Record** of the achievement.

3 Token Architecture Specification

The choice of token standard is crucial for ensuring the integrity and uniqueness of each credential.

3.1 NFTs vs. Fungible Tokens

The system must utilize **Non-Fungible Tokens (NFTs)**, specifically the **ERC-721** standard.

- **NFTs (✓ Perfect Fit)**: Each ERC-721 token is unique and non-interchangeable. A certificate issued to 'Alice' is fundamentally different from the same certificate issued to 'Bob', even if the course is identical. They represent unique ownership of a specific achievement.
- **Fungible Tokens (× Not Suitable)**: Fungible tokens (like ERC-20) are interchangeable (e.g., one currency unit is equal to another). Using them would imply that one person's degree could be traded for another's, which is logically unsound for credentials.

The entire architecture is designed around the **ERC-721** standard on an EVM-compatible blockchain.

3.2 Roles and Access Control

To maintain security and integrity, the system implements specific roles, typically managed via an access control mechanism like OpenZeppelin's `AccessControl`:

- **ADMIN_ROLE**: Manages the system (assigns/revokes `ISSUER_ROLE`, optional contract pausing).
- **ISSUER_ROLE**: Authorized to mint new credential NFTs and mark them as revoked.
- **STUDENT (Holder)**: Receives and owns the NFT credential.
- **VERIFIER**: Public role with read-only access to query the contract and verify credentials.

3.3 NFT Data Model (Metadata Fields)

Data is strategically split between on-chain and off-chain storage to optimize for cost (gas fees) and privacy.

3.3.1 On-Chain Data (Smart Contract $\mathbb{S}$)

The smart contract stores minimal, essential data within a struct (e.g., `CredentialData`) for fast, low-cost verification queries:

```
bluestruct CredentialData {
    address issuer;
    address recipient;
    CredentialType ctype; gray//gray grayEnumeratedgray graytypegray gray(grayegray.grayggray.,gray
        grayCOURSEgray,gray grayAWARDgray)
    string title;
    string institution;
    uint64 issueDate; gray//gray grayUnixgray graytimestamp
    uint64 expiryDate; gray//gray gray0gray grayifgray graynogray grayexpiry
    bytes32 credentialHash;gray//gray graySHAgray-256gray grayhashgray grayofgray grayoffgray-graychain
        gray graydata
    bool revoked;
}
```

The most critical field is `credentialHash`. This is a cryptographic hash (e.g., SHA-256) of the full, detailed off-chain document (PDF or JSON). Verifiers use this hash to cryptographically check that the retrieved off-chain file has not been tampered with.

3.3.2 Off-Chain Data (IPFS $\mathbb{I}$)

Rich, verbose data—such as full names, detailed descriptions, and project titles—is stored off-chain, typically on a decentralized file system like **IPFS (InterPlanetary File System)**. This off-chain data is referenced by the ERC-721 standard's `tokenURI` function.

3.4 Core Smart Contract Functions (API Outline)

Key functions enable the life-cycle management of a credential:

- **Issuance**: `function issueCredential(address to, CredentialInput input) external onlyRole(ISSUER_R` Mints a new `tokenId` to the student's address and stores the on-chain `CredentialData`.
- **Verification**: `function isValid(uint256 tokenId) external view returns (bool)`: Performs a quick check to ensure the token exists, is not revoked, and is not expired.
- **Metadata Access**: `function tokenURI(uint256 tokenId) public view override returns (string memory)`: Returns the IPFS URI pointing to the detailed off-chain metadata.
- **Revocation**: `function revokeCredential(uint256 tokenId, string calldata reason) external onlyRole(ISSUER_ROLE)`: Permanently flips the `revoked` flag to `true` in the `CredentialData`.

4 Bloc Architecture and Privacy

The "Bloc Architecture" describes how the data is written to and read from the underlying blockchain.

4.1 On-Chain Events and Transactions

Every key action is recorded as a transaction (Tx) and emits an Event, making the history of the credential fully auditable:

- **Credential Issuance**: The Tx calls `issueCredential()`, writing the new `CredentialData` to storage and updating ERC-721 ownership maps. A `CredentialIssued` event is logged.
- **Credential Revocation**: The Tx calls `revokeCredential()`, changing only the `revoked` state variable and logging a `CredentialRevoked` event.

Crucially, credentials in this system should be **non-transferable** (often called "Soulbound Tokens"). This is enforced by overriding the standard ERC-721 transfer functions (`transferFrom`, `safeTransferFrom`) to `revert()` the transaction, ensuring the credential stays linked to the original recipient's wallet.

4.2 Verification Data Flow

The full verification process requires reading both on-chain and off-chain data:

1. **Retrieve Pointers**: Verifier uses `tokenId` to call `getCredential()` and `tokenURI()`.
2. **Fetch Data**: The off-chain metadata (JSON/PDF) is retrieved from the URI (e.g., IPFS).
3. **Cryptographic Check**: The verifier's system recomputes the hash of the downloaded file and compares it to the `credentialHash` stored on-chain.
4. **Status Check**: The system confirms `!revoked` and `now` $\leq$ `expiryDate`.

If the hash matches and the status is valid, the credential is confirmed to be **authentic and untampered**.

4.3 Privacy Considerations

To meet privacy requirements (NFR3), Personally Identifiable Information (PII) like the student's full name or ID must be stored **off-chain** only. The on-chain record only stores the cryptographic `credentialHash` and the `recipient` wallet address, preserving the student's pseudonymity on the public ledger.

5 System Architecture and Requirements (SRS)

The system is defined by its functional and non-functional requirements, which dictate the high-level architecture.

5.1 Functional Requirements (FR)

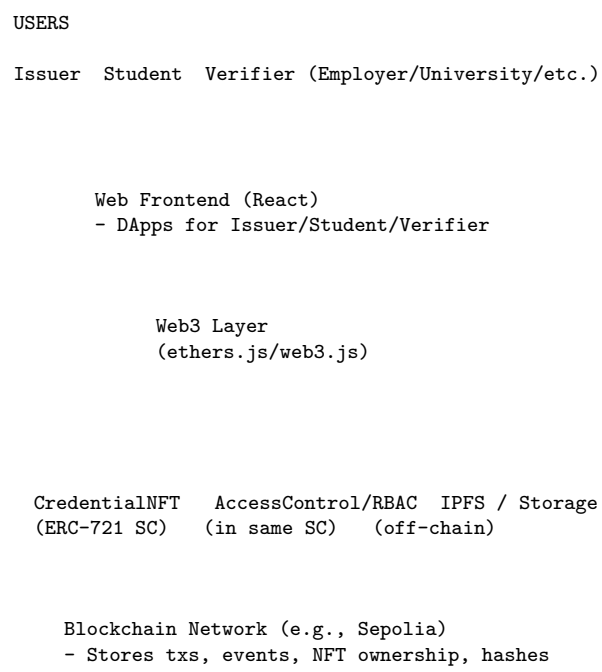
- **FR1 - Authorization**: Only authenticated `ISSUER_ROLE` addresses can perform minting/revocation.
- **FR2 - Issuance**: System must mint a new credential NFT with appropriate metadata and a cryptographic hash.
- **FR3 - Types**: Support for `COURSE_COMPLETION`, `COMPETITION_AWARD`, and `PROJECT_VALIDATION`.
- **FR5 - Public Verification**: Any party must be able to check the on-chain status (valid/revoked/expired) using the credential ID.
- **FR6 - Revocation**: Must allow issuers to permanently mark a credential as revoked, visible to all verifiers.

5.2 Non-Functional Requirements (NFR)

- **NFR1 - Security**: Rely on the blockchain's consensus and cryptographic primitives (ECDSA/EOA accounts).
- **NFR2 - Immutability**: Once issued, key data (hash, issuer, date) cannot be changed.
- **NFR4 - Scalability & Cost**: Minimal data stored on-chain to reduce gas costs; bulk data stored off-chain (IPFS).
- **NFR6 - Interoperability**: Use open standards (ERC-721, JSON) for compatibility with wallets and explorers.

5.3 High-Level Component Diagram

The architecture is divided into three layers: Users, the Web Frontend/Web3 layer, and the Decentralized Ledger.



6 Summary and Future Work

6.1 Summary

The Blockchain-based Credential Verification Project provides a robust, transparent, and auditable solution to credential fraud. By leveraging **ERC-721 NFTs** and ensuring core data integrity through the use of on-chain **cryptographic hashes** of off-chain metadata, the system eliminates the reliance on easily manipulated paper or centralized databases. The architecture is a pure decentralized application (DApp), relying solely on the frontend (DApp interface), the smart contract (backend logic), and the blockchain (database/ledger).

6.2 The Backend Question: Is it Necessary?

The short answer is no. For the core functionality of issuance, revocation, and public verification, a traditional centralized backend server is **not technically required**.

The **Smart Contract** acts as the decentralized, transparent backend logic. The **Blockchain** acts as the immutable, consensus-driven database. The **Frontend DApp** (using libraries like `ethers.js` or `web3.js`) interacts directly with these decentralized components.

A centralized backend would only be necessary if the project required:

- Complex off-chain data indexing (e.g., a high-speed search API not suitable for direct blockchain queries).
- Traditional user accounts and password management (if not relying solely on wallet connectivity like MetaMask).
- Advanced data analytics, bulk processing, or legacy system integration that the smart contract cannot handle.

For a basic, secure, and fully decentralized proof-of-verification system, the existing DApp architecture is sufficient and preferred.

6.3 Future Work

Potential areas for future expansion include:

- **Decentralized Identity (DID)**: Integrating with DID standards (e.g., W3C Verifiable Credentials) to link the credential NFT to a self-sovereign identity framework.
- **Layer 2 Integration**: Migrating the contract to a Layer 2 (L2) scaling solution (e.g., Polygon, Optimism) to further reduce gas costs for issuers and increase transaction throughput.
- **Advanced Revocation**: Implementing a more granular revocation system, such as a time-locked challenge period before final revocation.

7 The Technology Stack and Development Environment

This section details the complete technology stack used to develop and deploy the simple voting DApp. The architecture adheres to the standard Web3 development model, which separates the core, immutable application logic (On-Chain) from the user-facing interface (Off-Chain).

7.1 The Three-Layer DApp Stack

1. **On-Chain Layer (The Decentralized Backend)**: This layer is the permanent, trustless core of the application. It consists of the smart contract logic and the underlying blockchain network.
2. **Interface Layer (The Off-Chain Client)**: This is the user-facing web application (HTML/-JavaScript) that provides the user interface (UI) and acts as the bridge to the blockchain.
3. **Wallet/Provider Layer (The Bridge)**: This tool manages the user's digital identity and cryptographically signs transactions, connecting the interface to the network.

7.2 Core Tools and Components

The following table summarizes the specific tools utilized in the lab exercise based on these architectural layers.

Component	Technology / Tool	Role in the Lab (*Voting DApp*)
I. On-Chain Logic and Runtime		
Language	**Solidity**	High-level, statically-typed language used to define the core Voting.sol contract, including candidate lists and vote counting logic.
Platform	**Ethereum Test Network**	The runtime environment (e.g., Rinkeby or Goerli) where the contract is deployed. It processes and permanently records transactions.
Currency	**Test Ether (ETH)**	Used to pay for **Gas** (the computational fee) required for contract deployment and state-changing transactions (i.e., casting a vote).
II. Development and Deployment Environment		
IDE	**Remix IDE**	The browser-based Integrated Development Environment used to write, compile, debug, and facilitate the one-click deployment of the Solidity contract.
Provider	**Injected Web3**	The deployment setting in Remix that connects the IDE directly to the MetaMask provider in the browser for transaction signing.
III. Client Interface and Bridge		
Wallet	**MetaMask**	The essential browser extension that manages the user's accounts, provides an identity (EOA), and cryptographically signs transactional calls from the DApp.
Library	**Web3.js**	The JavaScript library used in the index.html file. It enables the frontend to initialize the contract object (ABI and Address) and send read (totalVotesFor) and write (voteForCandidate) calls to the deployed contract.
Frontend	**HTML/JavaScript**	Used to create the simple user interface, capture the candidate input, trigger the voting transaction, and display the vote results in real-time.