The social platform that realizes blockchain authentication even in environments without the internet

1. Certificate Validation Tools [1]

1.1 Functional Description

This system is a certificate validation tool designed for use in disaster situations.
It provides an offline-compatible validation environment that operates as a standalone system. The tool ensures the prevention of data tampering in certificates, verifies the legitimacy of issuers, and validates the integrity of blockchain data. By doing so, it supports highly reliable credential verification in disaster relief operations.

The system consists of the following two modules:

- Block Verification Module
- Certificate Verification Module

1.1.1 Block Verification Module (cardano_verifier.mjs)

1.1.1.1 Purpose

- To verify that the block data containing credential certificates maintains correct integrity.

1.1.1.2 Functions

- The system verifies the IPDC hash of each block and the link to the previous block to ensure that no data tampering has occurred.
- The results of block integrity verification are recorded as logs, providing trusted data that can be used in subsequent certificate verifications.

1.1.2 Certificate Verification Module (validation.html)

1.1.2.1 Purpose

Based on the credential certificate data, issuer information, and block data provided by the user, this module performs the following three types of verification:

- **Tamper-resistance Verification**

  ・Compares the certificate data and its digital signature to confirm that no tampering has occurred.

  ・Verifies the signature using the Ed25519 signature algorithm.

- **Issuer Legitimacy Verification**
  Confirms that the issuer is a trusted organization by matching the issuer information in the certificate against a known public key.
- **On-chain Integrity Verification**
  Verifies the transaction hash and Merkle path included in the certificate to ensure that the credential certificate has been properly recorded on the blockchain.

1.1.2.2 Main Functions

- Loads and processes the necessary files, including the credential certificate, issuer information, and IPDC reception data, for verification.
- Displays the verification results and detailed logs on screen, allowing users to confirm the integrity of the credential certificate.
- Enables export of the verification results in JSON format, allowing the data to be saved or shared.

1.1.2.3 Key Features

- Standalone Operation
  This system operates without requiring an internet connection, enabling verification in offline environments. It allows for rapid integrity checks of credential certificate data even in disaster relief sites or situations with no network access.

- Blockchain-Based Reliability
  Through block and certificate verification, the system ensures that credential certificate data has not been tampered with. It also confirms the legitimacy of both the issuer and the blockchain record.

- Simple Usability
  Users can complete all three verification steps—tamper-resistance, issuer legitimacy, and on-chain integrity—by simply loading the required data files and pressing the verification button.

- Clear Presentation of Results
  Verification results are displayed intuitively as success or failure. Detailed logs can also be reviewed to substantiate the reliability of the credential certificate data.

This system enables highly reliable verification of the qualifications of volunteers and support personnel during disaster relief operations. Even in environments with unstable network conditions, it can operate completely in a standalone mode, allowing for rapid and accurate credential verification.

1.2 System Architecture

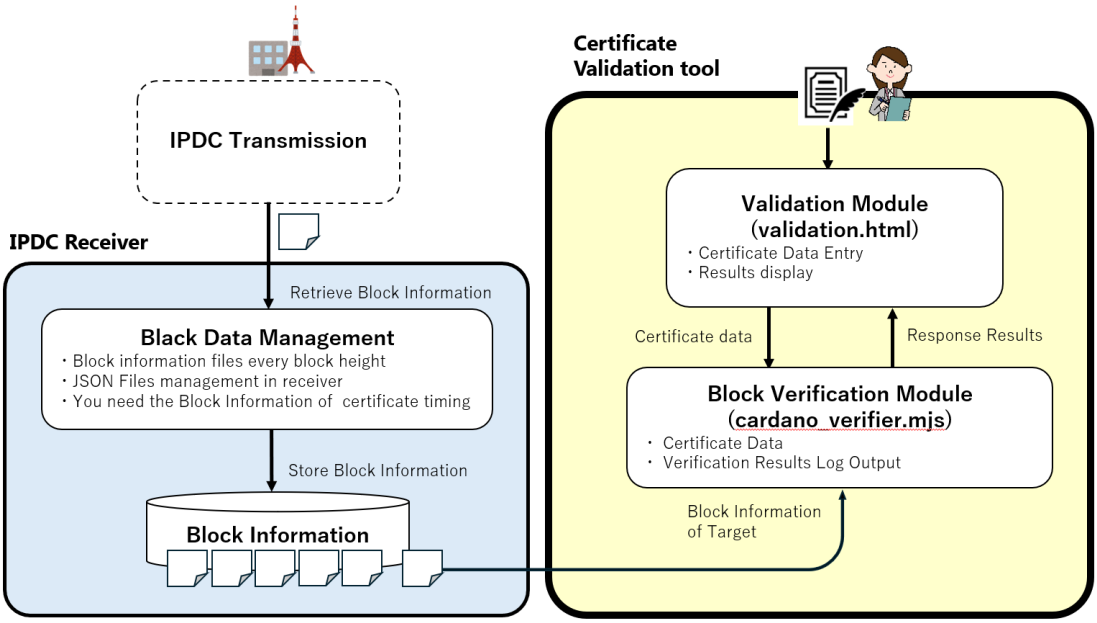
This system utilizes two modules in sequence to verify the integrity of credential certificate data in disaster relief environments.
First, the **Block Verification Module** checks the integrity of the blockchain data. Based on the results, the **Certificate Verification Module** then verifies the authenticity of the credential certificates.

1.2.1 System Configuration Diagram

This system consists of three main components:

IPDC Receiver for receiving block information via IPDC transmittion,
Verification Processing Module (cardano_verifier.mjs), and
Verification Display Module (validation.html).

The verification processing module operates in coordination with the block information files managed under the block data received via IPDC. Together, they implement the entire process from inputting certificate data to blockchain verification and presenting the verification results.



1.2.2 System Components

1.2.2.1 Block Verification Module

- Role

Verifies the integrity of block data provided in IPDC format, ensuring that each block is structured in the correct sequence and properly linked.

- Main Processing Tasks

  ・Calculates the IPDC hash of each block and verifies that it matches the hash of the previous block.

  ・Records the verification results in a log.

  ・Outputs successfully verified block header information in a format usable by the next step.

1.2.2.2 Certificate Verification Module

- Role

Utilizes the block header information output by the Block Verification Module to verify the integrity of credential certificate data.

- Main Processing Tasks

Based on the credential certificate data, issuer information, and block header information provided by the user, the following verifications are performed:

1. **Tamper-resistance Verification**:
   Compares the payload and digital signature of the credential certificate.
2. **Issuer Legitimacy Verification**:
   Matches the issuer information against a known public key list.
3. **On-chain Integrity Verification**:
   Verifies that the credential certificate is correctly recorded in the appropriate block using the transaction hash and Merkle path included in the certificate.

1.2.3 Input and Output Data Flow

This section defines the input and output data used in the system. Input data refers to the information required for verification, while output data refers to the results produced by the verification process.

1.2.3.1 Block Verification Module

This section defines the input and output data used by the Block Verification Module.

1. **Input Data**

- Block data in IPDC format

2. **Output Data**

- Verified block header information (passed to the next module)
- Verification log (in text format)

1.2.3.2 Certificate Verification Module

This section defines the input and output data used by the Certificate Verification Module.

1. **Input Data**

・ Verified block header information (output from the Block Verification Module)

・ Credential certificate data (vc.json)

・ Issuer information (in JSON format, including a list of public keys)

2. **Output Data**

・ Verification result of the credential certificate data (success/failure status)

・ Detailed log of the verification process (in text format)

1.2.4 Operation Flow

The operational flow of the system is described below.

1.2.4.1 Block Data Verification

- The IPDC-format block data provided by the user is verified using `cardano_validator.mjs`.
- The integrity of each block is checked, and verified block information is output.

1.2.4.2 Certificate Verification

- Based on the verified block information, `validator.html` verifies the credential certificate data.
- The verification result is displayed, and a detailed log is generated.

1.2.4.3 Output of Verification Results

- The verification result is presented as an intuitive status (success/failure).
- The detailed process used for verification can be saved as a log.

1.2.5 System Architecture Features

The following outlines the structure and key characteristics of this system.

1.2.5.1 Stepwise Verification

- By verifying the block data first, the reliability of the subsequent certificate verification is enhanced.

1.2.5.2 Fully Offline Environment

- All processes are executed in a standalone mode, making the system usable even in disaster areas or environments without internet connectivity.

1.2.5.3 Visualization of Results

- The system provides verification status and detailed logs, allowing users to clearly understand the verification process and results.

1.3 Module Design

1.3.1 Block Verification Module (cardano_verifier.mjs)

cardano_verifier.mjs is a module that verifies the integrity of block data based on the IPDC data format used in blockchain systems. It generates verified block information that can be utilized in the credential certificate verification process.

1.3.1.1 Module Responsibilities

- Calculates the IPDC hash for each block and ensures proper linkage with the previous block to guarantee block integrity.
- Saves verified block data in JSON format, making it available for the subsequent certificate verification step.
- Generates a verification log and records detailed information in case of errors.

1.3.1.2 Dependencies

- `fs`: Used for reading and writing files. Handles storage of block data and verification results.
- `crypto`: Used for hash calculations. Utilizes the sha256 algorithm to compute IPDC hashes.

1.3.1.3 Main Functions

1.3.1.3.1 `loadBlocks(filePath)`

- **Purpose:**
  Reads and parses block data in IPDC format from the specified file path.
- **Parameters:**
  filePath (string): Path to the block data file to be read.
- **Return Value:**
  An array of block data (in object format).
- **Processing Flow:**
  1. Asynchronously reads the file.
  2. Parses the JSON-formatted string and returns an array of block data.
- **Exception Handling:**
  Outputs an appropriate error message if the file does not exist.

1.3.1.3.2 verifyBlockIntegrity(blocks)

- **Purpose:**
  Verifies the integrity of each block by comparing the calculated IPDC hash with the previous block's hash value.
- **Parameters:**
  blocks (array): The block data to be verified.
- **Return Value:**
  Verification result (object): Contains information on successfully verified blocks and error logs.
- **Processing Flow:**
  1. For each block in the array:

     Perform hash calculation and compare the result with the hash value of the previous block.
  2. If verification fails, record detailed information in the error log.
  3. Return the verification results after all blocks have been processed.
- **Exception Handling:**
  If a calculation error or inconsistency occurs, record the affected block's information in the error log.

1.3.1.3.3 saveVerifiedBlocks(blocks, outputPath)

- **Purpose:**
  Saves the verified block data to the specified file.
- **Parameters:**
  blocks (array): Verified block data.
  outputPath (string): File path to save the data.
- **Return Value:**
  None.
- **Processing Flow:**
  1. Serialize the block data into JSON format.
  2. Write the data asynchronously to the specified file path.
- **Exception Handling:**
  If a write error occurs, output a detailed error message.

1.3.1.3.4 logVerificationResults(logs, logPath)

- **Purpose:**
  Records verification results and error information into a log file.
- **Parameters:**
  logs (array): Verification results and error information.
  logPath (string): Path to the log file to be saved.
- **Return Value:**
  None.
- **Processing Flow:**
  1. Convert the log data into text format.
  2. Asynchronously write the log to the specified file path.
- **Exception Handling:**
  If a write error occurs, output a detailed error message.

1.3.1.4 Data Flow

1.3.1.4.1 Block Data Loading

Reads IPDC-formatted data from a file and prepares it for verification.

1.3.1.4.2 Block Integrity Verification

Performs hash calculation for each block and compares it with the previous block's hash value.

1.3.1.4.3 Recording Verification Results

Saves block data whose integrity has been verified.
Logs any error information to a log file.

1.3.1.4.4 Output of Verified Data

Saves the data in a format that can be used by the Certificate Verification Module.

1.3.1.4 Example Usage

An example of how to use this module is shown below.

```
const { loadBlocks, verifyBlockIntegrity, saveVerifiedBlocks,
logVerificationResults } = require('./cardano_verifier');

// File Path Configuration
const inputFilePath = './ipdc_blocks.json';
const verifiedOutputPath = './verified_blocks.json';
const logFilePath = './verification_log.txt';

// Execution of Processing
async function main() {
  try {
    const blocks = await loadBlocks(inputFilePath);
    const results = verifyBlockIntegrity(blocks);
    saveVerifiedBlocks(results.verifiedBlocks, verifiedOutputPath);
    logVerificationResults(results.logs, logFilePath);
    console.log('Block verification completed.');
  } catch (error) {
    console.error('An error occurred:', error.message);
  }
}

main();
...
```

This module plays a critical role by handling block data integrity verification and log management, providing trusted data for use in the subsequent certificate verification step.

1.3.2 Certificate Verification Module (`validation.html `)

`validation.html `is a fully standalone tool designed to verify the integrity of credential certificate data.
It uses the verified block data generated by the Block Verification Module to perform three key checks:

- **Tamper-resistance** of the certificate
- **Legitimacy of the issuer**
- **Consistency of the blockchain record**

This module plays a central role in confirming the authenticity of credential certificates and provides users with highly reliable verification results.

1.3.2.1 Module Responsibilities

Based on the certificate data, issuer information, and verified block data provided by the user, this module performs a three-step verification:

- **Tamper-resistance Verification**
  Verifies the digital signature of the credential certificate.
- **Issuer Verification**
  Confirms that the issuer of the certificate is a trusted organization.
- **On-chain Consistency Verification**
  Verifies that the transaction data within the certificate is correctly recorded on the blockchain.

The verification results are visually presented to the user, and a detailed log is generated.

1.3.2.2 Dependencies

- cardano_serialization_lib.js

  Used for parsing Cardano transaction data and verifying signatures.

- sha3.min.js

  Provides the SHA3-256 hash function, used for Merkle tree verification.

- buffer.js

  Used for manipulating byte data and handling hexadecimal string conversions.

1.3.2.3 Main Functions

1.3.2.3.1 `handleFileUpload(event)`

- **Purpose:**
  Reads the file uploaded by the user and converts it into a format usable by the verification module.
- **Parameters:**
  ・ event (Event object): The file selection event.
- **Return Value:**
  Parsed JSON data.
- **Processing Flow:**
  1. Asynchronously reads the selected file.
  2. Parses it as JSON and stores it as verification target data.
- **Exception Handling:**
  If the file fails to load, an error message is displayed.

1.3.2.3.2 `verifySignature(payload, signature, publicKey)`

- **Purpose:**
  Verifies that the signature in the credential certificate data has not been tampered with.
- **Parameters:**
  - payload (object): The certificate's data payload.
  - signature (string): The digital signature attached to the certificate.
  - publicKey (string): The issuer's public key.
- **Return Value:**
  Boolean or parsed result indicating whether the signature is valid.
- **Processing Flow:**
  1. Asynchronously reads the selected file.
  2. Parses it as JSON and stores it as verification target data.
- **Exception Handling:**
  If the verification process fails (e.g., due to an invalid key or data), display an appropriate error message.

1.3.2.3.3 ` verifyIssuer(publicKey, issuerList)`

- **Purpose:**
  Confirms that the certificate issuer is a trusted organization.
- **Parameters:**
  - payload (object): The certificate's data payload.
  - issuerList (array): A list of trusted issuer information.
- **Return Value:**
  true if matched, or false if not matched.
- **Processing Flow:**

1. heck if the public key from the payload exists in the trusted issuer list.
2. If a match is found, return success; if not, log the failure in the error log.

1.3.2.3.4 ` verifyMerklePath(txHash, merklePath, merkleRoot)`

- **Purpose:**
  Verifies that the transaction data is correctly recorded on the blockchain.
- **Parameters:**

  ・ txHash (string): The transaction hash included in the certificate.

  ・ merklePath (array): The Merkle path included in the certificate.

  ・ merkleRoot (string): The Merkle root from the verified block data.
- **Return Value:**
  true if the calculated root matches, or false if it does not.
- **Processing Flow:**

  1. Calculate the Merkle root hash using the provided Merkle path and transaction hash.

  2. If it matches the expected Merkle root, return success; otherwise, record the failure in the error log.

1.3.2.3.5 `displayResults(results)`

- **Purpose:**
  Displays the outcome of each verification step to the user.
- **Parameters:**

  ・ results (object): The verification results, including success/failure status.
- **Return Value:**
  true if all checks passed, or false otherwise.
- **Processing Flow:**

  1. Visualize the success/failure status of each verification step (e.g., using ✅/❌ icons).
  2. Output detailed logs in a text area for user reference.

1.3.2.4 Data Flow

1.3.2.4.1 File Loading

Loads the credential certificate data, issuer information, and verified block data provided by the user.

1.3.2.4.2 Tamper-resistance Verification

Compares the payload and digital signature in the credential certificate to confirm that no tampering has occurred.

1.3.2.4.3 Issuer Verification

Checks the issuer information in the certificate against a trusted list of public keys.

1.3.2.4.4 On-chain Integrity Verification

Verifies the transaction hash and Merkle path against the verified block data to confirm correct blockchain recording.

1.3.2.4.5 Result Display

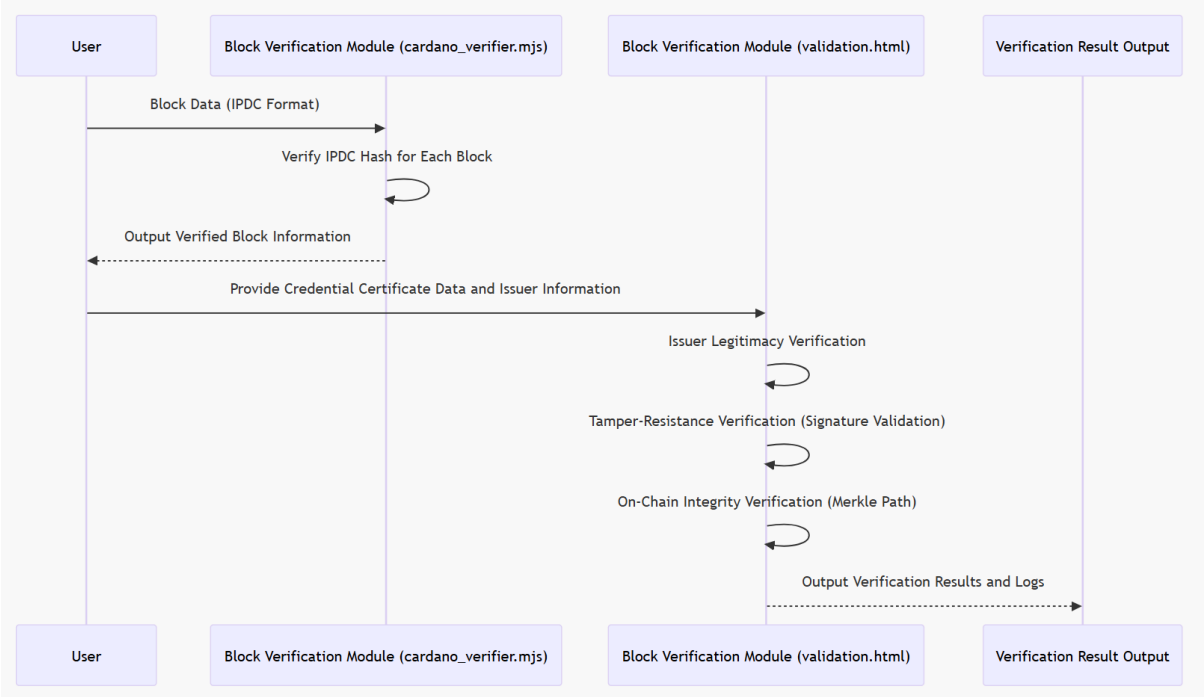
Presents the verification status (success/failure) and logs to the user.

1.4 Data Flow and Algorithms

This system performs staged processing of data between two modules to verify the integrity of credential certificates in a fully standalone environment.
The sections below describe the overall system data flow and the details of each algorithm used.

Through this data flow and its supporting algorithms, the system ensures the reliability of credential certificate data, enabling fast and accurate verification even in environments with limited network connectivity, such as disaster response sites.

1.4.1 Data Flow



1.4.1.1 Block Data Verification

- **Purpose**
  To verify the integrity of block data provided in IPDC format.
- **Input**
  Block data file in IPDC format.
- **Output**
  1. Verified block header information (in JSON format)
  2. Verification log (in text format)

1.4.1.2 Certificate Verification

- **Purpose**
  To verify the tamper-resistance, issuer legitimacy, and on-chain integrity of the credential certificate data based on the verified block header information.
- **Input**
  1. Verified block header information
  2. Credential certificate data (vc.json)
  3. Issuer information (in JSON format)
- **Output**
  1. Verification result (success/failure status)
  2. Detailed log of the verification process

1.4.2 Block Verification Algorithm

This section describes the block verification algorithm, including the overall processing flow and output specifications. It also outlines three assumed algorithms used in the verification process.

- Processing Flow
  1. Load block data in IPDC format.
  2. Calculate the hash value for each block.
  3. Check whether the calculated hash matches the IPDC hash of the previous block.
  4. Perform verification for all blocks and record the results.
  5. Save the verified block header information.
- Output
  - Verified block header information (if consistency is confirmed).
  - Error log (if inconsistencies are detected).
- Algorithms
  1. Algorithm 1: Issuer Legitimacy Verification

     Ensures that the issuer is trusted by verifying the public key against a known list.
  2. Algorithm 2: Tamper-resistance Verification

     Verifies that the block data has not been altered by checking the hash linkages.
  3. Algorithm 3: On-chain Integrity Verification

     Confirms that the transaction data is correctly recorded on the blockchain using Merkle proof.

1.4.2.1 Issuer Legitimacy Verification

- **Module**
  validation.html
- **Purpose**
  To confirm that the certificate issuer is included in a trusted public key list.
- **Processing Flow**
  1. Extract the issuer's public key.
  2. Check if it matches an entry in the trusted public key list.
  3. Record the verification result in the log.
- **Output**
  Success/Failure status

1.4.2.2 Tamper-Resistance Verification

- Module

  `validation.html`

- Purpose

  To ensure that the certificate data has not been tampered with.

- Processing Flow

  1. Extract the payload and signature from the credential certificate.
  2. Use the public key to verify the signature.
  3. Record the verification result in the log.

- Output

  Success/Failure status

1.4.2.3 On-Chain Integrity Verification

- **Module**
  validation.html
- **Purpose**
  To confirm that the credential certificate data is correctly recorded on the blockchain as a valid transaction.
- **Processing Flow**
  1. Retrieve the transaction hash and Merkle path from the certificate.
  2. Use the Merkle root from the verified block header to perform Merkle tree verification.
  3. Record the verification result in the log.
- **Output**
  Success/Failure status

1.4.3 Summary of Data Flow

This section summarizes the data flow and algorithms used in the verification process. The flow includes input/output specifications and outlines three core verification algorithms.

1. [cardano_verifier.mjs]

- **Input:** Block data in IPDC format
- **Output:** Verified block header information

2. [validation.html]

- Input:
    1. Issuer information
    2. Credential certificate data
    3. Verified block header information
- Verification Steps:
    1. Issuer legitimacy verification
    2. Tamper-resistance verification
    3. On-chain integrity verification
- Output:
    1. Verification result
    2. Detailed log

1.5 Data Design

In this system, the data formats and structures required for the verification process are clearly defined.
The data exchanged between modules is properly formatted and includes all essential information necessary for accurate verification.
Based on this data design, the system enhances its accuracy and reliability while also improving the efficiency of data interaction between modules.

1.5.1 Data Item Details

1.5.1.1 Block Data (Input)

- Format:

  IPDC Format（JSON）

- Important Fields:

  blockNumber(integer)　:The block number.

  ipdcHash(string)　:Hash value of the previous block.

  data(string)　:Transaction data contained in the block.

  timestamp(string)　:Timestamp of block creation.

1.5.1.2 Verified Block Information (Output)

- Format:

  JSON Format

- Important Fields:

  blockNumber(integer) :The block number.

  verified(bool) :Verification result (true/false).

  merkleRoot(string) :Merkle root of transaction hashes in the block.

  errors(array) :Details of any verification errors that occurred.

1.5.1.3 Credential Certificate Data (Input)

- Format:

  JSON(`vc.json`)

- Important Fields:

  id(string) :Unique identifier of the credential certificate.

  payload(object) :The payload data of the certificate.

  signature(string) :Digital signature.

  txHash(string) :Transaction hash recorded on the blockchain.

  merklePath(array):Path from the transaction hash to the Merkle root.

1.5.1.4 Issuer Information (Input)

- Format:

  JSON Format

- Important Fields:

  issuerId(string) :Unique identifier of the issuer.

  publicKey(string) :The issuer's public key.

  trusted(bool) :Flag indicating whether the issuer is trusted.

1.5.1.5 Verification Result (Output)

- Format:

  JSON Format

- Important Fields:

  signatureCheck(bool) :Result of tamper-resistance (signature) verification.

  issuerCheck(bool) :Result of issuer legitimacy verification.

  merkleCheck(bool) :Result of on-chain integrity verification.

  logs(array) :Detailed processing logs for each verification step.

1.5.2 Features of Data Design

1.5.2.1 Extensibility

- Flexible design that allows additional data fields to be added as needed.
- JSON schemas can be customized to support specific verification conditions.

1.5.2.2 Readability and Consistency

- Each data structure uses clear keys and a hierarchical format to ensure consistent processing.

1.5.2.3 Error Handling

- Error information is output through errors or logs, making it easier to diagnose and resolve issues.