International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 11 | Nov - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

D-SIVC: A Hybrid Blockchain Framework for Student Identity Management and Academic Credential Verification

KULDEEP BISHNOI, SIDDHI UKANDE, KHANDJOT SINGH, SUJAL HOTA, Prof. GOVIND SINGH

Department of Computer Science and Engineering, Shri Shankaracharya Technical Campus, Bhilai, CG, India

Abstract

The topic of this study is a usage of academic credentials in order to back up students' applications for further education and employment. Many institutions still use paper documentation, email queries or separate institutional web pages to verify such credentials, which can take a lot of time and impede a verifier from checking if the document matches the original one. The current paper presents D-SIVC - a practical prototype of the solution, which combines mobile application and institutional database and blockchain credential registry. In this architecture student and credential data stay off-chain and only SHA-256 digest of a credential and its unique identifier are stored on Polygon Amoy using Solidity smart contract. The system has functionalities for student identity view, credential issuance by administrator, QR lookup, public verification and revocation status check. The prototype was tested through source code review and contract testing. Fourteen Hardhat tests and application quality testing were successfully performed. But database behavior testing and Edge Function runtime testing are not done yet and the current public verification process doesn't check the on-chain digest independently. Thus, the result of this research is the functioning testnet prototype and also the list of tasks that need to be done before the institutional implementation.

Keywords: academic credential verification, blockchain, SHA-256, Polygon Amoy, smart contracts, hybrid architecture, student identity, credential revocation.

I. Introduction

Various types of records are issued by educational organizations, including student identity documents, completion certificates, transcripts, and achievement certificates. Such records influence decisions made by employers, admissions, and other institutions. In general, the task of a verifier is to determine whether a document is issued by the claimed institution and whether its content was unchanged since the moment of issuance. Manual verification through registrars or institutional portals may cause delays and duplicating efforts. On the other hand, digitization improves distribution, but a traditional database-driven portal depends on the availability of the organization that operates it.

Blockchain technology offers a distributed, append-only ledger in which published transactions are immutable under the consensus assumptions of the network [1]. In a case of credential systems, there is no need to keep the entire academic record in the ledger. Instead, a cryptographic digest can be stored on the blockchain, whereas the content of the credential stays in the off-chain database. A verifier can then check the digest of the provided record against the digest in the ledger. Thus, it is possible to reduce the exposure of personal data on the blockchain. However, it does not necessarily mean that the system guarantees confidentiality and prevents correlations.

There have been several studies devoted to blockchain-based academic credential systems, diploma generation and verification, and self-sovereign identity systems for education [2]– [5]. It is also clear from the literature review that there are more issues to consider during the deployment besides immutability. Formats of credentials, trust in issuers, privacy, revocation, usability, interoperability, and performance of the system are important aspects of design. W3C Verifiable Credentials Data Model 2.0 defines an ecosystem of issuers, holders, and verifiers and the data model for machine-verifiable credentials [6]. W3C Verifiable Credential Data Integrity 1.0 describes cryptographic methods of proving authenticity and integrity [7].

This paper describes D-SIVC, a hybrid prototype that combines off-chain identity and credential management with on-chain hash anchoring. The purpose of this paper is not to claim that the current prototype implements the full W3C credential ecosystem. The goal is to describe the implemented architecture, analyze the contract and application workflows of the system, and identify gaps that should be overcome before making any claims about security and interoperability.

A. Problem Statement

This research considers the lack of a lightweight and auditable workflow for both credential issuing process as well as credential history tampering detection. A proposed practical solution will have to follow three important constraints: (i) the student data cannot be revealed unnecessarily in the public ledger; (ii) the fingerprint of the credential should be anchored in the tamper-evident registry; and (iii) the credential issuing and revoking activities should be restricted to an authorized actor. In terms of current D-SIVC

approach, this problem is partly solved. Still, there are several limitations relating to role management, data privacy issues, and end-to-end blockchain integrity that are considered in the next chapter.

B. Objectives and Research Questions

• Develop the hybrid architecture that allows storing students and credential details off-chain while recording credential's unique identifier and its SHA-256 digest on-chain;

• Implement the smart-contracts with such capabilities as credential issuing, retrieving, digest comparison and revocation;

• Implement the application that works on mobile devices and QR codes that will serve as a public interface;

• Check the implementation in the aspects of access control, privacy, integrity, and life-cycle;

• Prepare a plan of empirical evaluation concerning latency, gas cost, consistency, and tamper-detection capabilities.

The following are the guiding research questions for this study:

1. RQ1: How to implement a hybrid architecture for storage of the academic credentials' records?
2. RQ2: Which of the integrity and life-cycle properties will be enforced by the smart contract?
3. RQ3: Are there ways how the current application and database can independently verify the on-chain anchor?
4. RQ4: Which limitations in terms of security, privacy and performance need to be taken care of prior to implementation at an educational institution?

II. Related Work

Rustemi et al. carried out the systematic review of blockchain-based academic credential verification research, selecting 34 relevant articles from 1,744 publications in the field dated from 2018 to 2022 [2]. Their study reveals that the field of blockchain credential verification matured enough to become a distinct research topic. Nevertheless, they highlight the need to focus on practical implementation decisions and open research questions in order to make the research progress in the area. Therefore, the evaluation of implemented prototypes is to be performed in addition to theoretical research in the matter.

Firstly, Rustemi et al. presented their DIAR blockchain-based system for creation and verification of academic diplomas [3]. The paper makes an important contribution to the system architecture, functionality of smart contracts and interoperation of such a system. Similarly to D-SIVC, DIAR uses a blockchain registry to maintain the integrity of credentials. However, at present, the implementation of D-SIVC uses Supabase as the underlying identity and credential database and Polygon Amoy as the anchoring blockchain.

Chan et al. provided a review on blockchain-enabled self-sovereign identity technologies in education [4]. It situates educational identity systems in the context of wider discussion on privacy, portability and reliability. It should be noticed that D-SIVC does not provide the full-scale self-sovereign identity, as students' accounts and credential storage is carried out using a centralized Supabase database with the power of issuing and revoking credentials reserved for the institution.

Fernández-Blanco et al. considered the design and energy-efficiency evaluation of blockchain-based system for academic credential verification [5]. They stressed the importance of adding resource and performance metrics when evaluating implementation solutions. The current repository of D-SIVC contains test code and documentation on implementation and deployment but does not contain a benchmarking data for issuance latency, verification latency, throughput and energy consumption.

W3C verifiable Credentials Data Model 2.0 and Data Integrity 1.0 offer the standards-based points for future interoperation of D-SIVC [6], [7]. The W3C verifiable credential cannot be equated with any arbitrary record kept in database with the hash committed to the blockchain. W3C Data Model specifies the properties of credentials and relationship between issuer, holder and verifier. W3C Data Integrity specifies the data proof and verification algorithms used. Currently, D-SIVC keeps the JSON payload and its SHA-256 digest in blockchain.

A. Positioning and Research Gap

The proposed project does not promote the use of blockchain as a solution to replace all other databases of institutions. This project aims at a much more modest goal, that is, the storage of students' data in a traditional application database and placing a tamper-proof digital signature of each issued certificate on a public testing network. In this paper, the working model of the application and the contract are described and analyzed in terms of how they can be better integrated.

III. Proposed D-SIVC Framework

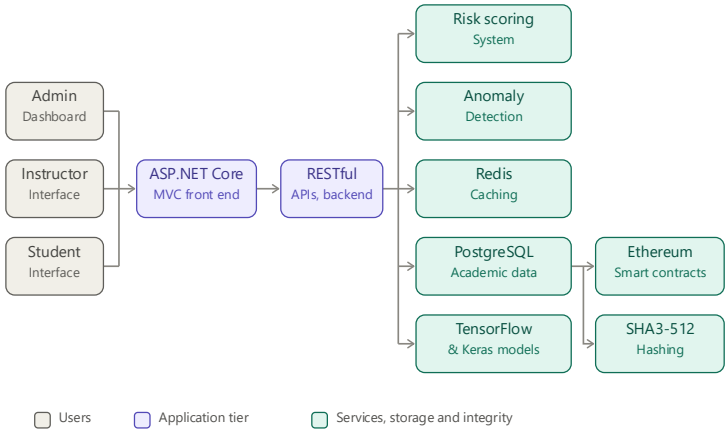
A. Architecture Overview

The application is divided into four parts. Firstly, it is the client which is an Expo and React Native application written in TypeScript. It includes user interfaces for both students and administrators including identity details, credential inventory, credential issuance, and QR-based verification. Secondly, Supabase offers authentication and PostgreSQL database for profiles and credential records. Thirdly, blockchain layer uses the StudentCredentialRegistry Solidity contract deployed to Polygon Mumbai (Polygon Amoy) network. It stores credential identifiers, SHA-256 digests, and lifecycle details instead of storing academic records. Finally, verification interface allows the user to fetch a credential and check its current status. While these parts work together, they do not provide equal amounts of trust: the database serves as a basis of the application's normal operation while the on-chain contract provides the public immutable record which an independent verifier will eventually use.

The application-service layer is built upon Supabase Authentication and PostgreSQL. The database includes the profiles table containing student and administrator profile properties and the credentials table which includes credential identifiers, credential type, JSON credential data, hash, transaction hash, blockchain status, issuance timestamp, and revocation timestamp. The RLS policies are defined for profiles and credentials. There are two Supabase Edge Functions issue-credential and revoke-credential performing blockchain transactions using Ethers.js along with server-side private key.

The blockchain layer includes the StudentCredentialRegistry Solidity contract compiled with Solidity version 0.8.20 and OpenZeppelin Ownable. The deployment configuration defines Polygon Amoy with chain identifier 80002. The contract stores a mapping from string credential identifiers to the records including a bytes32 digest, issuer address, issuance timestamp, revoked flag, and existence flag.

The verification layer currently describes two different capabilities which cannot be confused. The Solidity contract provides a public view function able to compare the input digest with the on-chain stored digest. However, the inspected application verification service uses the Supabase RPC function verify_public_credential. This RPC makes a query to PostgreSQL and determines validity from the database revocation timestamp and non-null hash. It does not call the smart contract. Thus, the public application interface works as a database-backed lookup and status display, but not an independent on-chain verification client.



B. Credential Data Model

Field / element	Location	Purpose
credential_id	PostgreSQL and contract	Unique public identifier used to locate a credential.
credential_data	PostgreSQL JSONB	Credential payload containing student and academic attributes.
credential_hash	PostgreSQL and contract	SHA-256 digest represented as a 32-byte value on-chain.
issuer	Contract	Address that submitted the issuance transaction.
issued_at	PostgreSQL and contract	Application issuance time and blockchain anchoring time, respectively.
blockchain_tx_hash	PostgreSQL	Transaction reference returned by the blockchain provider.
blockchain_status	PostgreSQL	PENDING, ANCHORED, FAILED or REVOKED state.
revoked / revoked_at	Contract / PostgreSQL	On-chain revocation flag and off-chain revocation timestamp.

A public credential identifier is created based on a timestamp and a random numeric suffix by the client. The identifier is then appended to the provided payload and serialized into an object with JSON.stringify, and its SHA-256 hash is calculated using Expo

Crypto. The hash is saved to PostgreSQL with the PENDING status, and the Edge Function reads the record, checks whether the hash has the correct 0x-prefixed 64-hex-character format for bytes32, and passes the identifier and digest to the contract.

There is no specified JSON canonicalization method at the moment. JSON.stringify generates deterministic output for the same JavaScript object and input ordering, but there is no canonicalization procedure specified for cross-language use in the project. This means that an independent verifier who tries to generate the same digest from a semantically equivalent JSON will have to use the same property ordering and encoding.

C. Credential Issuance Workflow

1. An admin chooses a student and provides credential type and payload via the mobile application.
2. Application generates credential identifier and appends it to the payload.
3. Application serializes the payload and calculates its SHA-256 hash digest.
4. The credential entry is added to PostgreSQL with blockchain_status set to PENDING.
5. Application calls issue-credential Edge Function providing credential identifier and database entry identifier.
6. Edge Function verifies caller's Supabase access token, gets the caller's profile role and rejects calls from those having role other than admin.
7. The function obtains the credential record, checks for any previously anchored record and whether student already has another credential.
8. Function calls issueCredential(credentialId, digest) of StudentCredentialRegistry and waits for one transaction confirmation.
9. Function updates the database entry with transaction hash and ANCHORED status if transaction succeeds. Otherwise, it marks it as FAILED.

D. Smart-Contract Design

The contract declares a struct called CredentialRecord which consists of five fields: credentialHash (bytes32), issuer (address), issuedAt (uint256), revoked (bool) and exists (bool). Records are stored in a private mapping that uses string identifier of the credential as a key. The contract also implements CredentialIssued and CredentialRevoked events. Constructor passes ownership to the account that deploys the contract using Ownable(msg.sender).

issueCredential function is restricted by onlyOwner modifier. It refuses to issue an empty identifier, zero digest or duplicate identifier. Upon success, it stores digest, caller's address, current block timestamp, non-revoked status and an existence flag. Function revokeCredential is restricted by onlyOwner modifier and refuses to revoke an unknown or already revoked record. Function verifyCredential is a public constant function which returns the tuple with the record existence, the equality between the supplied digest and the one in the record, the revocation status along with the revocation status and anchoring timestamp. Function getCredential returns the metadata of the record.

The contract implements a simple institution-controlled lifecycle. The contract does not contain any information about students and its read-only methods do not require a transaction. The owner model is quite simple, however it introduces a single administrative key and a single point of control. OpenZeppelin library states the following regarding Ownable contract: "This contract is based on the OpenZeppelin library, which provides a very simple ownership pattern to use for your contracts when there is a single administrative user" [8].

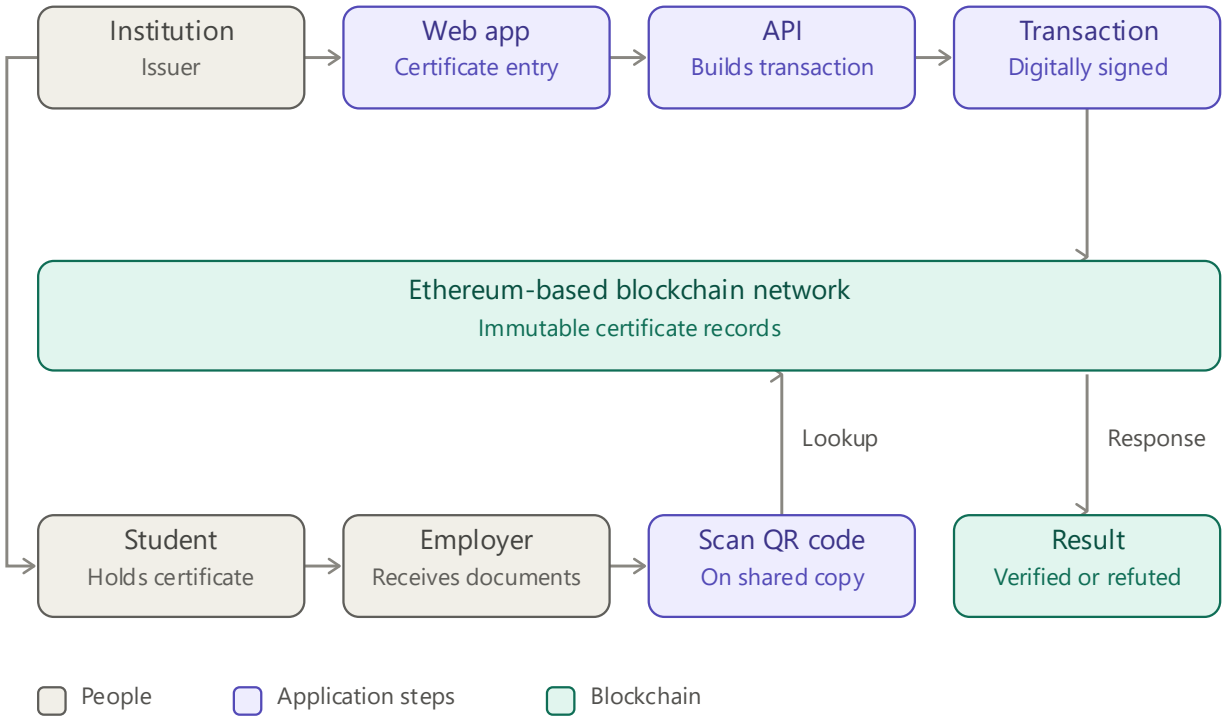
E. Revocation Workflow

The edge function revoke-credential verifies the caller's authentication, verifies the database profile role, retrieves the credential and rejects an already-revoked record in the database. In case that the status in the database is "ANCHORED" or that the transaction hash is provided, it calls the method "revokeCredential" on the smart contract and waits for one confirmation. Then, it saves in PostgreSQL the retrieved transaction hash, REVOKED status and revocation date. In case that the credential was not yet anchored, it revokes the credential only in the database without calling any on-chain revocation transaction. This should also hold in the meaning of the verifier status.

F. Public Verification Workflow

The public verification screen supports either the credential identifier or the SHA-256 digest. The verification service invokes the verify_public_credential Supabase RPC. The SQL function returns the matching credential row and calculates is_valid as the expression that revoked_at is null and credential_hash is not null. The screen shows the credential type, student name, roll number, institution, digest, status, and transaction reference, if any.

This process serves as a good prototype for a lookup interface, but it does not provide independent blockchain verification. The application must not assume a record to be valid merely because there is a hash stored in the database and no revocation timestamp. A proper verifier would retrieve the full credential, compute its digest according to a canonicalization scheme, get the on-chain record via a trusted RPC interface, and compare the digest and the existence flag and on-chain revocation status.



IV. Implementation Details

Layer	Technology / implementation notes
Mobile client	Expo SDK 57, React Native 0.86.3, TypeScript 6, Expo Router.
Authentication and data storage	Supabase Auth, PostgreSQL, JSONB and RLS policies.
Backend Integration	Supabase Edge Functions (Deno), Supabase.js and Ethers.js.
Cryptographic Digest	SHA-256, expo-crypto
Smart contract	Solidity 0.8.20, OpenZeppelin Ownable 5.x
Development and tests	Hardhat 2.x, ethers 6.x, Chai and Hardhat toolbox.
Blockchain network	Polygon Amoy Testnet (chain ID: 80002)

The private key for signing blockchain transactions is fetched from the Edge Function environment as BLOCKCHAIN_PRIVATE_KEY. The hardened version of the code alters the authorization path such that Edge Functions check the caller's Supabase access token, fetch the caller's role using the trusted profiles table via the privileged database client, and only authorize the caller to be exactly an admin before issuing credentials or revoking credentials. The client-specified role is not an authorization source in this case, and the service role secret is needed for privileged database access as opposed to falling back to the anonymous key. There is also permissive CORS origin configured in the repository, which the deployment policy can restrict origins where appropriate.

A. Application Interface and Prototype Screens

The images below are snapshots taken from the running application and reflect the various user flows. These images are added as proof of implementation and not for any performance or security testing purposes. The images reflect the student portal, QR credential sharing, administrator portal, student directory, and verifier portal.

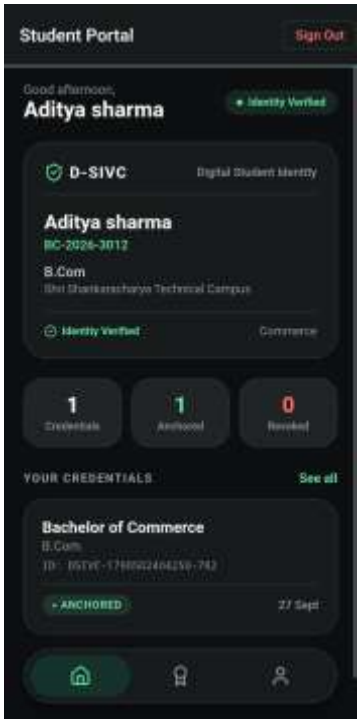


Figure 1. Student dashboard showing the digital identity summary, credential count and the status of an anchored credential.



Figure 2. Credential verification dialog displaying a QR code and the option to share a verification link.



Figure 3. Administrator dashboard showing student and credentia totals, the issue-credential action and recent issuance activity.



Figure 4. Student directory used by an administrator to locate a student and start the credential issuance process.

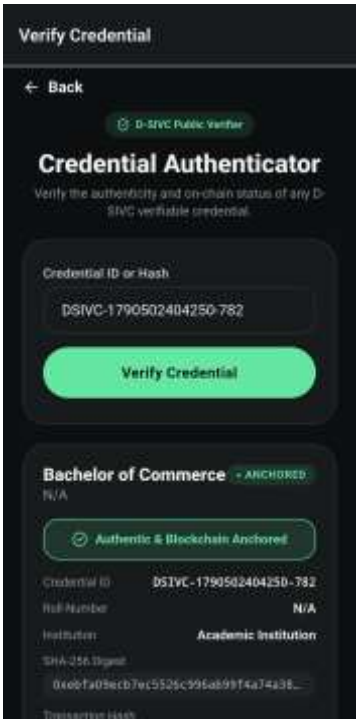


Figure 5. Public credential verifier with an identifier input and a result panel showing the credential's anchored status and digest information.

Privacy note: The above-mentioned screenshots provide examples of fake student names, numbers, credential IDs, along with the QR code that can be scanned. Before releasing the manuscript to public use, the authors have to replace the above data with dummy data, since the above QR-code cannot show the actual credential or personally identifiable information.

The screenshots show that the main screens and screen navigations are available in the functional prototype of the product. These screenshots alone do not prove that the shown identity or credentials have been separately validated through the blockchain. This is significant when analyzing public verification results in the following sections.

V. Security and Privacy Analysis

The following section assesses attributes present within the source code. This is a code-based analysis and not a penetration test or audit of the live Supabase project. The analysis is based on the interoperability and security principles of the W3C credential standards and blockchain security [1], [6]–[9].

Property	Observed implementation	Assessment
On-chain data minimization	The contract holds an identifier, digest, issuer address, timestamp, and status. However, no credential payload is saved.	Minimizes PII disclosure through direct on-chain storage; publicly available IDs and hashes can still be linked.
Contract issuance control	The issueCredential action is restricted only to the contract owner (onlyOwner).	Guaranteed by the contract; the vulnerability exists depending on the owner key's security.
Contract revocation control	The revokeCredential action is restricted only to the contract owner (onlyOwner).	Guaranteed by the contract; creates a single-owner centralization vulnerability.
Digest comparison	The verifyCredential action performs the comparison of input bytes32 with the stored one and the state of the revoked flag.	On-chain solution; the present public application does not use it.
Duplicate prevention	The contract prevents existing credential identifier.	Implemented on the contract layer.
Database role control	The role attribute in the user profile is derived from metadata; registration form allows choosing between a student or an admin role; there are no restrictions on updates of the role column in the user own-profile.	Privilege escalation vulnerability; needs correction before making any security claims.
Public data minimization	Public API returns the credential_data field in JSONB format. The user interface shows personal attributes.	Privacy exposure; requires minimizing public outputs and user permission.
Cross-layer consistency	Edge Functions update the PostgreSQL database after the blockchain transaction has taken place. The inspection code does not have any reconciliation worker.	Inconsistencies can occur due to the partial failures.
Hash reproducibility	The hash is generated based on JSON.stringify.	Canonicalization procedure is missing. Independent generation is not specified.
W3C VC conformance	The inspected source does not show the complete W3C Verifiable Credential data model implementation and Data Integrity proof.	Non-conformant; claims of compliance would be incorrect.

A. Role Escalation Risk

The audit of the initial configuration found a privilege escalation path: public user registration provided a role selector, registration flow could send role metadata, the initial handle_new_user trigger relied on that metadata, and the profile update policy was not strong enough to secure the role field. The subsequent hardening changes disable public administrator assignment, do not send a role as part of the registration payload, and modify the profile creation procedure to assign the student role when new profiles get created. Conflict handling updates ordinary profile fields without overwriting the existing role. The profile update policy requires a user role to be equal to the result of a trusted database function, while maintaining administrative access to perform valid administrative functions.

The hardening migration also creates a specific provision_admin(user_id, reason) database function, revokes its EXECUTE permission from everybody except Supabase service_role, and maintains an audit log in an audit table that captures account, user who is doing provisioning (where applicable), reason, and timestamp of an event. The Edge Functions use the database profile role of the authenticated user instead of a role that users control through user editable metadata or the role parameter of the request body. These are the source level remediations and do not constitute a proof of production security. In the version of migration used for the tests for this manuscript, the profile trigger bypasses the check based on a custom setting local to a transaction but does not validate the custom setting in terms of its binding to a JWT signed by the service_role. The custom setting alone is not a barrier of authorization. The trigger needs both a marker and a service_role claim to validate this bypass. The migration, as well as behavioral tests need to be updated accordingly and executed against a Supabase test database in order to have the administrative boundary properly validated.

The authorization model for administrator role assignment consists of separation between the normal registration and privileged account provisioning. New accounts are assigned student role by default; administrators are assigned explicitly on the server side with the reason logged. Table updates should continue to be restricted, even those done through the service_role database sessions unless they pass through the specified provisioning function.

B. Public Credential Data Exposure

The function verify_public_credential is defined as SECURITY DEFINER and returns credential data as JSONB. The public credential checker provides the details of student's name and roll number. Therefore, in case of public verification interface, the amount of information which gets revealed exceeds what should have been revealed in order to show presence of a particular credential or the fact that its digest verifies with an anchor. The public verification should respond only with the bare minimum data, such as, the type of credential, issuer of the credential, whether the credential is active or not, etc.

C. Integrity, Hashing and Authenticity

SHA-256 is used as a digest function, but a digest by itself doesn't prove the identity of the person who created the payload. The on-chain issuer address proves the account used to submit the transaction, but the current payload doesn't seem to have any issuer signature that could be verified outside of the application's trusted environment. Hashing is a way to check whether there was a mismatch between the expected digest and the calculated one; hashing does not prove that the claims were true at the time of issuing. The W3C Data Integrity Recommendation uses cryptographic proofs to establish integrity and authorship properties for secured credential data [7].

The credential digest is calculated based on a JavaScript JSON serialization. Some canonicalization scheme must define the ordering of properties, Unicode normalization, number formatting, handling of optional/null fields and encoding. Otherwise, two separate verifiers will serialize the same data in different ways and calculate different digests. The test suite must contain examples of reordering properties, Unicode names, optional fields and cross-language implementations.

D. Cross-Layer Consistency

The issuance function pushes the transaction to the blockchain and then updates the PostgreSQL entry. Should the blockchain transaction be successful but the database operation not happen for whatever reason, the blockchain would end up having an anchor while the database would continue showing a value of PENDING. The function will generate a warning message together with the transaction hash, but there does not seem to be any way of doing automatic reconciliation as part of the implementation in the source code examined. There should be proper transaction handling, where transaction hashes are saved whenever possible, idempotency keys are used, among others. For revocation, the boundary is almost the same.

E. Threat Model

Threat	Potential impact	Required control
Malicious registrant selects admin role	Unauthorized issuance/revocation via Edge Functions.	Server-controlled role assignment; immutable role field for users; authorization tests.
Unauthorized public lookup	Exposure of student attributes and correlation of credentials.	Minimal RPC response, explicit grants, consent and rate limiting.
Credential payload modified	Incorrect or failed integrity validation if verifier recomputes digest.	Canonical serialization and direct on-chain digest comparison.
Institutional owner key compromised	Attacker can issue or revoke credentials.	Multisig, hardware-backed key custody, rotation and incident response.
Database and chain become inconsistent	Stale status or false verification display.	Idempotent writes, event indexing and reconciliation.

RPC provider unavailable or dishonest	Verification unavailable or based on untrusted response.	Multiple providers, chain ID and contract address checks, failure-safe result.
Credential identifier leaked	Unwanted lookup or linkage of credential metadata.	Unpredictable identifiers, minimal disclosure and access logging.

VI. Functional Evaluation

The provided Hardhat test file includes fourteen test cases on deployment, ownership, issuance by the owner, record existence, digest storage, issuer registration, initial revocation state, verification with the correct digest, rejection of the incorrect digest, rejection of issuance by non-owners, revocation by the owner, verification after revocation, rejection of revocation by non-owners, and rejection of duplicate identifier. The most recent remediation report notes that all fourteen Hardhat tests have been run and passed. The report mentions successful linting of the application, TypeScript type checking, Expo Doctor check (21 out of 21 checks), and git diff whitespace check. These are development and contract-level tests, not database or end-to-end security tests.

Test category	Coverage in supplied Hardhat suite	Reported result
Deployment and ownership	Contract deployment and owner address.	2 tests; README reports pass.
Issuance and storage	Owner issuance, existence, stored digest and issuer address.	4 tests; README reports pass.
Verification	Correct digest and incorrect digest.	2 tests; README reports pass.
Authorization	Non-owner issuance and revocation rejected.	2 tests; README reports pass.
Revocation	Owner revocation and post-revocation state.	2 tests; README reports pass.
Duplicate handling	Duplicate credential identifier rejected.	1 test; README reports pass.
Other lifecycle state	Initial unrevoked flag.	1 test; README reports pass.
Total	Fourteen contract-level test cases.	14/14 reported passing; not independently rerun.

The SQL hardening and behavioral test cases have been written, including test cases for promoting the students as administrators, adding an administrator role, updating service roles, provisioning using RPC, creating an audit and accessing profiles. Nevertheless, the report says that these SQL test cases have not been run due to the absence of the Supabase CLI as well as Supabase database either locally or live. Furthermore, the deno could not be found and hence the test case for edge function could not be run. The live administrator inventory and deployed database policy/grants inventory have not yet been verified either.

Consequently, there has been no proof of RLS runtime security tests being run successfully.

A. Deployment Evidence

The repository has a deployment configuration for Polygon Amoy, as well as metadata for the deployment. The network name documented is 80002. The contract address according to the README file is 0x9ee5cd0d9D08d07d5547b2402240aD11F9761cA0. Transaction details for issuance and revocation transactions have also been noted in the README file. These values should be checked for their accuracy with respect to the public polygon explorer immediately before submission. Also, the bytecode of the contract deployed should be verified against the submission and compilation details provided.

Dependency auditing of the project shows that the total number of vulnerabilities reduced from twenty five to sixteen following patch updates for same major versions of expo-sdk-57. The remaining vulnerability list consisted of one critical, four high and eleven moderate vulnerabilities. Significant framework upgrade as well as forced dependency upgrade was intentionally kept at bay. These remaining issues warrant a separate dependency audit including reachability and production exposure evaluation.

VII. Performance Evaluation Plan and Missing Measurements

No reproducible performance tests have been provided in the source code package. It would be scientifically incorrect to state issuance latency, verification latency, gas consumption, throughput, scalability, or energy consumption as measurement results without conducting any controlled experiments. Below is the suggested protocol for the next evaluation step.

Metric	Measurement procedure	Reporting method
Client credential preparation time	Measure from submission action to completion of digest generation and local preparation.	Median, mean, standard deviation, sample count.
End-to-end issuance latency	Measure from authenticated issue request to one-confirmation transaction receipt and database status update.	Median, p95, mean, standard deviation.
Blockchain confirmation latency	Record transaction submission and receipt timestamps for each issuance/revocation transaction.	Median and p95, grouped by network conditions.
Public verification latency	Measure the complete verifier request. Separately measure RPC-only and direct-chain verification once implemented.	Median, p95, failure rate.
Gas consumption	Read gasUsed from each transaction receipt and record operation type.	Mean, median, range, transaction count.
Tamper detection	Change one payload field and recompute the digest; compare with stored on-chain digest.	Detected/undetected counts and test cases.
Revocation correctness	Verify before revocation, after confirmed revocation and during database/chain disagreement.	Correct state transitions and failure behaviour.

Scalability	Increase credential count and concurrent verifier requests under controlled conditions.	Throughput, p95 latency, error rate.

Every experiment needs to be documented with regard to the date, software version, device/server setup, RPC provider used, network chain name, contract address used, and number of repetitions (and any failures). Thirty repetitions per timing condition is a good starting point for initial evaluation via exploration, although the actual sampling size needs to be validated based on the variability observed and what the target venue wants. The gas measurement values need to be presented in gas units. To convert them into fiat values, one needs to know the token price at the time stamp.

VIII. Discussion

This contract gives a small registry of credential fingerprints and their lifecycle state. Keeping personal and academic information off-chain ensures that the complete student record is not made available as part of a public ledger. However, the application still relies on Supabase for user identity, credential data and some functionality of the public verifier. While this approach makes sense for the prototype, it demonstrates that blockchain anchoring alone does not decentralize the entire system and make it independently verifiable.

The present architecture demonstrates a compromise of the hybrid system as well. Supabase facilitates authentication, querying of the database and development of the application while the blockchain provides a public proof of the digest provided by the institutional owner. In any case, the database is a trusted component of the operation. In case the verifier only relies on that database, the verifier cannot independently verify that the record corresponds to the on-chain anchor. In the next version of the implementation, this decision needs to be moved into a verifier that validates both credential payload and on-chain registry.

Finally, the current owner-only approach to the management of the record is straightforward and convenient to test, but this is not a distributed institutional governance solution. In case the institution wants to run the system in production, the ability to issue and revoke credentials needs to be protected using multi-signature, institutional identity validation, logging of the records and key recovery procedure. The issue of role escalation in the current database needs to be solved first..

IX. Limitations

- The system is a prototype running on Polygon Amoy testnet, not a production deployment on mainnet.
- The present public verifier uses Supabase RPC and does not perform independent comparison between the credential digest and revocation status to the smart contract.
- The public verification RPC returns credential_data and the public screen shows personal fields. This is a privacy issue that needs to be fixed along with an audit of the granted permissions to the deployed RPC.
- The administrator privilege escalation has been solved in the source code but the provisioning trigger bypass still needs one more check of the service-role JWT and the behavioral SQL and Edge Function tests have not been run.
- The implementation performs JSON.stringify before performing SHA-256 hashing and does not specify canonical JSON serialization for cross-platform reproducibility.
- The smart contract uses a single Ownable administrator. If the owner key is compromised or lost, it affects all the operations of issuing and revoking credentials.
- The code does not show any implementation of the complete W3C Verifiable Credentials Data Model 2.0 and W3C Data Integrity proof.
- Validation of security is incomplete: fourteen Hardhat contract tests and application quality checks have passed, however, Supabase behavioral tests, Deno Edge Function tests, live administrator inventory, inspection of the deployed RLS policy and independent penetration test have not been performed.
- The off-chain database and blockchain are updated separately. There is no demonstration of reconciliation and idempotent recovery in the inspected implementation.
- The work does not investigate throughput at institutional scale, multiple institutions, interoperability with external credential wallets or selective disclosure..

X. Future Work

1. Strengthen the administrator provisioning process by linking the provisioned flag within transaction scope to the verified service_role JWT. Perform the behavioral tests of SQL provisioning, as well as edge functions authorization test suite inside the Supabase controlled environment.
2. Refine the public API for verification via an RPC call so that there is enough information and verification on the credential digest as well as revocation.
3. Define a canonical format and test vectors that will enable other implementations to produce the same SHA-256 digest as the end product.

4. The issuer's signature and W3C verifiable credentials 2.0 standard for data representation, plus the Data Integrity securing method (JOSE/COSE) shall be used to create such canonical serialization.
5. Provide for privacy preserving credential status checks and selective disclosures when necessary.
6. Use multilevel signing or governance based on roles with key rotation and key recovery plan documented.
7. Add transaction intent log, idempotence control, event indexing, and automatic PostgreSQL <-> Polygon reconciliation.

Make reproducible experiments (performance, usability, security and scalability tests, comparison with pure DB only solution).

XI. Conclusion

As a result of this project, D-SIVC, a working prototype of the system that manages student identity and verifies academic credentials has been created. The mobile application authenticates the user and saves off-chain records via Supabase, while a Solidity contract on Polygon Amoy holds a SHA-256 fingerprint and the state of the credential's lifecycle. The application has student and administrator views, the process of issuing a credential, QR lookup and a public verification page. The tests for the contract include the tests of the major issuance, ownership, duplicate prevention, digest comparison, and revocation functionalities. Fourteen Hardhat tests are said to pass successfully, together with linting, TypeScript checks and Expo Doctor checks.

The security review was also the source of some changes in the workflow of the account and administrator. Public self-registration for the administrators was deleted, the new profiles get the role of the student, and the Edge Functions now verify the role saved in the database rather than the one that is provided by the client. Also, an administrator provisioning function and an audit table were added to the system. All of these changes make the design better, but they should not be considered validated yet due to the trigger bypass issue and the lack of testing with SQL and Edge Functions in the controlled Supabase environment. The public verification RPC currently provides more student data than needed and should be simplified before using any real student records. D-SIVC can thus be referred to as a testnet prototype that is being hardened from the security point of view, but not as a production-grade or decentralized credential platform. W3C Verifiable Credentials conformance is also not claimed.

References

[1] D. Yaga, P. Mell, N. Roby, and K. Scarfone, “Blockchain Technology Overview,” NIST Interagency/Internal Report 8202, National Institute of Standards and Technology, Oct. 2018, doi: 10.6028/NIST.IR.8202.

[2] A. Rustemi, F. Dalipi, V. Atanasovski, and A. Risteski, “A Systematic Literature Review on Blockchain-Based Systems for Academic Certificate Verification,” IEEE Access, vol. 11, pp. 64679–64696, 2023, doi: 10.1109/ACCESS.2023.3289598.

[3] A. Rustemi et al., “DIAR: a blockchain-based system for generation and verification of academic diplomas,” Discover Applied Sciences, vol. 6, art. no. 297, 2024, doi: 10.1007/s42452-024-05984-1.

[4] W. Chan, K. Gai, J. Yu, and L. Zhu, “Blockchain-Assisted Self-Sovereign Identities on Education: A Survey,” Blockchains, vol. 3, no. 1, art. no. 3, 2025, doi: 10.3390/blockchains3010003.

[5] G. Fernández-Blanco, I. Froiz-Míguez, P. Fraga-Lamas, and T. M. Fernández-Caramés, “Design, Implementation and Practical Energy-Efficiency Evaluation of a Blockchain Based Academic Credential Verification System for Low-Power Nodes,” Applied Sciences, vol. 15, no. 12, art. no. 6596, 2025, doi: 10.3390/app15126596.

[6] M. Sporny, D. Longley, D. Chadwick, I. Herman, G. Cohen, and M. B. Jones, “Verifiable Credentials Data Model v2.0,” W3C Recommendation, 15 May 2025. [Online]. Available: https://www.w3.org/TR/vc-data-model/

[7] M. Sporny et al., “Verifiable Credential Data Integrity 1.0,” W3C Recommendation, 15 May 2025. [Online]. Available: https://www.w3.org/TR/vc-data-integrity/

[8] OpenZeppelin, “Access Control,” OpenZeppelin Contracts Documentation, version 5.x. [Online]. Available: https://docs.openzeppelin.com/contracts/5.x/access-control

[9] Solidity Documentation, “Security Considerations.” [Online]. Available: https://docs.solidity.org/en/latest/security-considerations.html