03 Unified Verifiable Credential Verification Platform

Create a Unified Verifiable Credential Verification Solution

Complexity Level: Medium

Overview:

The Challenge

The verifiable credential ecosystem suffers from **severe tooling fragmentation**, creating significant barriers for developers, solution architects, and organisations implementing digital identity systems. Despite the promise of interoperable digital credentials, the reality is a complex landscape of:

- **Multiple credential formats**: SD-JWT, W3C VC 1.1/2.0, ISO mDL, IETF SD-JWT VC, AnonCreds, etc.
- **Disparate verification libraries**: Each format requires different dependencies, APIs, and implementation approaches
- **Lack of unified tooling**: Developers must juggle multiple tools, libraries, and documentation sources

Current Pain Points:

1. **Development Complexity**
   - Teams must learn and integrate multiple different libraries for comprehensive VC support
   - Each library has different APIs, error handling, and configuration requirements
   - No single source of truth for verification logic

2. **Integration Nightmares**
   - Conflicting dependencies between verification libraries
   - Different programming languages and runtime requirements
   - Incompatible data formats requiring custom transformation layers
   - Libraries like MOSIP's vc-verifier handle only specific formats, requiring additional tools

3. **Testing & Debugging Challenges**
   - No unified tool to test credentials across formats
   - Difficult to diagnose whether issues are in credential generation or verification
   - Limited visibility into verification failures

4. **Maintenance Burden**
    - Keeping multiple libraries updated and secure
    - Tracking changes across different standards bodies
    - Managing breaking changes in verification APIs

5. **Barrier to Adoption**
    - High technical bar prevents smaller organizations from implementing VC solutions
    - Long development cycles due to complexity
    - Risk of implementation errors affecting credential acceptance

The Impact

This fragmentation affects every stakeholder:

- **Developers**: Spend majority of time on integration rather than building features
- **Organizations**: Face extended implementation timelines and higher costs
- **Solution Providers**: Struggle to support multiple credential formats reliably
- **End Users**: Experience inconsistent verification across different platforms

Current Landscape

Existing solutions address only parts of the problem:

- **MOSIP's vc-verifier**: Handles W3C 1.1 and 2.0 credentials with upcoming support for sd+jwt
- **DIF Universal Verifier**: Focused on DIDComm protocols
- **Spruce DIDKit**: Strong on DIDs but limited credential format support
- **Various vendor-specific tools**: Lock users into proprietary ecosystems

Each tool solves a piece of the puzzle, but none provides comprehensive multi-format verification.

Create a Unified Verifiable Credential Verification Solution that provides:

<u>Multi-Format Support</u>

- Auto-detection of credential format
- Support for SD-JWT, W3C VC 1.1/2.0, ISO mDL, and emerging standards

- Consistent verification response structure across all formats

Developer-Friendly Features

- Detailed verification reports showing exact failure points
- Built-in test credentials for each format
- Interactive documentation with live examples

Why This Matters

By solving the tooling fragmentation problem, we can:

- **Accelerate adoption** of verifiable credentials
- **Reduce implementation costs** substantially
- **Improve interoperability** between different credential ecosystems
- **Enable innovation** by lowering the technical barriers to entry
- **Strengthen MOSIP ecosystem** with comprehensive credential support

Exact Task:

Create a **Unified Verifiable Credential Verification Solution**

Mandatory Tasks:

- Auto-detection of credential format
- Support for SD-JWT, W3C VC 1.1/2.0, ISO mDL, and emerging standards
- Structural Validation of VC
- Signature Verification of Inji Supported Credential schemes
- Consistent verification response structure across all formats

Good-to-have Tasks:

Extensibility to be able to use any external api for verification if needed.

Recommended Tech Stack:

- Frontend: ReactJS
- Backend: Utilize existing libraries from MOSIP and other oss projects to achieve this.
- JS / Java

Deliverables:

1. **Core Components (Mandatory)**

- **Unified Verification** (Using existing libraries and tooling):
    - **Auto-detection Logic:** Functionality to automatically detect the format of the submitted verifiable credential (SD-JWT, W3C VC 1.1/2.0, ISO mDL, etc.).
    - **Format-Specific Parsers and Verifiers**:
        - Modules/libraries for parsing and validating the structure of each supported credential format.
        - Implementations of cryptographic signature verification for each format (e.g., JWS for JWT-based VCs, Data Integrity proofs for W3C VCs, cryptographic methods for mDL).
        - Support for revocation checking mechanisms (e.g., status lists, credential status registries) where applicable for each format.
    - **Standardized Verification Respons**e: A consistent JSON-based output format for all verification results, including:
        - Overall isValid boolean status.
        - Detected credentialFormat.
        - verificationReport object containing detailed success/failure reasons (e.g., proofValidationStatus, signatureVerificationStatus, expirationStatus, revocationStatus, schemaValidationStatus).
        - errorDetails array with specific error codes and human-readable messages for failures.
        - verifiedClaims object containing the selectively disclosed or fully revealed claims from the credential.
        - issuerDetails (DID or other identifier, public key information).
        - subjectDetails (DID or other identifier).

2. **System Design & Infrastructure (Mandatory)**

- **Modular Architecture:** A well-defined, extensible architecture that allows for easy addition of new credential formats and verification rules in the future.
- **Containerization Support:** Dockerfiles and deployment configurations for easy deployment.

Resources:

1. **Core Specifications & Standards**

- **W3C Verifiable Credentials (VC) Data Model:**
  - VC Data Model 1.1
  - VC Data Model 2.0 (Candidate Recommendation)
  - W3C DID Core
  - W3C JSON-LD 1.1 (for W3C VC 1.1/2.0 representations)

- **SD-JWT (Selective Disclosure for JWTs):**
  - IETF Draft: Selective Disclosure for JWTs (SD-JWT) (latest version)
  - IETF Draft: SD-JWT-based Verifiable Credentials (SD-JWT VC) (latest version)

- **ISO mDL (Mobile Driving License):**
  - ISO/IEC 18013-5:2021 - Personal identification — ISO-compliant mobile driving licence (mDL) application
  - ISO/IEC 18013-7:2022 - Personal identification — Mobile Driving License (mDL) — Part 7: Device Engagement

- **AnonCreds:**
  - AnonCreds Specification

2. **Existing Verification Libraries & Tools (for analysis, inspiration, and potential integration)**

- **MOSIP's vc-verifier:**
  - GitHub Repository (Focus on W3C 1.1/2.0, mdoc, upcoming SD-JWT support)
- **DIF Universal Verifier:** Explore DIF's work on universal verifiers, particularly around DIDComm protocols. (Note: Specific open-source implementation for "Universal Verifier" by DIF is less common than theoretical discussions; focus on underlying libraries they might use).

- **Spruce DIDKit/SpruceKit:**
  - DIDKit GitHub (Rust-based, cross-platform, good for DIDs and W3C VCs)
  - SpruceKit Documentation (Higher-level SDKs built on DIDKit, with mDL support)
  - http://walt.id **Verifiable Credentials SDK:**
  - walt.id VC Documentation (Open-source, comprehensive, multi-platform support for W3C VCs)

- **Community-driven Libraries (Language Specific):**
    - **Python:** py-ld-signatures, didkit-py, sd-jwt libraries.
    - **JavaScript/TypeScript:** did-jwt, @digitalcredentials/vc-js, jsonld-signatures, sd-jwt-js.
    - **Java/Kotlin:** sd-jwt (Authlete's Java lib), titanium-json-ld (for JSON-LD).
    - **Rust:** ssi (core library used by DIDKit), sd-jwt-rust.