Decentralized Identity Module Documentation

1. Overview

The Decentralized Identity module provides a comprehensive framework for managing decentralized identities, focusing on the creation, storage, and verification of cryptographic keys, DID documents, and verifiable credentials. It integrates several components to streamline identity operations, with cryptographic material managed through the `KeyManager`. This module supports multiple cryptographic algorithms and allows secure key storage and retrieval, encoding, and verification.

Key features of the Decentralized Identity module include:

- Secure cryptographic key management through `KeyManager`.
- Dynamic key pair generation using `PKIFactory`.
- DID document creation and verification.
- Issuance and validation of Verifiable Credentials (VCs).
- In-memory storage and identity management with `IdentityManager`.
- Unified error handling via `IdentityError`.

2. Core Components

2.1. `credential_issuance.rs`

Defines the `CredentialIssuer` structure for issuing Verifiable Credentials (VCs). This includes:

- **`fn new(did_document: DIDDocument, signing_key: PublicKey, key_manager: KeyManager)`** – Initializes a `CredentialIssuer` with a DID document and signing key.
- **`fn issue_credential`** – Creates and signs a new VC using the issuer's key.
- **`fn sign_credential(&self, vc: &VerifiableCredential)`** – Signs a credential to ensure its integrity and authenticity.

2.2. `did_document.rs`

Defines the `UserDocument` structure to manage user identities and Verifiable Credentials:

- **`fn new(did_document: DIDDocument, verifying_key: PublicKey)`** – Creates a new user identity document.
- **`fn add_credential(&mut self, credential: VerifiableCredential)`** – Adds a new credential.
- **`fn get_public_key_raw_bytes(&self)`** – Retrieves the raw public key bytes.
- **`fn display_vcs(&self)`** – Prints all Verifiable Credentials.
- **`fn to_json(&self) / fn from_json(json_str: &str)`** – Serializes and deserializes user data.

2.3. `did.rs`

Manages Decentralized Identifiers (DIDs) and associated cryptographic elements:

- **`fn new_with_keys(identity_suffix: &str, key_manager: &KeyManager, algorithm: Algorithm)`** – Creates a DID with an associated key.
- **`fn add_public_key(&mut self, key: &PKI, key_id: &str, algorithm: Algorithm)`** – Adds a public key to a DID document.
- **Structures**: `DIDDocument`, `PublicKey`, `Authentication`, `Service`, `Proof`.

2.4. `identity_flow.rs`

Defines high-level functions for DID management:

- **`fn create_did_with_algorithm`** – Creates a new DID with a specified algorithm.
- **`fn add_key_to_did`** – Adds a new key to an existing DID document.

2.5. `identity_error.rs`

Defines the `IdentityError` enum for handling decentralized identity errors:

- **Variants include**: `MissingPublicKey`, `InvalidDID`, `SerializationError`, `UnknownError`.

2.6. `identity_mgmt.rs`

Implements the `IdentityManager` for managing user identity documents:

- **`fn save_user_document`** – Stores a new `UserDocument`.
- **`fn remove_user_document(&mut self, did: &str)`** – Deletes a user document.
- **`fn get_user_document(&self, did: &str)`** – Retrieves a user document.
- **`fn upsert_user_document`** – Updates or inserts a `UserDocument`.

2.7. `key_mgmt.rs`

Implements `KeyManager` for cryptographic key handling:

- **`fn add_key(&mut self, key_id: String, pki: PKI)`** – Adds a cryptographic key.
- **`fn get_private_key(&self, key_id: &str)`** – Retrieves a private key.
- **`fn get_public_key(&self, key_id: &str)`** – Retrieves a public key.
- **`fn encode_key_to_base64 / fn decode_key_from_base64`** – Encodes and decodes keys in Base64 format.

2.8. `pki_factory.rs`

Defines the `PKIFactory` for creating key pairs dynamically:

- **`fn create_pki(algorithm: Algorithm)`** – Generates a key pair for a specified algorithm.
- **`fn sign(&self, data: &[u8]) / fn verify(&self, data: &[u8], signature: &[u8])`** – Provides signing and verification functionalities.

2.9. `vc.rs`

Manages the creation and signing of Verifiable Credentials:

- `fn new` – Initializes a new VC with an issuer, subject, and claims.
- `fn add_claim(&mut self, key: String, value: String)` – Adds claims to a credential.
- `fn sign(&mut self, proof_value: String, created: String, verification_method: String)` – Signs the VC.

3. Conclusion

This module provides a scalable, flexible, and secure system for managing decentralized identities, supporting multiple cryptographic algorithms and seamless DID-based authentication. The inclusion of robust key management, verifiable credential issuance, and an intuitive identity storage solution ensures reliability for decentralized applications.