Verifiable Credential Project Using DIDKit – Documentation

Project Overview

This project demonstrates how to generate Decentralized Identifiers (DIDs) and issue Verifiable Credentials (VCs) using the didkit library. It simulates the process of creating and verifying credentials, leveraging JSON-LD standards and cryptographic proofs.

Tech Stack

- Python
- didkit (DID/VC library)
- pyld (JSON-LD processing)
- requests (HTTP calls)
- libsodium-dev (Cryptography dependency)

Step-by-Step Code Breakdown

Step 1: Installing Dependencies

```
python
CopyEdit
!pip install didkit
!pip install pyld
!pip install requests
!apt-get install -y libsodium-dev
```

- Installs the DIDKit Python library (for DIDs and VCs)
- Installs `pyld` for handling JSON-LD
- Installs `requests` for API calls
- Installs `libsodium-dev`, a dependency for secure cryptographic operations

Step 2: Importing Libraries

```
python
CopyEdit
import didkit
import json
import requests
```

- `didkit`: core library to generate DIDs and Verifiable Credentials

- `json`: for formatting/verifying JSON data
- `requests`: to send/receive data from APIs (if needed)

I'll now extract and explain **all remaining cells**, including:

- DID generation
- Key creation
- Credential schema
- Signing and verifying credentials

Let me proceed to extract and summarize the rest of the notebook first.

Great — here's a complete **step-by-step explanation** of the remaining code from your notebook:

Verifiable Credential Project Using DIDKit – Full Explanation

Step 3: Generate a Cryptographic Key & DID

```
python
CopyEdit
key = didkit.generate_ed25519_key()
did = didkit.key_to_did("key", key)
print("DID:", did)
```

- `generate_ed25519_key()`: Creates a new Ed25519 private key.
- `key_to_did()`: Converts the key into a Decentralized Identifier (DID).
- Output: A DID like `did:key:z6Mk...`

Step 4: Get Verification Method

```
python
CopyEdit
did_document = didkit.key_to_verification_method("key", key)
print("Verification Method:", did_document)
```

- Converts the key to a DID *verification method*, used to sign and verify credentials.
- Example: `did:key:z6Mk...#z6Mk...`

Step 5: API Call to Dataverse

```
python
CopyEdit
base_url = "https://demo.dataverse.org"
endpoint = "/api/search?q=metadata"
response = requests.get(base_url + endpoint)
print(response.json())
```

- Sends an HTTP GET request to Dataverse API to fetch metadata-related datasets.
- Returns a JSON of datasets matching the query.

Step 6: Credential Verification (Placeholder)

```
python
CopyEdit
result = didkit.verify_credential(vc_signed, {})
print(result)
```

- This line tries to verify a credential `vc_signed`, but it's **not defined yet** at this point. The correct flow happens later.

Step 7: Asynchronous Function for Credential Issuing and Verifying

```
python
CopyEdit
import asyncio
import didkit
import json

async def issue_and_verify_credential(key):
```

- Defines an **async function** to issue and verify credentials using the key.

Step 8: Create and Sign a Verifiable Credential

```
python
CopyEdit
    did = didkit.key_to_did("key", key)
    verification_method = await didkit.key_to_verification_method("key", key)

    proof_options = {
        "proofPurpose": "assertionMethod",
        "verificationMethod": verification_method
    }

    vc = {
        "@context": ["https://www.w3.org/2018/credentials/v1"],
        "type": ["VerifiableCredential"],
        "issuer": did,
        "issuanceDate": "2025-07-05T00:00:00Z",
        "credentialSubject": {
            "id": did,
            "contribution": "Digitised 19th-century manuscript"
        }
    }
```

- A **Verifiable Credential (VC)** is created for the issuer and subject (same DID here).
- The subject's claim is a `"contribution"` to a digital manuscript.

Step 9: Sign the Credential

```
python
CopyEdit
    vc_signed = await didkit.issue_credential(json.dumps(vc),
json.dumps(proof_options), key)
    print("Signed VC:", vc_signed)
```

- Uses the `issue_credential` function to sign the VC using the private key and verification method.
- Output is a signed JSON credential.

Step 10: Verify the Credential

```
python
CopyEdit
    result = await didkit.verify_credential(vc_signed, json.dumps({}))
    print("Verification Result:", result)
```

- Verifies the signed VC to confirm that the cryptographic proof and structure are valid.

Step 11: Run the Async Function

```
python
CopyEdit
asyncio.run(issue_and_verify_credential(key))
```

- Executes the async function to issue and verify the VC end-to-end.

Expected Output

- DID
- Verification Method
- Signed Verifiable Credential
- Verification Result (success/failure)

How to Run This Project

1. Open Google Colab or Jupyter
2. Install dependencies
3. Run all cells in order
4. Verify signed credentials at the end

Project Structure

Single notebook file: `bi_2.ipynb`

Use Cases

- Academic project on Self-Sovereign Identity (SSI)
- Demo for Verifiable Credentials and DIDs
- Integration with platforms like Dataverse