# **Appendix 1 - Version 1.3 \_12/12/2023**

The following sections of code are taken from the source code that was amended so as to run the proof of concept. Each of the sections are delineated by a **~~~~~~** and each section of source code have interlaced screen shots from the code that was ran and committed to my GitHub repository.

# **README.md**

**# IOTA-IF-Certify**

This project is a proof of concept of decentralized identity and verifiable credentials on IOTA decentralized ledger technology. It contains an abstraction layer to the IOTA Identity library, to indicate what steps more clearly are needed to create and verify decentralised identities and verifiable credentials. It also contains an example using this abstraction layer, showcasing a typical flow of a holder getting a credential from an issuer, and the verification of the credential by a verifier.

**## Requirements**

- [[Node.js](https://nodejs.org/)](<https://nodejs.org/>) version 16.0 or greater.

- A code editor to inspect the code, like [Visual Studio Code](https://code.visualstudio.com/).

**## Usage**

1. Clone the repository:

```

git clone <https://github.com/mmclean/iota-if-certify>

```

2. Change directory to the root of the repository:

```

cd iota-certify

```

3. Install the dependencies:

```

npm install

```

4. Build the project:

```

npm run build

```

5. Run the Hans example:

```

npm run Hans-example

```

**## Scenario**

> TODO: Currently the program is not interactive yet and only verifies one credential.

This proof of concept aims to provide tools to support the scenario where a holder `H` is required by a verifier `V` to hold credentials, issued by an issuer `I` from the set of issuers approved by the verifier, that prove `H` has sufficiently recent experience to fulfil duties on behalf of some party `X`. Both `H` and `X` (on behalf of `H`), should be able to provide this proof to `V`, while `H` stays in control of what is shared and minimizing the need to query `I`.

**## Design**

**### Disclaimer**

> TODO: Nothing is persisted to disk yet.

As a proof of concept, we seek to get full insight in the process of creating, managing, and verifying DID documents and credentials. As such, this proof of concept does not adhere to security standards when dealing with secret management, where a production grade implementation must. For example, seeds and private keys are committed to disk unsecured.

**### Overview**

As a proof of concept, the code contained in this project is being optimized for readability and understanding, rather than performance. It contains an abstraction layer, which wraps functionality of the IOTA SDK and IOTA Identity libraries in an interface using a [[builder pattern](https://refactoring.guru/design-patterns/builder](https://refactoring.guru/design-patterns/builder)) for readability. It also contains a suite of functions to programmatically fulfil some requirements to be able to run the program to completion, but which are not core to the DID functionality being shown here, like getting enough tokens to hold a DID document in the IOTA ledger.

**### DID derivation**

The [[IOTA Identity](https://wiki.iota.org/identity.rs/introduction/)](<https://wiki.iota.org/identity.rs/introduction/>) framework builds on top of [alias outputs](https://wiki.iota.org/tips/tips/TIP-0018/#alias-output) provided by the IOTA protocol, which in turn are controlled by addresses derived from a seed using [[BIP32](https://en.bitcoin.it/wiki/BIP_0032)](<https://en.bitcoin.it/wiki/BIP_0032>) key derivation. While this allows the derivation of a near limitless amount of DIDs from a single seed, in this proof of concept we limit ourselves to the creation of a single DID document and consequently a single DID per seed for each entity.

**### Storage deposit**

IOTA is a decentralized ledger technology (DLT) aiming to provide fair access to anyone. To prevent intentional or unintentional misuse of available bandwidth and storage space in the network, any DLT has to limit the use of those resources. IOTA is no exception to that. The IOTA protocol limits storage space usage by requiring a [[storage deposit](https://wiki.iota.org/learn/protocols/stardust/core-concepts/storage-deposit/)](<https://wiki.iota.org/learn/protocols/stardust/core-concepts/storage-deposit/>) depending on the amount of storage space used. This storage desposit is returned to the user when storage space is no longer required by the user. Since we are storing DID documents in alias outputs on the ledger, we use storage space and are required to deposit an amount of tokens while holding the DID document on the ledger. Normally these tokens have to be acquired, but in this proof of concept we connect to the [[testnet](https://wiki.iota.org/build/networks-endpoints/#public-testnet)]([https://wiki.iota.org/build/networks-endpoints/#public-testnet),](https://wiki.iota.org/build/networks-endpoints/%23public-testnet),%20) which has a token faucet available where we can request tokens for free.

**### DID key management**

> TODO: This is not implemented yet.

At its core, DID works by providing proof of authenticity in the form of signatures which are verified against public keys. Anyone with access to a private key is able to produce a valid signature. Therefore, private keys must be handled in a secure manner, preferably using a secure key management system. IOTA Identity was designed to work with any key management system through an [abstraction layer](<https://wiki.iota.org/identity.rs/concepts/key_storage/>), requiring a program to implement the `JwkStorage` and `KeyIdStorage` interfaces for a specific key management system. In this proof of concept, we implement an unsecure key management system which stores private keys unencrypted, to be able to inspect them.

**## TODO**

**### Proof of concept**

- [ ] Persist data.

- [ ] Persist secrets.

- [ ] Persist private keys.

- [ ] Implement CLI interaction.

**### Production**

- [ ] Persist secrets securely.

- [ ] Persist private keys securely.

- [ ] Use specification like [OpenID](https://openid.net/specs/openid-4-verifiable-presentations-1\_0.html) for standardized presentation requests and exchange.

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**