Aries Framework JavaScript

Discussion about Wallet API - June 15th 2023

Intro

As most aspects of AFJ, Wallet is a module that has been widely inspired in Indy SDK. Now that the framework aims to be truly agnostic of Indy stack, it could be interesting to think a bit about how we can support different wallet types without the need of “hacking” around them in order to adapt them to work with AFJ.

We’ll discuss three main topics:

- Agent Initialization
- Wallet API
- Wallet interface

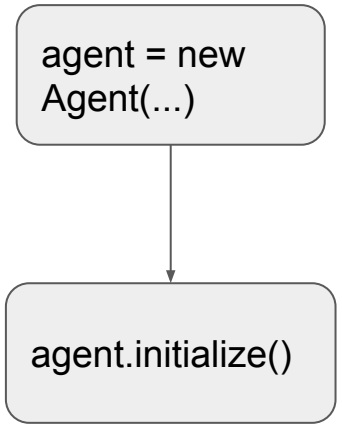
Agent initialization

An agent needs two steps to be up and running: construction and initialization.

In order to be properly initialized, a wallet must be configured or opened at the initialization step (as modules such as mediation require to access wallet and storage).

Currently, we provide an optional WalletConfig object to automate the process, containing fields coming originally from Indy SDK like id, key, keyDerivationMethod, etc.

Even though we could adapt this to work with Askar, this generic WalletConfig is not optimal for every wallet type. So an alternative could be to extract this from Agent constructor and use it right on the wallet provider module.



Agent initialization (cont.)

```
const agent = new Agent({
  config: {
    label: 'agentLabel',
    walletConfig: {
      id: 'walletId',
      key: 'walletKey',
      keyDerivationMethod: KeyDerivationMethod.Raw,
      storage: { type: 'storageType' },
    },
  },
  dependencies: agentDependencies,
  modules: {
    askar: new AskarModule({
      ariesAskar,
    }),
  },
})
```

```
const agent = new Agent({
  config: {
    label: 'agentLabel',
  },
  dependencies: agentDependencies,
  modules: {
    askar: new AskarModule({
      ariesAskar,
      walletConfig: {
        id: 'walletId',
        key: 'walletKey',
        keyDerivationMethod: KeyDerivationMethod.Raw,
        storage: { type: 'storageType' },
      },
    }),
  },
})
```

Wallet API

Current Wallet module API:

- `initialize(walletConfig: WalletConfig): Promise<void>`
- `create(walletConfig: WalletConfig): Promise<void>`
- `createAndOpen(walletConfig: WalletConfig): Promise<void>`
- `open(walletConfig: WalletConfig): Promise<void>`
- `rotateKey(walletConfig: WalletConfigRekey): Promise<void>`
- `close(): Promise<void>`
- `delete(): Promise<void>`
- `export(exportConfig: WalletExportImportConfig): Promise<void>`
- `import(walletConfig: WalletConfig, importConfig: WalletExportImportConfig): Promise<void>`
- `createKey(options: WalletCreateKeyOptions): Promise<Key>`
- `generateNonce(): Promise<string>`

Wallet API

In most cases, Wallet API is basically calling the underlying Wallet object. This currently makes sense because the module itself does not know how to actually handle wallets. The issue is that most of those functionalities are also available through **AgentContext.wallet**, so it might be confusing.

A proposal to deal with this could be to restrict Wallet API to administrative tasks, leaving operations for current Wallet instance to the Wallet object itself.

Wallet API will make use of a “**WalletService**” that must be implemented and registered by the wallet module (e.g. AskarModule, IndySdkModule).

Question: do we really need a Wallet module and API on the Core? Could all these administrative stuff be defined at the external wallet module level?

Wallet API

Proposed Wallet module API:

- `initialize(): Promise<void>`
- `create(options: WalletCreateOptions): Promise<Wallet>`
- `open(options: WalletOpenOptions): Promise<Wallet>`
- `close(): Promise<void>`

Options dependent on wallet type. Should go directly on external wallet module?

- `rotateKey(options: WalletRekeyOptions): Promise<void>`
- `find(options: WalletFindOptions): Promise<string>`
- `delete(options: WalletDeleteOptions): Promise<void>`
- `export(options: WalletExportOptions): Promise<void>`
- `import(options: WalletImportOptions): Promise<void>`

Wallet interface (cont.)

Current Wallet interface:

- `create(walletConfig: WalletConfig): Promise<void>`
- `createAndOpen(walletConfig: WalletConfig): Promise<void>`
- `open(walletConfig: WalletConfig): Promise<Wallet>`
- `rotateKey(walletConfig: WalletConfigRekey): Promise<void>`
- `close(): Promise<void>`
- `delete(): Promise<void>`
- `export(exportConfig: WalletExportImportConfig): Promise<void>`
- `import(walletConfig: WalletConfig, importConfig: WalletExportImportConfig): Promise<void>`
- `createKey(options: WalletCreateKeyOptions): Promise<Key>`
- `sign(options: WalletSignOptions): Promise<Buffer>`
- `verify(options: WalletVerifyOptions): Promise<boolean>`
- `pack(payload: Record<string, unknown>, recipientKeys: string[], senderVerkey?: string): Promise<EncryptedMessage>`
- `unpack(encryptedMessage: EncryptedMessage): Promise<UnpackedMessageContext>`
- `generateNonce(): Promise<string>`
- `generateWalletKey(): Promise<string>`

Wallet interface

Current Wallet interface includes methods for different types of operations:

- Wallet creation/deletion
- Key management and usage (create, sign, verify)
- Wallet import/export/rekey (not tied to the current instance)
- DIDComm message packing/unpacking

Following the previously mentioned strategy, Wallet interface can contain only those methods that work on a specific wallet instance.

Moreover, some methods can be improved, such as the ones for Key management (add a way to find and delete keys).

For message packing/unpacking, Wallet can probably use another object (e.g. “Packer/Unpacker”) that might not be restricted to DIDComm but other envelopes in the future. The Wallet module must somehow tell Agent Core which message packing and unpacking mechanisms it is capable of.

Wallet interface (cont.)

Proposed Wallet interface:

- `open(): Promise<void>`
- `close(): Promise<void>`
- `delete(): Promise<void>`
- `createKey(options: WalletCreateKeyOptions): Promise<Key>`
- `findKey(options: WalletCreateKeyOptions): Promise<Key>`
- `deleteKey(options: WalletCreateKeyOptions): Promise<Key>`
- `sign(options: WalletSignOptions): Promise<Buffer>`
- `verify(options: WalletVerifyOptions): Promise<boolean>`

- `pack(payload: Record<string, unknown>, recipientKeys: string[], senderVerkey?: string): Promise<EncryptedMessage>`
- `unpack(encryptedMessage: EncryptedMessage): Promise<UnpackedMessageContext>`

- `generateNonce(options: WalletGenerateNonceOptions): Promise<string>`
- `generateKey(options: GenerateKeyOptions): Promise<string>`