ADAvault Vault White Paper

Overview

ADAvault are a Cardano Stake Pool Operator (SPO) with a successful track record operating reliable and secure stake pools for over 100 epochs.

We plan to offer a non-custodial vault service which can be used by any person with a Cardano wallet that supports dApps via CIP-030. The service will permit users to define under which circumstances assets are released from the vault.

Potential applications for the service include; escrow for digital assets, secure document transfer and signature, digital inheritance, trust funds, security deposits and vaulting for digital assets.

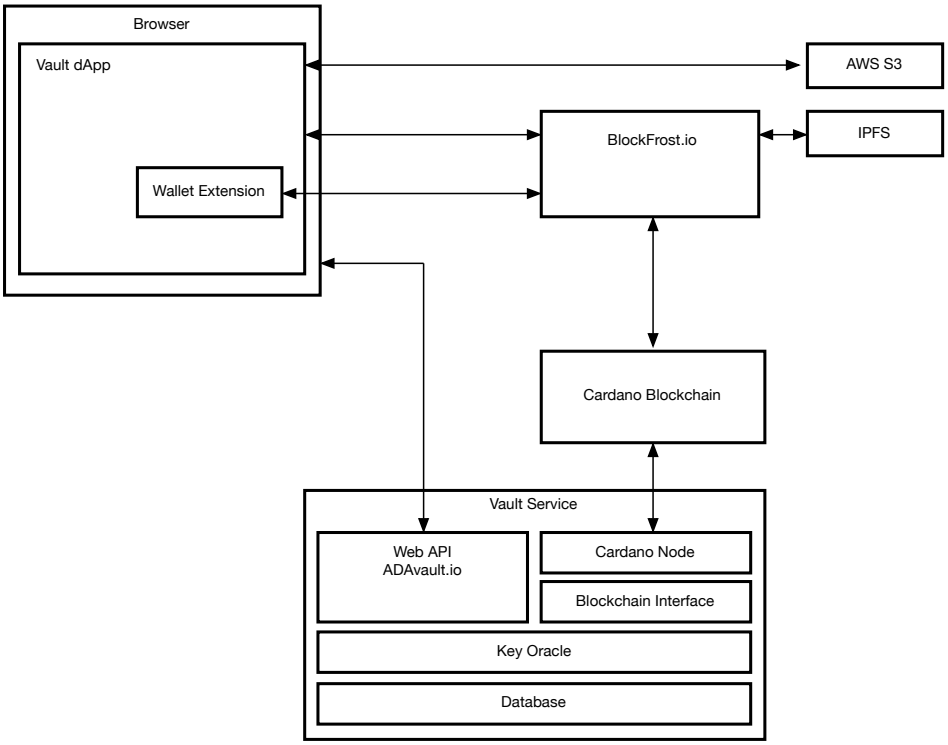
The service uses a combination of smart contract and an associated oracle and will be provided via an API at ADAvault.io. All components of the framework will be provided open source to the community to use and extend. We plan to provide a number of reference implementations that have been audited to provide a high level of assurance.

The service can use a variety of oracles as these become available on-chain. Community members (for example Stake Pool Operators) may choose to create and run vault services, either based on the ADAvault oracle, or using their own implementations.

The service will be provided for a defined nominal transaction fee for the smart contract, with associated fees for the digital asset to be stored dependent upon the storage size of the asset. Storage services that will be supported include IPFS and AWS S3. We will also explore using native tokens (for example ADV utility tokens) or other payment tokens and services, with the goal to create a thriving ecosystem on the base service.

High level Design

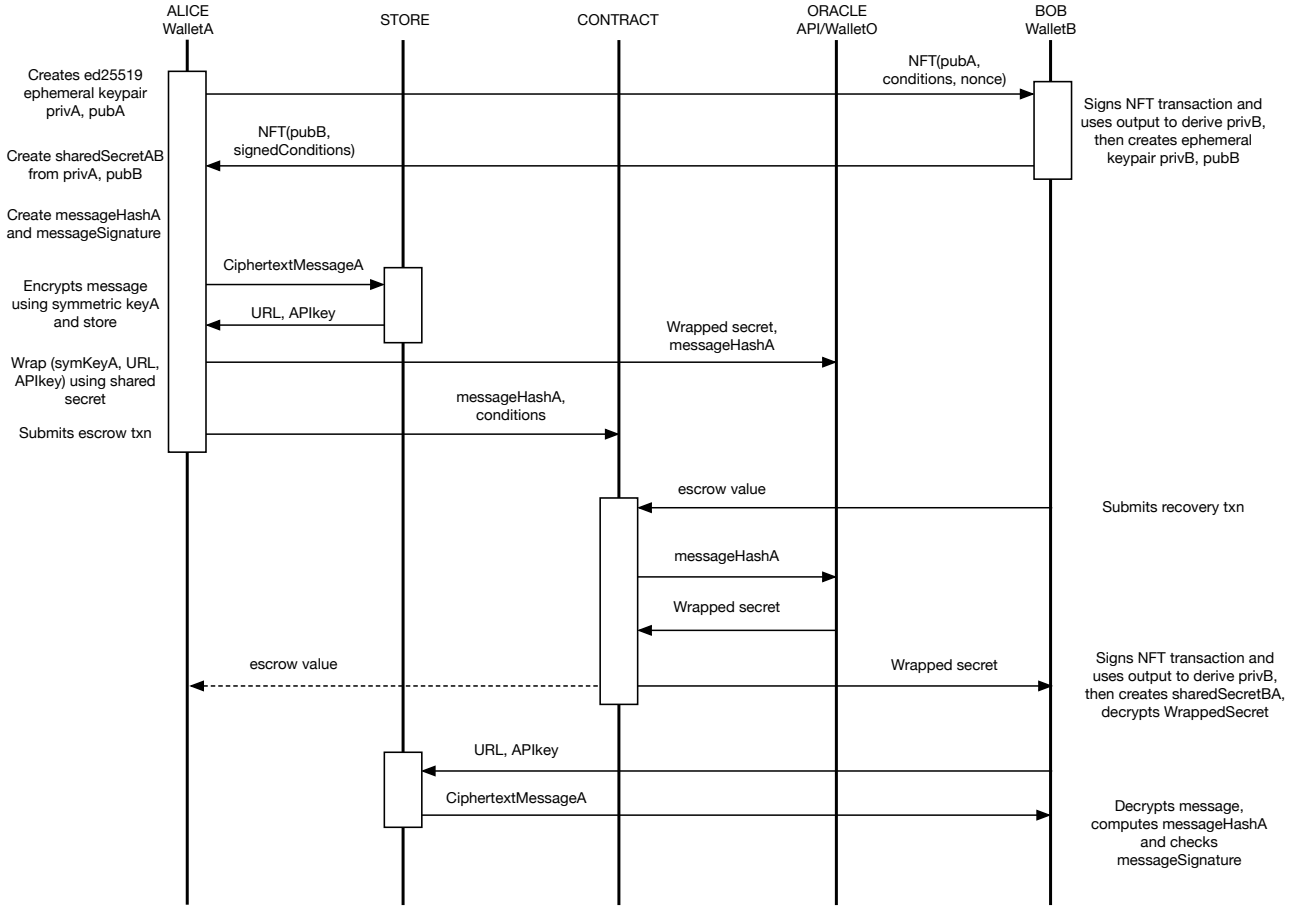
The main components of the design are shown in the architecture overview diagram.



The users Alice and Bob interact with the solution using a browser, connecting the dApp via a CIP-030 compliant Cardano wallet. The smart contract and oracle are used to provide a trusted escrow service that can execute on conditions defined by Alice and communicated to Bob at the initiation of the contract. The exchange of NFT's and associated metadata at initiation allows Alice and Bob to keep a co-signed record of the contract (this function may be performed using native tokens, signed transactions and associated metadata rather than NFTs).

The key oracle is designed for minimal trust and has one role to store the wrapped secret and release the wrapped secret to Bob (via the smart contract) under control of the contract. The oracle cannot access the secret and therefore cannot be compromised to access secret material.

The following sequence digram shows a typical interaction for execution of an escrow contract using the vault service.



Strengths of the design include:

- Design minimises trust that must be placed in the key oracle.
- Symmetric key generation is under control of the sender.
- Allows for the sender to abort contract/remove message or irrevocably rescind the right to abort.
- Design minimises on chain data required and therefore size and cost of the contract and associated transactions.
- Possible to send to multiple recipients by collecting multiple recipient wallets addresses and public keys and submitting to the oracle against the same message hash.
- URL and API key used to access the encrypted message are not public.
- Contract conditions are only limited by availability of reliable and trusted oracles.

- Simple charging model for contract initiation and execution.Time based and potential for timely contract execution.
- Simple design with potential for high assurance possible for key oracle and associated contracts.
- Option to run within layer 2 Hydra head(s) supports very high transaction volumes and lower cost model than layer 1.

The weakness of the design remains the need to place trust in a single oracle to release the key reliably. However it may be possible to remove that trust in a future iteration as capabilities for blockchain secrets to be stored become available[1].

Implementation

We adhere to standards when implementing:

- RFC3565 Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS)
- RFC7748 Elliptic Curves for Security
- RFC8032 Edwards-Curve Digital Signature Algorithm (EdDSA)
- RFC7049 Concise Binary Object Representation (CBOR)
- RFC8152 CBOR Object Signing and Encryption (COSE)
- CIP-008/030 Cardano Improvement Proposals for message signing and wallet integration.

Open Source libraries used in the application will be chosen to confirm to these standards.

Browser based dApp

A browser is used used to present the interface to user, and provides the environment for interaction with user wallets.

The dApp must perform cryptographic functions to permit secure communication between Alice and Bob. The dApp uses:
- The wallet via a CIP-030 interface to perform signing functions, access wallet addresses and associated public keys, and submit transactions.
- Javascript cryptographic libraries to perform ed25519 DH key exchange, hashing, signing, and symmetric encryption (AES-256).

The dApp allows users to specify escrow (contract) conditions, recipients, and store/recover digital assets.

Smart contract

The smart contract will be implemented on the Cardano Blockchain using the Plutus smart contract functionality which is implemented in Haskell. This contract language is based on a functional programming language and therefore can be audited to provide a high level of assurance that it will operate as specified.

Contract conditions can include time based or event based inputs. The initial implementation will be time based, with event based conditions depending on availability of the relevant oracles.

Oracle

The Oracle monitors the API and stores escrow registration requests in the form:
[hashMa, @SCaddress, @recipientWallet, wrappedSecret]

[1] Can a blockchain keep a secret? https://eprint.iacr.org/2020/464.pdf

Escrow requests are only permanently stored if an associated transaction is registered within a defined period at the Smart Contract address (@SCaddress) in order to prevent resource exhaustion attacks[2].

The oracle monitors the chain state (via the oracle wallet) for transactions from the smart contract that request the wrapped secret for a specific asset.

The oracle checks the smart contract address and conditions passed, and if valid, submits a transaction back to the contract to return the wrapped secret(s) associated with the message hash passed. This transaction will return the wrapped secret to Bob and fulfil any escrow conditions such as transfer of funds to Alice.

The Oracle implementation will use the Plutus application backend (PAB) once available. In the near term it will be implemented using OGMIOS or similar interface to a local instance of the Cardano Node. Persistent state will be stored locally in a mysql instance, but could be distributed amongst multiple instances.

Wrapped secrets stored by the Oracle must be protected from the recipient Bob to prevent claims that are outside of the crow contract. Compromise of the Oracle by any other party does not result in the disclosure of confidential information, however the Oracle should be secured adequately to mitigate threats against the integrity and availability of the service.

[2] Rate limiting can also be implemented