ERC721 Controlled Vaults

APYMON

Project Overview

The purpose of this project is to allow token holders of a preexisting ERC721 token contract (Key Contract) to create and control individual Vaults. A Vault is an entity in which ETH, ERC20, ERC721, ERC1155, and CryptoPunks can be stored. Each token (Key Token) from the Key Contract can be used once to create a Vault. Ownership of the created Vault is directly tied to the Key Token used to create it and therefore ownership of the Vault is transferred when the Key Token is transferred. Owning a Vault allows for the tokens inside to be withdrawn or timelocked.

Requirements

Roles

- **Admin -** The entity that is responsible for deploying and managing the contracts that handle Vault creation. Does not have access to any Vaults.
- **User -** A holder of one or more of the Key Tokens.

Definitions

- **Vault** - An entity that tokens (ERC20, ERC721, ERC1155, CryptoPunks) can be deposited into and allows the User that owns it to withdraw tokens and timelock withdraws. Ownership is tied to the Key Token that created it.
- **Key Contract -** The preexisting ERC721 token contract whose tokens will act as keys to the Vaults.
- **Key Token -** A single token from the Key Contract that can be used to create and control a Vault instance.

Features

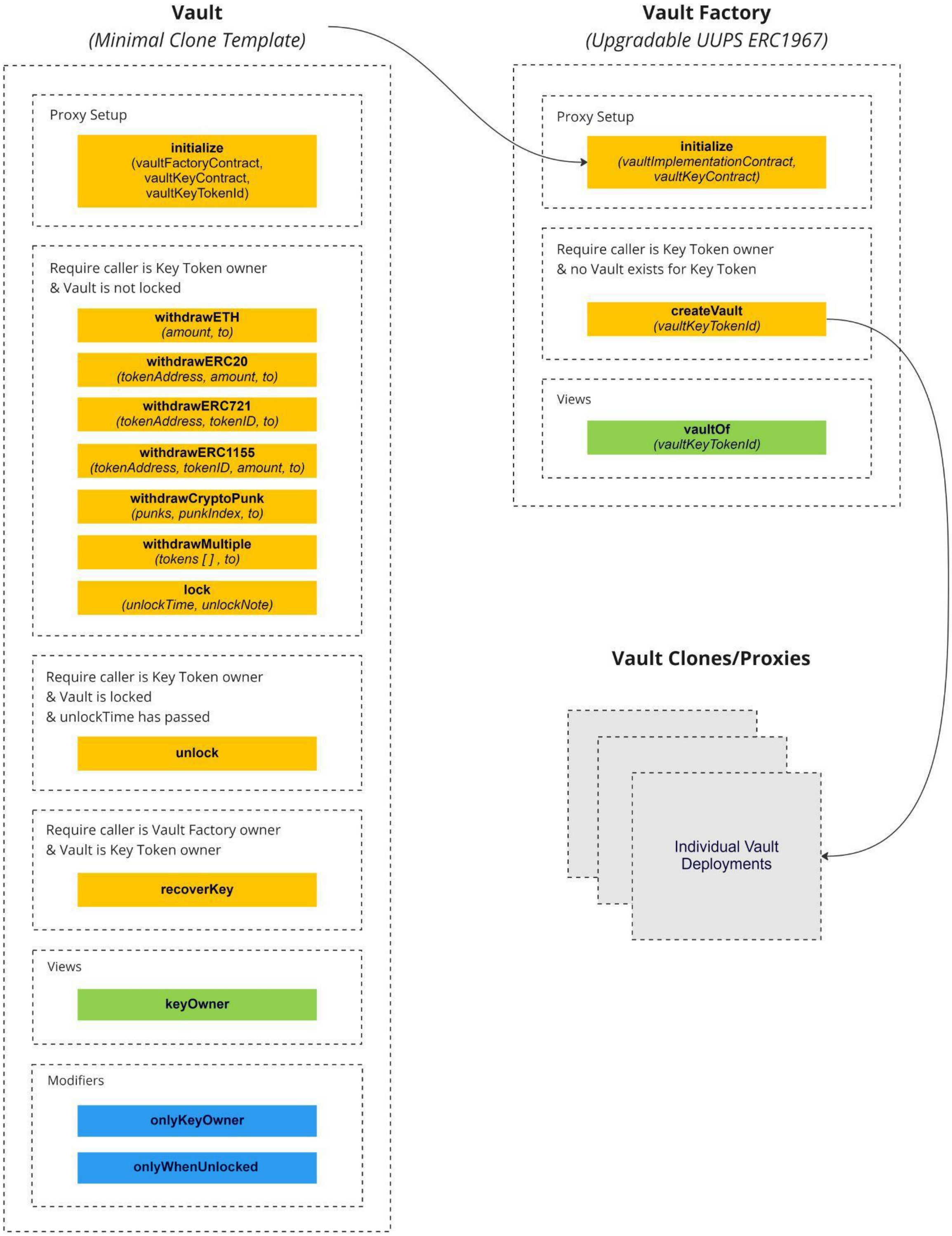
- Create a Vault for a Key Token if one does not exist. (User)
- Determine if a Key Token has been used to create a Vault, and if so, what Vault it created. (Anyone)
- Determine the owner of a Vault / determine who owns a Vault's Key Token. (Anyone)
- Deposit ETH and other tokens (ERC20, ERC721, ERC1155, CryptoPunks) to a Vault. (Anyone)
- Withdraw ETH and other tokens (ERC20, ERC721, ERC1155, CryptoPunks) from a Vault. (User that owns the Key Token of the Vault)
- Timelock a Vault to prevent withdrawals for a period of time and leave an accompanying note. (User that owns the Key Token of the Vault)
- Recover the Key Token from a Vault in the event it owns the Key Token that controls it. (Admin)

Use Cases

- A user can store tokens in a Vault and easily transfer all of them at once between wallets in a single low cost transaction by simply transferring the Key Token.
- A user can sell a bundle of tokens by depositing them in a Vault and selling the Key Token.
- A user can create a time capsule of sorts by depositing tokens in a Vault and timelocking it to ensure tokens are not withdrawn prior to a given date. A note can be included for posterity.

Architecture

The diagram below outlines the design of the vault contracts. It makes use of a factory pattern to deploy minimal proxy clones of a Vault implementation contract. The design consists of two contracts; a Vault implementation contract to be cloned, and a VaultFactory contract that handles the process and stores the addresses of each Vault deployment. Upon deployment of a new Vault, an ERC721 key token is set in the initializer and cannot be changed. This design results in each Vault being completely separate from the others and only being controlled by the holder of the key token it was created with.



Contracts

This section outlines the contracts in the project and gives more details around each of the functions they implement.

VaultFactory.sol

The contract responsible for allowing users to create Vaults for a given Key Token and keeping track of the Vaults created. It is implemented as an upgradable contract following OpenZeppelin's UUPS ERC1967 process.

- initialize(address vaultImplementationContract, address vaultKeyContract): Sets the ERC721 contract whose tokens will be acting as keys for the Vaults created, and sets the implementation contract to be used when creating Vault clone/proxy deployments.

- createVault(uint256 vaultKeyTokenId): Creates a Vault deployment using the key token id provided. A Vault must not already exist for the given key token, and the caller must own the key token provided. Vault deployments are done using OpenZeppelin's Minimal Clone library with the implementation contract address previously set in the initializer.

- vaultOf(uint256 vaultKeyTokenId): A view function that returns the address of a Vault deployment for a given key token id. This allows anyone to see if a Vault has been created for a given key token.

Vault.sol

The contract responsible for handling deposits, withdraws, and timelocks for the following token types; ETH, ERC20, ERC721, ERC1155, and CryptoPunks. The Key Token that controls withdrawals and timelocks can only be set once per deployment. It is implemented as a non-upgradable contract following a minimal clone pattern for each deployment.

- withdrawETH(address to, uint256 amount): Withdraws ETH, can only be called by the key token owner and when the vault is unlocked.

- withdrawERC20(address token, address to, uint256 amount): Withdraws an ERC20 token, can only be called by the key token owner and when the vault is unlocked.

- withdrawERC721(address token, uint256 tokenId, address to): Withdraws an ERC721 token, can only be called by the key token owner and when the vault is unlocked.

- withdrawERC1155(address token, uint256 tokenId, address to, uint256 amount): Withdraws an ERC1155 token, can only be called by the key token owner and when the vault is unlocked.

- withdrawCryptoPunk(address punks, uint256 punkIndex, address to): Withdraws a CryptoPunk, can only be called by the key token owner and when the vault is unlocked.

- withdrawMultiple(TokenWithdraw[ ] calldata tokens, address to): Withdraws multiple tokens at once, can only be called by the key token owner and when the vault is unlocked. The TokenWithdraw struct is defined in IVault.sol.

- lock(uint256 _unlockTime, string calldata _unlockNote): Locks the vault until the provided time has passed and unlock() is called, can only be called by the key owner and when the vault is unlocked. The time is measured in seconds since the unix epoch. The unlock time is emitted as event data.

- unlock(): Unlocks the vault, can only be called by the key owner and when the unlock time has passed. The note provided when the vault was last locked is emitted as event data and returned.

- recoverKey(): Recovers the ERC721 key token to the VaultFactory contract owner, can only be called if the vault owns the key token.

- keyOwner(): A view function that returns the address of the owner of the key token.