Contract Design

In this paper, we will explain the overall structure of the contract group that makes up unWallet and the behavior of Module Contract, which is responsible for the basic functions of the wallet.

1. Overall structure

1-1. Overview

As shown in Figure 1, unWallet consists of multiple contracts. First, outlines of these major contracts are described below.

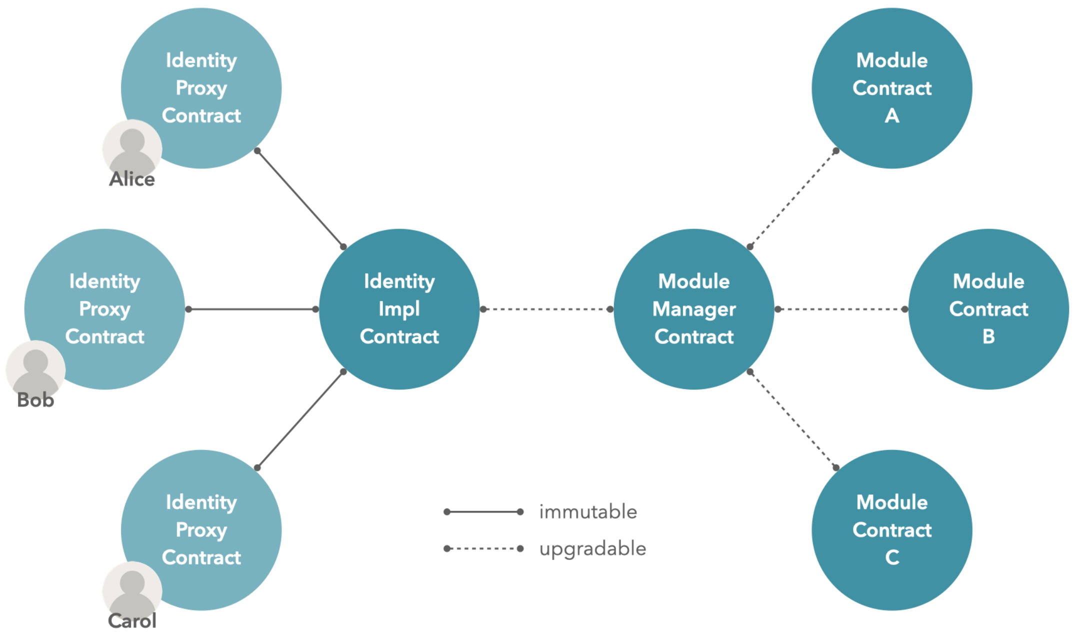

Figure 1. Main contract group

1-1-1. Identity Proxy Contract

A very lightweight proxy contract generated per end-user. The address of this contract will be the wallet address of each end user. As its name suggests, it is a proxy, so all processing is delegated to Identity Impl Contract via delegatecall.

1-1-2. Identity Impl Contract

A contract that implements only the core functionality of the wallet, such as wallet owner change and external calls. Since almost all functions can only be used through Module Contracts, it is necessary to pass through the verification process implemented in each Module Contract in order to execute the above process. Access control from the Module Contract is based on the Module Manager Contract set for this contract.

It also has a mechanism for delegating arbitrary function calls (onERC721Received, etc.) for this contract to the previously specified Module Contract. This designation is also done in the Module Manager Contract.

1-1-3. Module Manager Contract

A contract that manages a Module Contract accessible to an Identity Impl Contract. Only Module Contracts registered with this contract can use the core functions implemented in the Identity Impl Contract.

In order to avoid a situation where all Module Contracts are unregistered for some reason and the wallet stops functioning, there is also a mechanism that locks the specified Module Contract so that it cannot be permanently unregistered.

1-1-4. Module Contracts

It is a contract group that implements each function of the wallet, and is responsible for the entry point of each function. The basic behavior is to perform some kind of verification process (such as confirming whether the entered electronic signature is from the wallet owner or not), and finally to activate the core function implemented in the Identity Impl Contract. is.

Examples of functions implemented by the Module Contract are shown below.

- A meta-transaction feature that executes external calls from the Identity Proxy Contract only when certain conditions are met
- Recovery function that changes the wallet owner only when certain conditions are met

1-2. Detail

1-2-1. Upgradability

Since unWallet has a module structure as described in 1-1, upgrades are performed in a different format from general proxy structure contracts. Specifically, as shown in Figure 2, the Module Manager Contract upgrades by registering and canceling the registration of the Module Contract.

This upgrade will be applied to all Identity Proxy Contracts connected to the Identity Impl Contract that the corresponding Module Manager Contract is set, so even if a large number of Identity Proxy Contracts are generated, it will be easy to deal with. (However, since the wider the scope of application, the greater the risk, it will be important to divide the Module Manager Contract into appropriate units in the future).

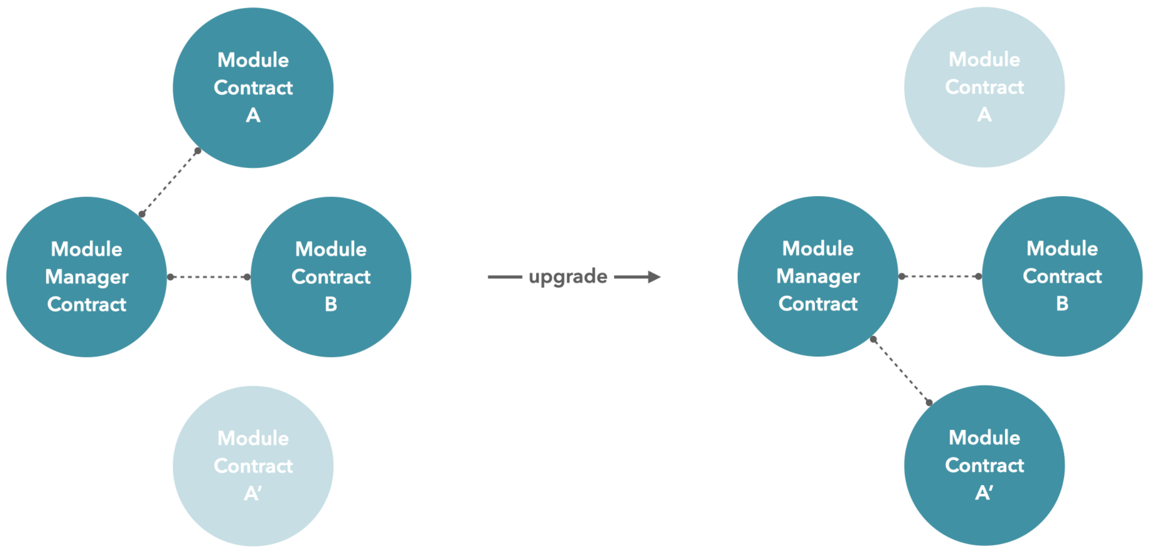

Figure 2. Modular structure based upgrade

In a general proxy model, multiple implementation contracts cannot be set for one proxy contract, so it is difficult to partially upgrade. It can be easily realized by replacing only Contract A with Module Contract A'.

The Module Manager Contract owner (single EOA or contract account) can freely register and unregister Module Contracts. In the initial phase, when there is a high possibility that urgent bug fixes and vulnerability resolution will occur, use the module contract locking mechanism described in 1-1-3 The development team plans to manage the Module Manager Contract while establishing a certain amount of time from the application for registration or cancellation until it is executed, etc., but in the medium to long term , we plan to set a contract account with some kind of governance mechanism as the owner of the Module Manager Contract, gradually weakening the power of the development team.

The modular structure of unWallet also benefits the upgradability of “calling” contracts such as contract wallets. Specifically, a step-by-step upgrade is possible while the old and new version implementations coexist. This means, for example, if you want to upgrade Module Contract A to its new version, Module Contract A', you can do so with the following flow:

1. Register Module Contract A’
2. While Module Contract A and Module Contract A’ coexist, wait for implementation support of the client group (an application that becomes the UI of the wallet).
3. Unregister Module Contract A after all clients have responded to Module Contract A’

Of course, if there is a client who wants to continue using Module Contract A for some reason, there is also the option of "do not perform step 3".

As various protocols and Dapps are being developed, wallets are important products that need to function as the “subject” of daily on-chain actions while adapting to those changes. It has flexible upgradability as mentioned above.

Note that the upgradability of a "called party" contract, such as a token contract, is centered around the mechanisms that control the delegates of incoming calls, which are essentially a single delegate at a given time. Therefore, it is not possible to coexist with multiple versions of the implementation for a particular call as described above, but the upgradability of the “calling side” contract, such as the contract wallet, controls the path to making an external call. The mechanism is central, and there is no problem if there are multiple routes (even if the same external call is made as a result) as described above, and the wallet owner can decide which route to use. When designing upgradability, it is also important to choose mechanisms appropriate to the nature of such contracts.

2. Module

The behavior of Module Contract responsible for the basic functions of unWallet is explained.

2-1. Relayer Module Contract

Module Contract for executing arbitrary external calls (Meta-transactions) from the Identity Proxy Contract.

The function responsible for executing the meta-transaction takes the following arguments:

- Data that specifies the content of the external call you want to make and the refund method of the fee required for it
    - From which Identity Proxy Contract to execute which function of which contract with what arguments
    - Whether the Identity Proxy Contract refunds the transaction fee paid by the Relayer EOA (the EOA that is the starting point of the meta-transaction)
        - If so, in which token (ETH, ERC20 token, etc.)
- Electronic signature by the wallet owner on the above data
    - Strictly speaking, the data to be signed includes data to prevent replay attacks.

If these can pass the verification within this Module Contract, the specified external call will be executed with the flow shown in Figure 3 (Refunds can also be considered a type of external call).

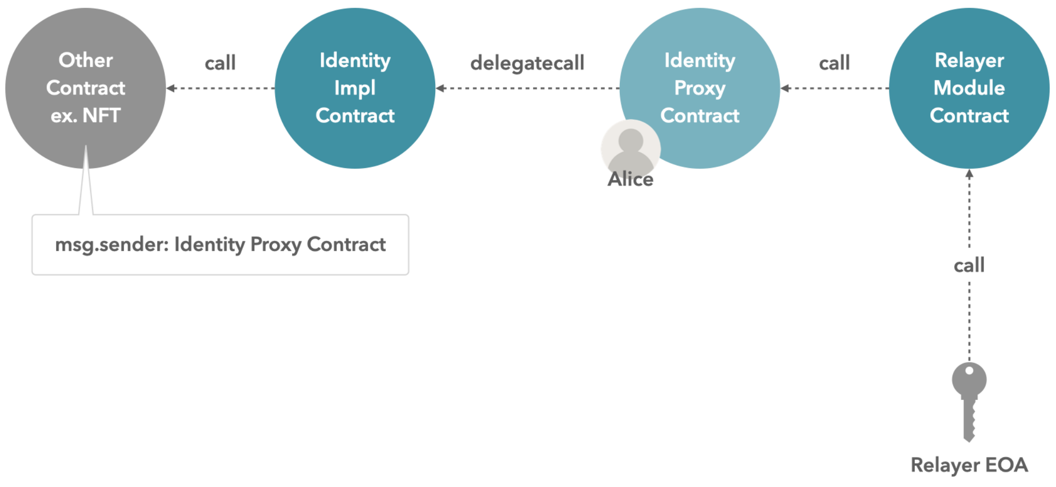

Figure 3. Meta-transaction execution flow

Note that any EOA can become a Relayer EOA as long as it can pass valid arguments. In addition, not refunding means that the Relayer EOA takes over the transaction fee that should be paid by the end user who is the wallet owner. Therefore, for example, in order to lower the hurdles for using Dapps, an organization that provides some Dapps should be responsible for Relayer EOA only for meta-transactions that make external calls to contracts belonging to their own Dapps, and shoulder end-user fees. is also possible.

2-2. Delegate Module Contract

A Module Contract that delegates any function calls to an Identity Proxy Contract.

For function calls (such as onERC721Received) that you want to delegate processing to this Module Contract, you need to register the function selector and the address of this Module Contract in the Module Manager Contract in advance. For function calls set to perform delegation, delegation is performed according to the flow shown in Figure 4.

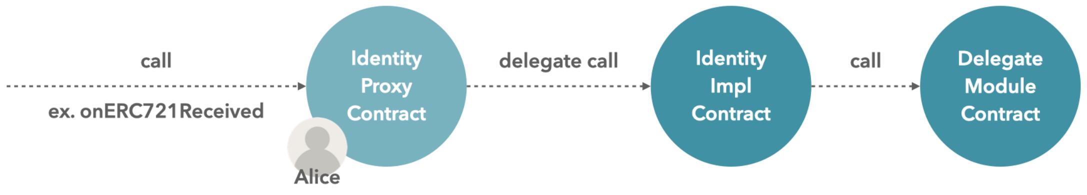

Figure 4. Function Call Delegation Flow for Identity Proxy Contract

2-3. Ownership Manager Module Contract

A Module Contract that manages wallet owners (basically a single EOA).

In the case of EOA, if the corresponding private key is lost, all the assets held by the account will be lost. By making it possible to change the wallet owner when the conditions are met, it is possible to deal with the loss of the private key (corresponding to the EOA corresponding to the wallet owner). Even if the wallet owner is changed, the address of the wallet (Identity Proxy Contract) will not change.

There are various conditions for changing the wallet owner, but these days, the condition is called “social recovery,” which is defined as “getting the consent of more than half of the accounts (guardians) set in advance by the wallet owner.” method (Fig. 5) is considered to be the most promising method, and unWallet also adopts this method.

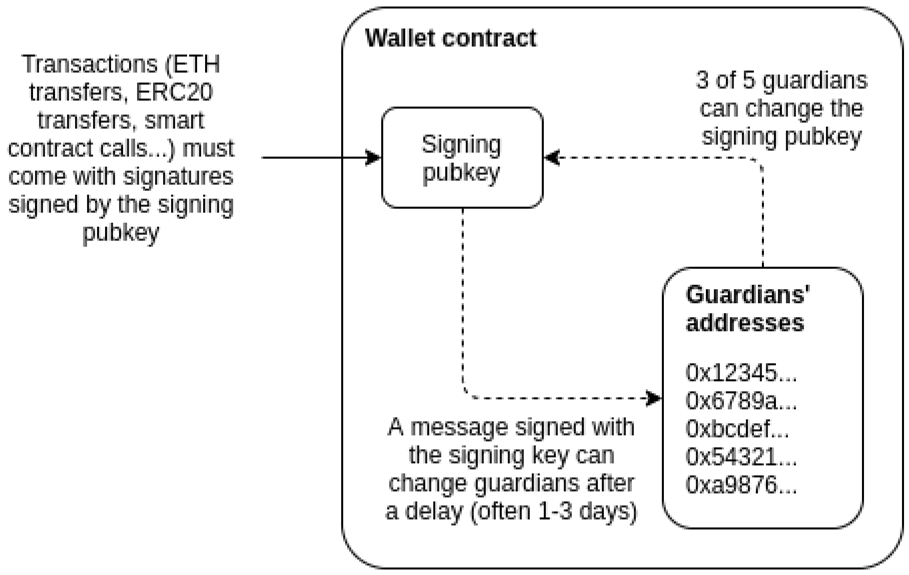
Transactions (ETH transfers, ERC20 transfers, smart contract calls...) must come with signatures signed by the signing pubkey
Wallet contract
3 of 5 guardians can change the signing pubkey
Signing pubkey
Guardians' addresses
0x12345...
0x6789a...
0xbcdef...
0x54321...
0xa9876...
A message signed with the signing key can change guardians after a delay (often 1-3 days)

Figure 5. Social recovery
(Quoted from https://vitalik.ca/general/2021/01/11/recovery.html)

2-4. Transfer Manager Module Contract

A module to restrict the transfer of FT (ERC20 token) and NFT (ERC721 token and ERC1155 token).

In recent years, fraud methods such as luring users to fake websites of well-known projects and making them execute transactions that transfer their FTs and NFTs to fraudster accounts have become popular. Hundreds of millions of yen in damage has been caused, but by using this Module Contract and imposing the following restrictions, it is possible to counter such fraud.

- Lock your own FTs and NFTs (make them untransferable)
- Set up a whitelist of accounts to which you can transfer your FTs and NFTs
- A certain amount of time must pass from application to application of the above restrictions.

In order to actually make such restrictions work, it is necessary to cooperate with a Relayer Module Contract that makes an external call equivalent to token transfer. In other words, it is necessary to implement a mechanism in the Relayer Module Contract that ``checks the constraints set by this Module Contract before making an external call, and if the condition is met, does not make an external call".

Currently, even highly literate users find it cumbersome and difficult to precisely verify the details of a requested transaction, making it difficult to prevent sophisticated fraud simply by being careful. With the progress of mass adoption, the above-mentioned mechanism that realizes "no damage even if you are caught in a trap" will become more and more important.