Argent Smart Wallet Specification

v2.5

April 23, 2021

1 Specifications

1.1 Introduction

The Argent wallet is an Ethereum Smart Contract based mobile wallet. The wallet's user keeps an Ethereum account (Externally Owned Account) secretly on his mobile device. This account is set as the owner of the Smart Contract. User's assets (e.g. ETH, ERC20, ERC721 or ERC1155 tokens) are stored on the Smart Contract. With that model, logic can be added to the wallet to improve both the user experience and the wallet security. For instance, the wallet is guarded, recoverable, transferable, lockable and upgradable.

1.2 Guardians

The wallet security model is based on the ability to add *guardians*. A guardian is an account (EOA or smart contract) that has been given permission by the wallet's owner to execute certain specific operations on their wallet. In particular guardians can lock, unlock, and trigger a recovery procedure on the wallet as well as approve the execution of transactions to unknown accounts or the creation of a session key.

We do not impose restrictions on who or what guardians are. They can be a friend's Argent wallet, a friend's EOA, a hardware wallet, or even a paid third-party service.

Adding a guardian is an action triggered by the wallet owner. While the first guardian is added immediately when the wallet is created, all subsequent additions must be confirmed after 36 hours and no later than 48 hours after

the addition was requested. This confirmation window ensures that a pending addition will be canceled (expire) should the wallet be locked or recovered.

Removing a guardian is an action triggered by the wallet owner. It must always be confirmed after 36 hours and no later than 48 hours after the removal was requested. This leaves the legitimate wallet owner enough time to notice and prevent the appointment of an illegitimate guardian (or the dismissal of a legitimate guardian) in case the owner lost control over their mobile device.

1.3 Locking

In case the wallet owner suspects his account (i.e. device) is compromised (lost, stolen, ...), he can ask any of his guardians to lock the wallet for a security period of 5 days. Once the wallet is locked only a limited set of actions can be operated on the wallet, namely the recovery procedure, the unlock procedure, or the revocation of Guardians. All other operations (add guardian, assets transfer, ...) are blocked.

To unlock a wallet before the end of the security period, any guardian should trigger a wallet unlock.

1.4 Recovery

Wallet recovery is a process requested by a user who asserts ownership of a wallet while not being in possession of the owner key. A successful recovery sets a new account as the wallet owner. This process should be validated by the wallet's guardians to be executed and thus requires the wallet to have at least 1 guardian. Once a recovery has been executed it may be finalised after 48 hours, unless it has been cancelled.

The number of signatures needed to execute a recovery is given by

$$\left\lceil \frac{n}{2} \right\rceil$$

where n is the total number of guardians and $\lceil \rceil$ is the ceiling function.

A recovery can be cancelled before finalisation. The number of signatures (owner and/or guardians) needed to cancel a recovery is given by

$$\left\lceil \frac{n+1}{2} \right\rceil$$

where n is the total number of guardians when the recovery was executed.

Once a recovery is started the wallet is automatically locked. The wallet can only be unlock by finalising or cancelling the ongoing procedure, i.e. guardians cannot unlock during a recovery.

1.5 Ownership Transfer

In addition to recovery it is possible for a user to transfer ownership of his wallet to a new device while still being in possession of the actual phone. This transfer is immediate to avoid service interruption but must be approved by guardians. The number of required signatures is given by

$$1 + \left\lceil \frac{n}{2} \right\rceil$$

where the first signature is the owner and n is the total number of guardians.

1.6 Multi-Call

The wallet can interact with the ethereum ecosystem to e.g. send an asset or interact with the smart-contracts of a decentralized application through multi-calls. A multi-call is a sequence of transactions executed by the wallet one after the other and such that the multi-call fails if any of the transaction fails.

By design the wallet will block a multi-call unless one of the following conditions is satisfied:

1. the multicall is triggered by the wallet owner and each transaction of the multi-call:
    - transfers or approves an asset (ETH, ERC20, ERC721, ERC1155) to a trusted contact, see Section 1.7
    - transfers or approves an asset (ETH, ERC20, ERC721, ERC1155) or calls an authorised Dapp, see Section 1.8
2. the multicall is approved by the owner and a majority of guardians, see Section 1.9
3. the multicall is executed with a valid session key, see Section 1.10

1.7 Trusted Contacts

The wallet keeps a list of trusted addresses managed by the wallet owner. Adding an address to this list is an action triggered by the wallet owner and takes 36 hours to be effective. Removing an address is triggered by the owner and is immediate.

Interacting with a trusted contact to e.g. send an asset, can be triggered by the wallet owner and does not require any additional authorisation.

1.8 Dapp Registry

In addition to trusted contacts, the wallet supports 2 registries of addresses that have been authorised by Argent.

The first registry is enabled by default (opt-out) for all wallets and contains all the dapps that are natively integrated in the Argent client application (Paraswap, Compound, Maker, Aave, etc). The second registry is disabled by default (opt-in) and contains a list of curated dapps that the user can access through WalletConnect.

Each registry entry is a mapping between an authorised address and a filter containing the set of conditions that a transaction from a wallet to the authorised address must satisfy. Adding or removing an entry in a registry is a timelocked operation that can only be initiated by the Argent multisig. The timelock is set to 7 days to give enough time for a user to disable the correspoding registry should they not approve the new addition.

Enabling or disabling a registry for a wallet is achieved via a multicall and requires guardian approvals. The number of required signatures is given by

$$1 + \left\lceil \frac{n}{2} \right\rceil$$

where the first signature is the owner and n is the total number of guardians.

Interacting with an authorised dapp to e.g. invest an asset in a DeFi protocol, can be triggered by the owner provided that the corresponding registry is enabled for the wallet and the transaction passes the associated filter.

1.9 Guardian Approval

Any sequence of transactions (multi-call) can be executed by the wallet owner provided that they obtain approval from their guardians. The number of

required signatures is given by

$$1 + \left\lceil \frac{n}{2} \right\rceil$$

where the first signature is the owner and n is the total number of guardians.

1.10 Session

Users that plan to execute many transactions in a given time window while only requiring their guardian approval once can do so by creating a session. A session defines a temporary session key that can be used to execute any multi-call for as long as the session key is valid. The duration of the session is defined by the owner when the session is created and the session key will automatically become invalid at the end the session.

The number of required signatures to start a session is given by

$$1 + \left\lceil \frac{n}{2} \right\rceil$$

where the first signature is the owner and n is the total number of guardians.

The wallet owner can always close a session before its expiration.

1.11 Upgradability

The wallet is upgradable to add new features and fix potential bugs. The choice of whether to upgrade or not a wallet is left to the wallet owner. In particular, it is not possible for a centralised party such as Argent to force a wallet upgrade and change an implementation that is assumed to be immutable by the owner.

1.12 ETH-less Account

Owner and guardians can execute wallet operations without the need to pay transaction fees and own ETH, i.e. they are ETH-less account. This is achieved by enabling accounts to sign a message showing intent of execution, and allowing a third party relayer to execute the transaction and pay the fee on their behalf. The party signing the transaction can specify if the wallet should refund the gas (partially or totally) required to execute the transaction

to the third party relayer. This pattern, now called meta-transactions, is described in EIP 1077[1] (see Section 2.5).

1.13 Summary of Guardian Operations

	Lock/Unlock	Execute Recovery	Cancel Recovery	Transfer Ownership	Approve Multi-call / Start Session	Enable Registry
	Guardians	Guardians	Owner OR Guardians	Owner AND Guardians	Owner AND Guardians	Owner AND Guardians
1	1	1	1	2	2	2
2	1	1	2	2	2	2
3	1	2	2	3	3	3
4	1	2	3	3	3	4
5	1	3	3	4	4	4

Table 1: Number of signatures required to perform operations. Depending the type of the operation, signatures can be required from guardians only or by a combination of guardians and/or owner.

2 Implementation

2.1 Smart Contracts architecture

Our architecture is made up of multiple contracts. A first group of contracts form the infrastructure required to deploy or update user wallets as well as to expose on-chain information needed to secure the wallets. These infrastructure contracts are meant to be deployed only once:

- **Multisig Wallet:** Custom-made multi-signatures wallet which is the owner of most of other infrastructure contracts. All calls on those contracts will therefore need to be approved by multiple persons.

[1] https://eips.ethereum.org/EIPS/eip-1077

- **Wallet Factory:** Wallet factory contract used to create proxy wallets using CREATE2 and assign them to users.
- **ENS Manager:** The ENS Manager is responsible for registering ENS subdomains (e.g. mike.argent.xyz) and assigning them to wallets.
- **ENS Resolver:** The ENS Resolver keeps links between ENS subdomains and wallet addresses and allows to resolve them in both directions.
- **Module Registry:** The Module Registry maintains a list of the registered *Module* contracts that can be used with user wallets. It also maintains a list of registered *Upgrader* contracts that a user can use to migrate the module(s) used with their wallet (see Section 2.2).
- **Token Registry:** On-chain registry of ERC20 tokens that can be safely traded.
- **Dapp Registry:** Registries of dapps authorised by Argent. The *DappRegistry* currently supports 2 registries.

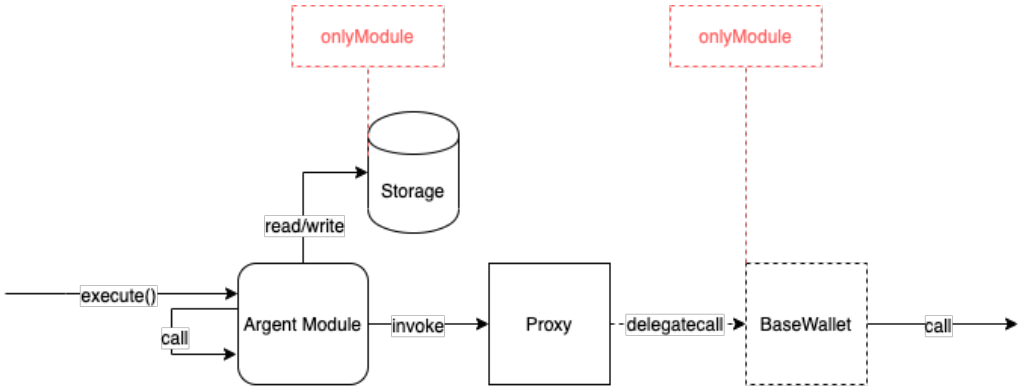

Figure 1: Call flow in Argent

A second group of contracts implements the functionalities of the wallet:

- **Proxy Wallet:** Lightweight proxy contract that delegates all calls to a Base Wallet library-like contract. There is one proxy deployed per wallet. Note that the rationale for using the Proxy-Implementation

design pattern[2] is *not* to enable wallet upgradability (we use upgradable modules/features for that) but simply to reduce the deployment cost of each new wallet.

- **Base Wallet:** The Base Wallet is a simple library-like contract implementing basic wallet functionalities used by proxy wallets (via delegatecalls), that are not expected to ever change. These functionalities include changing the owner of the wallet, (de)authorizating modules and performing (value-carrying) internal transactions to third-party contracts.

- **Modules:** The different functionalities of the wallet are encapsulated in *Modules*. Each Module contract is used by all wallets to handle a specific set of operations (e.g. adding and revoking guardians, doing multicalls, etc). Registered Modules can be added and removed by the wallet owner to update the functionalities of its wallet but the wallet must always have at least 1 module. Modules are grouped in bundles and each bundle is given a unique version number. A wallet is typically upgraded by its owner to use a certain bundle of modules (i.e. version number).

- **Storages:** Some Modules store part of their states in a dedicated storage contract (see Section 2.4).

2.2 Upgradability

Argent maintains an evolving set of registered *Module* contracts. A subset of these *Module* contracts are *SimpleUpgrader* modules that define a migration path from a particular set of old modules to a new set of registered modules, i.e. it contains the list of modules to disable and the list of modules to enable to go from the old set to the new set. A user can perform an upgrade of their modules using one of the registered *SimpleUpgrader* contracts.

2.3 Modules

All the functionalities of the Argent wallet are currently implemented in a single module. To structure the code, the *ArgentModule* inherits from several abstract modules that each implements a coherent subset of the logic.

[2]introduced by Nick Johnson in https://gist.github.com/Arachnid/4ca9da48d51e23e5cfe0f0e14dd6318f

2.3.1 RelayerManager

The RelayerManager module is the entry point for meta-transactions. A relayer calls the *execute()* method of the *RelayerManager*, which validates the signatures provided, calls the required target method to run the business logic and optionally refunds the relayer's gas cost.

2.3.2 SecurityManager

The SecurityManager module contains all the logic related to the security of the wallet; namely adding and removing guardians, locking and unlocking the wallet, recovering and transfering ownership.

2.3.3 TransactionManager

The TransactionManager module contains all the logic to execute multi-calls, including adding or removing a trusted contact, starting or closing a session, and executing multi-calls with the correct authorisation.

2.3.4 BaseModule

The BaseModule is a base contract containing logic common to all modules.

2.4 Storage

To enable cost-efficient upgrades, the part of the wallet state that must be permanent across upgrades is stored on dedicated storage contracts.

The current version of the wallet uses two storage contracts:

- **Transfer Storage:** Used to store the trusted contacts of a wallet.
- **Guardian Storage:** Used to store the list of guardians of a wallet.

2.5 Meta-Transactions

Meta-Transactions are implemented in the abstract *RelayerManager* from which the *ArgentModule* inherit. It implements a permissionless method *execute()* that is meant to be called by a relayer account. The relayer must pass to the *execute()* function an intention and the signature(s) of this intention by the originator(s) of that intention. As described in Section 1.12,

this pattern allows ether-less accounts to perform operations on the wallet without the need to directly call the corresponding feature methods to do so.

2.6 Dapp Registry

The DappRegistry maintains a list of registries, where each registry is a mapping between a set of authorised addresses and *Filter* contracts that implement the conditions that must be satisfied to send value and/or data to these authorised addresses. In theory up to 256 registries can be defined on the DappRegistry but the current version only defines two: registry Id[0] containing all the dapps that are natively integrated in the Argent client application and registry Id[1] containing a list of curated dapps that can be accessed through WalletConnect.

Each registry has an owner who is the only account that can request the addition or update of an entry in the registry. Once the addition/update has been requested a timelock period of 7 days must pass before the operation can be completed. Anybody can trigger the confirmation transaction provided that the timelock period has expired. Removing an entry from a registry must be triggered by the owner of the registry and is immediate. The Argent multisig is the owner of registry Id[0] and Id[1].

In addition, the owner of registry Id[0] has a special role as it is the only account authorised to create a new registry or to change the timelock period of the *DappRegistry* contract. Changing the timelock is also a 2 step process where the confirmation transaction can only be executed after the old timelock period as expired. We note that it is not possible to remove an existing registry because that would allow the owner of registry Id[0] to replace registries that have already been enabled by users with a new (potentially maliciously populated) registry.

2.7 Filters

Filters are smart-contracts that expose a single method that validates:

- that a contract call from a wallet to an authorised address is considered secure
- that a contract call from a wallet to an ERC20/ERC721/ERC1155 token where the authorised address is the spender is considered secure

- that an ETH transfer from the wallet to the authorised address is considered secure

We note that *secure* is to be understood in the context of an Argent wallet whose assets need to be protected even if the owner key is compromised.

Some example of filters added to registry Id[0]:

2.7.1 CompoundCTokenFilter

The CompoundCTokenFilter applies to all cTokens but there must be one filter per cToken as defined by its underlying. It only accepts contract calls to a cToken to *mint*, *redeem*, *redeemUnderlying*, *borrow*, and *repayBorrow*. All other methods are blocked. The CompoundCTokenFilter blocks all contract calls to ERC20/ERC721/ERC1155 tokens where the spender is the cToken to prevent e.g. transfering DAI to the cEther contract. The only exception is an ERC20 approve on the underlying of the cToken.

2.7.2 UniswapV2UniZapFilter

The UniswapV2UniZapFilter lets a wallet add/remove liquidity to Uniswap V2 pools via the UniZap contract deployed at 0xbCc492DF37bD4ec651D46d72230E340c9ec1950c. It will only allow adding/removing liquidity to pools that have been flagged as tradable in the *TokenRegistry* based on their underlying tokens and liquidity. It also checks that the beneficiary of the resulting tokens is the calling wallet. The UniswapV2UniZapFilter lets a wallet approve the UniZap contract as the spender of ERC20 tokens to enable adding liquidity from a token or removing liquidity by burning LP tokens. All other standard ERC20/ERC721/ERC1155 methods are blocked.

2.7.3 LidoFilter

The LidoFilter lets a wallet swap ETH for stETH by allowing calls to the *submit* method of the Lido contract. All other methods calls are disabled.

2.7.4 AaveV2Filter

The AaveV2Filter lets a wallet add/remove liquidity to the V2 version of the Aave protocol provided that the beneficiary of the resulting tokens is the calling wallet.