OpenZeppelin

Smart account design

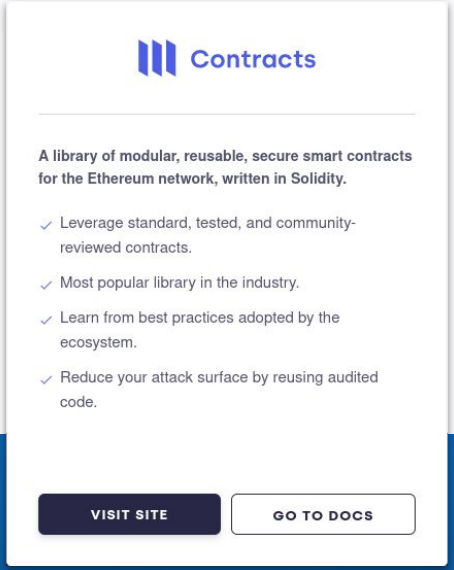
The different path toward modularity

Hadrien Croubois – Account Abstraction Community Hub – November 2025

Contracts

@openzeppelin/contracts@5.5.0

@openzeppelin/contracts-upgradeable@5.5.0



How is contract modularity achieved ?

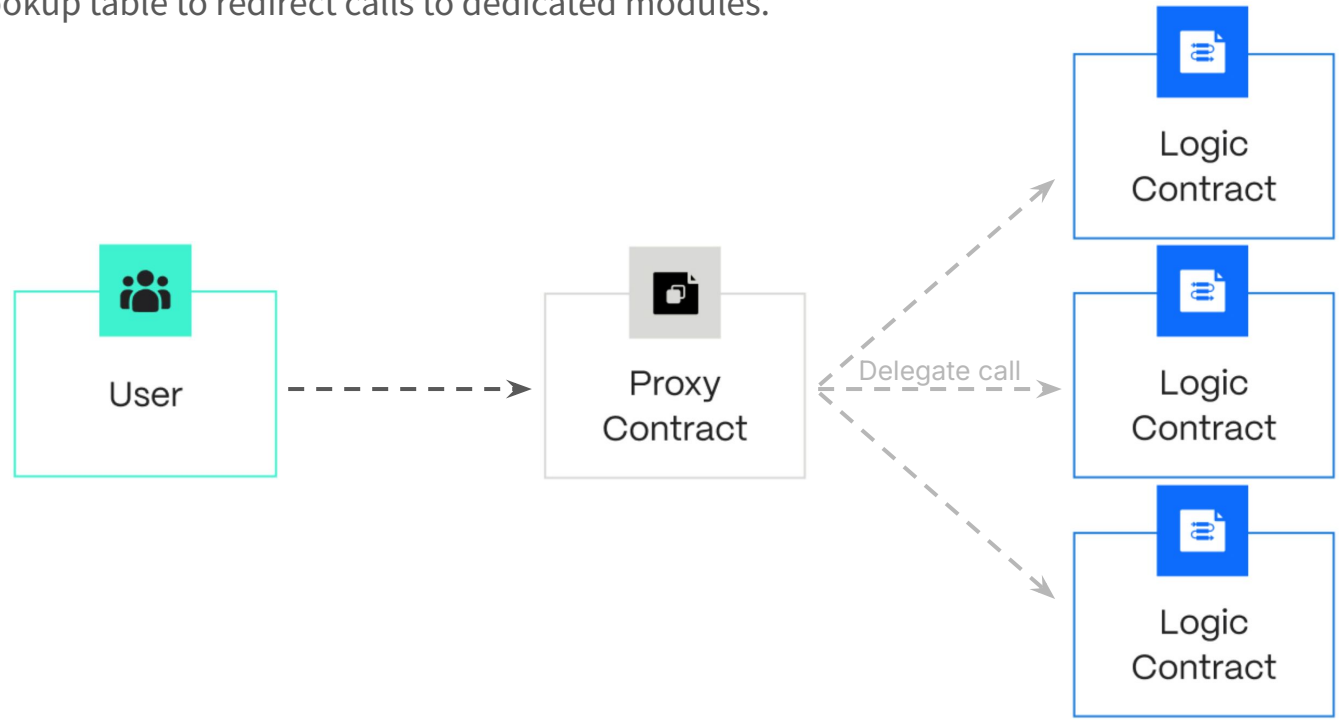
- **Compile time modularity:**
  Combining “modules” using inheritance. Get a monolithic artefact.

How is contract modularity achieved ?

- **Simple upgradeability:**
  Give the ability to “jump” between versions of a monolithic artefacts.

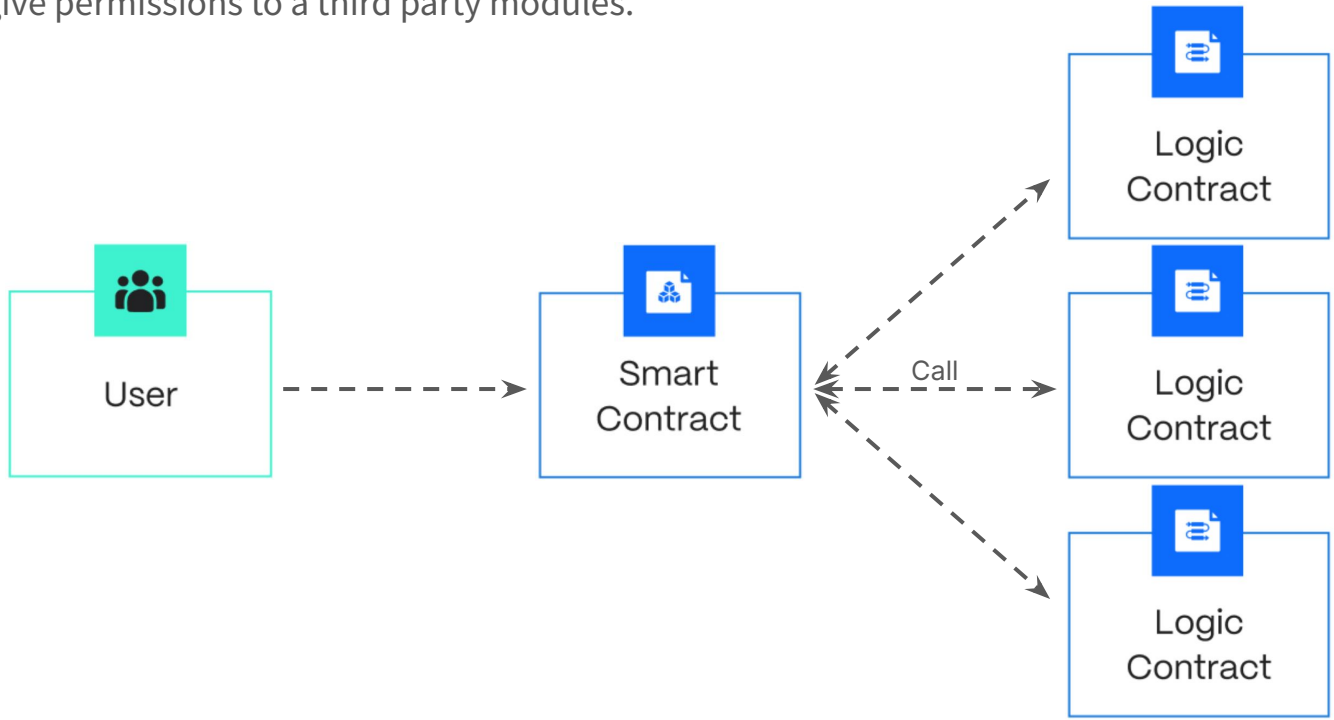
How is contract modularity achieved ?

- **“Diamond” upgradeability:**
  Use a lookup table to redirect calls to dedicated modules.



How is contract modularity achieved ?

- **“Runtime” modularity:**
  Call or give permissions to a third party modules.



How can we make smart accounts modular ?

- ~~**Compile time modularity**~~**:**
  Not enough by itself, we want accounts to be “living”.

- ~~**Simple upgradeability**~~**:**
  Would possibly work, but risks are high, and it would be a nightmare for EIP-7702 whitelists.

- ~~**“Diamond” upgradeability**~~**:**
  That works for some big DeFi contracts, but it feels very dangerous and brittle for accounts.

- **“Runtime” modularity:**
  That is what we are left with, and it’s honestly not a bad option!

Current account situation

- **Wallets “hide” EIP-7702 away for user protection**
  Prevents malicious apps from asking users to delegate to a malicious implementation

- **Wallets expose high-level feature**
  EIP-5792 (wallet_sendCalls), EIP-7715 (wallet_grantPermissions)

- **Each wallet has its own implementation**
  No code reuse between wallet & mostly incompatible interfaces

- **Hardware wallets that want to implement whitelists need to whitelist many instances**
  Additional source of trust, that is not easily maintainable.

- **Users that put the same signer/private key in multiple wallets may constantly re-delegate**

Why the current situation is problematic

- **Metamask user sets up social recovery**

  Using ERC-7715, the user requests a permission for a “social recovery” contract to access the wallet funds in some specific circumstances. This permission is not used immediately. It has an expiry date. Users shares it with someone else that is authorized to perform social recovery. User feels protected !

- **User tries a “band new” wallet with special features**

  Gaming, DeFi, social media, … there are many reason for a user to try a shiny new wallet that it trust (and that we assume is safe). They leverage EIP-7702 to provide the great UX. They use their own implementation. When the user shares its PK/signer, the wallets re-delegate. User may see a warning and discard them.

- **Something happens, social recovery of the user funds is needed**

  Unfortunately, the redelegation means the ERC-7715 permission can no longer be used. It is valid, but the implementation that the EOA delegates to does not give execution permission to Metamask’s ERC-7710 delegation manager.

How we could fix that

- **Use a standard implementation**

  Once the EOA has performed the EIP-7702 delegation, any wallet should be able to use it and not have to re-delegate

- **Make the implementation “runtime” modular**

  All wallets should be able to add modules to extend the user experience. Modules could implement gaming session keys, social recovery, improved AMM integration …

- **Make the modularity standard**

  All wallets should be able to perform introspection on the wallet, see all modules installed, and remove them selectively upon users requesting it.

Properties I think we should have

- ERC-4337 support
- Support for third party execution by “executors modules”
- Support for signing logic beside the EOA’s ECDSA signature
- Some redirection mechanism for the fallback (onERCxxxReceived)

ERC-7579

ERC20 | ERC721 | ERC1155 | Stablecoin* | Real-World Asset* | **Account** | Governor | Custom

Open in Remix | Download

SETTINGS

Name

UniversalModularAccount

FEATURES

- [x] Signature Validation
  - [x] Account Bound
- [x] ERC721 Holder
- [x] ERC1155 Holder
- [x] Batched Execution
- [x] Modules
  - [ ] Hooked

[x] SIGNER

- ( ) ECDSA
- (•) EOA Delegation
- ( ) Multisig
- ( ) Multisig Weighted
- ( ) P256
- ( ) RSA
- ( ) WebAuthn

[ ] UPGRADEABILITY

EOAs can upgrade by redelegating to a new account

- ( ) Transparent
- ( ) UUPS

INFO

Security Contact

security@example.com

License

MIT

```
// SPDX-License-Identifier: MIT
// Compatible with OpenZeppelin Contracts ^5.5.0
pragma solidity ^0.8.27;

import {AbstractSigner} from "@openzeppelin/contracts/utils/cryptography/signers/AbstractSigner.sol";
import {Account} from "@openzeppelin/contracts/account/Account.sol";
import {AccountERC7579} from "@openzeppelin/contracts/account/extensions/draft-AccountERC7579.sol";
import {EIP712} from "@openzeppelin/contracts/utils/cryptography/EIP712.sol";
import {ERC1155Holder} from "@openzeppelin/contracts/token/ERC1155/utils/ERC1155Holder.sol";
import {ERC721Holder} from "@openzeppelin/contracts/token/ERC721/utils/ERC721Holder.sol";
import {ERC7739} from "@openzeppelin/contracts/utils/cryptography/signers/draft-ERC7739.sol";
import {PackedUserOperation} from "@openzeppelin/contracts/interfaces/draft-IERC4337.sol";
import {SignerEIP7702} from "@openzeppelin/contracts/utils/cryptography/signers/SignerEIP7702.sol";

contract UniversalModularAccount is Account, EIP712, ERC7739, AccountERC7579, SignerEIP7702, ERC721Holder, ERC1155Holder
    constructor() EIP712("UniversalModularAccount", "1") {}

    function isValidSignature(bytes32 hash, bytes calldata signature)
        public
        view
        override(AccountERC7579, ERC7739)
        returns (bytes4)
    {
        // ERC-7739 can return the ERC-1271 magic value, 0xffffffff (invalid) or 0x77390001 (detection).
        // If the returned value is 0xffffffff, fallback to ERC-7579 validation.
        bytes4 erc7739magic = ERC7739.isValidSignature(hash, signature);
        return erc7739magic == bytes4(0xffffffff) ? AccountERC7579.isValidSignature(hash, signature) : erc7739magic;
    }

    // The following functions are overrides required by Solidity.

    function _validateUserOp(PackedUserOperation calldata userOp, bytes32 userOpHash, bytes calldata signature)
        internal
        override(Account, AccountERC7579)
        returns (uint256)
    {
        return super._validateUserOp(userOp, userOpHash, signature);
    }

    // IMPORTANT: Make sure SignerEIP7702 is more derived than AccountERC7579
    // in the inheritance chain (i.e. contract ... is AccountERC7579, ..., SignerEIP7702)
    // to ensure the correct order of function resolution.
    // AccountERC7579 returns false for _rawSignatureValidation
    function _rawSignatureValidation(bytes32 hash, bytes calldata signature)
        internal
        view
        override(SignerEIP7702, AbstractSigner, AccountERC7579)
        returns (bool)
    {
        return super._rawSignatureValidation(hash, signature);
    }
}
```

AI Assistant

ERC20 | ERC721 | ERC1155 | Stablecoin* | Real-World Asset* | **Account** | Governor | Custom

Open in Remix | Download

SETTINGS

Name

UniversalModularAccount

FEATURES

- [x] Signature Validation
  - [x] Account Bound
- [x] ERC721 Holder
- [x] ERC1155 Holder
- [x] Batched Execution
- [x] Modules
  - [ ] Hooked

[ ] SIGNER

- ECDSA
- EOA Delegation
- Multisig
- Multisig Weighted
- P256
- RSA
- WebAuthn

[x] UPGRADEABILITY

- Transparent
- (•) UUPS

INFO

Security Contact

security@example.com

License

MIT

```
// SPDX-License-Identifier: MIT
// Compatible with OpenZeppelin Contracts ^5.5.0
pragma solidity ^0.8.27;

import {Account} from "@openzeppelin/contracts/account/Account.sol";
import {AccountERC7579Upgradeable} from "@openzeppelin/contracts-upgradeable/account/extensions/draft-AccountERC7579Upgr
import {EIP712} from "@openzeppelin/contracts/utils/cryptography/EIP712.sol";
import {ERC1155Holder} from "@openzeppelin/contracts/token/ERC1155/utils/ERC1155Holder.sol";
import {ERC721Holder} from "@openzeppelin/contracts/token/ERC721/utils/ERC721Holder.sol";
import {ERC7739} from "@openzeppelin/contracts/utils/cryptography/signers/draft-ERC7739.sol";
import {Initializable} from "@openzeppelin/contracts/proxy/utils/Initializable.sol";
import {PackedUserOperation} from "@openzeppelin/contracts/interfaces/draft-IERC4337.sol";
import {UUPSUpgradeable} from "@openzeppelin/contracts/proxy/utils/UUPSUpgradeable.sol";

contract UniversalModularAccount is Initializable, Account, EIP712, ERC7739, AccountERC7579Upgradeable, ERC721Holder, ER
    /// @custom:oz-upgrades-unsafe-allow-reachable constructor
    constructor() EIP712("UniversalModularAccount", "1") {
        _disableInitializers();
    }

    function initialize(uint256 moduleTypeId, address module, bytes calldata initData)
        public
        initializer
    {
        __AccountERC7579_init();

        require(moduleTypeId == MODULE_TYPE_VALIDATOR || moduleTypeId == MODULE_TYPE_EXECUTOR);
        _installModule(moduleTypeId, module, initData);
    }

    function isValidSignature(bytes32 hash, bytes calldata signature)
        public
        view
        override(AccountERC7579Upgradeable, ERC7739)
        returns (bytes4)
    {
        // ERC-7739 can return the ERC-1271 magic value, 0xffffffff (invalid) or 0x77390001 (detection).
        // If the returned value is 0xffffffff, fallback to ERC-7579 validation.
        bytes4 erc7739magic = ERC7739.isValidSignature(hash, signature);
        return erc7739magic == bytes4(0xffffffff) ? AccountERC7579Upgradeable.isValidSignature(hash, signature) : erc773
    }

    function _authorizeUpgrade(address newImplementation)
        internal
        override
        onlyEntryPointOrSelf
    {}

    // The following functions are overrides required by Solidity.

    function _validateUserOp(PackedUserOperation calldata userOp, bytes32 userOpHash, bytes calldata signature)
        internal
        override(Account, AccountERC7579Upgradeable)
        returns (uint256)
    {
        return super._validateUserOp(userOp, userOpHash, signature);
```

AI Assistant

Upsides of this approach

- **Only one wallet implementation used by everyone**

  Less maintenance burden by the wallets (shared effort)

  Security audit/review focused on one piece of code instead of many (less expensive to individual wallets)

  Only one address to whitelist in hardware wallets (less maintenance effort and security risks)

- **Create an ecosystem of modules**

  Third parties, that are not wallet devs, could build application/feature specific modules. Wallet could decide to integrate some of them to provide interesting features.

- **Wallet ecosystem that provides properties aligned with the users interests**

  See Vitalik's talk at EthCC[8]

How much work is necessary to get there?

- **Metamask already uses an “account <> module” design**

  There is only one module (the EIP-7710 delegation manager) that cannot be removed. Other modules cannot be added. Changing the module management would not question EIP-7710 & ERC-7715 design.

- **What about other wallets?**

Immediate actionnables

- **Share support for this initiative**
- **Let’s start a working group with wallets, hardware signers and module developers**