

Hooks everywhere

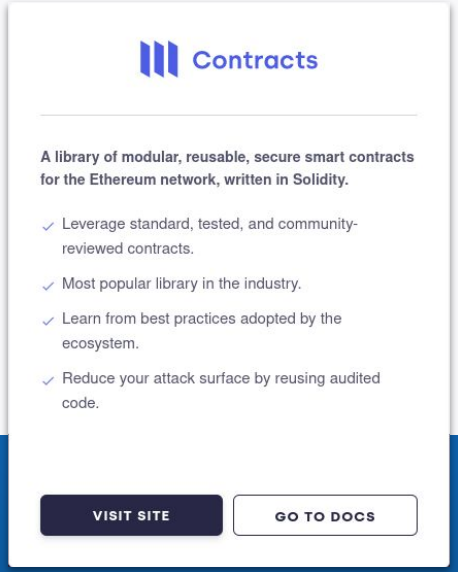
The age of dynamic contracts

Hadrien Croubois
OpenZeppelin

Contracts

@openzeppelin/contracts@5.7.0

@openzeppelin/contracts-upgradeable@5.7.0



Code onchain is mostly immutable

- Smart contracts code cannot be modified (including upgradeable proxy)
- Storage can be reconfigured: this is how upgradeability is achieved.
- SELFDESTRUCT was deprecated: once a contract has been deployed, it's here to stay
- EIP-7702 allows code changes, but only on clearly identifiable instances (EOA, 0xEF0100 prefix)
- EIP-8298 (SETCODEFROM) changes that. Currently CFI in Hegotá

How do we make smart contracts more dynamic ?

Notification hooks



Notification hooks

- Present in the official token tutorial since 2015~2016.
- Standardized in ERCs in 2017.
- Present in all major token standards besides (native) ERC-20.
- Great for composability.
- ERC-777 used ERC-1820 to notify third parties (on a per-account basis).
- Not suitable for adding compliance checks.
- Can lead to re-entrancy if not handled properly.

```
function transferFrom(address from, address to, uint256 tokenId) public virtual {
    require(to != address(0), ERC721InvalidReceiver(address(0)));
    // Setting an "auth" arguments enables the `_isAuthorized` check which verifies that the token exists
    // (from != 0). Therefore, it is not needed to verify that the return value is not 0 here.
    address previousOwner = _update(to, tokenId, _msgSender());
    require(previousOwner == from, ERC721IncorrectOwner(from, tokenId, previousOwner));
}
function safeTransferFrom(address from, address to, uint256 tokenId, bytes memory data) public virtual {
    transferFrom(from, to, tokenId);
    ERC721Utils.checkOnERC721Received(_msgSender(), from, to, tokenId, data);
}
function checkOnERC721Received(
    address operator,
    address from,
    address to,
    uint256 tokenId,
    bytes memory data
) internal {
    if (to.code.length > 0) {
        try IERC721Receiver(to).onERC721Received(operator, from, tokenId, data) returns (bytes4 retval) {
            if (retval != IERC721Receiver.onERC721Received.selector) {
                // Token rejected
                revert IERC721Errors.ERC721InvalidReceiver(to);
            }
        } catch (bytes memory reason) {
            if (reason.length == 0) {
                // non-IERC721Receiver implementer
                revert IERC721Errors.ERC721InvalidReceiver(to);
            } else {
                assembly ("memory-safe") {
                    revert(add(reason, 0x20), mload(reason))
                }
            }
        }
    }
}
```

Compile time hooks

~2020

_beforeXxx and _afterXxx

Compile time hooks

- Uses virtual functions to allow behavior customization.
- Dedicated “hook” functions are placed at location where extra logic may be injected.
- Resolved at compilation, produces static bytecode. Not modifiable after deployment.

```
function _transfer(
    address sender,
    address recipient,
    uint256 amount
) internal virtual {
    require(sender != address(0), "ERC20: transfer from the zero address");
    require(recipient != address(0), "ERC20: transfer to the zero address");

    _beforeTokenTransfer(sender, recipient, amount);

    uint256 senderBalance = _balances[sender];
    require(senderBalance >= amount, "ERC20: transfer amount exceeds balance");
    unchecked {
        _balances[sender] = senderBalance - amount;
    }
    _balances[recipient] += amount;

    emit Transfer(sender, recipient, amount);

    _afterTokenTransfer(sender, recipient, amount);
}

function _beforeTokenTransfer(
    address from,
    address to,
    uint256 amount
) internal virtual {}

function _afterTokenTransfer(
    address from,
    address to,
    uint256 amount
) internal virtual {}
```

Construction hooks

ERC20Pausable.sol

```
function _beforeTokenTransfer(
    address from,
    address to,
    uint256 amount
) internal virtual override {
    super._beforeTokenTransfer(from, to, amount);

    require(!paused(), "ERC20Pausable: token transfer while paused");
}
```

ERC20Snapshot.sol

```
function _beforeTokenTransfer(
    address from,
    address to,
    uint256 amount
) internal virtual override {
    super._beforeTokenTransfer(from, to, amount);

    if (from == address(0)) {
        // mint
        _updateAccountSnapshot(to);
        _updateTotalSupplySnapshot();
    } else if (to == address(0)) {
        // burn
        _updateAccountSnapshot(from);
        _updateTotalSupplySnapshot();
    } else {
        // transfer
        _updateAccountSnapshot(from);
        _updateAccountSnapshot(to);
    }
}
```

Solady's ERC20.sol

```
function transfer(address to, uint256 amount) public virtual returns (bool) {
    _beforeTokenTransfer(msg.sender, to, amount);
    /// @solidity memory-safe-assembly
    assembly {
        // Compute the balance slot and load its value.
        mstore(0x0c, _BALANCE_SLOT_SEED)
        mstore(0x00, caller())
        let fromBalanceSlot := keccak256(0x0c, 0x20)
        let fromBalance := sload(fromBalanceSlot)
        // Revert if insufficient balance.
        if gt(amount, fromBalance) {
            mstore(0x00, 0xf4d678b8) // `InsufficientBalance()`.
            revert(0x1c, 0x04)
        }
        // Subtract and store the updated balance.
        sstore(fromBalanceSlot, sub(fromBalance, amount))
        // Compute the balance slot of `to`.
        mstore(0x00, to)
        let toBalanceSlot := keccak256(0x0c, 0x20)
        // Add and store the updated balance of `to`.
        // Will not overflow because the sum of all user balances
        // cannot exceed the maximum uint256 value.
        sstore(toBalanceSlot, add(sload(toBalanceSlot), amount))
        // Emit the {Transfer} event.
        mstore(0x20, amount)
        log3(0x20, 0x20, _TRANSFER_EVENT_SIGNATURE, caller(), shr(96, mload(0x0c)))
    }
    _afterTokenTransfer(msg.sender, to, amount);
    return true;
}
```

Static configuration hooks

June 2023

Uniswap V4

Uniswap v4 hooks

- Each pool is identified by a key. The IHooks address is part of that key.
- If enabled, hook functions are called at key moments of the pool lifecycle.
- Allows pools with custom behavior to live in the same singleton (PoolManager).
- The binding is static (hook address and permissions are fixed per pool), but the hook itself may be upgradeable.

```
function donate(PoolKey memory key, uint256 amount0, uint256 amount1, bytes calldata hookData)
    external
    onlyWhenUnlocked
    noDelegateCall
    returns (BalanceDelta delta)
{
    PoolId poolId = key.toId();
    Pool.State storage pool = _getPool(poolId);
    pool.checkPoolInitialized();

    key.hooks.beforeDonate(key, amount0, amount1, hookData);

    delta = pool.donate(amount0, amount1);

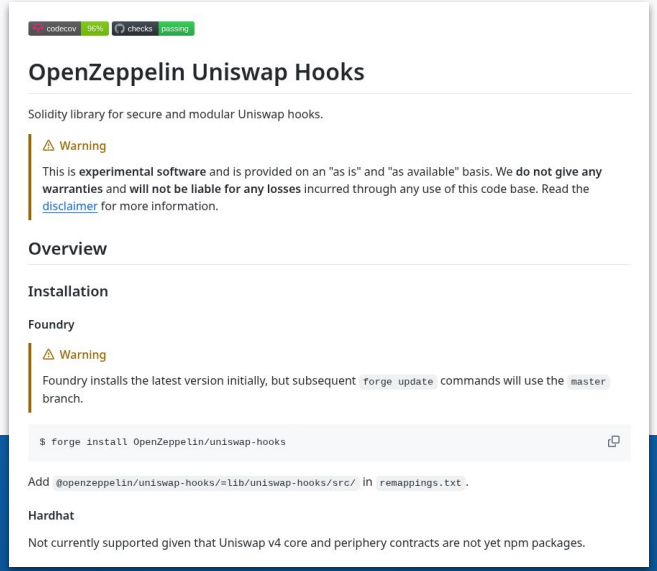
    _accountPoolBalanceDelta(key, delta, msg.sender);

    // event is emitted before the afterDonate call to ensure events are always emitted in order
    emit Donate(poolId, msg.sender, amount0, amount1);

    key.hooks.afterDonate(key, amount0, amount1, hookData);
}
function afterDonate(IHooks self, PoolKey memory key, uint256 amount0, uint256 amount1, bytes calldata hookData)
    internal
    noSelfCall(self)
{
    if (self.hasPermission(AFTER_DONATE_FLAG)) {
        self.callHook(abi.encodeCall(IHooks.afterDonate, (msg.sender, key, amount0, amount1, hookData)));
    }
}
function beforeDonate(IHooks self, PoolKey memory key, uint256 amount0, uint256 amount1, bytes calldata hookData)
    internal
    noSelfCall(self)
{
    if (self.hasPermission(BEFORE_DONATE_FLAG)) {
        self.callHook(abi.encodeCall(IHooks.beforeDonate, (msg.sender, key, amount0, amount1, hookData)));
    }
}
function hasPermission(IHooks self, uint160 flag) internal pure returns (bool) {
    return uint160(address(self)) & flag != 0;
}
```

OpenZeppelin Uniswap Hooks

https://github.com/OpenZeppelin/uniswap-hooks@1.3.0



codecov 96% checks passing

OpenZeppelin Uniswap Hooks

Solidity library for secure and modular Uniswap hooks.

> ⚠ Warning
>
> This is **experimental software** and is provided on an "as is" and "as available" basis. We **do not give any warranties** and **will not be liable for any losses** incurred through any use of this code base. Read the disclaimer for more information.

Overview

Installation

Foundry

> ⚠ Warning
>
> Foundry installs the latest version initially, but subsequent `forge update` commands will use the `master` branch.

```
$ forge install OpenZeppelin/uniswap-hooks
```

Add `@openzeppelin/uniswap-hooks/=lib/uniswap-hooks/src/` in `remappings.txt`.

Hardhat

Not currently supported given that Uniswap v4 core and periphery contracts are not yet npm packages.

Dynamic configuration hooks

May 2021

Safe 1.3.0

Safe’s guard

- A safe can set a guard. This guard is called before and after each transaction.
- A similar guard for module operations was introduced in 1.5.0.
- Guard can be updated at any point.
- Can implement compliance checks, timelock, …

```
function execTransaction(
    address to,
    uint256 value,
    bytes calldata data,
    Enum.Operation operation,
    uint256 safeTxGas,
    uint256 baseGas,
    uint256 gasPrice,
    address gasToken,
    address payable refundReceiver,
    bytes memory signatures
) external payable override returns (bool success) {
    onBeforeExecTransaction(to, value, data, operation, safeTxGas, baseGas, gasPrice, gasToken, refundReceiver, signatures);
    bytes32 txHash;
    // Use scope here to limit variable lifetime and prevent `stack too deep` errors
    {
        txHash = getTransactionHash(
            // Transaction info
            to,
            value,
            data,
            operation,
            safeTxGas,
            // Payment info
            baseGas,
            gasPrice,
            gasToken,
            refundReceiver,
            // Signature info
            // We use the post-increment here, so the current nonce value is used and incremented afterwards.
            nonce++
        );
        checkSignatures(msg.sender, txHash, signatures);
    }
    address guard = getGuard();
    {
        if (guard != address(0)) {
            ITransactionGuard(guard).checkTransaction(
                // Transaction info
                to,
                value,
                data,
                operation,
                safeTxGas,
                // Payment info
                baseGas,
                gasPrice,
                gasToken,
                refundReceiver,
                // Signature info
                signatures,
                msg.sender
            );
        }
    }

    // We require some gas to emit the events (at least 2500) after the execution and some to perform code until the execution (500)
    // We also include the 1/64 in the check that is not sent along with a call to counteract potential shortings because of EIP-150
```

Examples

Safe Lockdown's Lockdown.sol

```
function checkTransaction(
    address to,
    uint256 value,
    bytes calldata data,
    Enum.Operation operation,
    uint256 safeTxGas,
    uint256 baseGas,
    uint256 gasPrice,
    address gasToken,
    address payable refundReceiver,
    bytes calldata,
    address
) public virtual override {
    (State state,) = getState(msg.sender);
    if (state == State.Uninitialised) {
        // Fail-open for uninitialised configs (anti-brick on install order).
    } else if (state == State.Normal) {
        // Owner path: quorum may administer the singleton.
        _checkCall(msg.sender, to, value, data, operation, Privilege.Owners);
    } else if (state == State.LockedDown) {
        revert LockedDown();
    } else {
        // Recovering: the Recovery privilege routes every leaf through the
        // unconditional _isRecoveryCall gate (narrow whitelist), so the
        // Owners/Module admin branches are never consulted on this path.
        _checkCall(msg.sender, to, value, data, operation, Privilege.Recovery);
        require(
            safeTxGas == 0 && baseGas == 0 && gasPrice == 0 && gasToken == address(0)
                && refundReceiver == address(0),
            RecoveryRefundDenied()
        );
    }
}
```

Optimism's TimelockGuard.sol

```
function checkTransaction(
    address _to,
    uint256 _value,
    bytes memory _data,
    Enum.Operation _operation,
    uint256 _safeTxGas,
    uint256 _baseGas,
    uint256 _gasPrice,
    address _gasToken,
    address payable _refundReceiver,
    bytes memory, /* signatures */
    address _msgSender
)
    external
    override
{
    Safe callingSafe = Safe(payable(msg.sender));

    if (_currentSafeState(callingSafe).timelockDelay == 0) {
        // We return immediately. This is important in order to allow a Safe which has the
        // guard set, but not configured, to complete the setup process.

        // It is also just a reasonable thing to do, since an unconfigured Safe must have a
        // delay of zero.
        return;
    }

    // Limit execution of transactions to owners of the Safe only.
    // This ensures that an attacker cannot simply collect valid signatures, but must also
    // control a private key. It is accepted as a trade-off that paymasters, relayers or UX
    // wrappers cannot execute transactions with the TimelockGuard enabled.
    if (!callingSafe.isOwner(_msgSender)) {
        revert TimelockGuard_NotOwner();
    }

    // Get the nonce of the Safe for the transaction being executed,
    // since the Safe's nonce is incremented before the transaction is executed,
    // we must subtract 1.
    uint256 nonce = callingSafe.nonce() - 1;

    // Get the transaction hash from the Safe's getTransactionHash function
    bytes32 txHash = callingSafe.getTransactionHash(
        _to, _value, _data, _operation, _safeTxGas, _baseGas, _gasPrice, _gasToken, _refundReceiver, nonce
    );

    // Get the scheduled transaction
    ScheduledTransaction storage scheduledTx = _currentSafeState(callingSafe).scheduledTransactions[txHash];
```

More dynamic configuration hooks

December 2023

ERC-7579

June 2025

ERC-20 Hooked

June 2026

Ascend

ERC-7579 hook extension

- Generic modifier that can be added to any function.
- Hook contracts have a unique endpoint that receives the context (sender, value, calldata).
- Hook can revert to prevent an operation, keep track of activity, trigger additional operations, …
- Only a single hook contract. Using a hook multiplexer is an option.
- ERC-6900 allows multiple hooks, at the cost of a more complex, underspecified, logic.

```
modifier withHook() {
    address hook_ = hook();
    bytes memory hookData;

    // slither-disable-next-line reentrancy-no-eth
    if (hook_ != address(0)) hookData = IERC7579Hook(hook_).preCheck(msg.sender, msg.value, msg.data);
    _;
    if (hook_ != address(0)) IERC7579Hook(hook_).postCheck(hookData);
}
function _installModule(
    uint256 moduleTypeId,
    address module,
    bytes memory initData
) internal virtual override withHook {
    if (moduleTypeId == MODULE_TYPE_HOOK) {
        require(_hook == address(0), ERC7579HookModuleAlreadyPresent(_hook));
        _hook = module;
    }
    super._installModule(moduleTypeId, module, initData);
}
function _execute(
    Mode mode,
    bytes calldata executionCalldata
) internal virtual override withHook returns (bytes[] memory) {
    return super._execute(mode, executionCalldata);
}
function _fallback() internal virtual override withHook returns (bytes memory) {
    return super._fallback();
}
```

1inch's ERC20Hooked

- Accounts register their own hooks.
- Token movements trigger all hooks from both the sender and receiver.
- If hooks are shared, only one call.
- Best effort: gas limited, reverts are ignored.

```
mapping(address => AddressSet.Data) private _hooks;

function _addHook(address account, address hook) internal virtual {
    if (hook == address(0)) revert InvalidHookAddress();
    if (IHook(hook).TOKEN() != IERC20Hooks(address(this))) revert InvalidTokenInHook();
    if (!_hooks[account].add(hook)) revert HookAlreadyAdded();
    if (_hooks[account].length() > MAX_HOOKS_PER_ACCOUNT) revert HooksLimitReachedForAccount();

    emit HookAdded(account, hook);
    uint256 balance = balanceOf(account);
    if (balance > 0) {
        _updateBalances(hook, address(0), account, balance);
    }
}

function addHook(address hook) public virtual {
    _addHook(msg.sender, hook);
}
```

```
function _update(address from, address to, uint256 amount) internal nonReentrant(_guard) override virtual {
    super._update(from, to, amount);

    unchecked {
        if (amount > 0 && from != to) {
            address[] memory hooksFrom = _hooks[from].items.get();
            address[] memory hooksTo = _hooks[to].items.get();
            uint256 hooksFromLength = hooksFrom.length;
            uint256 hooksToLength = hooksTo.length;

            for (uint256 i = 0; i < hooksFromLength; i++) {
                address hook = hooksFrom[i];

                uint256 j;
                for (j = 0; j < hooksToLength; j++) {
                    if (hook == hooksTo[j]) {
                        // Both parties are participating in the same hook
                        _updateBalances(hook, from, to, amount);
                        hooksTo[j] = address(0);
                        break;
                    }
                }

                if (j == hooksToLength) {
                    // Sender is participating in a hook, but receiver is not
                    _updateBalances(hook, from, address(0), amount);
                }
            }

            for (uint256 j = 0; j < hooksToLength; j++) {
                address hook = hooksTo[j];
                if (hook != address(0)) {
                    // Receiver is participating in a hook, but sender is not
                    _updateBalances(hook, address(0), to, amount);
                }
            }
        }
    }
}
```

Ascend

Ascend is a DeFi protocol that implements compliant finance. Compliance is implemented using hooks and modules.

- Separate set of hooks for each selector.
- Hook contracts have a unique endpoint that receives the context (sender and calldata).
- Hooks implement compliance checks that can be added, removed or updated at any point.

```
mapping(bytes4 selector => EnumerableSet.AddressSet) private _hooks;

modifier withHook() {
    (address[] memory hooks, bytes[] memory hookDatas) = _preHook(msg.sig);
    _;
    _postHook(hooks, hookDatas);
}
function _preHook(bytes4 selector) internal virtual returns (address[] memory hooks, bytes[] memory hookDatas) {
    hooks = _hooks[selector].values();
    hookDatas = new bytes[](hooks.length);
    for (uint256 i = 0; i < hooks.length; ++i) {
        // hook can reenter but is considered trusted since it is installed by the manager of the facility
        // slither-disable-next-line reentrancy-no-eth
        hookDatas[i] = IHook(hooks[i]).preCheck(msg.sender, msg.data);
    }
}

function _postHook(address[] memory hooks, bytes[] memory hookDatas) internal virtual {
    for (uint256 i = 0; i < hooks.length; ++i) {
        // hook can reenter but is considered trusted since it is installed by the manager of the facility
        // slither-disable-next-line reentrancy-no-eth
        IHook(hooks[i]).postCheck(hookDatas[i]);
    }
}
```

Limits of configuration hooks

April 2026

ERC-7984 Hooked

ERC-7984 Hooked

- Impossible to branch based on confidential values.
- Hooks cannot revert to implement compliance checks
- Instead, hooks return an encrypted boolean that nullifies the amount transferred.
- Unlike ERC-7579 or Ascend, no data is passed from pre-check to post-check.

```
EnumerableSet.AddressSet private _modules;

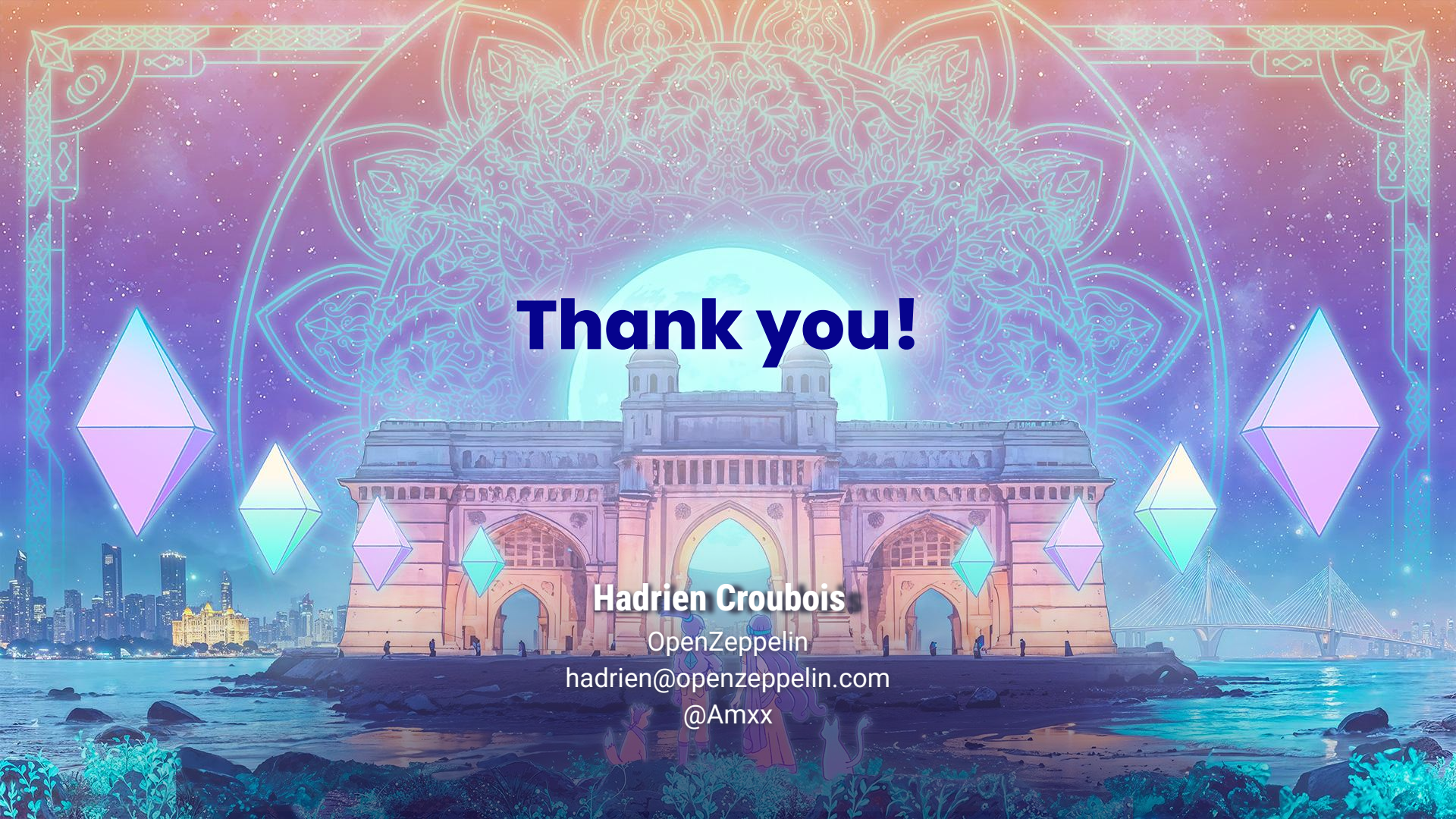
function _update(
    address from,
    address to,
    euint64 encryptedAmount
) internal virtual override returns (euint64 transferred) {
    euint64 amountToTransfer = FHE.select(
        _runPreTransferHooks(from, to, encryptedAmount),
        encryptedAmount,
        FHE.asEuint64(0)
    );
    transferred = super._update(from, to, amountToTransfer);
    _runPostTransferHooks(from, to, transferred);
}

function _runPreTransferHooks(
    address from,
    address to,
    euint64 encryptedAmount
) internal virtual returns (ebool compliant) {
    address[] memory modules_ = modules(0, type(uint256).max);
    uint256 modulesLength = modules_.length;
    compliant = FHE.asEbool(true);
    for (uint256 i = 0; i < modulesLength; ++i) {
        if (FHE.isInitialized(encryptedAmount)) FHE.allowTransient(encryptedAmount, modules_[i]);
        compliant = FHE.and(compliant, IERC7984HookModule(modules_[i]).preTransfer(from, to, encryptedAmount));
    }
}

function _runPostTransferHooks(address from, address to, euint64 encryptedAmount) internal virtual {
    address[] memory modules_ = modules(0, type(uint256).max);
    uint256 modulesLength = modules_.length;
    for (uint256 i = 0; i < modulesLength; i++) {
        if (FHE.isInitialized(encryptedAmount)) FHE.allowTransient(encryptedAmount, modules_[i]);
        IERC7984HookModule(modules_[i]).postTransfer(from, to, encryptedAmount);
    }
}
```

Conclusions

- The options form a spectrum from static (compile time) to dynamic (reconfigurable at any time).
- Each point on the spectrum serves a purpose. Static isn’t bad, and dynamic isn’t good.
- Every hook is an external call. It can re-enter, revert, or burn gas. Either place it before checks, after effects or trust it.
- Dynamic hooks turn code risk into governance risk: whoever can swap the hook controls the behavior.
- With dynamic hooks, the code you audit is not necessarily the code that runs.
- A single hook ERC is unlikely: binding time, call context, and failure semantics differ too much between use cases.


Thank you!
Hadrien Croubois
OpenZeppelin
hadrien@openzeppelin.com
@Amxx