Kakarot governance

TL;DR - Recommendation

We recommend creating a single smart contract, `ProtocolHandler`, governed by KKRT Labs (called `Operator` henceforth), the Kakarot Security Council (referred to as `Security Council`) and a list of guardians (referred as `Guardians`) in order to manage the Smart Contracts Ops & on-chain security needs of `Kakarot Core` contracts on Starknet.

That `ProtocolHandler` contract will have simple methods with access controls to call the Kakarot core contract and will be controlled by the Operator through a Timelock from OZ. This approach is the most lean (least code and complexity), most auditable (reusing existing bricks with separation of concerns) as well as most pragmatic (KKRT Labs starts with having a lot of agency). This recommendation is detailed in the first proposal.

Context

The `Kakarot Core` contract is critical and needs to be handled carefully. The owner of the contract has privileged permissions like upgrading or pausing the protocol.

At this stage of Kakarot, decentralized governance is too early and not relevant. There will be no DAO for now and KKRT labs will be the entity that will submit proposals and execute them through a multisig. Decentralization of the governance should done in at a second time. This document propose solutions in the meantime to safely handle the governance of the Kakarot contract.

See also
⚔️ Security Council multisig and Guardians (assumes first proposal accepted)

Vision

A security council will be here to ensure the security of Kakarot contract and any proposal submitted. It will also have the power to pause / unpause or push an emergency upgrade. It will be a multisig and it will be comprised of different actors (each of them should be a multisig):

- Starkware
- Starknet Fondation
- KKRT labs
- Zellic
- L2BEAT
- Others? (Security Council target size ~=12)

Guardians are selected entities that can initiate a soft pause in case of suspicious activity: each of them SHOULD be a multisig contract.

- Starkware
- Starknet Fondation
- KKRT labs

Current Situation

Kakarot Core only owner functions (currently deployed)

ADMIN

- `upgrade`
- `transfer_ownership`
- `pause`
- `unpause`

STORAGE

- `set_base_fee`
- `set_coinbase`
- `set_prev_randao`
- `set_block_gas_limit`
- `set_account_contract_class_hash`
- `set_uninitialized_account_class_hash`
- `set_authorized_cairo_precompile_caller`
- `set_cairo1_helpers_class_hash`
- `upgrade_account`
- `set_authorized_pre_eip155_tx`
- `set_l1_messaging_contract_address`

First proposal: Separate Timelock and protocol logic

Governance Management Strategy

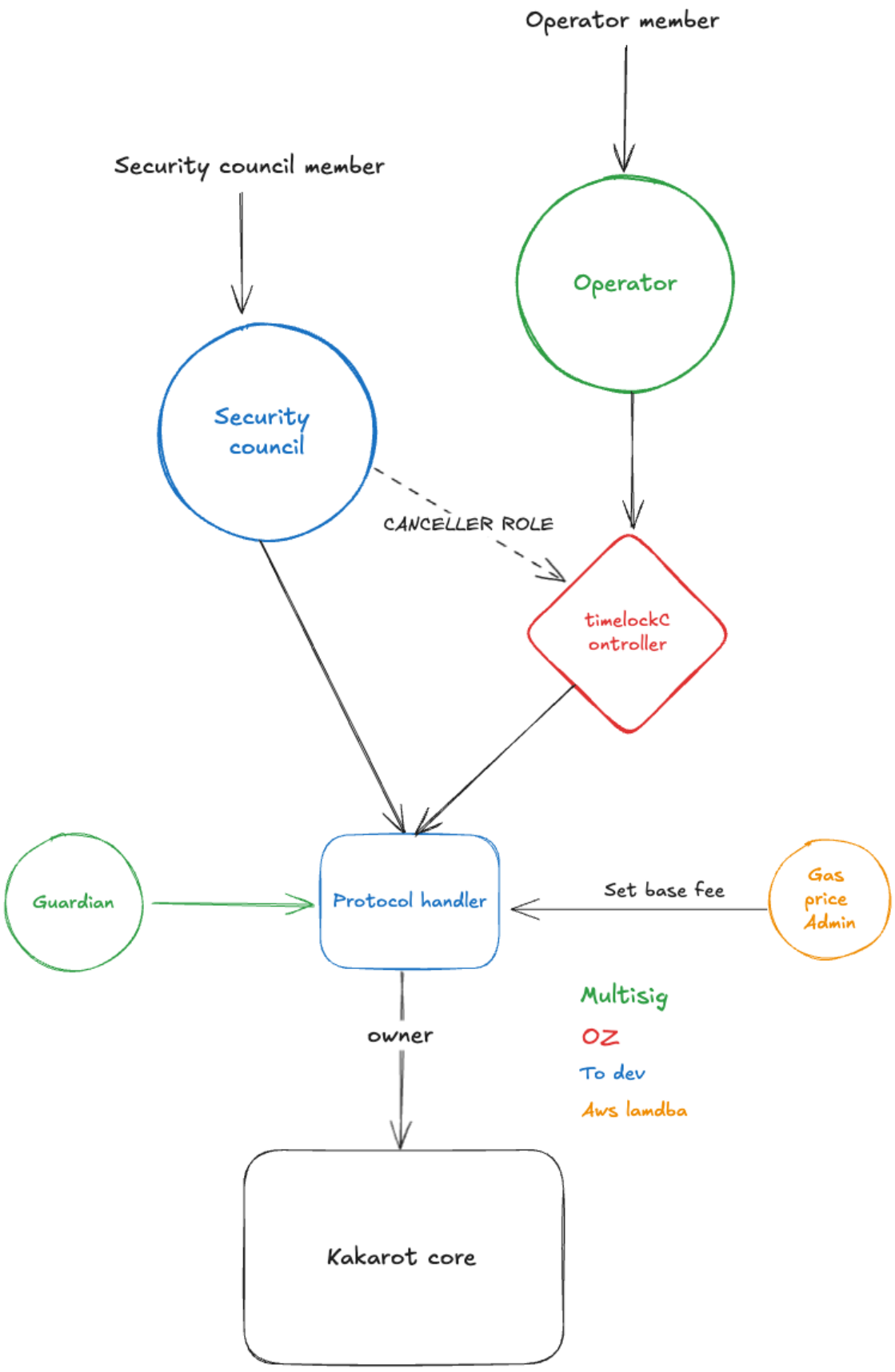
The owner of the `Kakarot Core` contract will be an intermediary contract: the `ProtocolHandler`.
Proposals will be scheduled and executed by KKRT Labs through a

`TimelockController` from Open Zeppelin (OZ) targeting the `ProtocolHandler`. The security council will have the canceller role in the `TimelockController` of Kakarot in order to veto any proposal if not deemed secure. Guardians can initiates a soft pause in the `ProtocolHandler` in case of suspicious activity.

The `TimelockController` will have the following roles

- `PROPOSER`: Operator
- `EXECUTOR`: Operator
- `CANCELLER`: SecurityCouncil
- NO ADMIN / NO OPEN ROLE

The minimum delay for proposal to be executed will be 7 days


Operator member
Security council member
Operator
Security council
CANCELLER ROLE
timelockC ontroller
Guardian
Protocol handler
Set base fee
Gas price Admin
Multisig
OZ
To dev
Aws lamdba
owner
Kakarot core

Spec

`ProtocolHandler` (owner of Kakarot):

```
// Contants
soft_pause_expiration = 12 hours
hard_pause_expiration = 7 days

// Storage
struct Storage {
    kakarot_core: ContractAddress
    security_council: ContractAddress
    operator: ContractAddress
    guardians Vec<ContractAddress>
    gas_price_admin: ContractAddress
    protocol_frozen_until: felt252
    authorized_operator_selector: Map<felt252, bool>
}

// Authorized operator selectors are set in construcor

// Admin
* fn emergency_execution(call: Call) SECURITY_COUNCIL
* fn upgrade(new_class_hash: felt252) OPERATOR
* fn transfer_ownership(new_owner: ContractAddress) SECURITY_COU
* fn soft_pause() GUARDIAN
* fn hard_pause() SECURITY_COUNCIL
* fn unpause() SECURITY_COUNCIL or delay passed

// Storage modification
// check if the selector is authorized
* fn execute_call(ref self: ContractState, call: Call) OPERATOR

// Self management
* fn change_operator(new_address_operator: ContractAddress) SECU
* fn change_security_council(new_security_council_address: Contr
```

```
 * fn add_guardian(new_guardians_address: ContractAddress) SECURI
 * fn remove_guardian(guardian_to_remove_address: ContractAddress
 * fn change_gas_price_admin(new_gas_price_admin: ContractAddres
```

Second proposal: merge Timelock and protocol logic

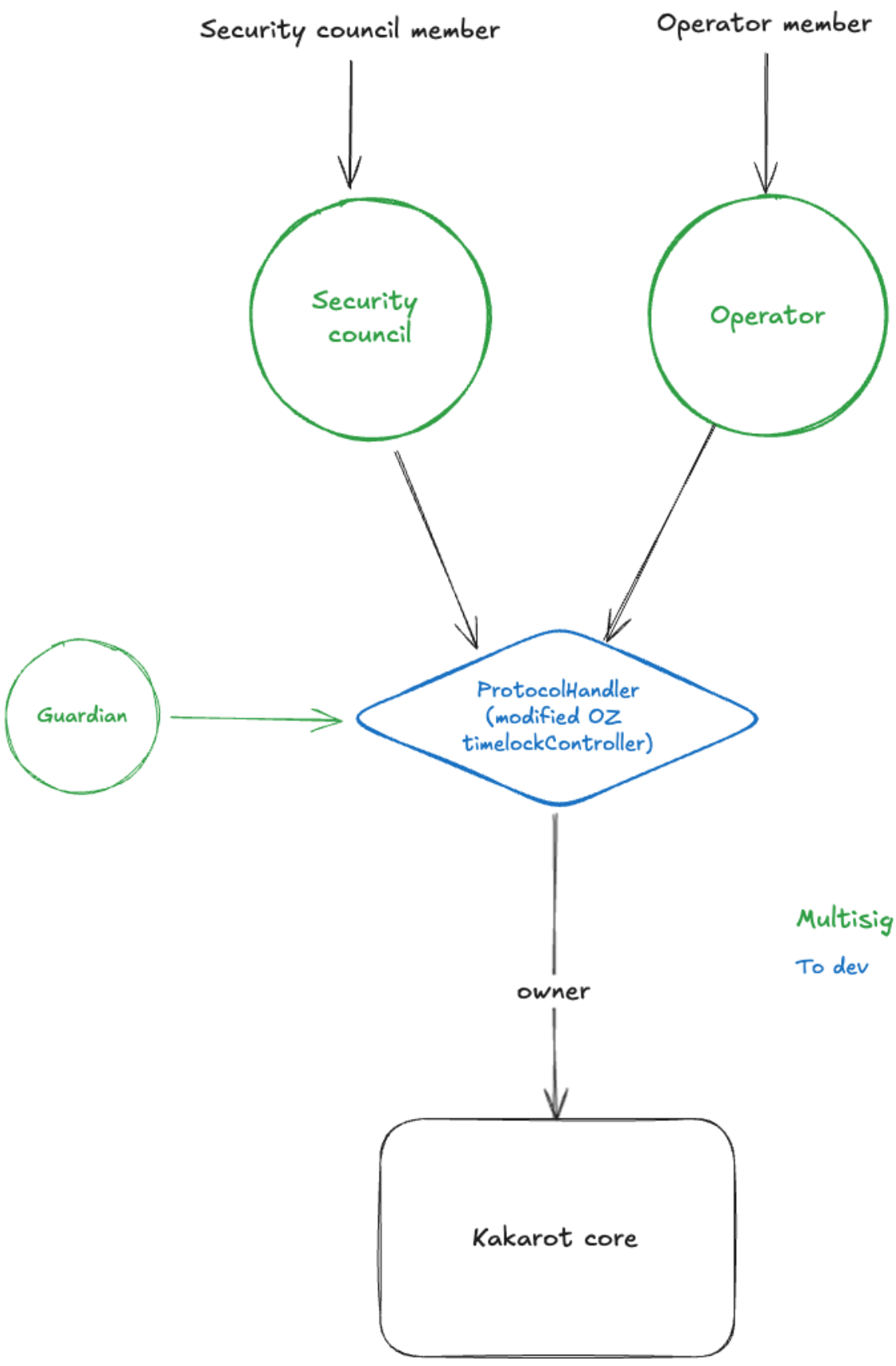
How the governance of kakarot is going to be managed ?

The owner of the `Kakarot Core` will be a modified `TimelockController` from OZ. The latter will be the owner of the `Kakarot Core` contract directly.
Proposals will be scheduled and executed through the modified `TimelockController` with a delay. The security council multisig will have access to special functions in the modified `TimelockController` to defend user interest like pausing or initiate an emergency upgrade. It will also have the `CANCELLER` role to have the possibility to veto any proposal.

The `TimelockController` will have the following roles

- `PROPOSER`: Operator
- `EXECUTOR`: Operator
- `CANCELLER`: SecurityCouncil
- `SECURITY_COUNCIL`: SecurityCouncil
- `GUARDIAN`: Guardians
- NO ADMIN / NO OPEN ROLE

The minimum delay for proposal to be executed will be 7 days


Security council member
Operator member
Security council
Operator
Guardian
ProtocolHandler (modified OZ timelockController)
Multisig
To dev
owner
Kakarot core

Spec

```
// Will inherit all functions from the timelock controller from
// It will be customized through implementation of the componen
// https://docs.openzeppelin.com/contracts-cairo/0.17.0/compone

// Contants
soft_pause_expiration = 12 hours
hard_pause_expiration = 7 days

// Storage
struct Storage {
    min_delay: felt252 = 7 days
    kakarot_core: ContractAddress
    security_council: ContractAddress
    operator: ContractAddress
    gas_price_admin: ContractAddress
    mapping(ContractAddress => bool) guardians
    protocol_frozen_until: felt252
}

// Admin bypassing the delay
* fn emergency_execution(call: Call, predecessor: felt252, salt
* fn transfer_ownership(new_owner: felt252) SECURITY_COUNCIL
* fn soft_pause() GUARDIAN
* fn hard_pause() SECURITY_COUNCIL
* fn unpause() SECURITY_COUNCIL or delay passed

// Custom Timelock functions
* fn cancel(id: felt252) CANCELLER
// !!! Need to check the function to execute is in a whitelist
* fn execute(call: Call, predecessor: felt252, salt: felt252) OR

// Self management
* fn change_operator(new_address_operator: ContractAddress) SECU
```

```
* fn change_security_council(new_security_council_address: Contr
* fn add_guardian(new_guardians_address: ContractAddress) SECURI
* fn remove_guardian(guardian_to_remove_address: ContractAddress
* fn change_gas_price_admin(new_gas_price_admin: ContractAddress
```

Third proposal: doing everything in house for maximum flexibility

Multisigs and intermediary contracts would be done in house to have flexibility on thresholds to set. More work to write hence more risks. Would not advice for this proposal at this stage of development.

Decision

	Audit scope + cost	Maintainability / Complexity
1/ OZ Timelock + Custom Protocol Handler	✅	✅
2/ OZ Timelock merged with protocol handler	✅	❌
3/ Fully custom	❌	❌

The third propose should be discard for several reason: there is no need to do everything in house for the current need.
The decision should be between proposal 1 and 2

1. Pros: protocol logic and time lock logic are separated. Cons: Ops can be more difficult.
2. Pros: only one contract. Cons: More complexity in code.

Proposal 1 should be preferred.

Multisigs

See Security Council multisig for the implementation discussion

The security council, the guardians and the Operator have to be mutlsig in order to avoid any takeover or loss of control of the different contracts. Suggestion is also that each member of those entity should be multisigs.