Eternal Permission Kernel (EPK) — v1 Specification (Final)

Immutable Core • Modular Validators • Revocable Capabilities

0) Design Principles

- **Immutable Kernel:** ядро не апгрейдится.
- **No implicit approvals:** модель не требует approve(∞) как базовый примитив.
- **Capability-based execution:** агент получает **Policy**, а не доступ к активам.
- **Revocability-first:** мгновенный отзыв прав (panic/revoke).
- **Minimal attack surface:** в ядре нет бизнес-логики (DEX/oracles/price math).
- **Extensible via Validators:** вся "умность" и эволюция — во внешних модулях.

---

1) Kernel (Immutable Executor)

1.1 Policy Model

```
struct Policy {
    address owner;            // policy owner (signer)
    bool    active;           // quick revoke switch
    uint48  validUntil;       // policy TTL (0 = no expiry)
    uint96  maxValuePerCall;  // max native value per call
    address validator;        // optional validator (0 = none)
}
```

1.2 Agent Permissions (with TTL)

```
struct AgentPermission {
    bool   allowed;
    uint40 validUntil;        // agent TTL (0 = no expiry)
}
```

1.3 Storage (normative)

- `policies[policyId] -> Policy` (policy exists iff `owner != address(0)`)
- `nonces[policyId] -> uint256`
- `agentPermission[policyId][agent] -> AgentPermission`
- `callAllowed[policyId][callKey] -> bool`
  where `callKey = keccak256(abi.encodePacked(target, selector))`

---

2) Execution Interface

2.1 Kernel function

```
execute(
    uint256 policyId,
    address target,
```

```
    uint256 value,
    bytes calldata data,
    uint256 deadline,
    bytes calldata signature
) external payable returns (bytes memory);
```

2.2 Strict execution rules (normative order)

Kernel MUST enforce, in this order:

1. **Policy exists & active**

- `Policy p = policies[policyId]`
- `require(p.owner != address(0))`
- `require(p.active)`

2. **Deadline required & fresh**

- `require(deadline > block.timestamp)`
  (prevents "timeless" signatures)

3. **Effective expiry**

- Define:
  - `policyExpiry = (p.validUntil == 0) ? MAX_UINT256 : uint256(p.validUntil)`
  - `effectiveExpiry = min(policyExpiry, deadline)`
- `require(block.timestamp <= effectiveExpiry)`

4. **Agent permission**

- `AgentPermission ap = agentPermission[policyId][msg.sender]`
- `require(ap.allowed)`
- if `ap.validUntil != 0` then `require(block.timestamp <= ap.validUntil)`

5. **Target + selector allowlist**

- `bytes4 selector = bytes4(data[0:4])`
- `callKey = keccak256(abi.encodePacked(target, selector))`
- `require(callAllowed[policyId][callKey])`

6. **Value bound**

- `require(value <= p.maxValuePerCall)`
- `require(msg.value == value)` (no hidden value)

7. **Nonce match**

- `uint256 nonce = nonces[policyId]` (current)

8. **EIP-712 signature check (owner only)**
   Signature MUST match typed data:

```
Execute(policyId, target, value, keccak256(data), nonce, deadline)
```

Recovered signer MUST equal `p.owner`.

9. **Optional validator**
   If `p.validator != address(0)` then kernel MUST call:

- `validator.validate(policyId, p.owner, msg.sender, target, value, data)`
- **Any revert MUST propagate unchanged** (no catch/rethrow).

10. **Nonce increment then call**

- increment nonce (strictly monotonic)
- perform `target.call{value: value}(data)`
- if call fails: revert (bubble reason if possible)

11. **Audit event**
    Kernel MUST emit an event with at least:
    `policyId, owner, agent, target, selector, value`.

---

3) Nonce Management (Revocation-grade)

3.1 Rule

- `nonces[policyId]` MUST be strictly monotonic increasing per successful execution.

3.2 Emergency bump (normative)

Owner MUST be able to invalidate pending/queued signatures:

```
emergencyNonceBump(uint256 policyId, uint256 newNonce)
```

- `require(msg.sender == policy.owner)`
- `require(newNonce > nonces[policyId])`
- set nonce to `newNonce` (cannot decrease)

---

4) Revocation (Panic Mode)

Owner MUST be able to instantly disable policy:

```
setPolicyActive(uint256 policyId, bool active)
```

Immediate effect. No timelock inside kernel.

---

5) Validator Interface (Modular, revert-on-fail)

Kernel MUST treat validators as pure policy constraints.

```
interface IPolicyValidator {
    function validate(
        uint256 policyId,
        address owner,
        address agent,
        address target,
        uint256 value,
        bytes calldata data
    ) external;
}
```

- Validator MUST revert with **custom errors** on failure.
- Kernel MUST NOT mask validator errors.

---

6) Registry (Recommended Layer, not required by Kernel)

Purpose: trust layer for validators/versions (governance + timelock). Kernel независим.

Minimum:

```
isValidatorApproved(address validator, uint256 version) external view returns
(bool);
getValidatorMeta(address validator) external view returns (bytes32 codeHash,
uint256 version, address approvedBy, uint256 timestamp);
```

- Updates via multisig/DAO + timelock.
- UI/SDK SHOULD default to registry-approved validators.

---

7) Threat Feed (Optional Layer, Merkle-root only)

Purpose: обновляемая “информация угроз” без изменения ядра.

Minimum:

- `bytes32 root`
- `uint256 epoch`

Update via governance:

```
updateRoot(bytes32 newRoot, uint256 newEpoch, bytes signatures)
```

Verification:

```
verifyProof(bytes32 itemHash, bytes32[] calldata proof) external view returns
(bool);
```

Kernel MUST NOT depend on ThreatFeed directly. Only validators may.

---

8) EIP-712 Domain (normative)

Domain MUST include:

- `name = "Eternal Permission Kernel"`
- `version = "1"`
- `chainId`
- `verifyingContract` (kernel address)

---

9) Required v1 Validators (reference set)

- **SpendLimitValidator** (rolling window, per-token/per-tx)
- **TargetSelectorGuard** (strict allowlist + hard-block approve/permit/delegatecall by default)
- **ThreatFeedBlocklistValidator** (blocks target/selector/token in feed root)

(Остальное — v1.1+)

---

10) Security Invariants (Must Always Hold)

1. Execution cannot occur without valid owner signature.
2. Nonce strictly increases after each successful execution.
3. Revoked policy blocks all executions immediately.
4. Expired policy or expired signature blocks execution.
5. Unapproved agent blocks execution (incl. agent TTL).
6. Unapproved (target, selector) blocks execution.
7. Validator revert MUST block execution and propagate unchanged.
8. Kernel contains no protocol-specific business logic.