SLOWMIST SLOWMIST AML MISTTRACK MISTEYE

A Deep Dive into EIP-7702 with Best Practices

By: Kong　Leader of Security Audit Team

Ethereum Pectra Upgrade

Ethereum Pectra Upgrade

EIP-7251: Increase the Maximum Effective Balance

EIP-7002: Execution Layer Triggerable Exits

EIP-6110: Supply Validator Deposits On-Chain

EIP-7685: General Purpose Execution Layer Requests

EIP-7623: Increase Calldata Cost

EIP-7691: Blob Throughput Increase

EIP-7840: Add Blob Schedule to Execution Layer Config Files

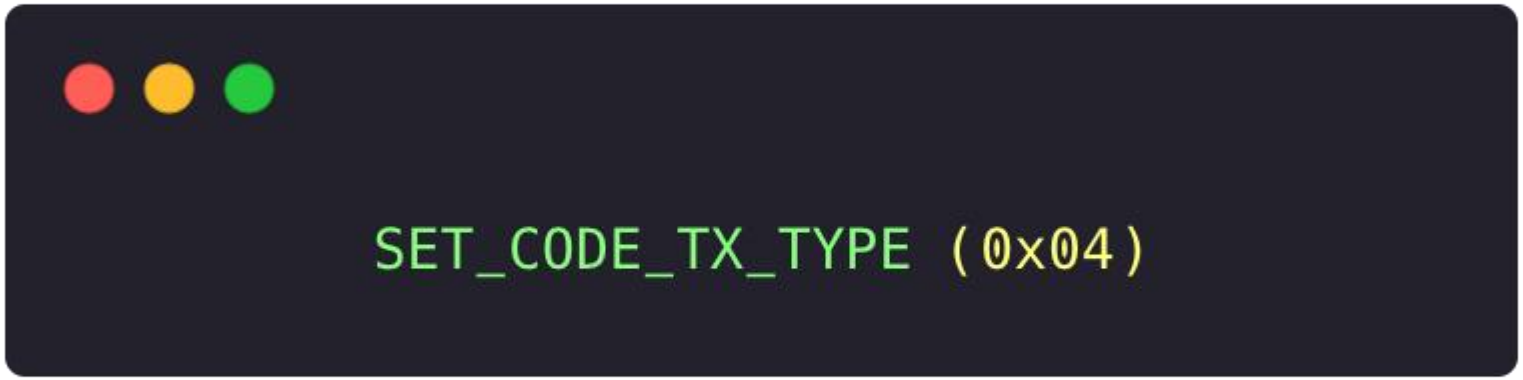
EIP-2537: Precompile for BLS12-381 Curve Operations

EIP-2935: Save Historical Block Hashes in State

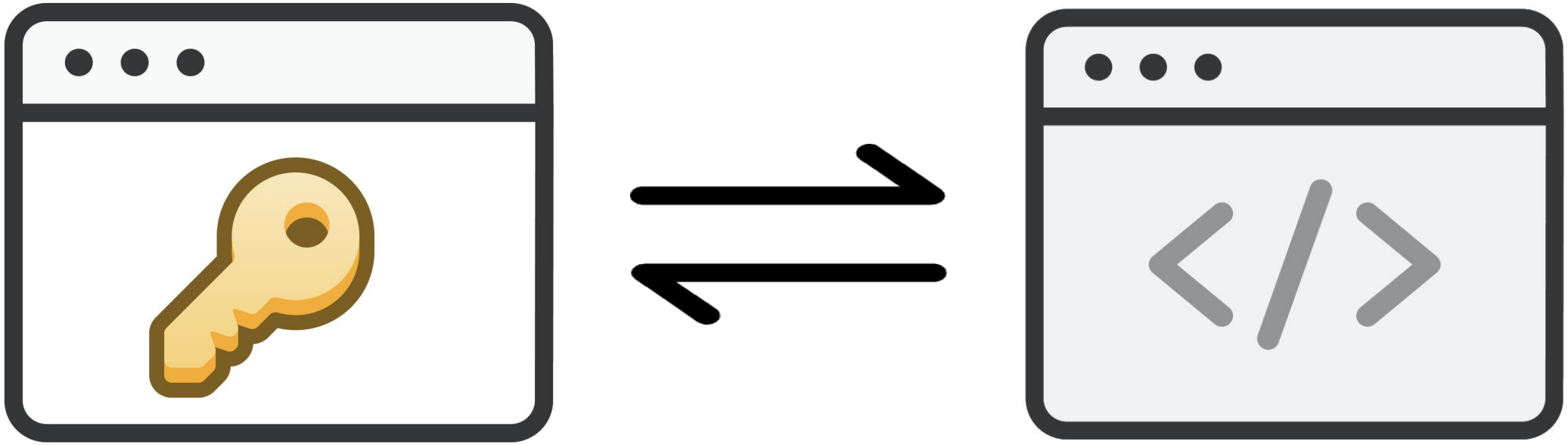
EIP-7702: Set EOA Account Code

What is EIP-7702

What is EIP-7702



What is EIP-7702



Externally Owned Account

Smart Contract

Implementation of EIP - 7702

Implementation of EIP - 7702

EIP-7702

```
rlp([
      chain_id,
      nonce,
      max_priority_fee_per_gas,
      max_fee_per_gas,
      gas_limit,
      destination,
      value,
      data,
      access_list,
      authorization_list,
      signature_y_parity,
      signature_r,
      signature_s
    ])
```

EIP-1559

```
rlp([
      chain_id,
      nonce,
      max_priority_fee_per_gas,
      max_fee_per_gas,
      gas_limit,
      destination,
      value,
      data,
      access_list,
      signature_y_parity,
      signature_r,
      signature_s
    ])
```

Implementation of EIP - 7702

```
authorization_list = [
    [chain_id, address, nonce, y_parity, r, s],
    [chain_id, address, nonce, y_parity, r, s],
    [chain_id, address, nonce, y_parity, r, s],
    [chain_id, address, nonce, y_parity, r, s],
    [chain_id, address, nonce, y_parity, r, s],
    [chain_id, address, nonce, y_parity, r, s],
    [chain_id, address, nonce, y_parity, r, s],
    ...
]
```

Implementation of EIP - 7702

```
authorization = [chain_id, address, nonce, y_parity, r, s]
```

chain_id: The chain on which the authorization is available

address: The target address entrusted by the authorizer (usually a smart contract)

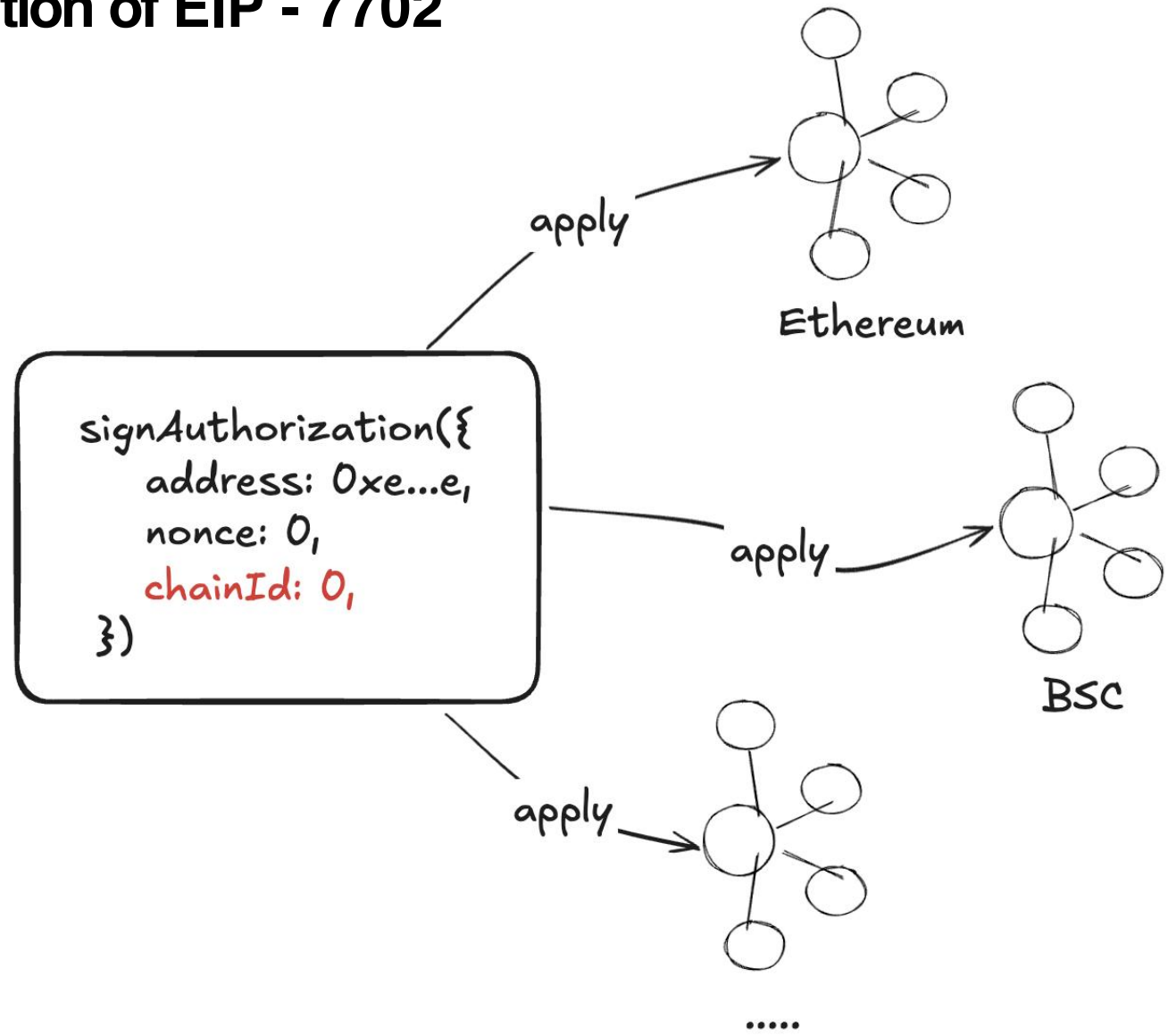
nonce: The nonce of the account

y_parity, r, s: The signed data after signing the chain_id, address, and nonce

Implementation of EIP - 7702

```go
// Go-ethereum/core/types/tx_setcode.go#L109-L113
func (a *SetCodeAuthorization) sigHash() common.Hash {
    return prefixedRlpHash(0x05, []any{
        a.ChainID,
        a.Address,
        a.Nonce,
    })
}
```

Implementation of EIP - 7702



Implementation of EIP - 7702

Implementation of EIP - 7702

EIP-7702

```
rlp([
      chain_id,
      nonce,
      max_priority_fee_per_gas,
      max_fee_per_gas,
      gas_limit,
      destination,
      value,
      data,
      access_list,
      authorization_list,
      signature_y_parity,
      signature_r,
      signature_s
    ])
```

```
// Go-ethereum/core/state_transition.go#L388-L390
if msg.To == nil {
            return fmt.Errorf("%w (sender %v)",
ErrSetCodeTxCreate, msg.From)
}
```

Implementation of EIP - 7702

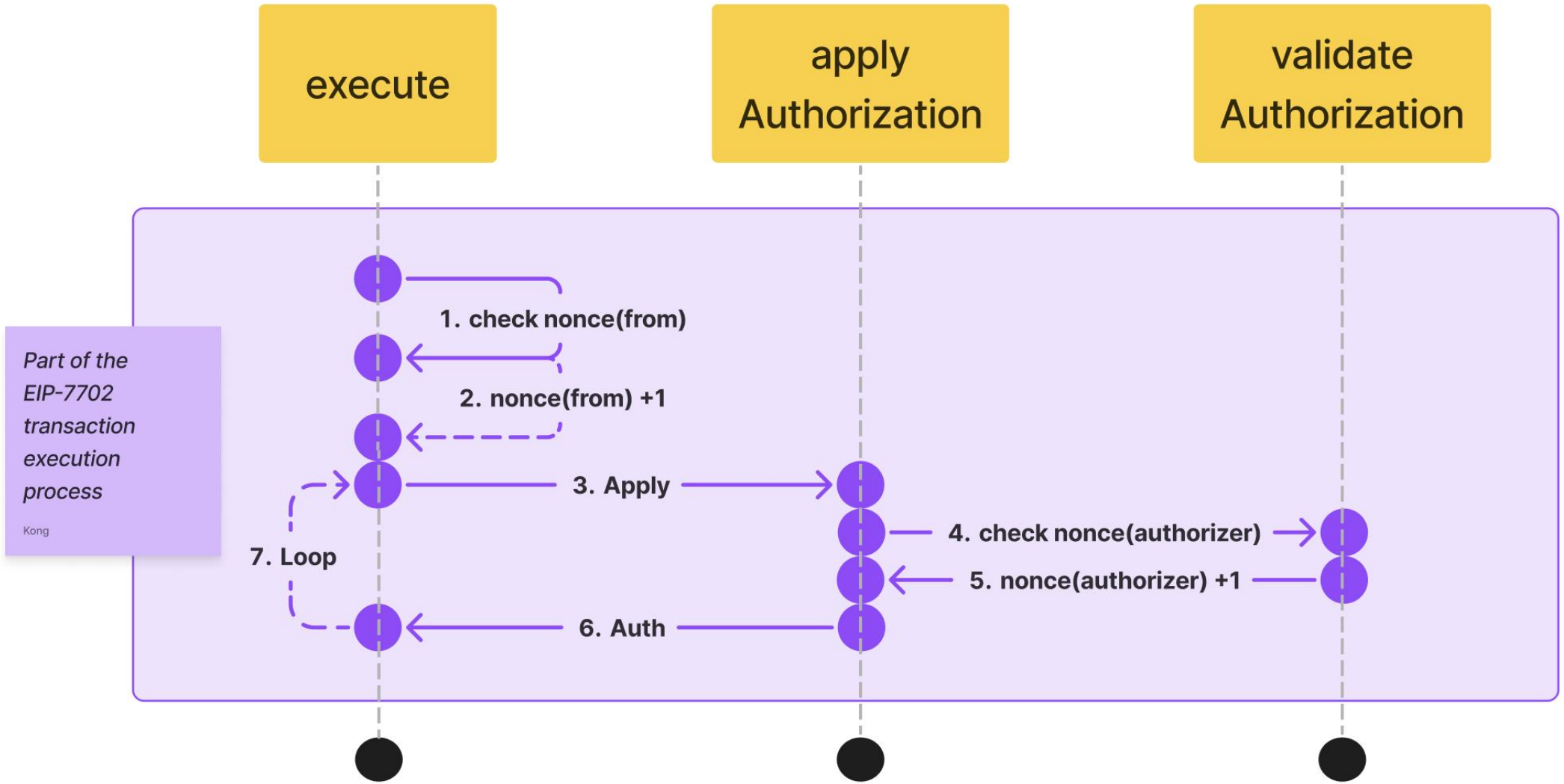
EIP-7702

```
rlp([
      chain_id,
      nonce,
      max_priority_fee_per_gas,
      max_fee_per_gas,
      gas_limit,
      destination,
      value,
      data,
      access_list,
      authorization_list,
      signature_y_parity,
      signature_r,
      signature_s
    ])
```

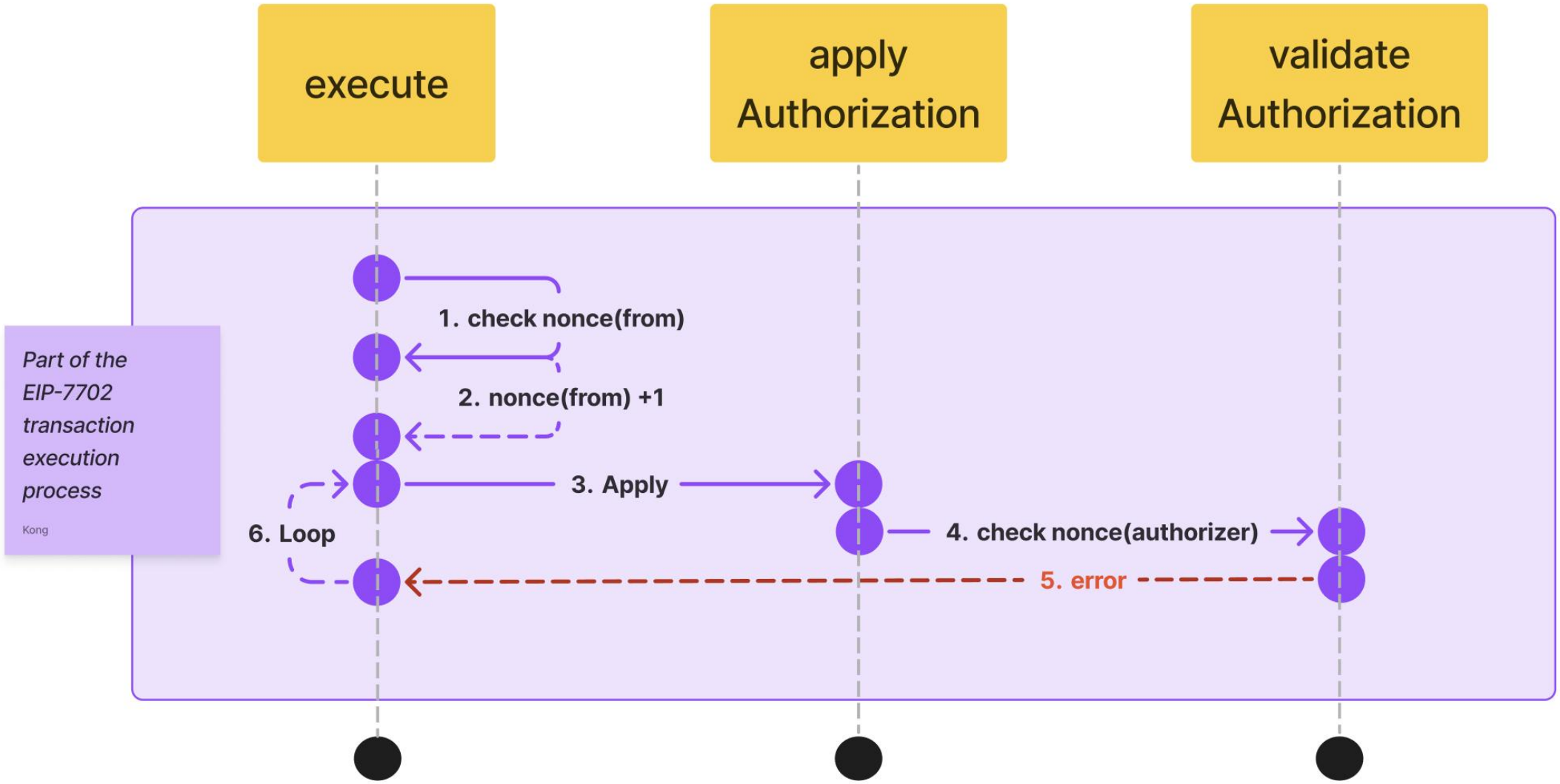
```
// Go-ethereum/core/state_transition.go#L391-L393
if len(msg.SetCodeAuthorizations) == 0 {
            return fmt.Errorf("%w (sender %v)",
ErrEmptyAuthList, msg.From)
}
```

Implementation of EIP - 7702

Implementation of EIP - 7702



Implementation of EIP - 7702



Implementation of EIP - 7702

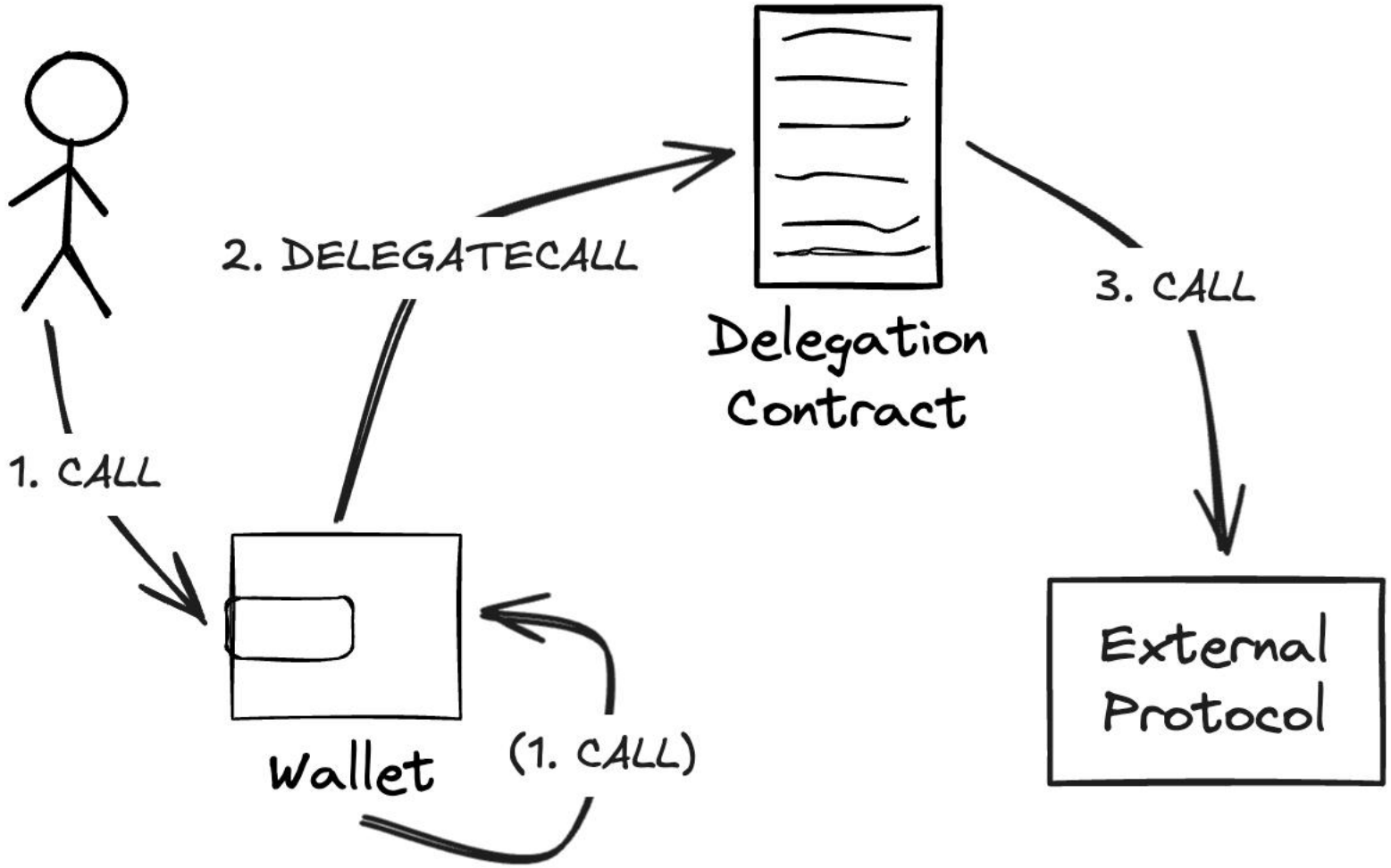
```go
// Go-ethereum/core/state_transition.go#L612
st.state.SetCode(authority, types.AddressToDelegation(auth.Address))

// Go-ethereum/core/types/tx_setcode.go#L45
var DelegationPrefix = []byte{0xef, 0x01, 0x00}

func AddressToDelegation(addr common.Address) []byte {
    return append(DelegationPrefix, addr.Bytes()...)
}
```

```
(0xef0100 || 0xbd026d743285610450ea11c7652710f75b6a082c)
```

- **Implementation of EIP - 7702**



Best Practices

Best Practices

Privatekey Management

- Even though EOAs (Externally Owned Accounts) have measures like social recovery after delegation to prevent losses from private key loss, the risk of private key leakage remains.
- After delegation, the EOA private key still retains the highest control rights and can dispose of assets at will.
- Locally deleting the private key cannot eliminate the risk of leakage, especially in supply chain attack scenarios.

For users: When using delegated accounts, private key protection should still be your top priority.

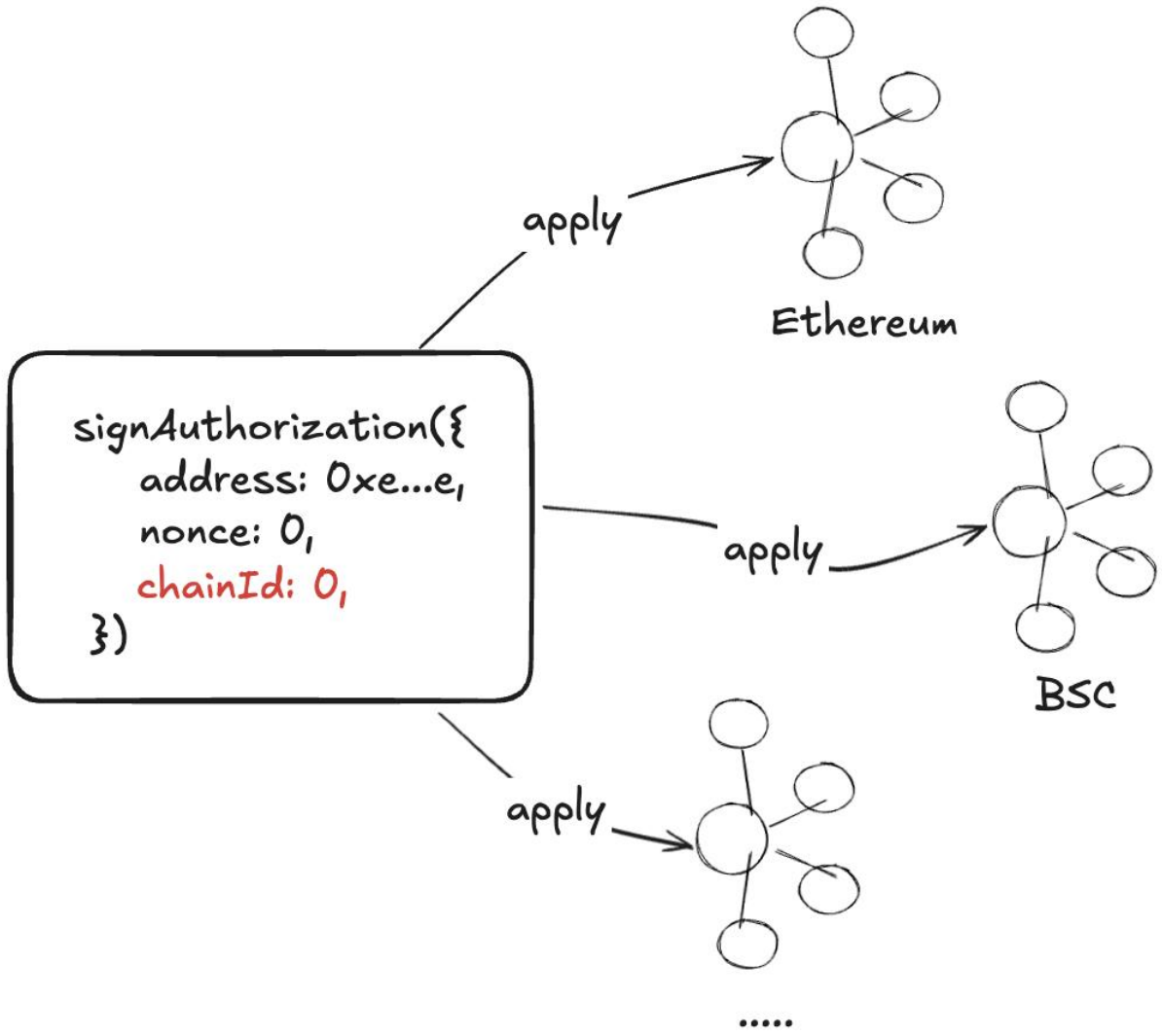
Remember: Not your keys, not your coins.

Best Practices

Multi-chain replay

Users can select the chain where delegation takes effect through chainId, and setting chainId to 0 allows for multi-chain replay.

However, the same contract address may have different implementation code across multiple chains.



Best Practices

Multi-chain replay

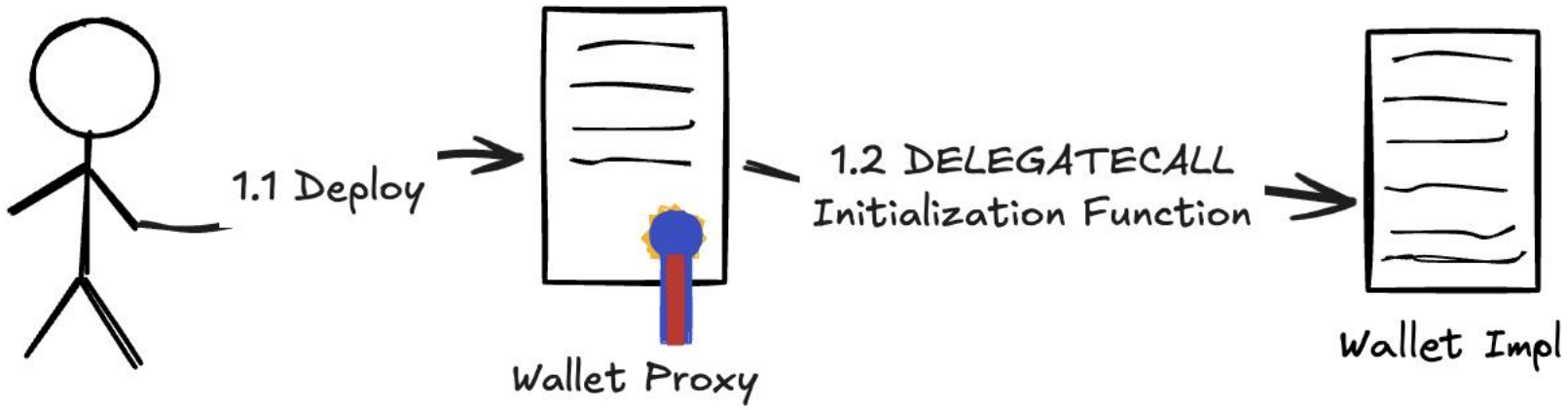
For wallet service providers: When users delegate, you should verify that the chain where delegation takes effect matches the currently connected network. Also, remind users about the potential risks of signing delegations with chainId set to 0.

For users: You should be aware that identical contract addresses on different chains may not always have the same contract code. You should clearly understand your delegation target before proceeding.

Best Practices

No initcode

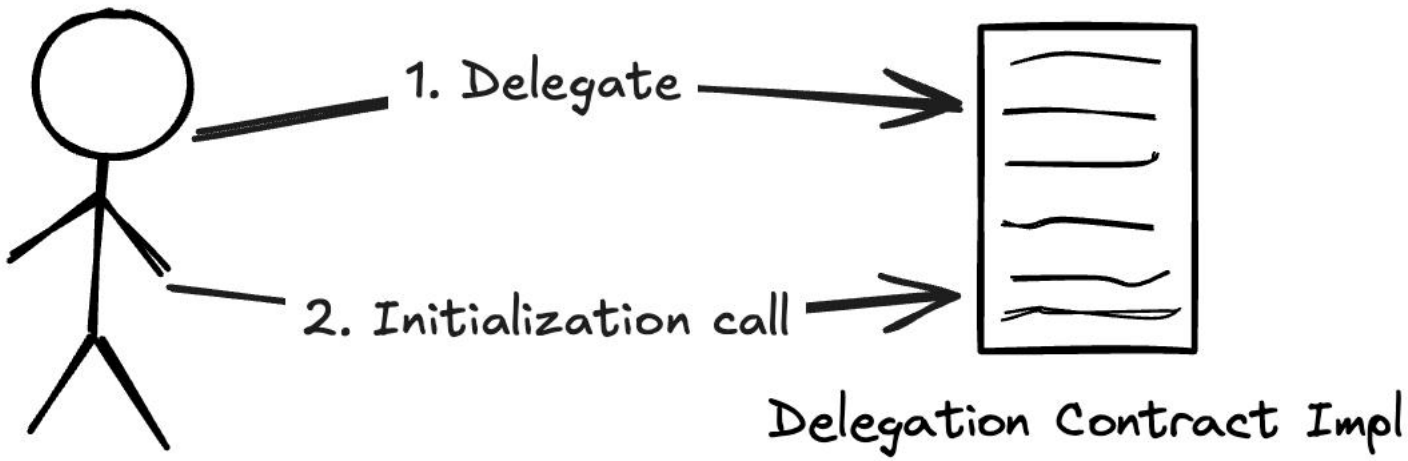
Mainstream smart contract wallets mostly adopt an upgradeable (proxy) model, initializing through DELEGATECALL during deployment.



Best Practices

No initcode

The EIP-7702 delegation only updates the address code field and cannot call the delegated address for initialization.



Best Practices

No initcode

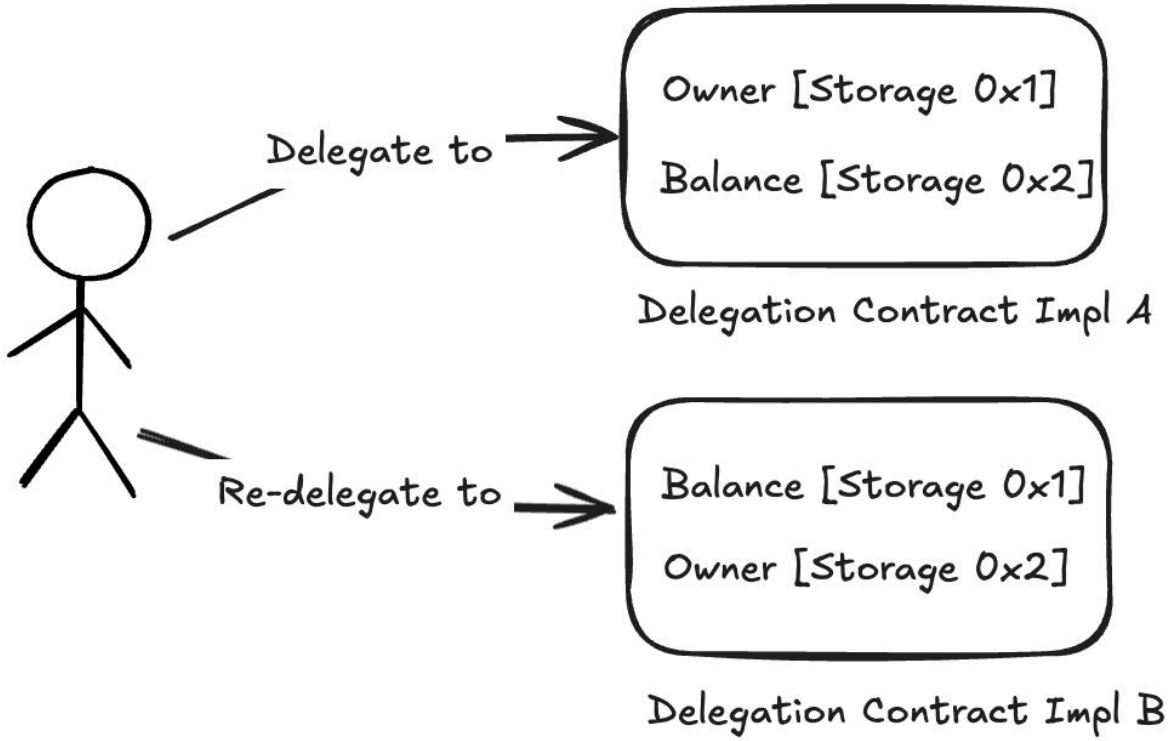
The EIP-7702 delegation only updates the address code field and cannot call the delegated address for initialization.

For developers: When adapting EIP-7702 to work with existing smart contract wallets like EIP-4337, attention should be paid to implementing permission checks during wallet initialization (for example, by using ecrecover to verify the signing address for permission validation) to prevent the risk of front-running wallet initialization operations.

Best Practices

Storage Management

When users redelegate to different contract addresses due to functionality changes or other reasons, differences in storage structures between contracts may exist. This can potentially cause the new contract to reuse old data, leading to account lockouts, financial losses, and other issues.



Best Practices

Storage Management

For users: Exercise caution when redelegating. Before standardized redelegation protocols are established, withdraw your funds before making any redelegation.

For developers: Follow the "Namespace Formula" proposed in ERC-7201 during development, assigning variables to designated independent storage locations.

For wallet service providers: If possible, after ERC-7779 is approved, verify storage compatibility before redelegation and clean up old storage data through the interface of the previous delegation.

Best Practices

False Top-up

After delegation, EOAs can function as smart contracts, which may lead to exchanges (CEXs) facing widespread smart contract deposits.

Best Practices

False Top-up

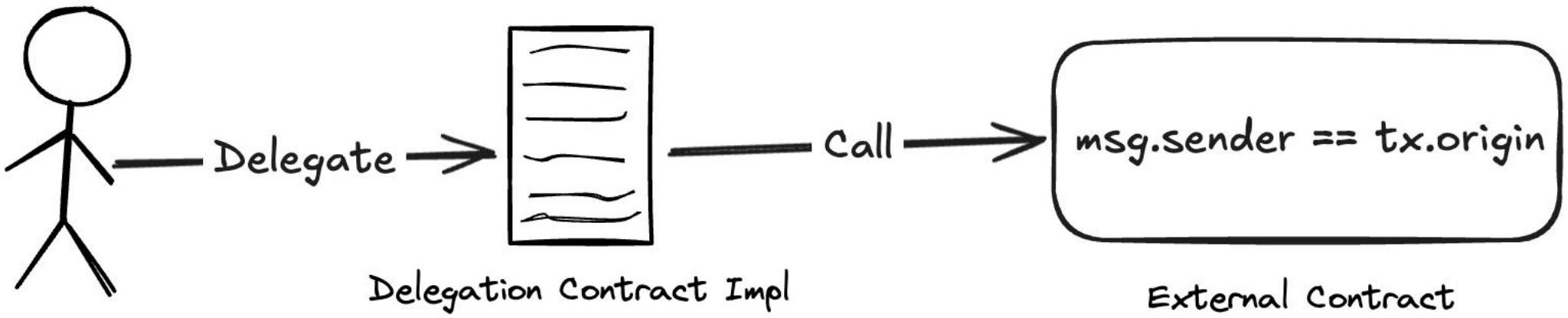
After delegation, EOAs can function as smart contracts, which may lead to exchanges (CEXs) facing widespread smart contract deposits.

For CEXs: They should examine the status of each deposit transaction through trace, to guard against the risk of false top-up from smart contracts.

Best Practices

Account Conversion

After implementing EIP-7702 delegation, accounts can be converted between EOA and SC, capable of both initiating transactions and being called.



Best Practices

Account Conversion

After implementing EIP-7702 delegation, accounts can be converted between EOA and SC, capable of both initiating transactions and being called.

For contract developers: Assuming tx.origin is always an EOA will no longer be viable. Similarly, using `msg.sender == tx.origin` checks to defend against reentrancy attacks will become ineffective. Developers should assume that future participants might all be smart contracts.

Best Practices

Compatibility

When an account is used as a smart contract, compatibility with existing infrastructure must be considered. For example: existing ERC-721 and ERC-777 tokens have Hook functions when transferring to contracts, meaning recipients must implement corresponding callback functions to successfully receive tokens.

For developers: The target contracts for user delegation should implement interfaces compatible with mainstream infrastructure to ensure compatibility with mainstream tokens/protocols.

Best Practices

Phishing

After implementing EIP-7702 delegation, assets in user accounts will be controlled by smart contracts. If a user delegates their account to a malicious contract, it becomes trivially easy for attackers to steal funds.

(Key Data Indicators of Wallet Drainer Attacks in 2024)

Best Practices

Phishing

After implementing EIP-7702 delegation, assets in user accounts will be controlled by smart contracts. If a user delegates their account to a malicious contract, it becomes trivially easy for attackers to steal funds.

For wallet service providers: It is especially important to support EIP-7702 type transactions as soon as possible, and to prominently display the target contract to users when they sign delegation transactions, to mitigate the risk of phishing attacks.

If possible, conducting more in-depth automatic analysis of the target delegation contracts (open source verification, permission checks, etc.) can better help users avoid such risks.

Best Practices

- Privatekey Management
- Multi-chain replay
- No initcode
- Storage Management
- False Top-up
- Account Conversion
- Compatibility
- Phishing

In Conclusion

Official Website
https://slowmist.com

Email
team@slowmist.com

Twitter
@SlowMist_Team

GitHub
https://github.com/slowmist

Medium
https://slowmist.medium.com