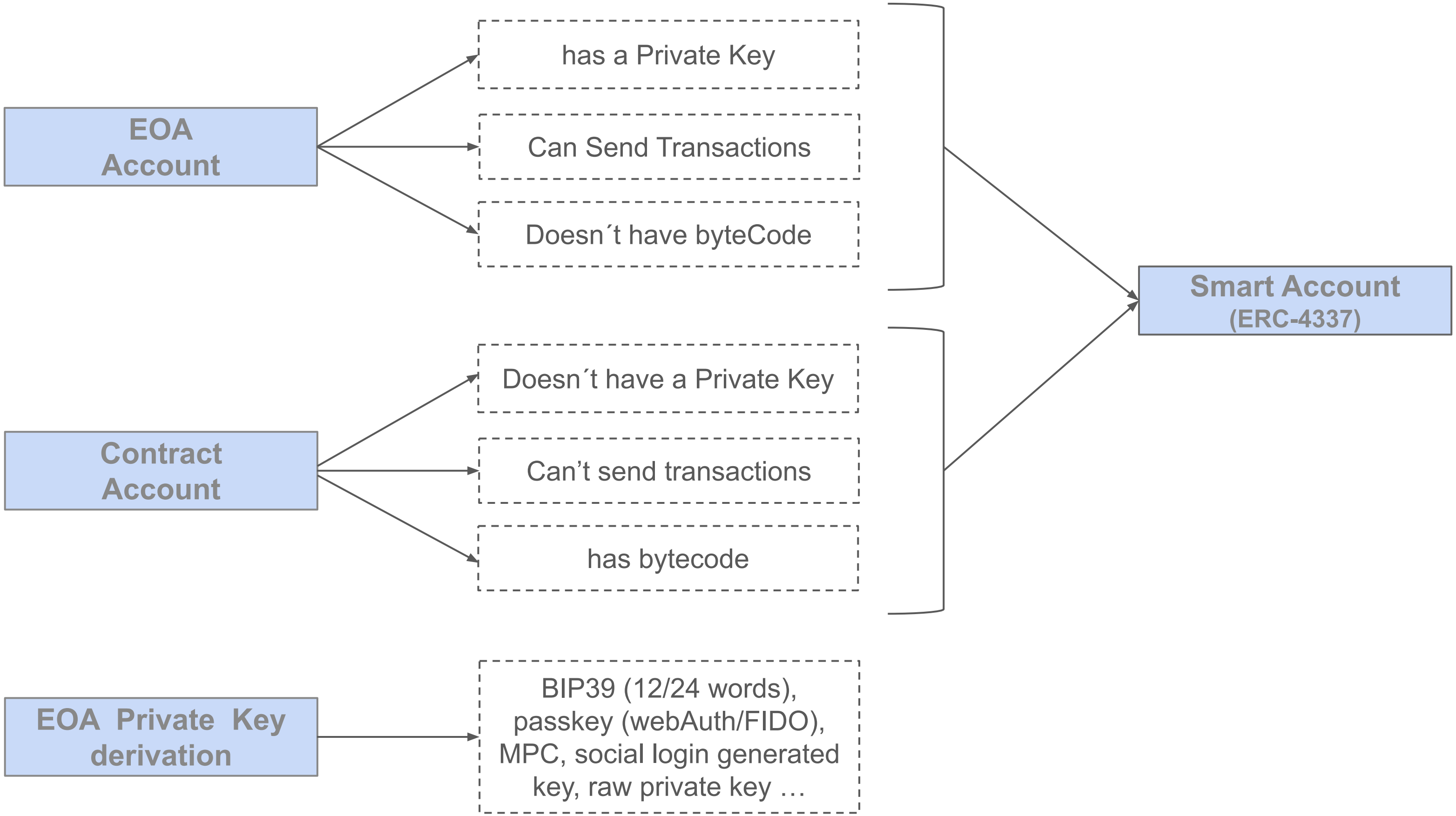
Account Abstraction

Under The Hood

What makes the **magic** happen.

Accounts Type



EOA Account

- has a Private Key
- Can Send Transactions
- Doesn´t have byteCode

Contract Account

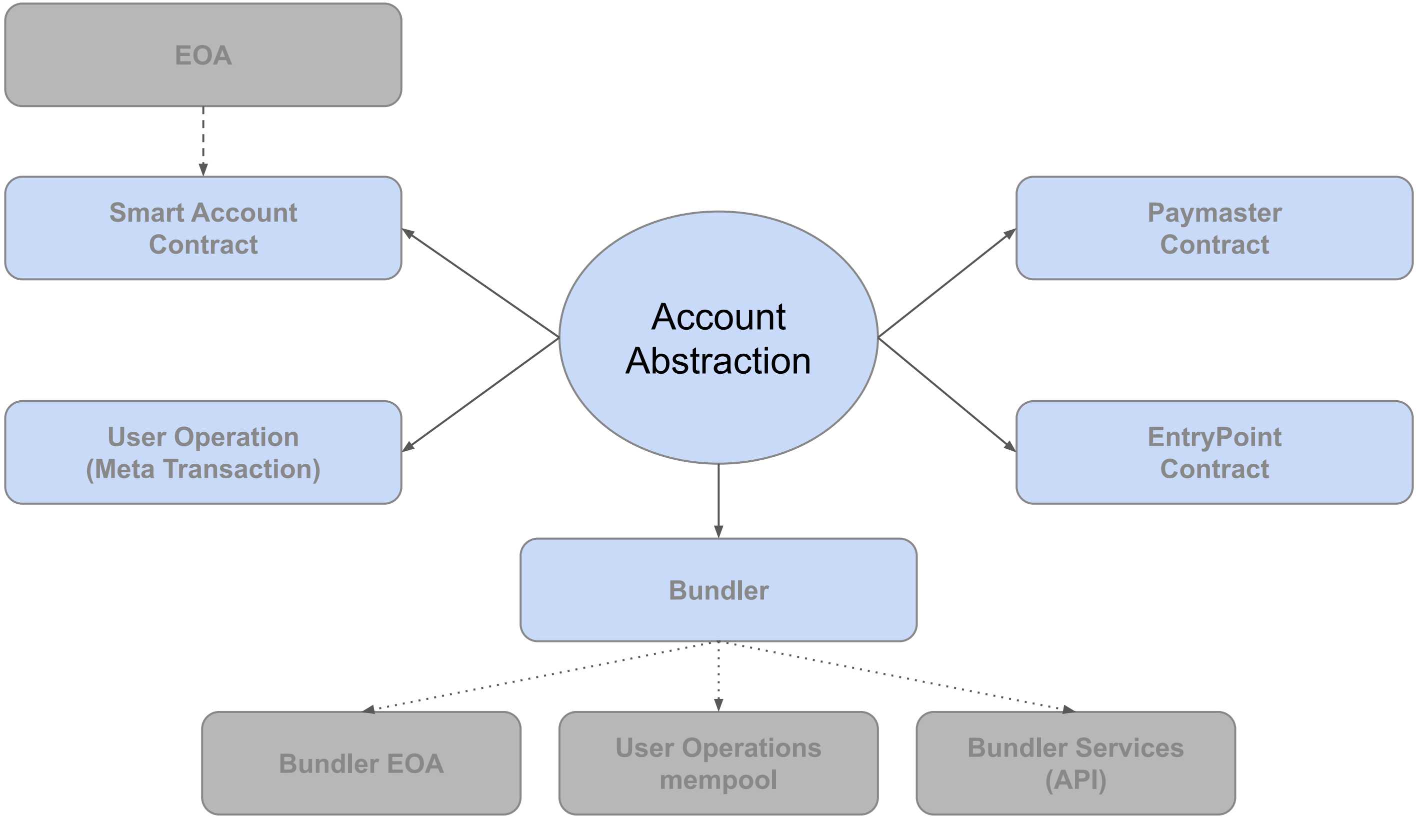
- Doesn´t have a Private Key
- Can’t send transactions
- has bytecode

Smart Account (ERC-4337)

EOA Private Key derivation

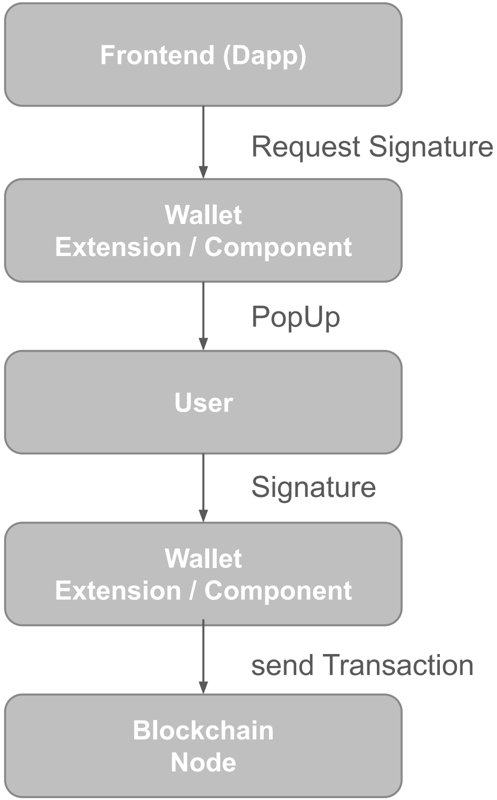
BIP39 (12/24 words), passkey (webAuth/FIDO), MPC, social login generated key, raw private key …

Players


EOA
Smart Account Contract
Account Abstraction
Paymaster Contract
User Operation (Meta Transaction)
EntryPoint Contract
Bundler
Bundler EOA
User Operations mempool
Bundler Services (API)

Transactions Flow Options

Flow 1: Frontend Sign w/EOA



Frontend (Dapp)

Request Signature

Wallet
Extension / Component

PopUp

User

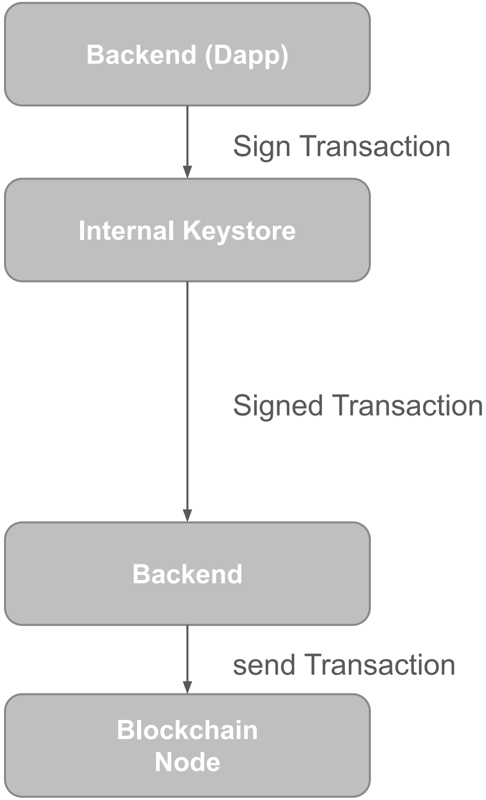
Signature

Wallet
Extension / Component

send Transaction

Blockchain
Node

Flow 2: Backend Sign w/EOA



Backend (Dapp)

Sign Transaction

Internal Keystore

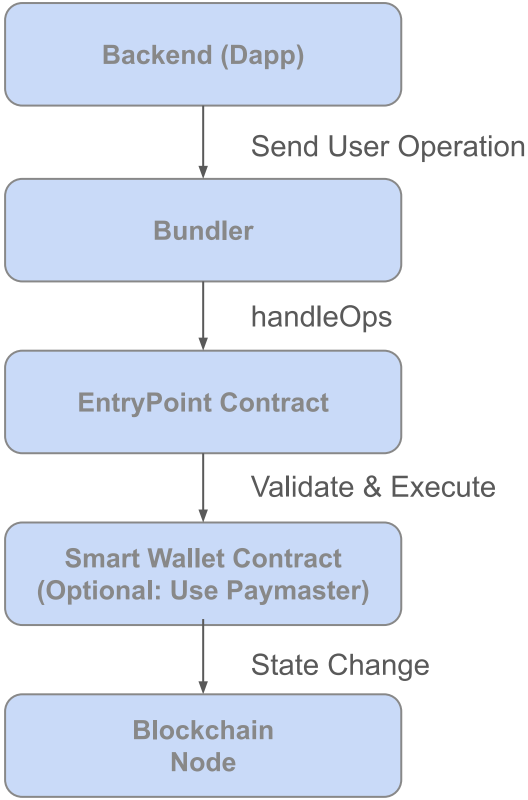
Signed Transaction

Backend

send Transaction

Blockchain
Node

Flow 3: Account Abstraction



Backend (Dapp)

Send User Operation

Bundler

handleOps

EntryPoint Contract

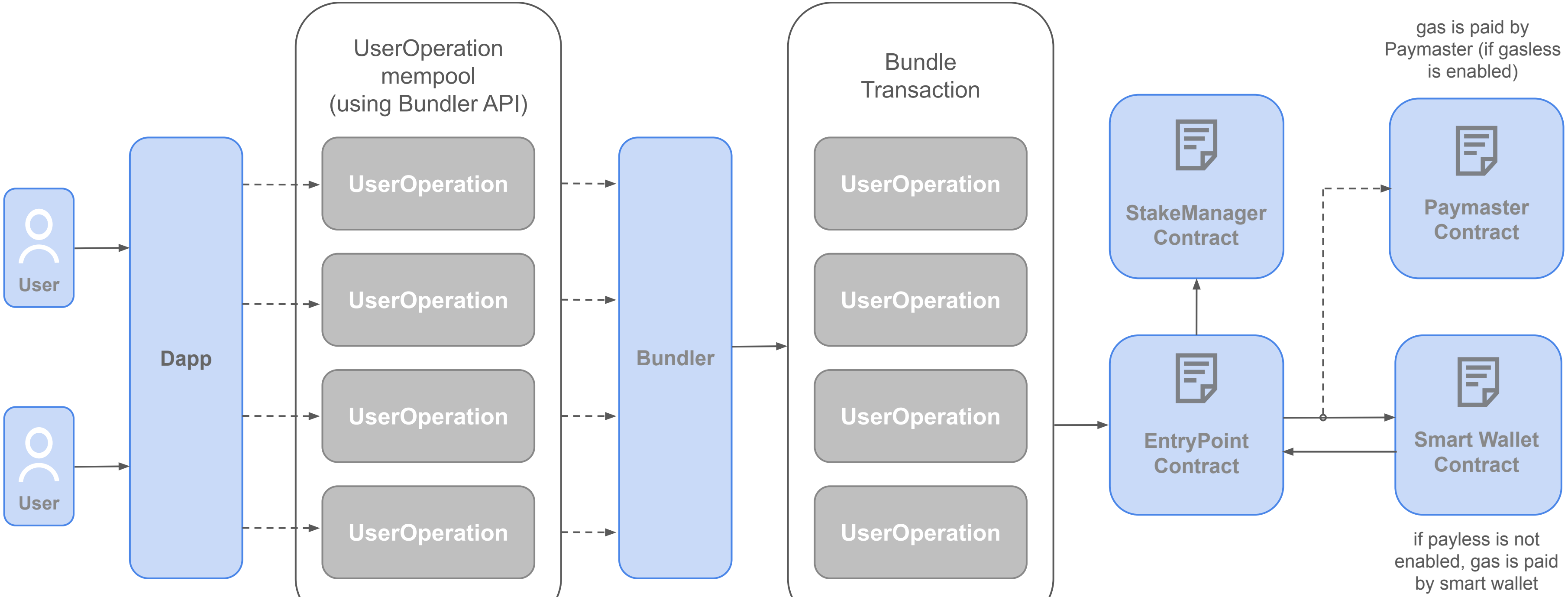
Validate & Execute

Smart Wallet Contract
(Optional: Use Paymaster)

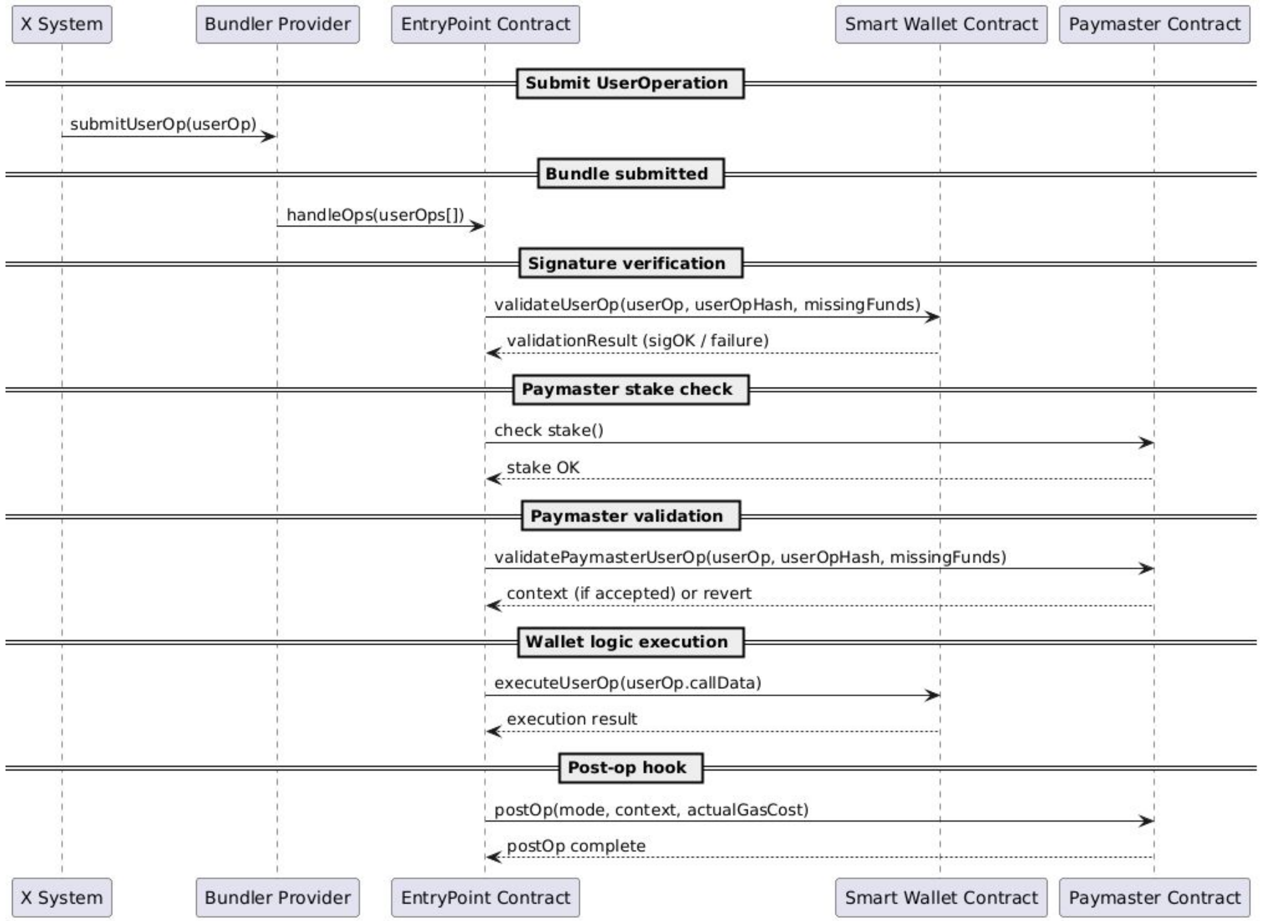
State Change

Blockchain
Node

AA Flow


User
User
Dapp
UserOperation mempool (using Bundler API)
UserOperation
UserOperation
UserOperation
UserOperation
Bundler
Bundle Transaction
UserOperation
UserOperation
UserOperation
UserOperation
StakeManager Contract
EntryPoint Contract
gas is paid by Paymaster (if gasless is enabled)
Paymaster Contract
Smart Wallet Contract
if payless is not enabled, gas is paid by smart wallet

AA Steps



X System | Bundler Provider | EntryPoint Contract | Smart Wallet Contract | Paymaster Contract

Submit UserOperation

submitUserOp(userOp)

Bundle submitted

handleOps(userOps[])

Signature verification

validateUserOp(userOp, userOpHash, missingFunds)

validationResult (sigOK / failure)

Paymaster stake check

check stake()

stake OK

Paymaster validation

validatePaymasterUserOp(userOp, userOpHash, missingFunds)

context (if accepted) or revert

Wallet logic execution

executeUserOp(userOp.callData)

execution result

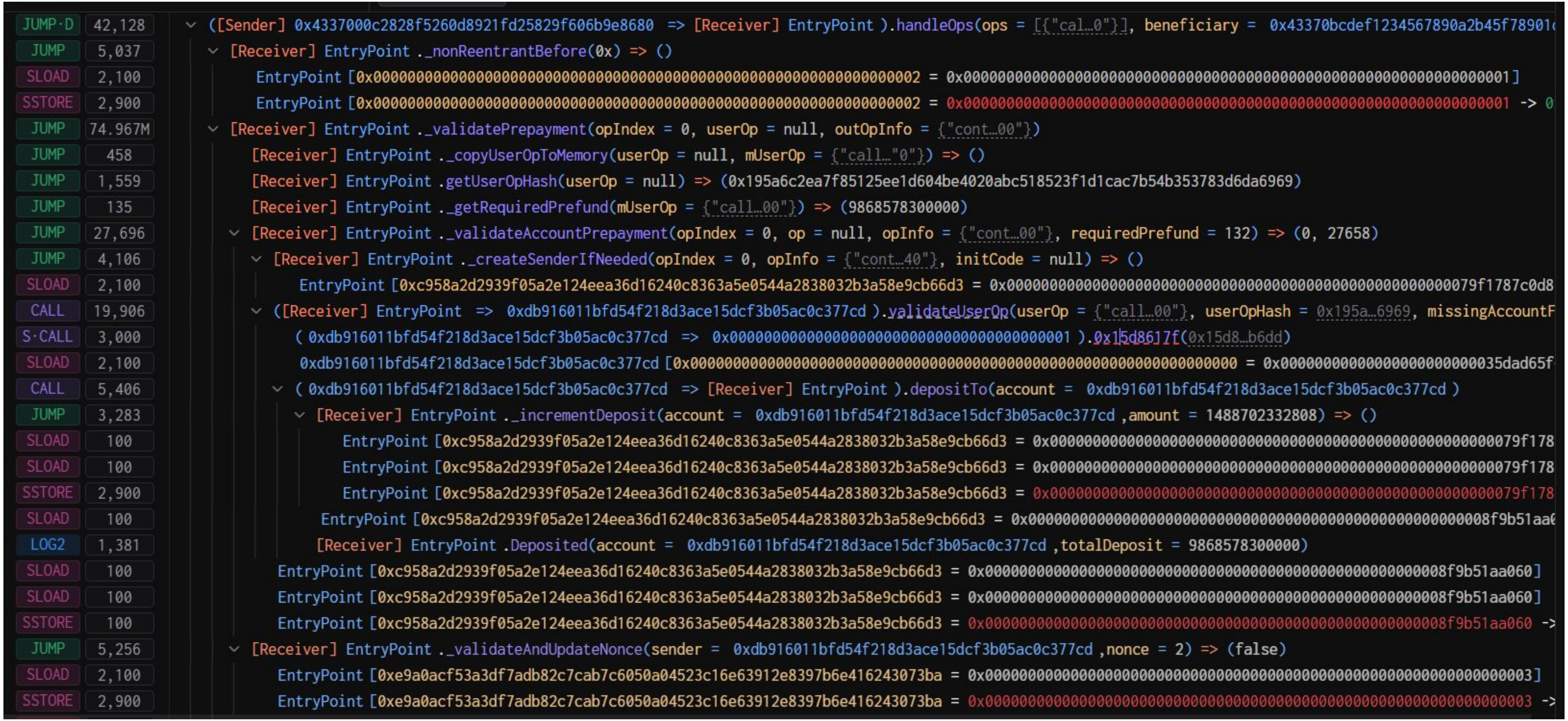
Post-op hook

postOp(mode, context, actualGasCost)

postOp complete

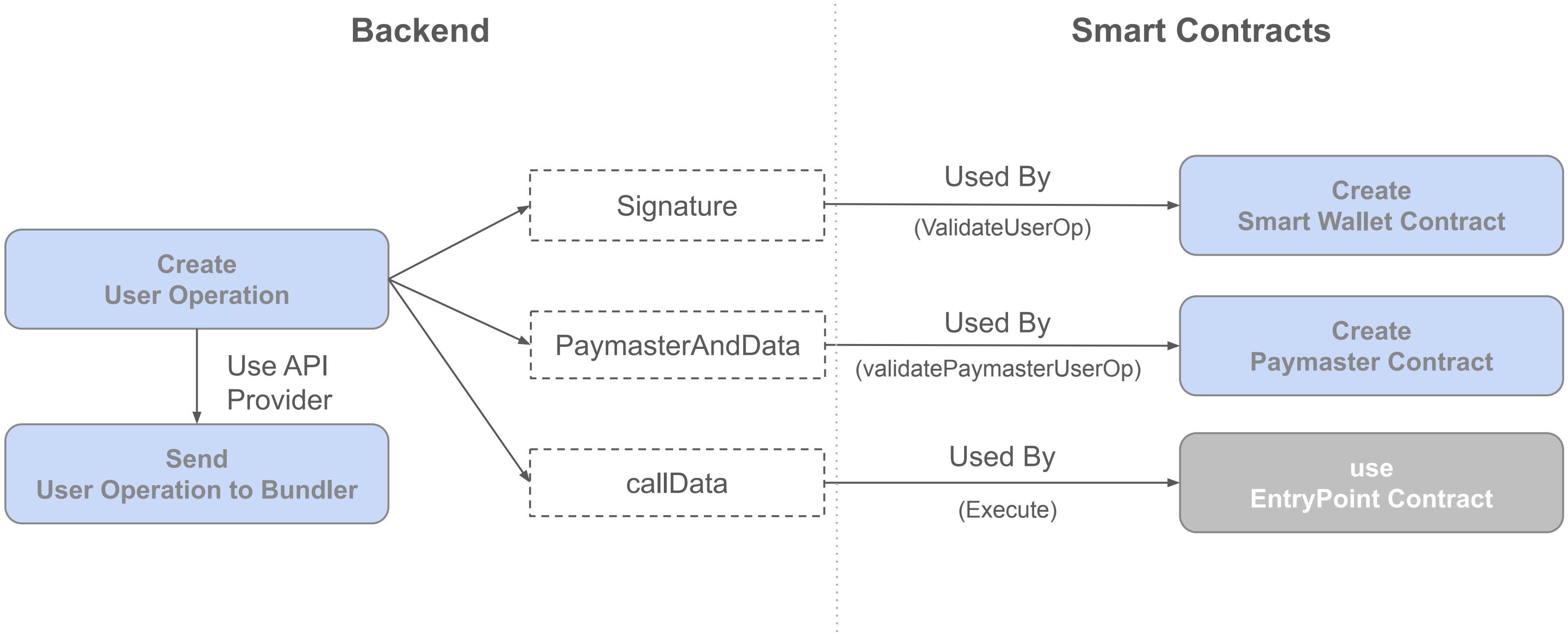
X System | Bundler Provider | EntryPoint Contract | Smart Wallet Contract | Paymaster Contract

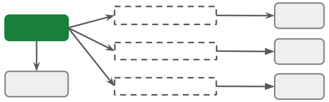
Entry Point Functions



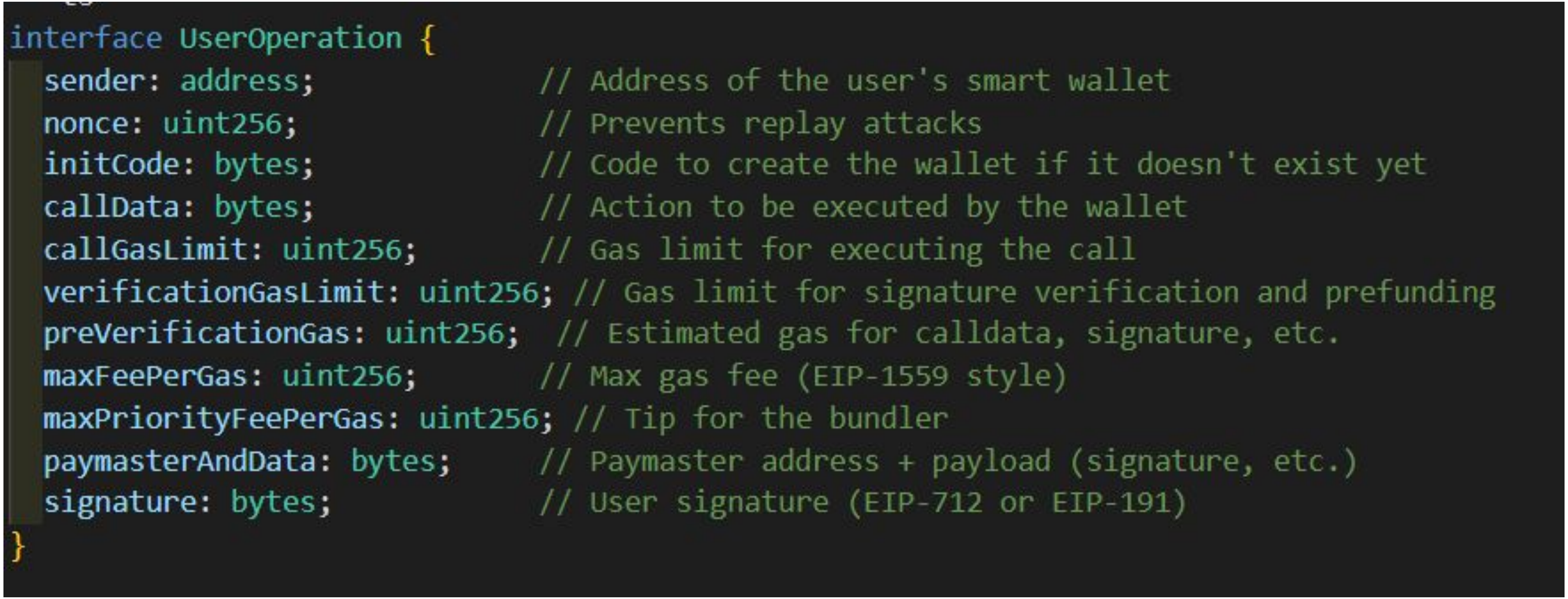
```
JUMP·D  42,128   ⌄ ([Sender] 0x4337000c2828f5260d8921fd25829f606b9e8680  => [Receiver] EntryPoint ).handleOps(ops = [{"cal…0"}], beneficiary =  0x43370bcdef1234567890a2b45f78901
JUMP    5,037      ⌄ [Receiver] EntryPoint ._nonReentrantBefore(0x) => ()
SLOAD   2,100          EntryPoint [0x0000000000000000000000000000000000000000000000000000000000000002 = 0x0000000000000000000000000000000000000000000000000000000000000001]
SSTORE  2,900          EntryPoint [0x0000000000000000000000000000000000000000000000000000000000000002 = 0x0000000000000000000000000000000000000000000000000000000000000001 -> 0
JUMP    74.967M    ⌄ [Receiver] EntryPoint ._validatePrepayment(opIndex = 0, userOp = null, outOpInfo = {"cont…00"})
JUMP    458            [Receiver] EntryPoint ._copyUserOpToMemory(userOp = null, mUserOp = {"call…"0"}) => ()
JUMP    1,559          [Receiver] EntryPoint .getUserOpHash(userOp = null) => (0x195a6c2ea7f85125ee1d604be4020abc518523f1d1cac7b54b353783d6da6969)
JUMP    135            [Receiver] EntryPoint ._getRequiredPrefund(mUserOp = {"call…00"}) => (9868578300000)
JUMP    27,696       ⌄ [Receiver] EntryPoint ._validateAccountPrepayment(opIndex = 0, op = null, opInfo = {"cont…00"}, requiredPrefund = 132) => (0, 27658)
JUMP    4,106          ⌄ [Receiver] EntryPoint ._createSenderIfNeeded(opIndex = 0, opInfo = {"cont…40"}, initCode = null) => ()
SLOAD   2,100              EntryPoint [0xc958a2d2939f05a2e124eea36d16240c8363a5e0544a2838032b3a58e9cb66d3 = 0x000000000000000000000000000000000000000000000000000079f1787c0d8
CALL    19,906         ⌄ ([Receiver] EntryPoint  =>  0xdb916011bfd54f218d3ace15dcf3b05ac0c377cd ).validateUserOp(userOp = {"call…00"}, userOpHash = 0x195a…6969, missingAccountF
S·CALL  3,000              ( 0xdb916011bfd54f218d3ace15dcf3b05ac0c377cd  =>  0x0000000000000000000000000000000000000001 ).0x15d8617f(0x15d8…b6dd)
SLOAD   2,100              0xdb916011bfd54f218d3ace15dcf3b05ac0c377cd [0x0000000000000000000000000000000000000000000000000000000000000000 = 0x000000000000000000000000035dad65f
CALL    5,406            ⌄ ( 0xdb916011bfd54f218d3ace15dcf3b05ac0c377cd  => [Receiver] EntryPoint ).depositTo(account =  0xdb916011bfd54f218d3ace15dcf3b05ac0c377cd )
JUMP    3,283              ⌄ [Receiver] EntryPoint ._incrementDeposit(account =  0xdb916011bfd54f218d3ace15dcf3b05ac0c377cd ,amount = 1488702332808) => ()
SLOAD   100                    EntryPoint [0xc958a2d2939f05a2e124eea36d16240c8363a5e0544a2838032b3a58e9cb66d3 = 0x000000000000000000000000000000000000000000000000000079f178
SLOAD   100                    EntryPoint [0xc958a2d2939f05a2e124eea36d16240c8363a5e0544a2838032b3a58e9cb66d3 = 0x000000000000000000000000000000000000000000000000000079f178
SSTORE  2,900                  EntryPoint [0xc958a2d2939f05a2e124eea36d16240c8363a5e0544a2838032b3a58e9cb66d3 = 0x000000000000000000000000000000000000000000000000000079f178
SLOAD   100                  EntryPoint [0xc958a2d2939f05a2e124eea36d16240c8363a5e0544a2838032b3a58e9cb66d3 = 0x000000000000000000000000000000000000000000000000000008f9b51aa
LOG2    1,381                [Receiver] EntryPoint .Deposited(account =  0xdb916011bfd54f218d3ace15dcf3b05ac0c377cd ,totalDeposit = 9868578300000)
SLOAD   100              EntryPoint [0xc958a2d2939f05a2e124eea36d16240c8363a5e0544a2838032b3a58e9cb66d3 = 0x000000000000000000000000000000000000000000000000000008f9b51aa060]
SLOAD   100              EntryPoint [0xc958a2d2939f05a2e124eea36d16240c8363a5e0544a2838032b3a58e9cb66d3 = 0x000000000000000000000000000000000000000000000000000008f9b51aa060]
SSTORE  100              EntryPoint [0xc958a2d2939f05a2e124eea36d16240c8363a5e0544a2838032b3a58e9cb66d3 = 0x000000000000000000000000000000000000000000000000000008f9b51aa060 ->
JUMP    5,256        ⌄ [Receiver] EntryPoint ._validateAndUpdateNonce(sender =  0xdb916011bfd54f218d3ace15dcf3b05ac0c377cd ,nonce = 2) => (false)
SLOAD   2,100            EntryPoint [0xe9a0acf53a3df7adb82c7cab7c6050a04523c16e63912e8397b6e416243073ba = 0x0000000000000000000000000000000000000000000000000000000000000003]
SSTORE  2,900            EntryPoint [0xe9a0acf53a3df7adb82c7cab7c6050a04523c16e63912e8397b6e416243073ba = 0x0000000000000000000000000000000000000000000000000000000000000003 ->
```

AA Developer Tasks

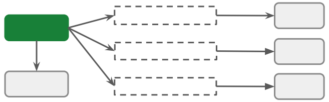




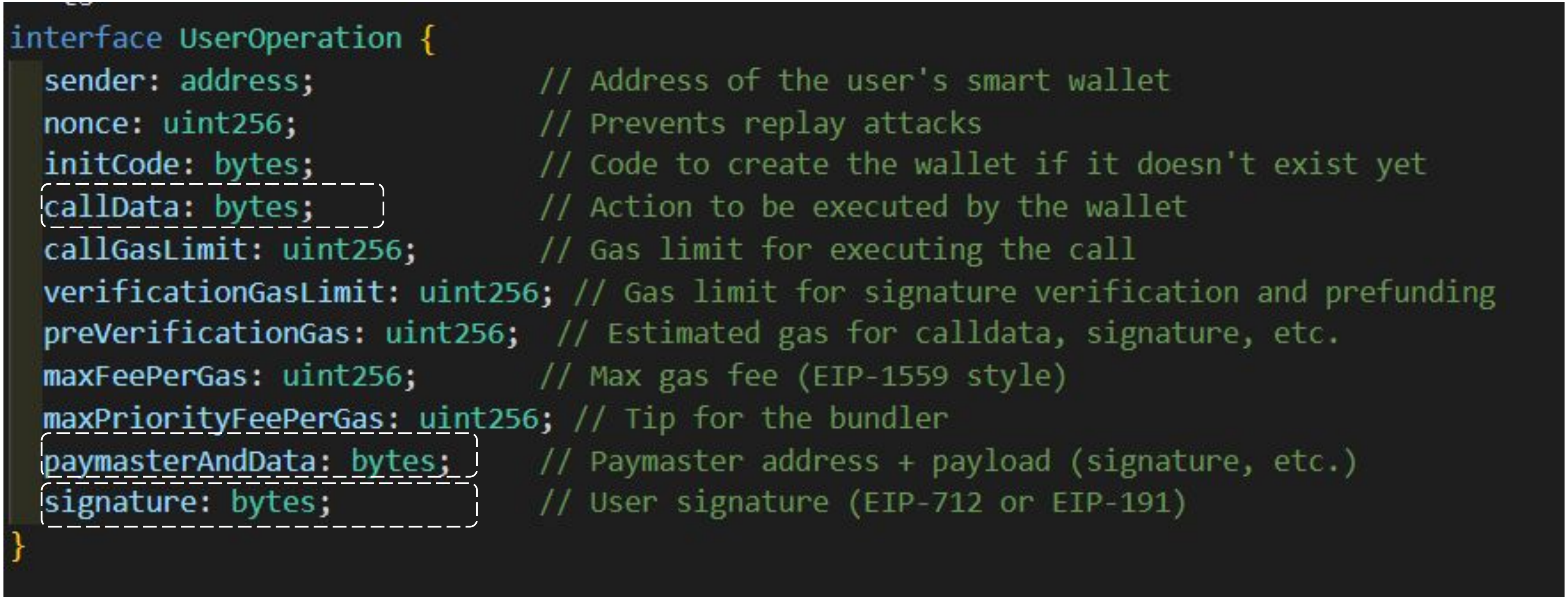
Bkd: User Operation



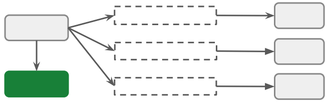
```
interface UserOperation {
  sender: address;             // Address of the user's smart wallet
  nonce: uint256;              // Prevents replay attacks
  initCode: bytes;             // Code to create the wallet if it doesn't exist yet
  callData: bytes;             // Action to be executed by the wallet
  callGasLimit: uint256;       // Gas limit for executing the call
  verificationGasLimit: uint256; // Gas limit for signature verification and prefunding
  preVerificationGas: uint256;  // Estimated gas for calldata, signature, etc.
  maxFeePerGas: uint256;       // Max gas fee (EIP-1559 style)
  maxPriorityFeePerGas: uint256; // Tip for the bundler
  paymasterAndData: bytes;     // Paymaster address + payload (signature, etc.)
  signature: bytes;            // User signature (EIP-712 or EIP-191)
}
```



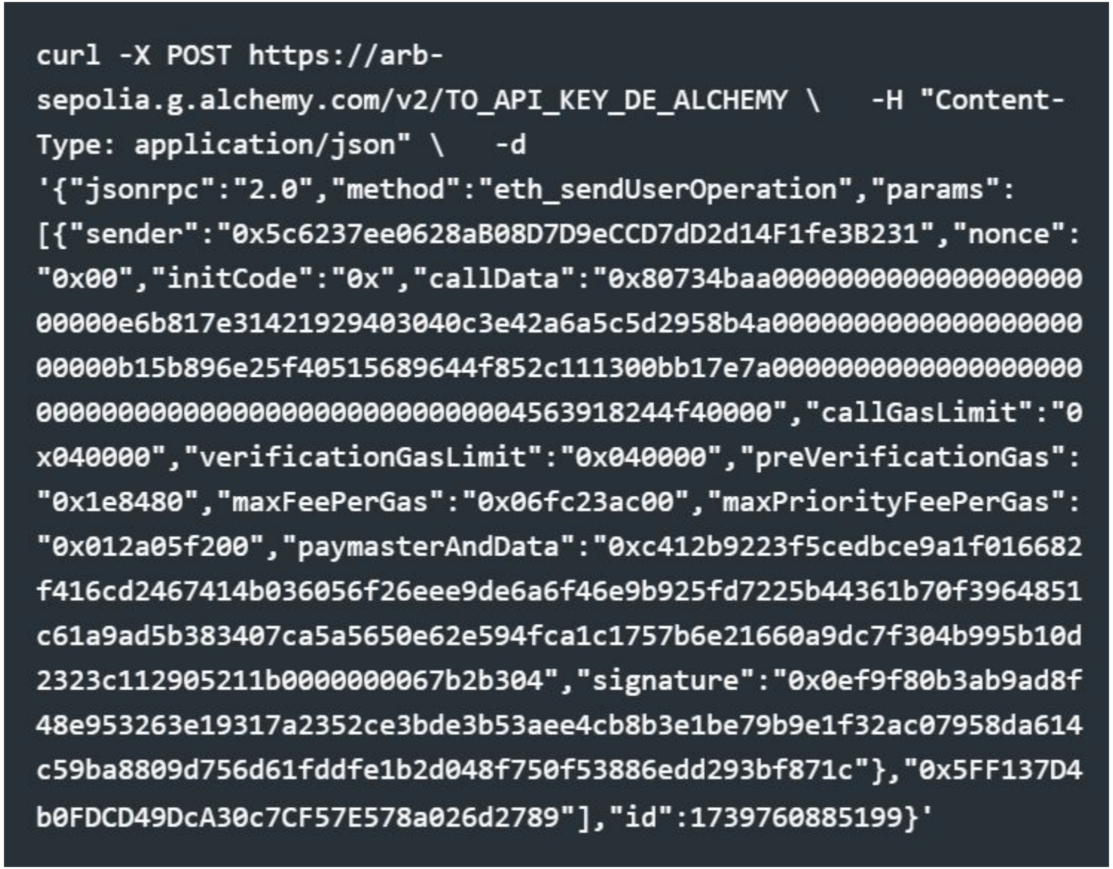
Bkd: User Operation



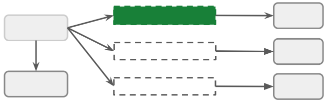
```
interface UserOperation {
  sender: address;                // Address of the user's smart wallet
  nonce: uint256;                 // Prevents replay attacks
  initCode: bytes;                // Code to create the wallet if it doesn't exist yet
  callData: bytes;                // Action to be executed by the wallet
  callGasLimit: uint256;          // Gas limit for executing the call
  verificationGasLimit: uint256; // Gas limit for signature verification and prefunding
  preVerificationGas: uint256;  // Estimated gas for calldata, signature, etc.
  maxFeePerGas: uint256;          // Max gas fee (EIP-1559 style)
  maxPriorityFeePerGas: uint256; // Tip for the bundler
  paymasterAndData: bytes;        // Paymaster address + payload (signature, etc.)
  signature: bytes;               // User signature (EIP-712 or EIP-191)
}
```



Bkd: Post User Operation



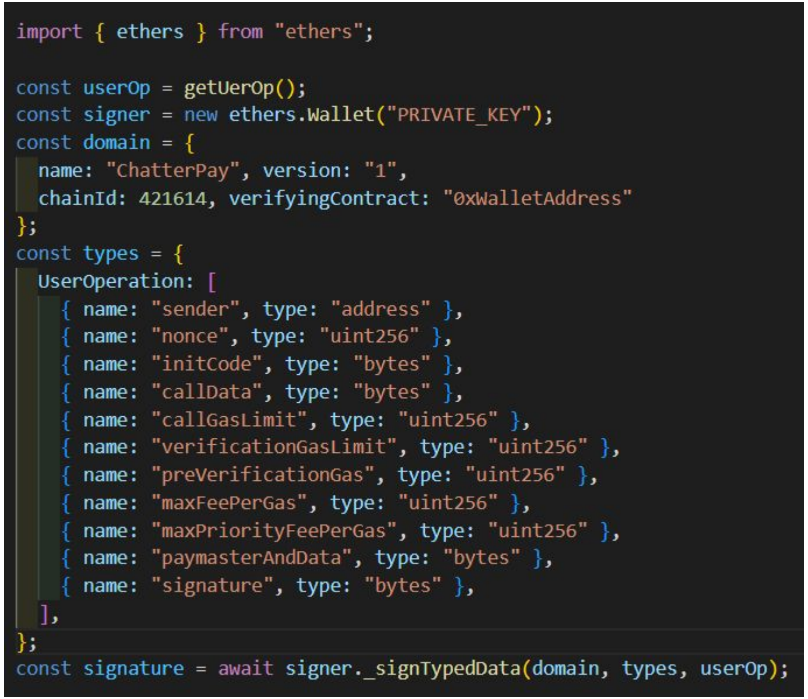
```
curl -X POST https://arb-
sepolia.g.alchemy.com/v2/TO_API_KEY_DE_ALCHEMY \   -H "Content-
Type: application/json" \   -d
'{"jsonrpc":"2.0","method":"eth_sendUserOperation","params":
[{"sender":"0x5c6237ee0628aB08D7D9eCCD7dD2d14F1fe3B231","nonce":
"0x00","initCode":"0x","callData":"0x80734baa0000000000000000000
00000e6b817e31421929403040c3e42a6a5c5d2958b4a0000000000000000000
00000b15b896e25f40515689644f852c111300bb17e7a0000000000000000000
000000000000000000000000000000004563918244f40000","callGasLimit":"0
x040000","verificationGasLimit":"0x040000","preVerificationGas":
"0x1e8480","maxFeePerGas":"0x06fc23ac00","maxPriorityFeePerGas":
"0x012a05f200","paymasterAndData":"0xc412b9223f5cedbce9a1f016682
f416cd2467414b036056f26eee9de6a6f46e9b925fd7225b44361b70f3964851
c61a9ad5b383407ca5a5650e62e594fca1c1757b6e21660a9dc7f304b995b10d
2323c112905211b0000000067b2b304","signature":"0x0ef9f80b3ab9ad8f
48e953263e19317a2352ce3bde3b53aee4cb8b3e1be79b9e1f32ac07958da614
c59ba8809d756d61fddfe1b2d048f750f53886edd293bf871c"},"0x5FF137D4
b0FDCD49DcA30c7CF57E578a026d2789"],"id":1739760885199}'
```



Bkd: Signature

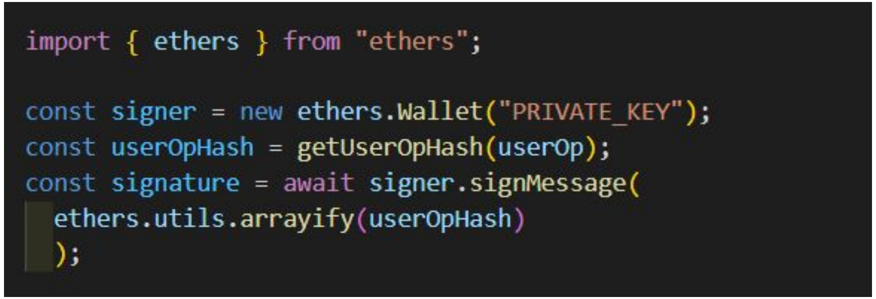
This field contains the Signature.

EIP-712 (structured signatures)

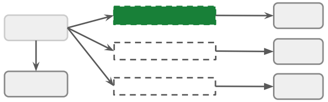


```
import { ethers } from "ethers";

const userOp = getUerOp();
const signer = new ethers.Wallet("PRIVATE_KEY");
const domain = {
  name: "ChatterPay", version: "1",
  chainId: 421614, verifyingContract: "0xWalletAddress"
};
const types = {
  UserOperation: [
    { name: "sender", type: "address" },
    { name: "nonce", type: "uint256" },
    { name: "initCode", type: "bytes" },
    { name: "callData", type: "bytes" },
    { name: "callGasLimit", type: "uint256" },
    { name: "verificationGasLimit", type: "uint256" },
    { name: "preVerificationGas", type: "uint256" },
    { name: "maxFeePerGas", type: "uint256" },
    { name: "maxPriorityFeePerGas", type: "uint256" },
    { name: "paymasterAndData", type: "bytes" },
    { name: "signature", type: "bytes" },
  ],
};
const signature = await signer._signTypedData(domain, types, userOp);
```

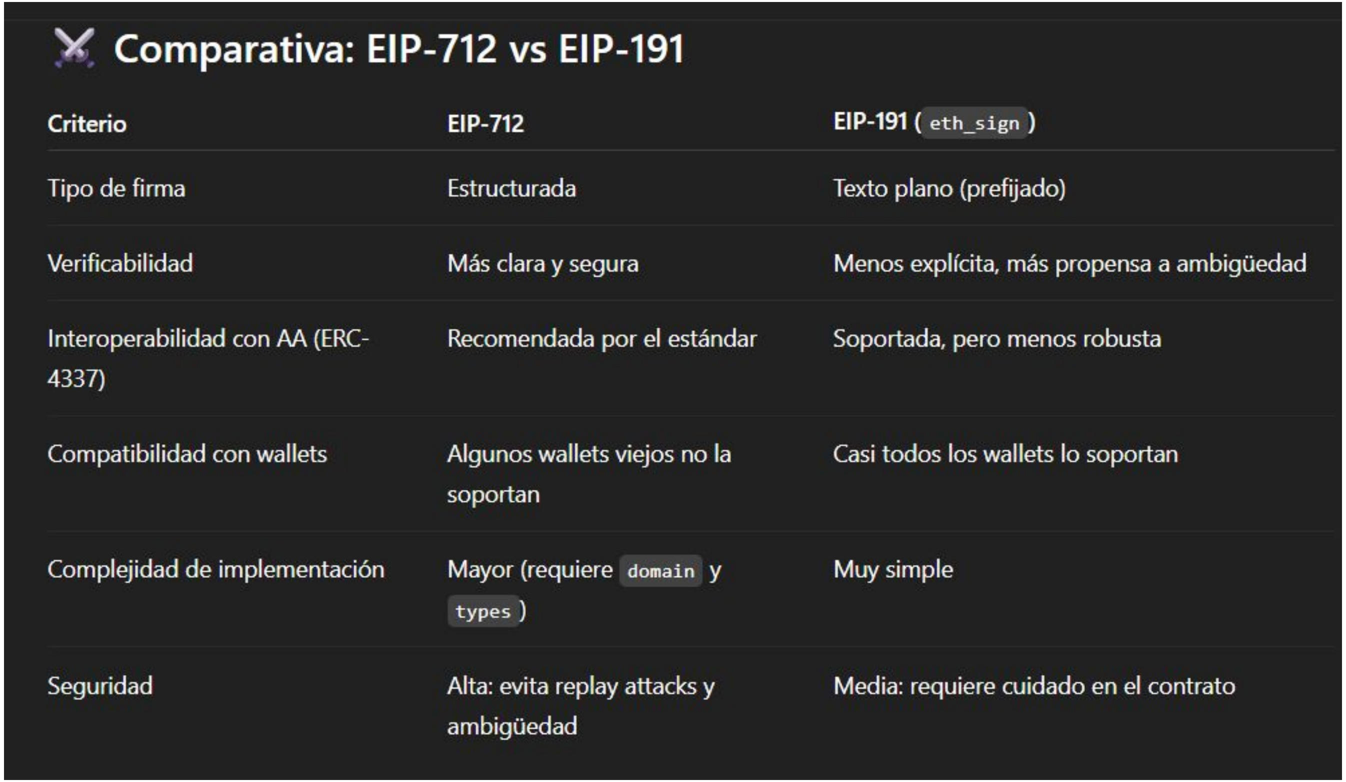
EIP-191



```
import { ethers } from "ethers";

const signer = new ethers.Wallet("PRIVATE_KEY");
const userOpHash = getUserOpHash(userOp);
const signature = await signer.signMessage(
  ethers.utils.arrayify(userOpHash)
);
```

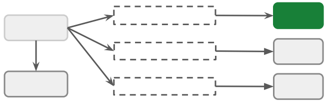


EIP-712 vs EIP-191

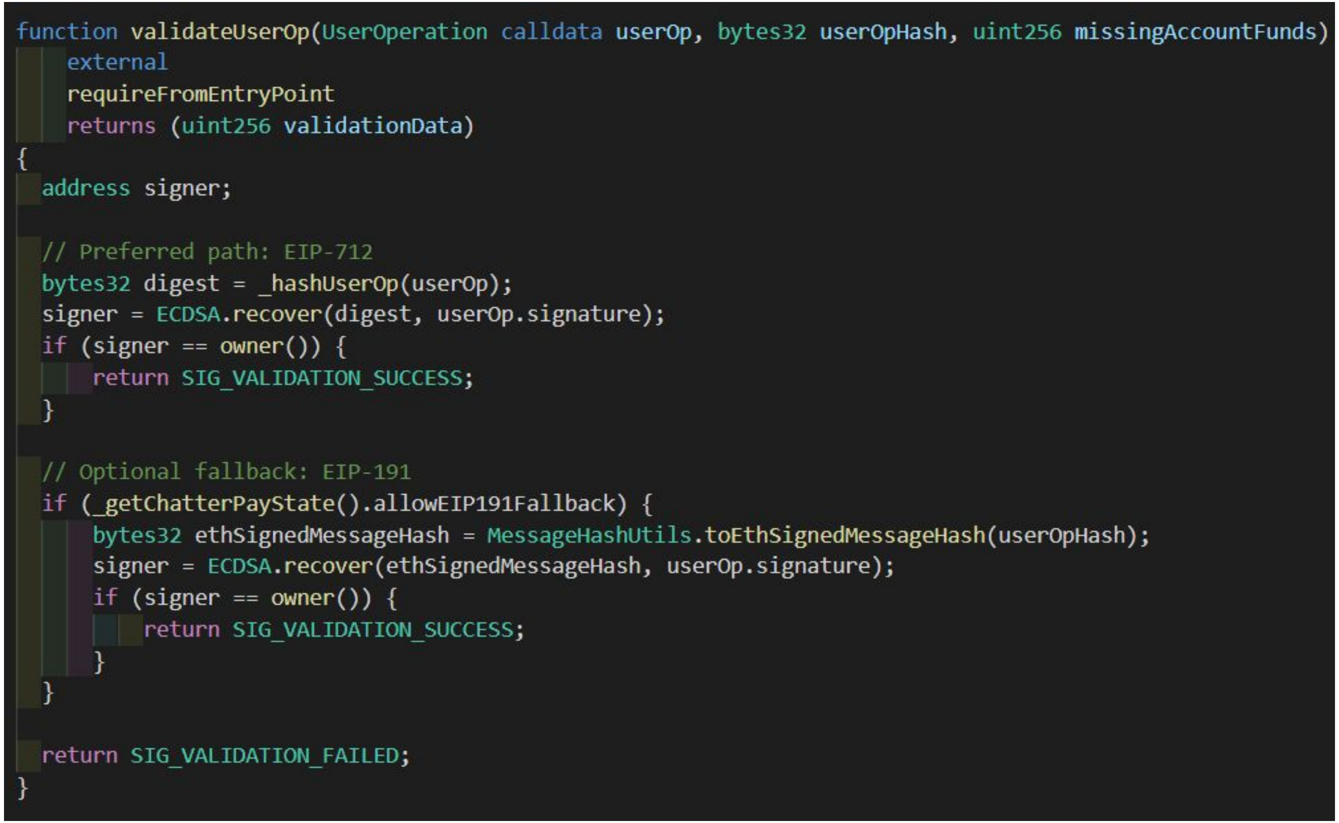


⚔️ Comparativa: EIP-712 vs EIP-191

Criterio	EIP-712	EIP-191 (`eth_sign`)
Tipo de firma	Estructurada	Texto plano (prefijado)
Verificabilidad	Más clara y segura	Menos explícita, más propensa a ambigüedad
Interoperabilidad con AA (ERC-4337)	Recomendada por el estándar	Soportada, pero menos robusta
Compatibilidad con wallets	Algunos wallets viejos no la soportan	Casi todos los wallets lo soportan
Complejidad de implementación	Mayor (requiere `domain` y `types`)	Muy simple
Seguridad	Alta: evita replay attacks y ambigüedad	Media: requiere cuidado en el contrato



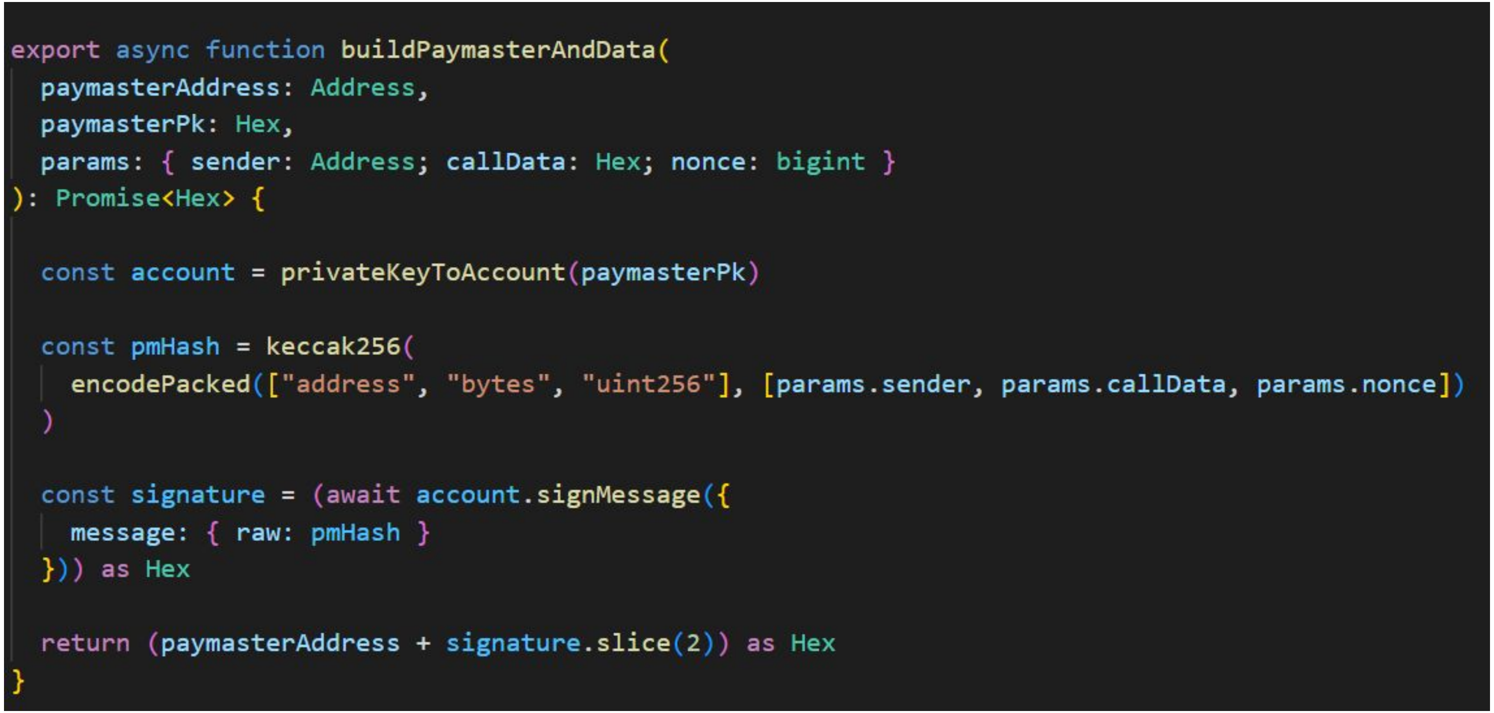
SC: Smart Wallet



```
function validateUserOp(UserOperation calldata userOp, bytes32 userOpHash, uint256 missingAccountFunds)
    external
    requireFromEntryPoint
    returns (uint256 validationData)
{
  address signer;

  // Preferred path: EIP-712
  bytes32 digest = _hashUserOp(userOp);
  signer = ECDSA.recover(digest, userOp.signature);
  if (signer == owner()) {
      return SIG_VALIDATION_SUCCESS;
  }

  // Optional fallback: EIP-191
  if (_getChatterPayState().allowEIP191Fallback) {
      bytes32 ethSignedMessageHash = MessageHashUtils.toEthSignedMessageHash(userOpHash);
      signer = ECDSA.recover(ethSignedMessageHash, userOp.signature);
      if (signer == owner()) {
          return SIG_VALIDATION_SUCCESS;
      }
  }

  return SIG_VALIDATION_FAILED;
}
```

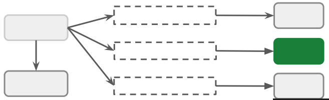
Bkd: PaymasterAndData

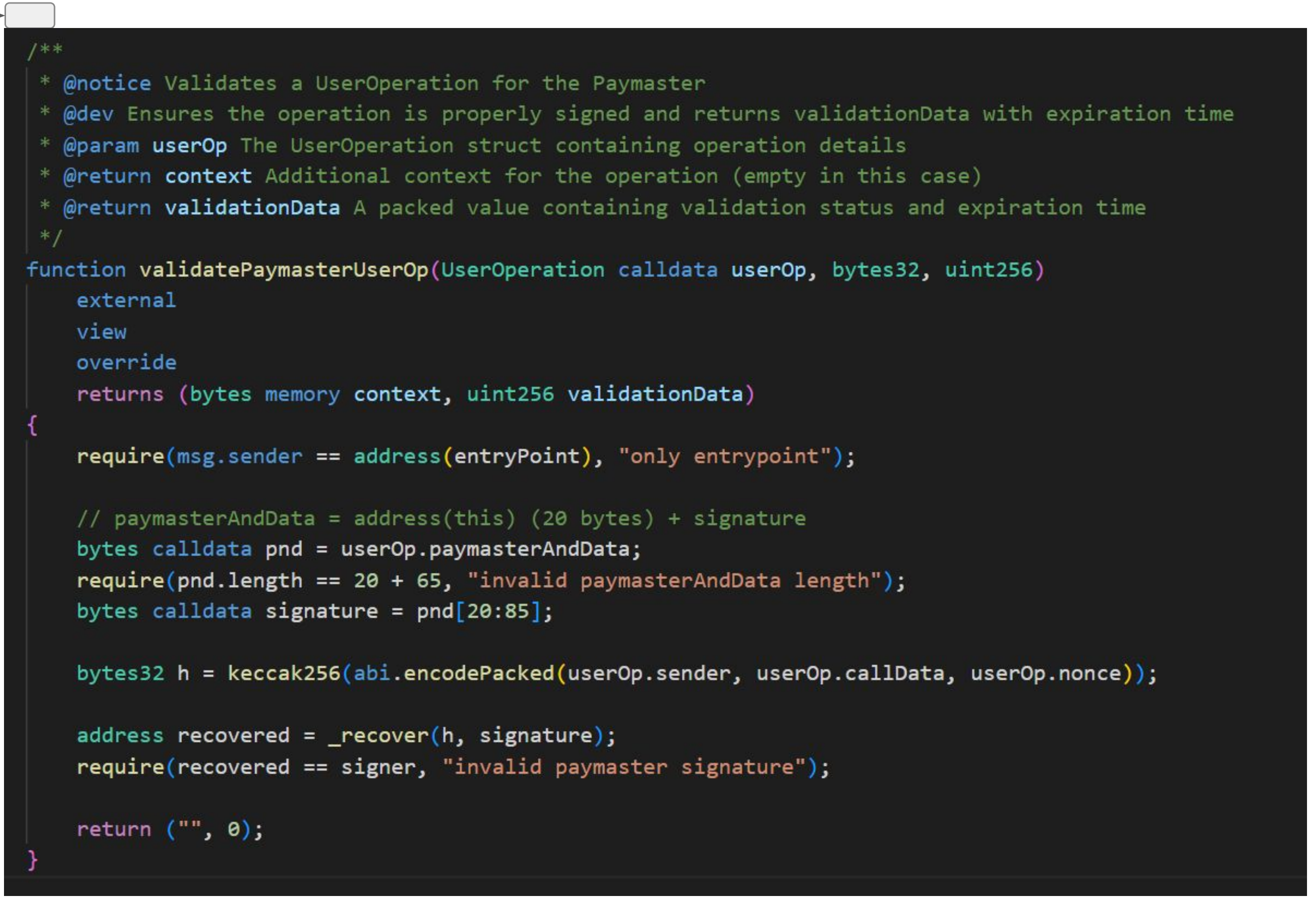
This field contains the Paymaster address and additional data (such as a signature) that enable it to sponsor the operation.



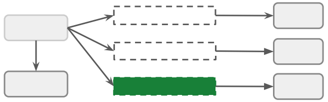
```
export async function buildPaymasterAndData(
  paymasterAddress: Address,
  paymasterPk: Hex,
  params: { sender: Address; callData: Hex; nonce: bigint }
): Promise<Hex> {

  const account = privateKeyToAccount(paymasterPk)

  const pmHash = keccak256(
    encodePacked(["address", "bytes", "uint256"], [params.sender, params.callData, params.nonce])
  )

  const signature = (await account.signMessage({
    message: { raw: pmHash }
  })) as Hex

  return (paymasterAddress + signature.slice(2)) as Hex
}
```

SC: Paymaster



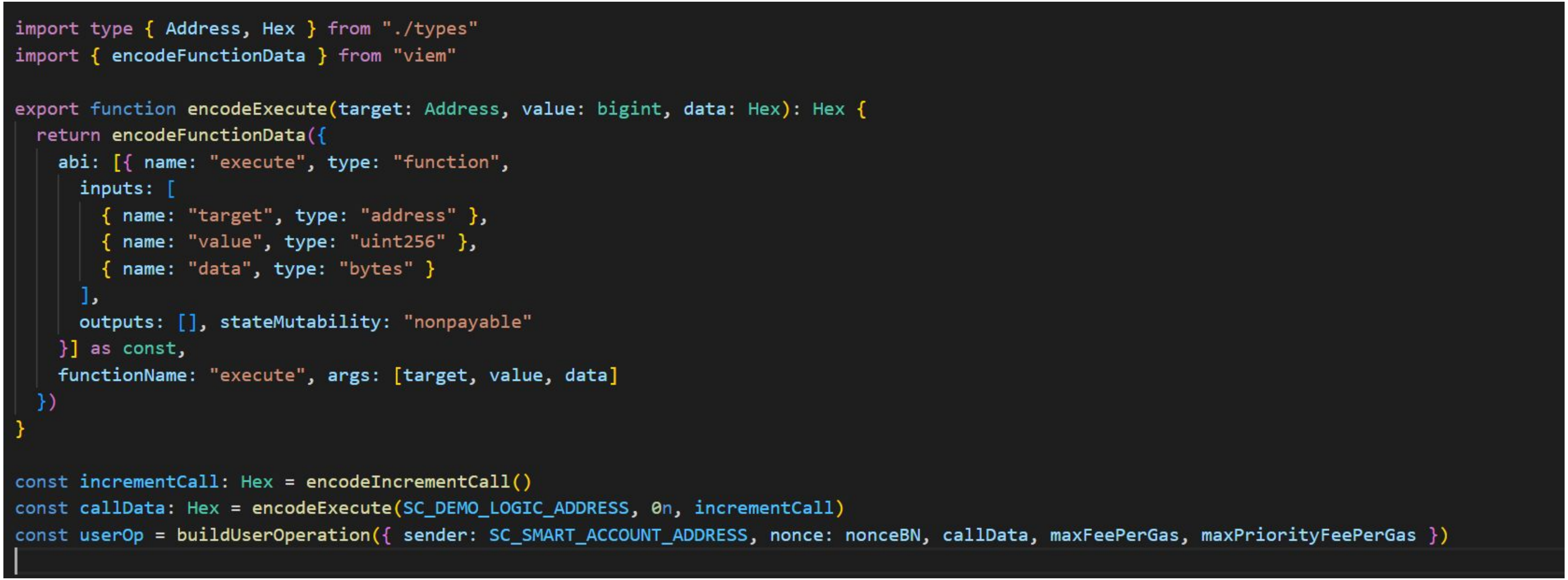


```
/**
 * @notice Validates a UserOperation for the Paymaster
 * @dev Ensures the operation is properly signed and returns validationData with expiration time
 * @param userOp The UserOperation struct containing operation details
 * @return context Additional context for the operation (empty in this case)
 * @return validationData A packed value containing validation status and expiration time
 */
function validatePaymasterUserOp(UserOperation calldata userOp, bytes32, uint256)
    external
    view
    override
    returns (bytes memory context, uint256 validationData)
{
    require(msg.sender == address(entryPoint), "only entrypoint");

    // paymasterAndData = address(this) (20 bytes) + signature
    bytes calldata pnd = userOp.paymasterAndData;
    require(pnd.length == 20 + 65, "invalid paymasterAndData length");
    bytes calldata signature = pnd[20:85];

    bytes32 h = keccak256(abi.encodePacked(userOp.sender, userOp.callData, userOp.nonce));

    address recovered = _recover(h, signature);
    require(recovered == signer, "invalid paymaster signature");

    return ("", 0);
}
```

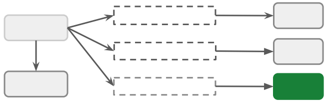


User Operation: Call Data

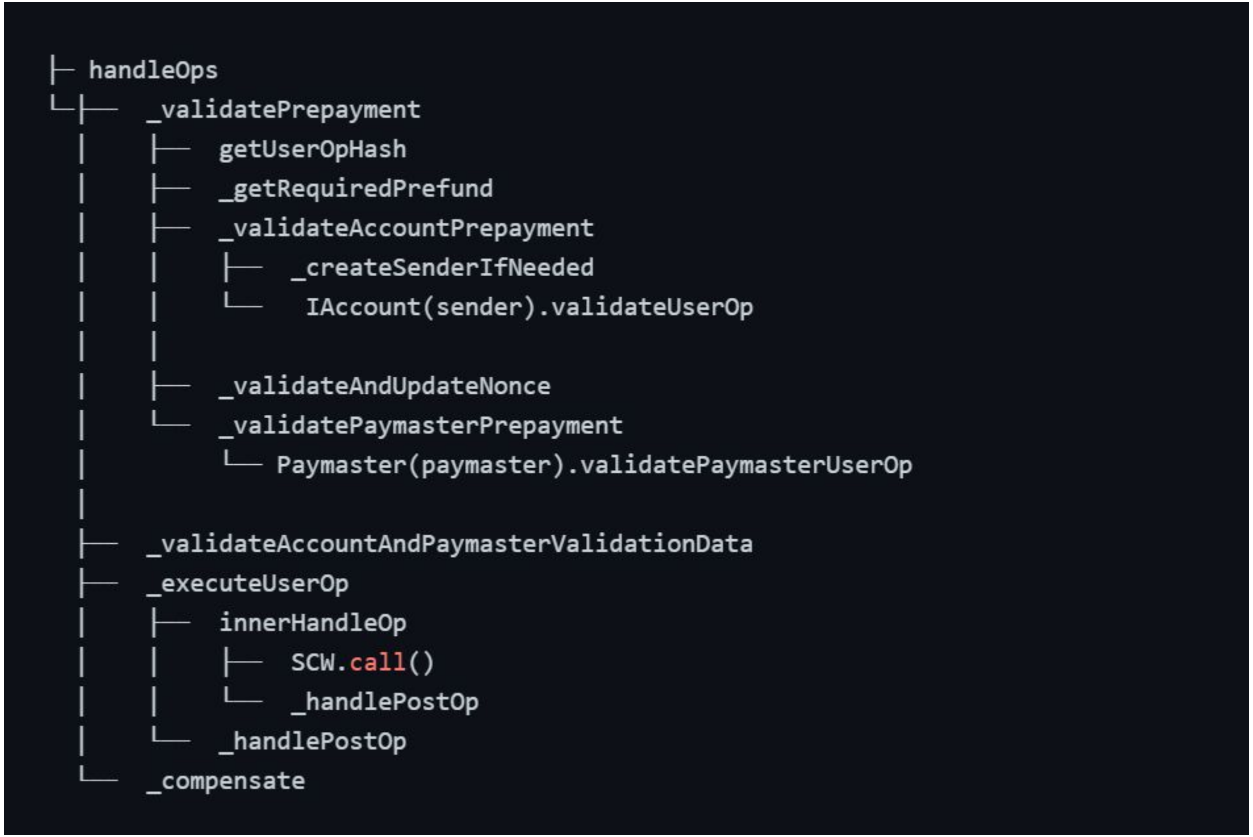
This field represents the action that the smart wallet will execute.



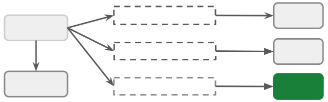
```
import type { Address, Hex } from "./types"
import { encodeFunctionData } from "viem"

export function encodeExecute(target: Address, value: bigint, data: Hex): Hex {
  return encodeFunctionData({
    abi: [{ name: "execute", type: "function",
      inputs: [
        { name: "target", type: "address" },
        { name: "value", type: "uint256" },
        { name: "data", type: "bytes" }
      ],
      outputs: [], stateMutability: "nonpayable"
    }] as const,
    functionName: "execute", args: [target, value, data]
  })
}

const incrementCall: Hex = encodeIncrementCall()
const callData: Hex = encodeExecute(SC_DEMO_LOGIC_ADDRESS, 0n, incrementCall)
const userOp = buildUserOperation({ sender: SC_SMART_ACCOUNT_ADDRESS, nonce: nonceBN, callData, maxFeePerGas, maxPriorityFeePerGas })
```



Entry Point Execution

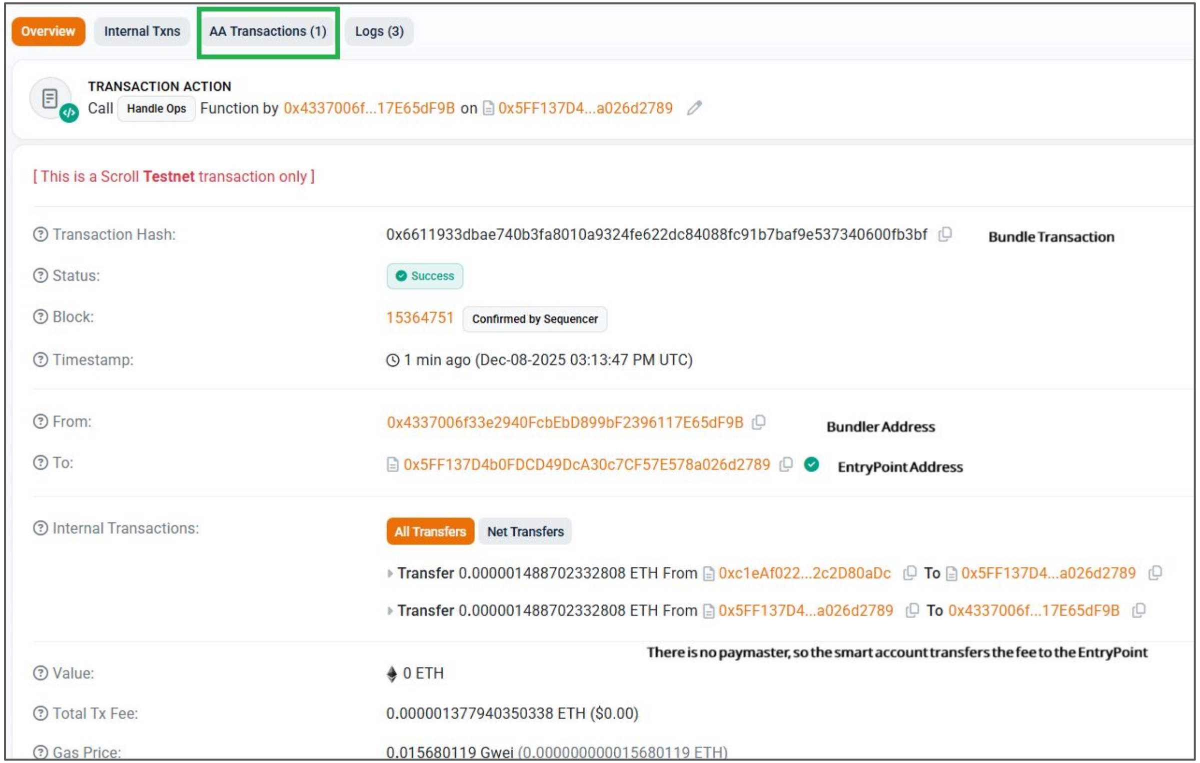


```
├─ handleOps
└─├──  _validatePrepayment
  │    ├──  getUserOpHash
  │    ├──  _getRequiredPrefund
  │    ├──  _validateAccountPrepayment
  │    │    ├──  _createSenderIfNeeded
  │    │    └──   IAccount(sender).validateUserOp
  │    │
  │    ├──  _validateAndUpdateNonce
  │    └──  _validatePaymasterPrepayment
  │         └── Paymaster(paymaster).validatePaymasterUserOp
  │
  ├──  _validateAccountAndPaymasterValidationData
  ├──  _executeUserOp
  │    ├──  innerHandleOp
  │    │    ├──  SCW.call()
  │    │    └──  _handlePostOp
  │    └──  _handlePostOp
  └──  _compensate
```



User Operation: Tx

https://sepolia.scrollscan.com/tx/0x6611933dbae740b3fa8010a9324fe622dc84088fc91b7baf9e537340600fb3bf



TRANSACTION ACTION

Call Handle Ops Function by 0x4337006f...17E65dF9B on 0x5FF137D4...a026d2789

[This is a Scroll **Testnet** transaction only]

Transaction Hash: 0x6611933dbae740b3fa8010a9324fe622dc84088fc91b7baf9e537340600fb3bf **Bundle Transaction**

Status: Success

Block: 15364751 Confirmed by Sequencer

Timestamp: 1 min ago (Dec-08-2025 03:13:47 PM UTC)

From: 0x4337006f33e2940FcbEbD899bF2396117E65dF9B **Bundler Address**

To: 0x5FF137D4b0FDCD49DcA30c7CF57E578a026d2789 ✓ **EntryPoint Address**

Internal Transactions: All Transfers Net Transfers

Transfer 0.000001488702332808 ETH From 0xc1eAf022...2c2D80aDc To 0x5FF137D4...a026d2789

Transfer 0.000001488702332808 ETH From 0x5FF137D4...a026d2789 To 0x4337006f...17E65dF9B

There is no paymaster, so the smart account transfers the fee to the EntryPoint

Value: 0 ETH

Total Tx Fee: 0.000001377940350338 ETH ($0.00)

Gas Price: 0.015680119 Gwei (0.000000000015680119 ETH)

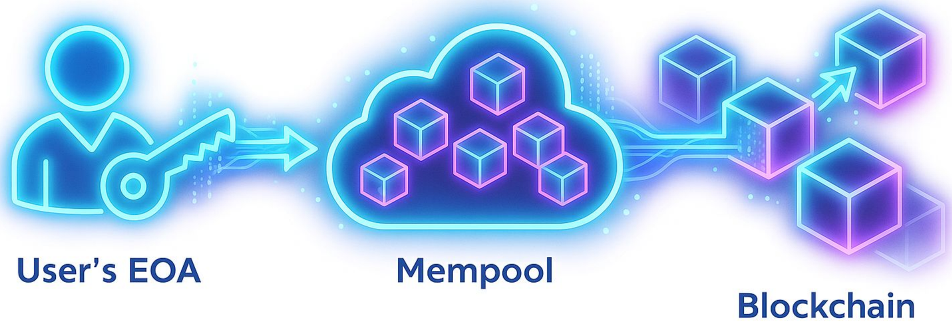
Q&A

have any **Questions** in mind?

Question 1

Who signs and sends a traditional transaction to blockchain?

The **user's EOA** signs the transaction, and the **user's EOA** sends it directly to the mempool.

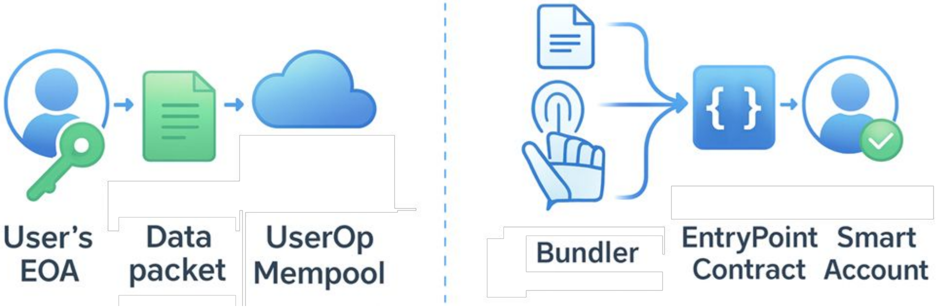


Question 2

In an AA transaction, who signs and sends the transaction to the blockchain? And the UserOperation?

The **user's EOA** (the smart account's owner) signs the UserOperation **off-chain** (that signature goes into the signature field). The smart account validates that signature on-chain when the EntryPoint executes it.

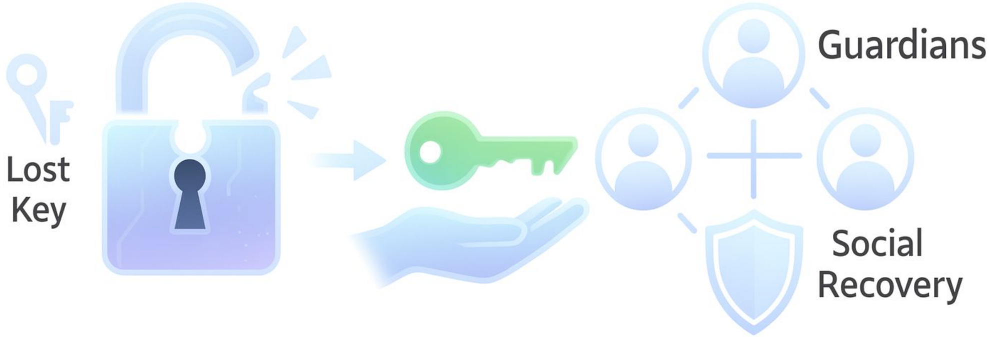
The **on-chain** transaction is sent by the **bundler's EOA**, not the user's.



Question 3

What happens if I lose the EOA that controls my smart account? Can I replace it?

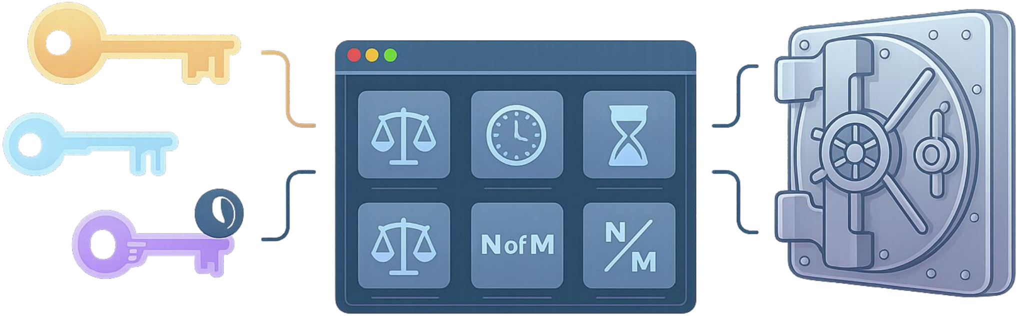
If the **smart account supports it** (social recovery, owner rotation, guardians), **then yes**.
If it doesn't include that logic, then no.



Question 4

Can the smart account have multiple signers?

Yes, absolutely. **The logic is whatever you define**.
It can have one, many, weighted rules, time-locks, whatever you need.

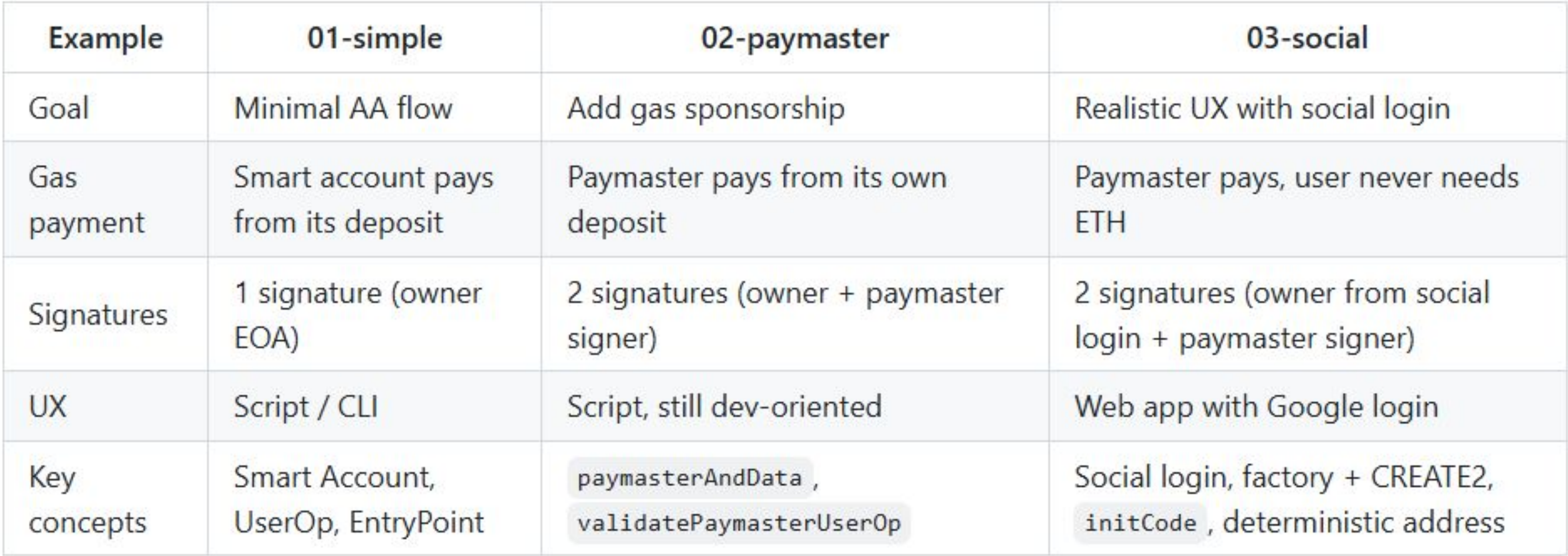


Demo

A Practical glimpse into code's power

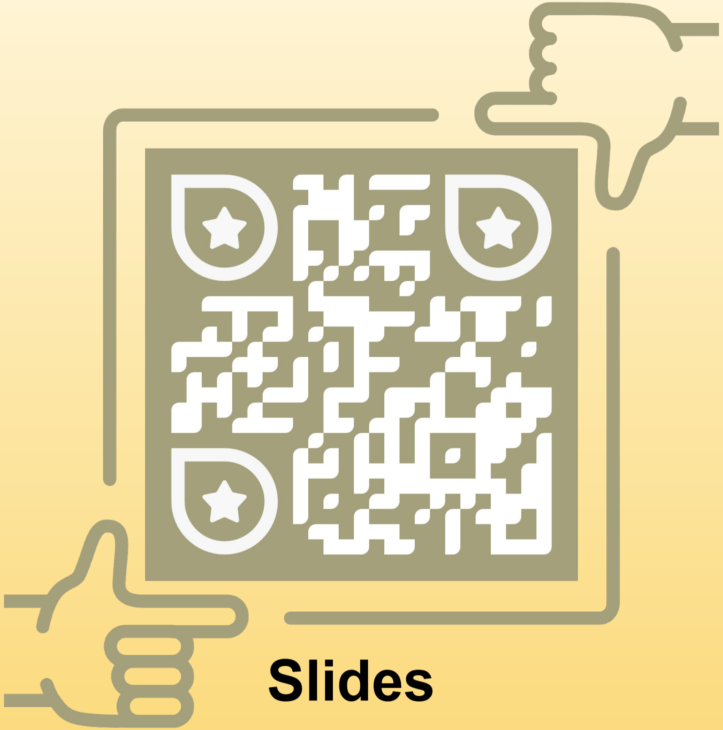
github.com/dappsar/scroll-demo-20251211

Examples



Example	01-simple	02-paymaster	03-social
Goal	Minimal AA flow	Add gas sponsorship	Realistic UX with social login
Gas payment	Smart account pays from its deposit	Paymaster pays from its own deposit	Paymaster pays, user never needs ETH
Signatures	1 signature (owner EOA)	2 signatures (owner + paymaster signer)	2 signatures (owner from social login + paymaster signer)
UX	Script / CLI	Script, still dev-oriented	Web app with Google login
Key concepts	Smart Account, UserOp, EntryPoint	`paymasterAndData` , `validatePaymasterUserOp`	Social login, factory + CREATE2, `initCode` , deterministic address

github.com/dappsar/scroll-demo-20251211



Slides

Thank you!

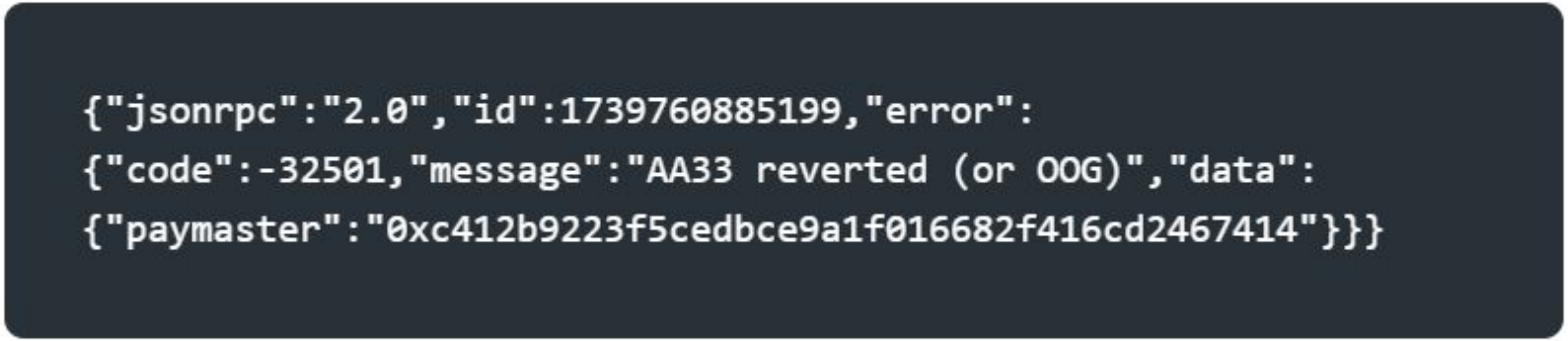
for helping make mass adoption of crypto a reality.

“Bit” Slapped

When **zeroes** and **ones fight back.**

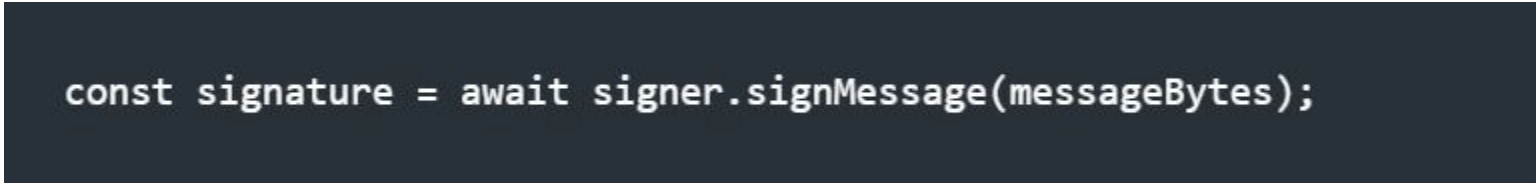
dappsar

P1 - Signature Mismatch: Problem

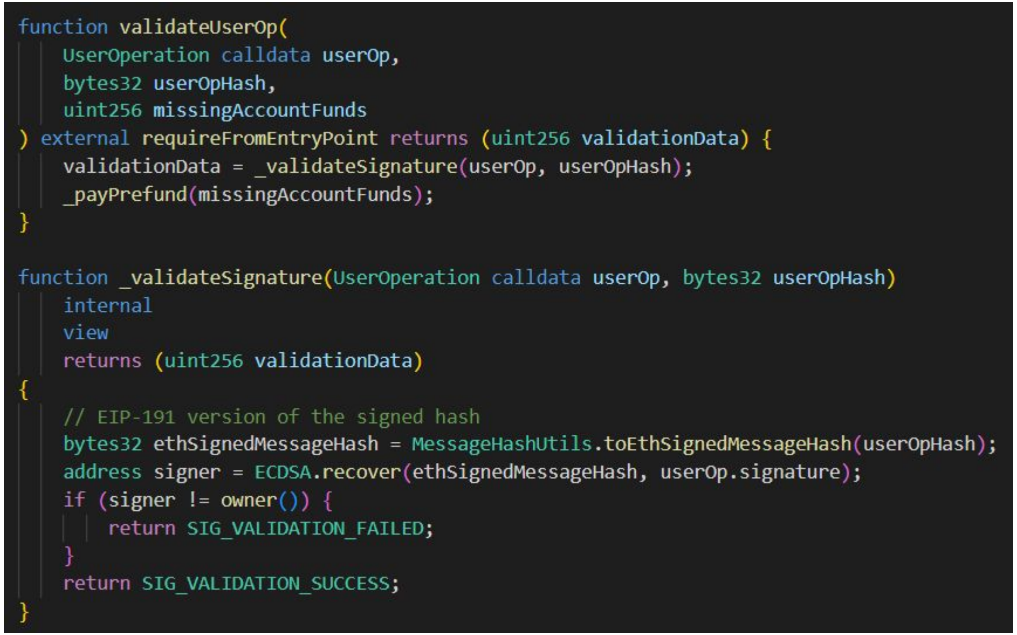


```
{"jsonrpc":"2.0","id":1739760885199,"error":
{"code":-32501,"message":"AA33 reverted (or OOG)","data":
{"paymaster":"0xc412b9223f5cedbce9a1f016682f416cd2467414"}}}
```

P1 - Signature Mismatch: Trace

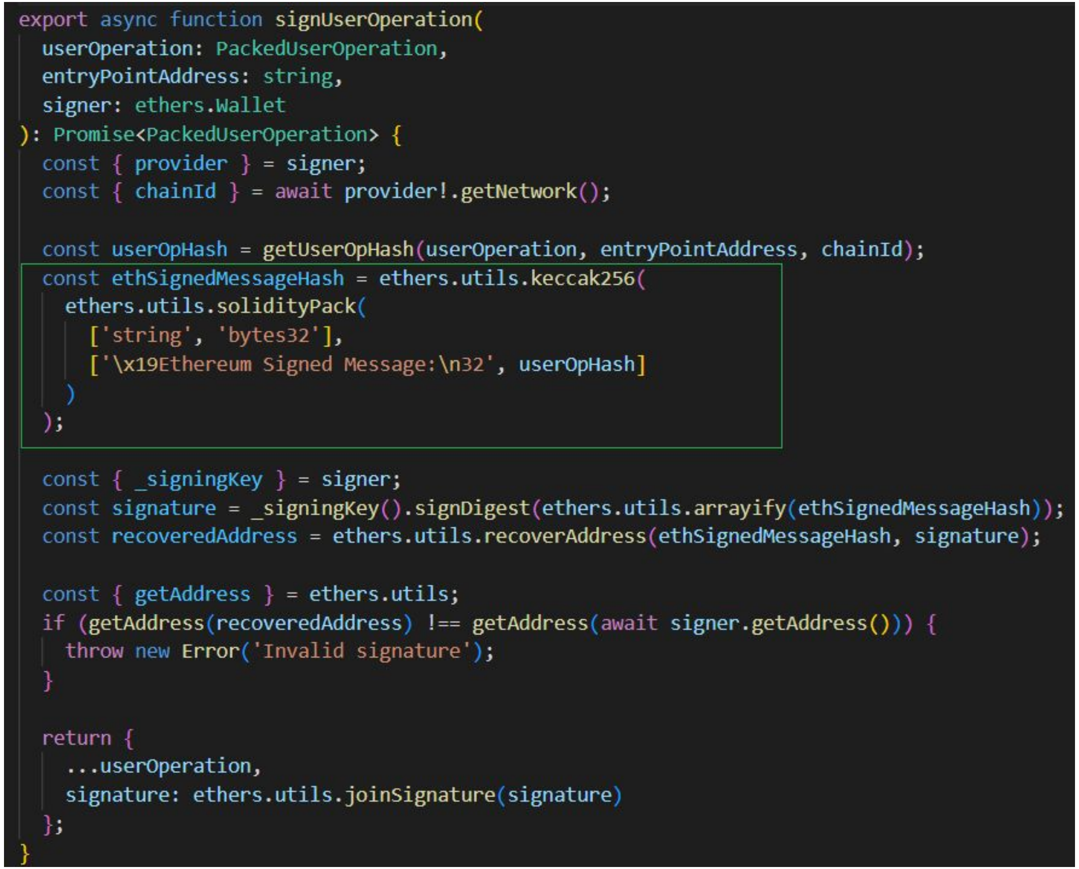


```
const signature = await signer.signMessage(messageBytes);
```



```
function validateUserOp(
    UserOperation calldata userOp,
    bytes32 userOpHash,
    uint256 missingAccountFunds
) external requireFromEntryPoint returns (uint256 validationData) {
    validationData = _validateSignature(userOp, userOpHash);
    _payPrefund(missingAccountFunds);
}

function _validateSignature(UserOperation calldata userOp, bytes32 userOpHash)
    internal
    view
    returns (uint256 validationData)
{
    // EIP-191 version of the signed hash
    bytes32 ethSignedMessageHash = MessageHashUtils.toEthSignedMessageHash(userOpHash);
    address signer = ECDSA.recover(ethSignedMessageHash, userOp.signature);
    if (signer != owner()) {
        return SIG_VALIDATION_FAILED;
    }
    return SIG_VALIDATION_SUCCESS;
}
```

P1 - Signature Mismatch: Trace

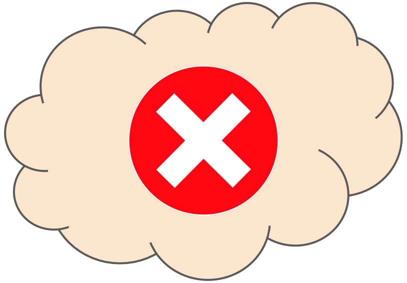


```
export async function signUserOperation(
  userOperation: PackedUserOperation,
  entryPointAddress: string,
  signer: ethers.Wallet
): Promise<PackedUserOperation> {
  const { provider } = signer;
  const { chainId } = await provider!.getNetwork();

  const userOpHash = getUserOpHash(userOperation, entryPointAddress, chainId);
  const ethSignedMessageHash = ethers.utils.keccak256(
    ethers.utils.solidityPack(
      ['string', 'bytes32'],
      ['\x19Ethereum Signed Message:\n32', userOpHash]
    )
  );

  const { _signingKey } = signer;
  const signature = _signingKey().signDigest(ethers.utils.arrayify(ethSignedMessageHash));
  const recoveredAddress = ethers.utils.recoverAddress(ethSignedMessageHash, signature);

  const { getAddress } = ethers.utils;
  if (getAddress(recoveredAddress) !== getAddress(await signer.getAddress())) {
    throw new Error('Invalid signature');
  }

  return {
    ...userOperation,
    signature: ethers.utils.joinSignature(signature)
  };
}
```

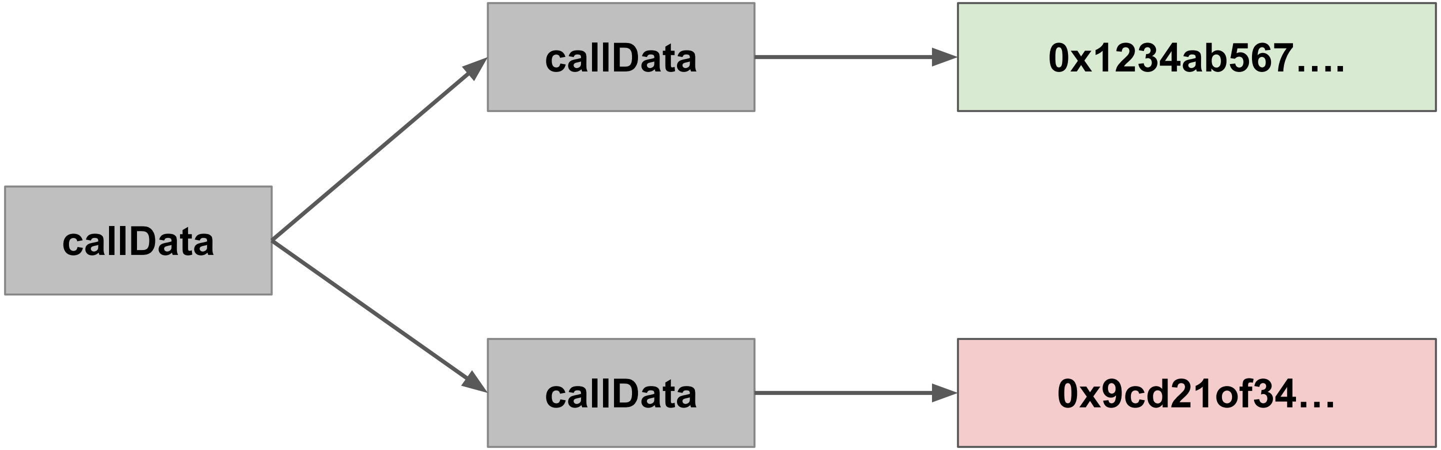
P2 - CallData Caos

LOCAL ENVIRONMENT

CLOUD ENVIRONMENT



P2 - CallData Caos



callData

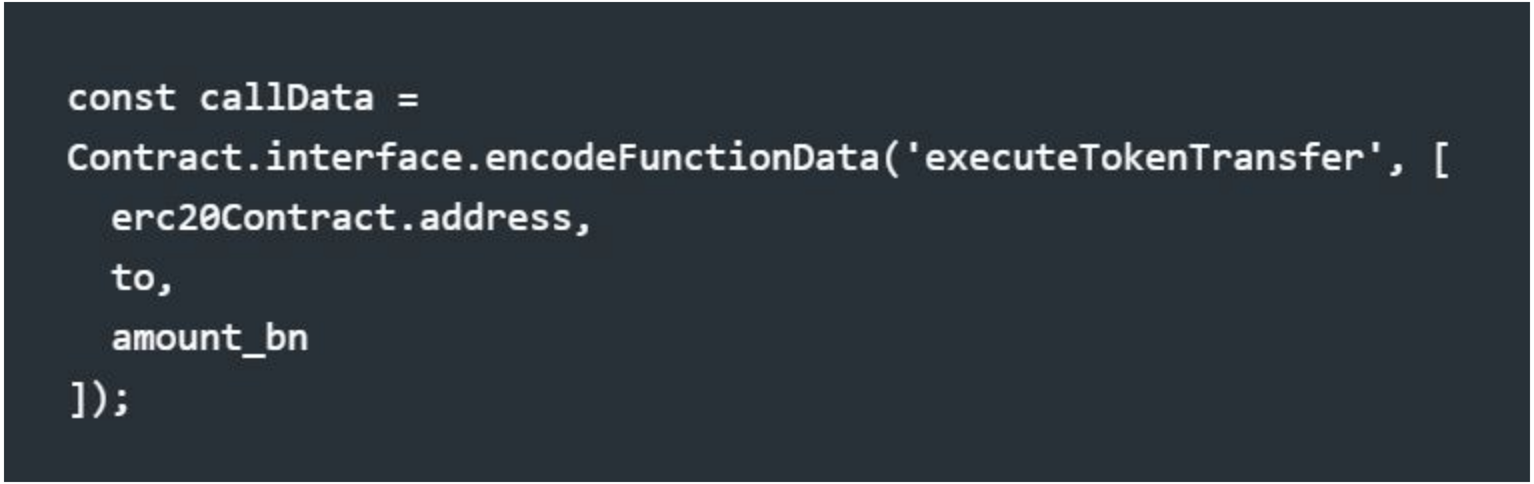
callData

0x1234ab567….

callData

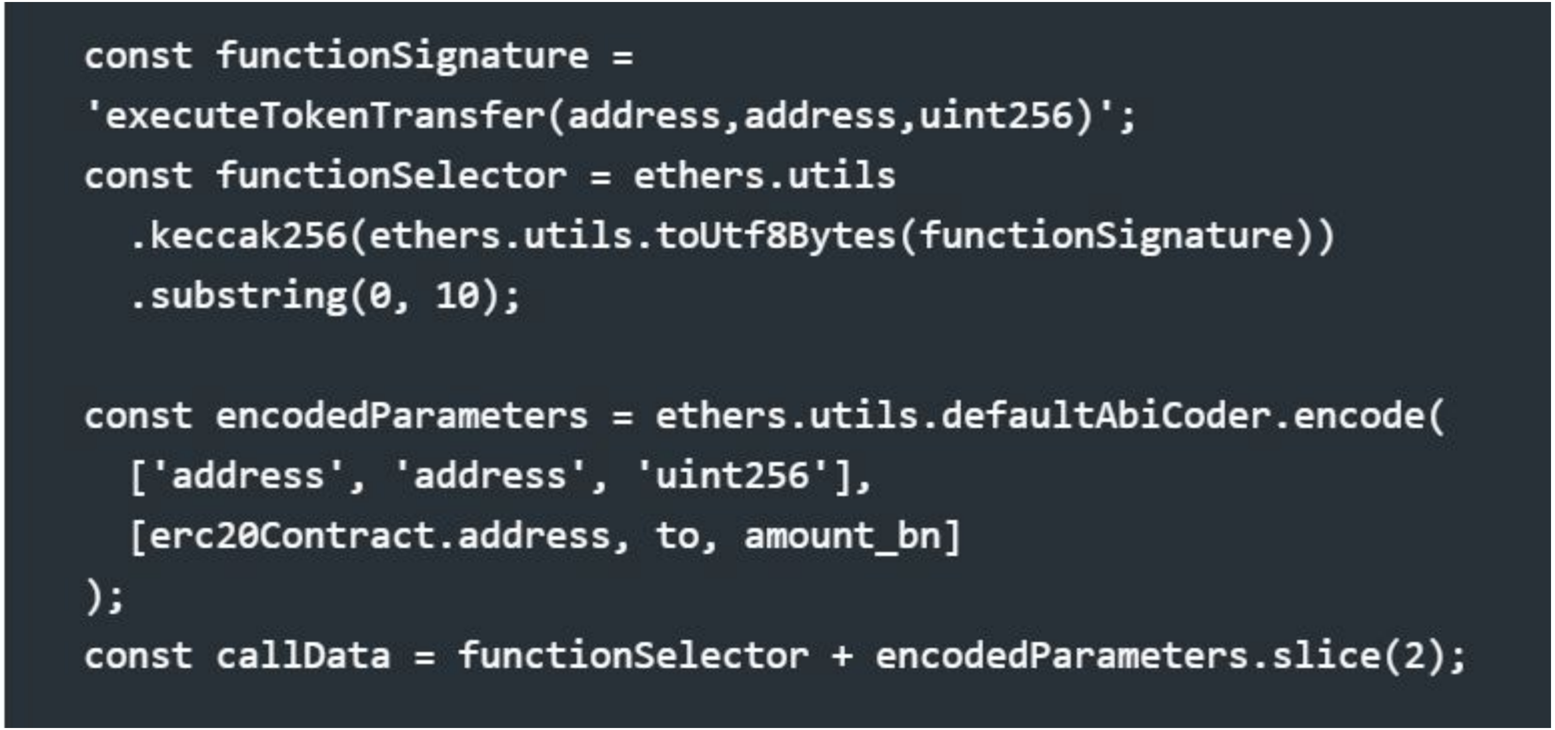
0x9cd21of34…

P2 - CallData Caos: Problem



```
const callData =
Contract.interface.encodeFunctionData('executeTokenTransfer', [
  erc20Contract.address,
  to,
  amount_bn
]);
```

P2 - CallData Caos: Solution



```
const functionSignature =
'executeTokenTransfer(address,address,uint256)';
const functionSelector = ethers.utils
  .keccak256(ethers.utils.toUtf8Bytes(functionSignature))
  .substring(0, 10);

const encodedParameters = ethers.utils.defaultAbiCoder.encode(
  ['address', 'address', 'uint256'],
  [erc20Contract.address, to, amount_bn]
);
const callData = functionSelector + encodedParameters.slice(2);
```

P3 - No Stake, No Party

entity stake/unstake delay too low

No stake, no trust

Bundlers require Paymasters to stake ETH as collateral to guarantee good behavior.