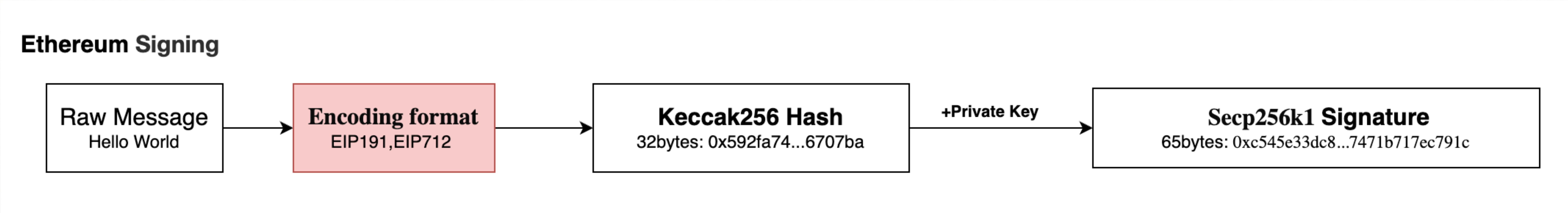
OPENSPACE × 登链社区

以太坊签名格式

七哥

https://x.com/0xqige

签名原理



Ethereum Signing

Raw Message
Hello World

Encoding format
EIP191,EIP712

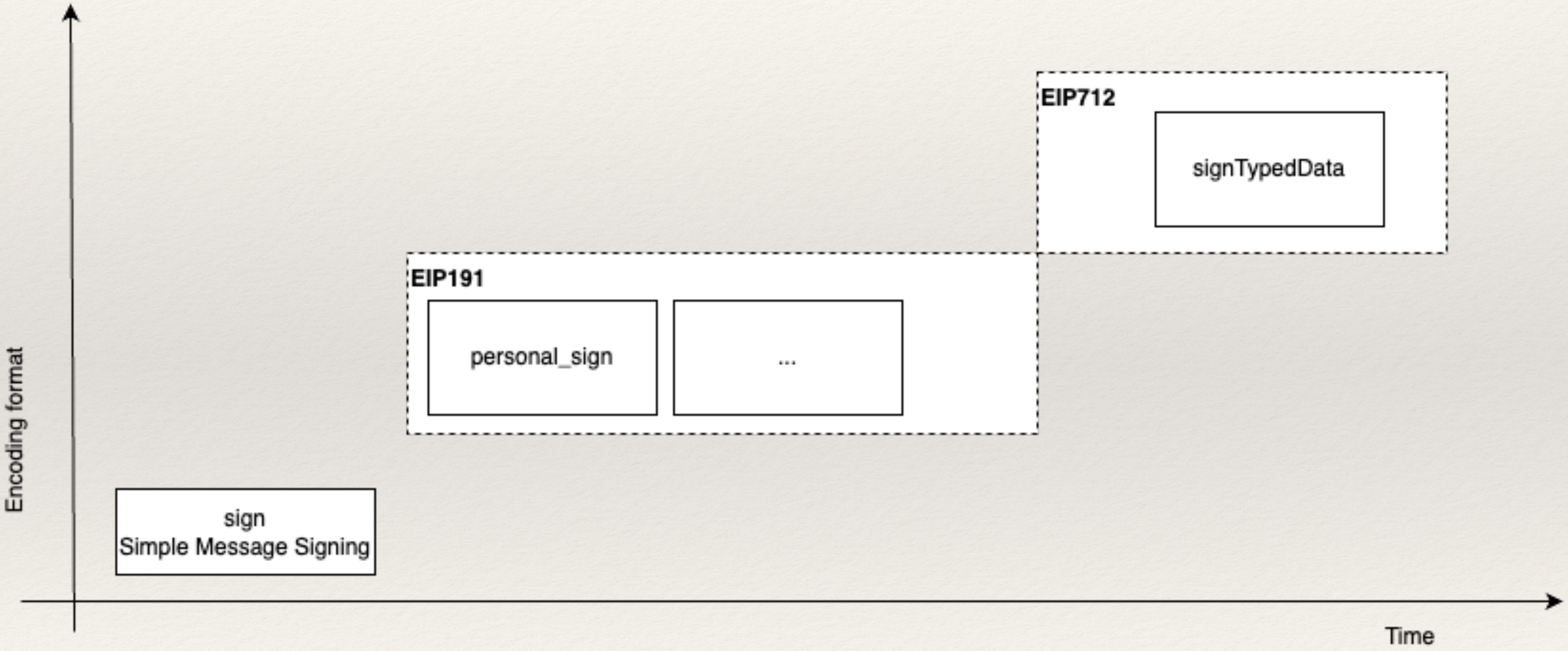
Keccak256 Hash
32bytes: 0x592fa74...6707ba

+Private Key

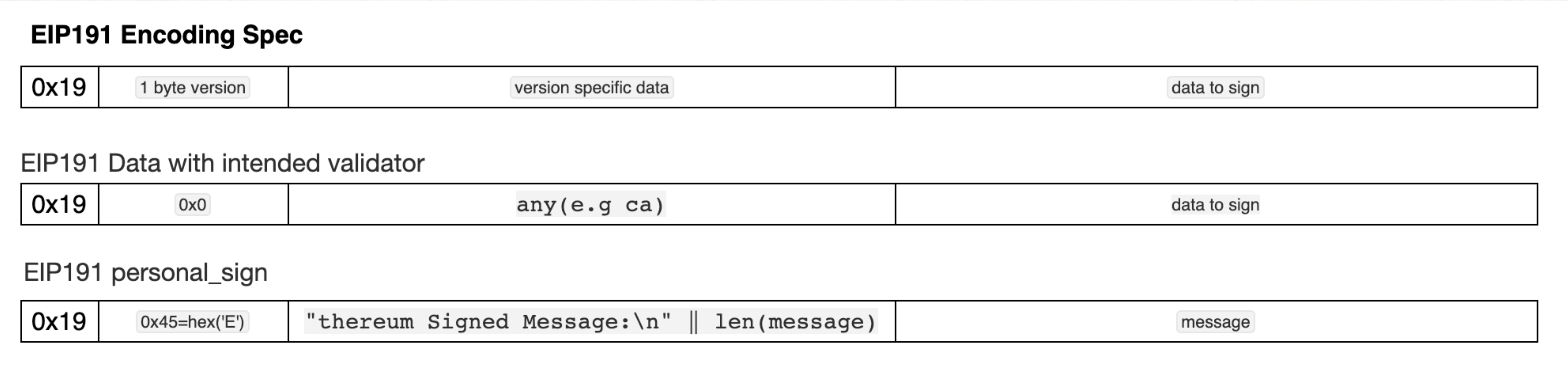
Secp256k1 Signature
65bytes: 0xc545e33dc8...7471b717ec791c

- 编码格式：EIP191，EIP712，……
- Keccak256 哈希算法，是 SHA-3 族：安全性高，灵活，速度快
- 椭圆曲线加密算法 Secp256k1：高效性，安全性，标准化

编码格式


Encoding format
Time
sign
Simple Message Signing
EIP191
personal_sign
...
EIP712
signTypedData

EIP191



EIP191 Encoding Spec

0x19	1 byte version	version specific data	data to sign

EIP191 Data with intended validator

0x19	0x0	any(e.g ca)	data to sign

EIP191 personal_sign

0x19	0x45=hex('E')	"thereum Signed Message:\n" ‖ len(message)	message

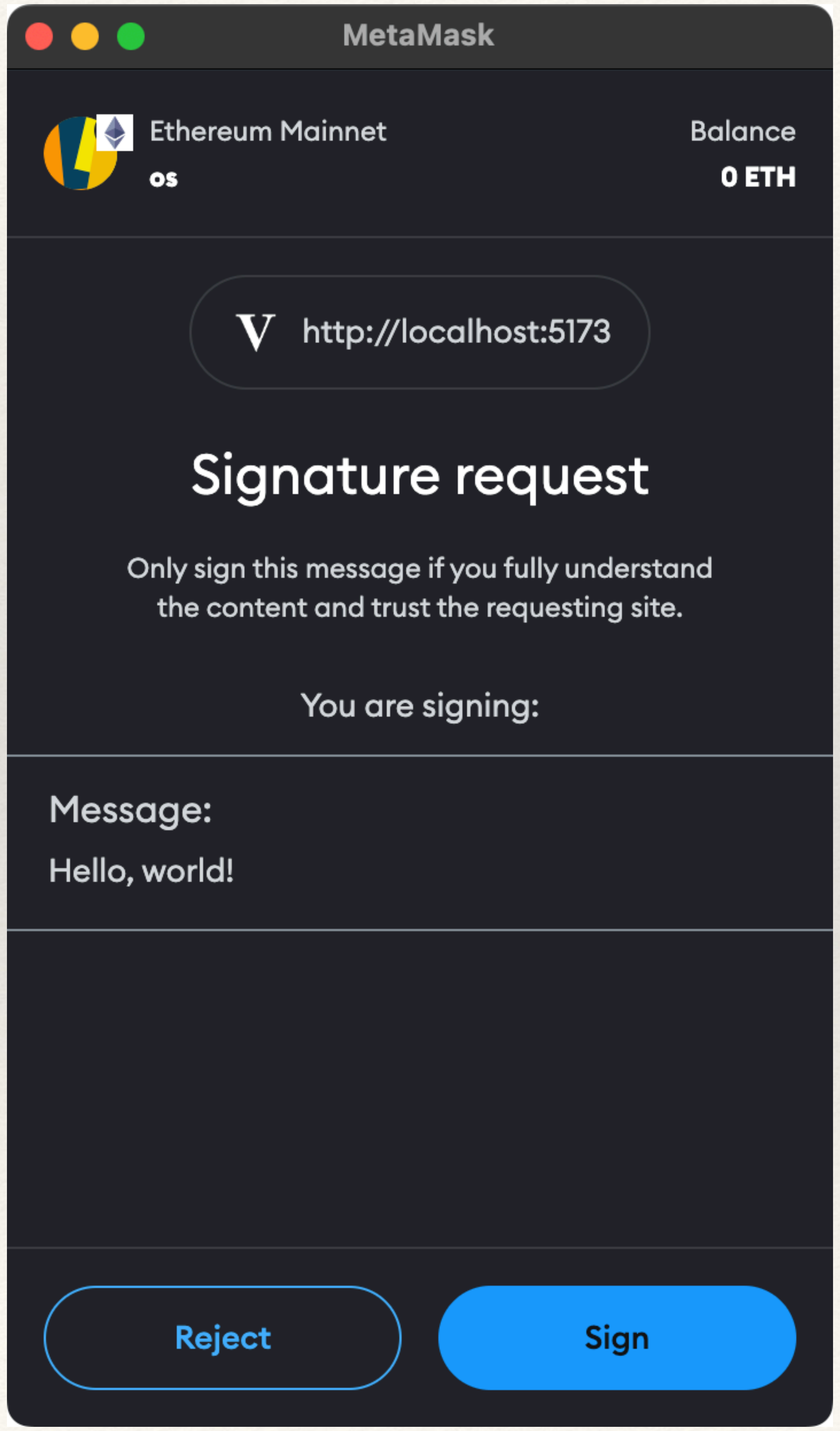
0x19 初始字节： 这个初始字节确保 signed_data 不是有效的 RLP 编码，从而防止签名数据被误解为以太坊交易。

<1 byte version>： 可以自行定义签名数据版本，占用一字节，可以是0；

<version sepific data>： 不定长的消息头数据；

<data to sign>： 原始签名数据;

默认使用 person_sign



MetaMask

Ethereum Mainnet

os

Balance

0 ETH

http://localhost:5173

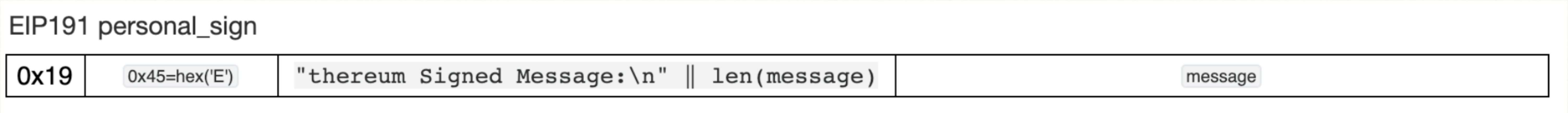
Signature request

Only sign this message if you fully understand the content and trust the requesting site.

You are signing:

Message:

Hello, world!



EIP191 personal_sign

0x19	0x45=hex('E')	"thereum Signed Message:\n" ‖ len(message)	message

```
import { hashMessage } from "viem";
const message = "Hello, World!”;

// hash message with default encoding(EIP-191)
const messageHash = hashMessage(message);
console.log("Message Hash1:", messageHash);
```

等价

```
// hash message with EIP-191 encoding
import { Hex, keccak256, stringToHex } from "viem";
function eip191EncodeAndHash(message: string): Hex {
  const prefix = `\x19Ethereum Signed Message:\n${message.length}`;
  const prefixedMessage = prefix + message;
  return keccak256(stringToHex(prefixedMessage));
}
console.log("Message Hash2:", eip191EncodeAndHash(message));
```

person_sign 签名与验证

签名

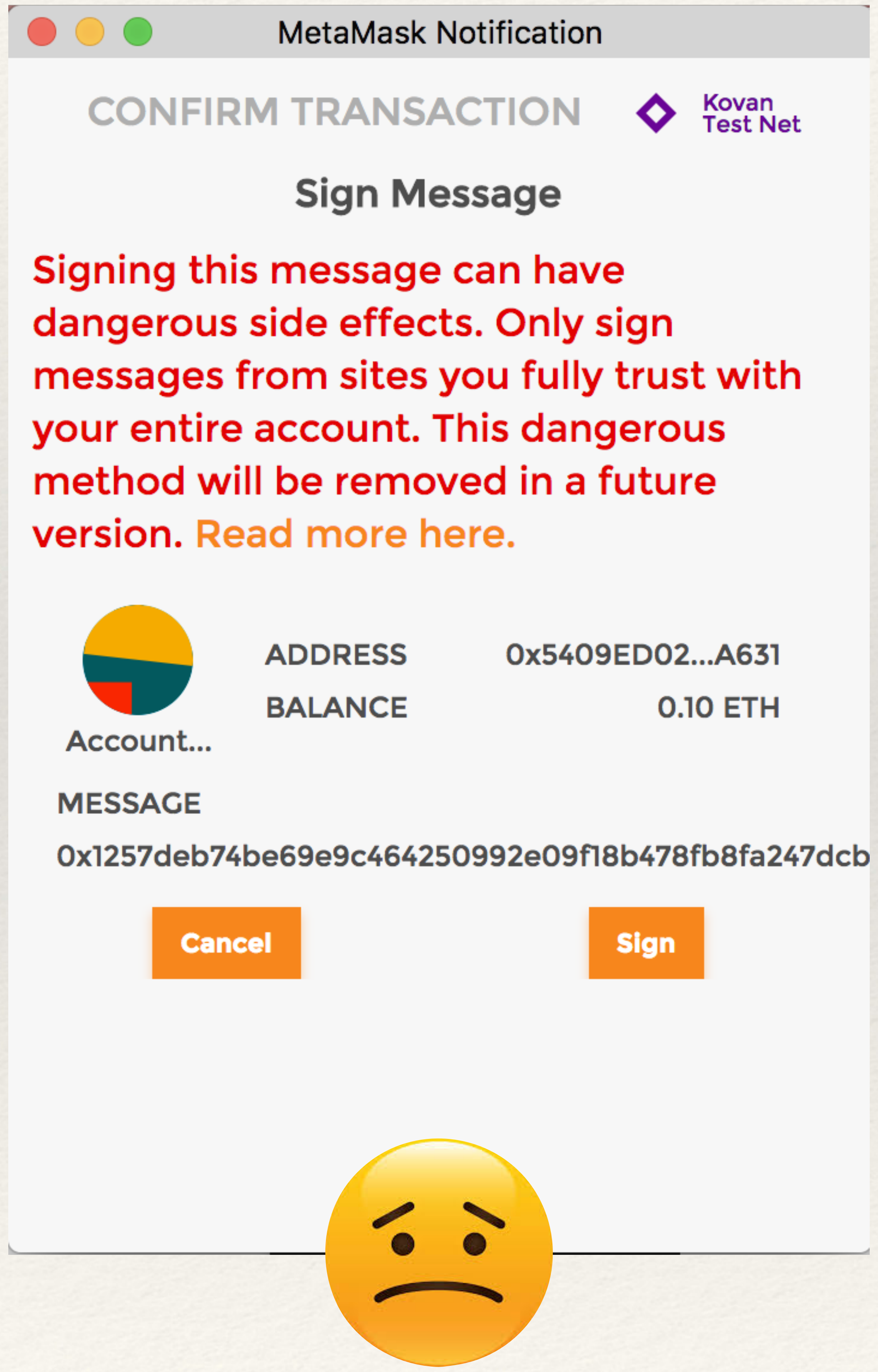
```
import { privateKeyToAccount } from "viem/accounts";

const account =
privateKeyToAccount("0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80");
console.log("Address:", account.address);
// Address:0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266


const signature = await account.signMessage({ message });
console.log("Signature:", signature);
```

```
import "@openzeppelin/contracts/utils/Strings.sol";

contract HashExample {
    function hashPersonal(bytes memory message) public pure returns (bytes32) {
        return
            keccak256(bytes.concat("\x19Ethereum Signed Message:\n",
bytes(Strings.toString(message.length)), message));
    }


}
```

验签

```
import { isAddressEqual } from "viem";
import { recoverMessageAddress } from "viem";
const recoveredAddress = await recoverMessageAddress({
  message: message,
  signature: signature,
});

console.log("Recovered Address:", recoveredAddress);
const wantSigner = "0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266";
const pass = isAddressEqual(recoveredAddress, wantSigner);
console.log("Valid Signature:", pass);
```

EIP-712结构化数据签名



Sign Message

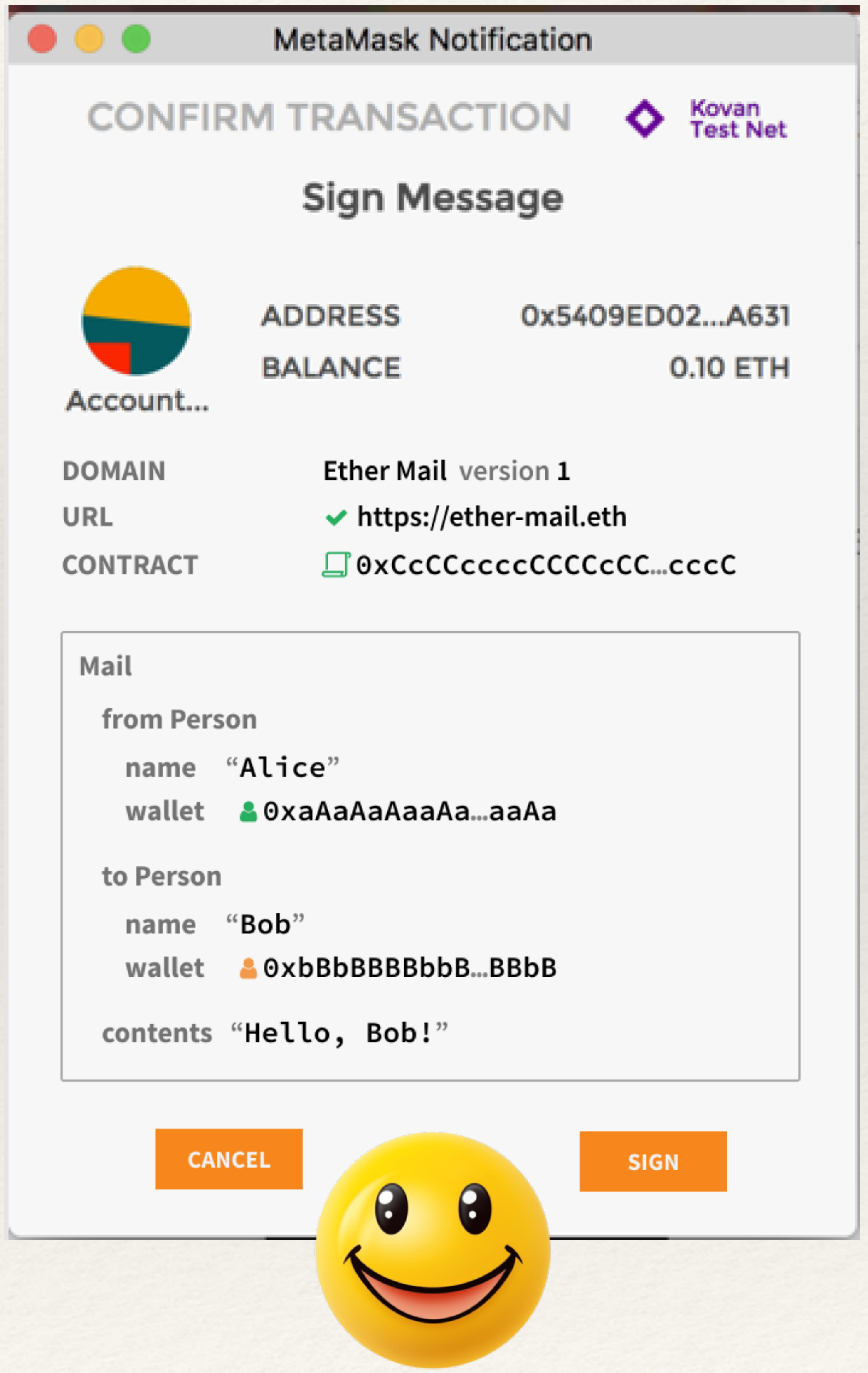
Signing this message can have dangerous side effects. Only sign messages from sites you fully trust with your entire account. This dangerous method will be removed in a future version. Read more here.

ADDRESS	0x5409ED02...A631
BALANCE	0.10 ETH

Account...

MESSAGE

0x1257deb74be69e9c464250992e09f18b478fb8fa247dcb



Sign Message

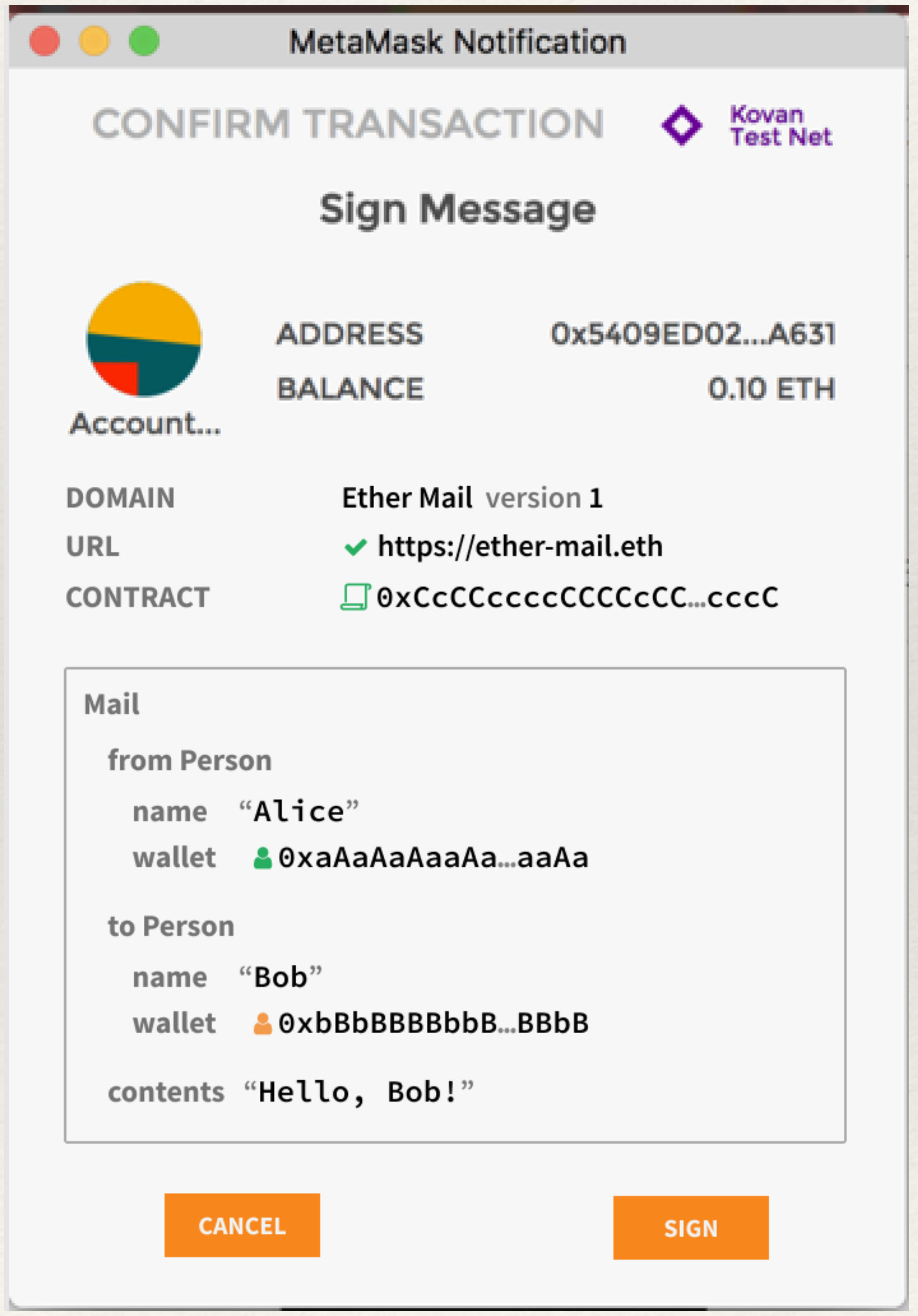
ADDRESS	0x5409ED02...A631
BALANCE	0.10 ETH

Account...

DOMAIN	Ether Mail version 1
URL	✓ https://ether-mail.eth
CONTRACT	0xCcCCccccCCCCcCC...cccC

```
Mail
  from Person
    name   "Alice"
    wallet  0xaAaAaAaaAa...aaAa
  to Person
    name   "Bob"
    wallet  0xbBbBBBBbbB...BBbB
  contents "Hello, Bob!"
```

EIP-712结构化数据



MetaMask Notification

CONFIRM TRANSACTION

Kovan Test Net

Sign Message

Account...

ADDRESS 0x5409ED02...A631

BALANCE 0.10 ETH

DOMAIN Ether Mail version 1

URL ✓ https://ether-mail.eth

CONTRACT 0xCcCCccccCCCCcCC...cccC

Mail

from Person

name “Alice”

wallet 0xaAaAaAaaAa...aaAa

to Person

name “Bob”

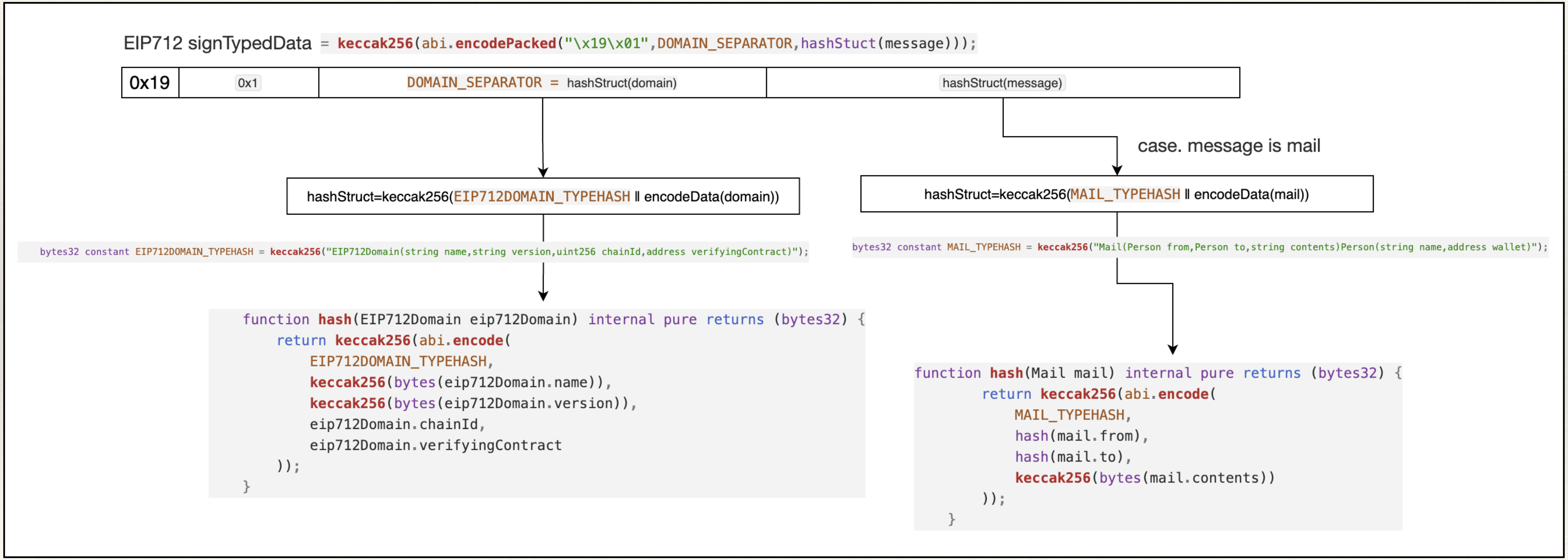
wallet 0xbBbBBBBbbB...BBbB

contents “Hello, Bob!”

CANCEL SIGN

```
const domain = {
  name: "Ether Mail",
  version: "1",
  chainId: 1,
  verifyingContract: "0xCcCCccccCCCCcCCCCCCcCcCccCcCCCcCcccccccC",
} as const;


const types = {
  Person: [
    { name: "name", type: "string" },
    { name: "wallet", type: "address" },
  ],
  Mail: [
    { name: "from", type: "Person" },
    { name: "to", type: "Person" },
    { name: "contents", type: "string" },
  ],
} as const;

const message = {
  from: {
    name: "Cow",
    wallet: "0xCD2a3d9F938E13CD947Ec05AbC7FE734Df8DD826",
  },
  to: {
    name: "Bob",
    wallet: "0xbBbBBBBbbBBBbbbBbbBbbbbBBbBbbbbBbBbbBBbB",
  },
  contents: "Hello, Bob!",
} as const;
```

EIP-712结构化数据格式



EIP712 signTypedData = `keccak256(abi.encodePacked("\x19\x01",DOMAIN_SEPARATOR,hashStuct(message)));`

0x19	0x1	DOMAIN_SEPARATOR = hashStruct(domain)	hashStruct(message)

case. message is mail

hashStruct=keccak256(EIP712DOMAIN_TYPEHASH ‖ encodeData(domain))

hashStruct=keccak256(MAIL_TYPEHASH ‖ encodeData(mail))

```
bytes32 constant EIP712DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)");
```

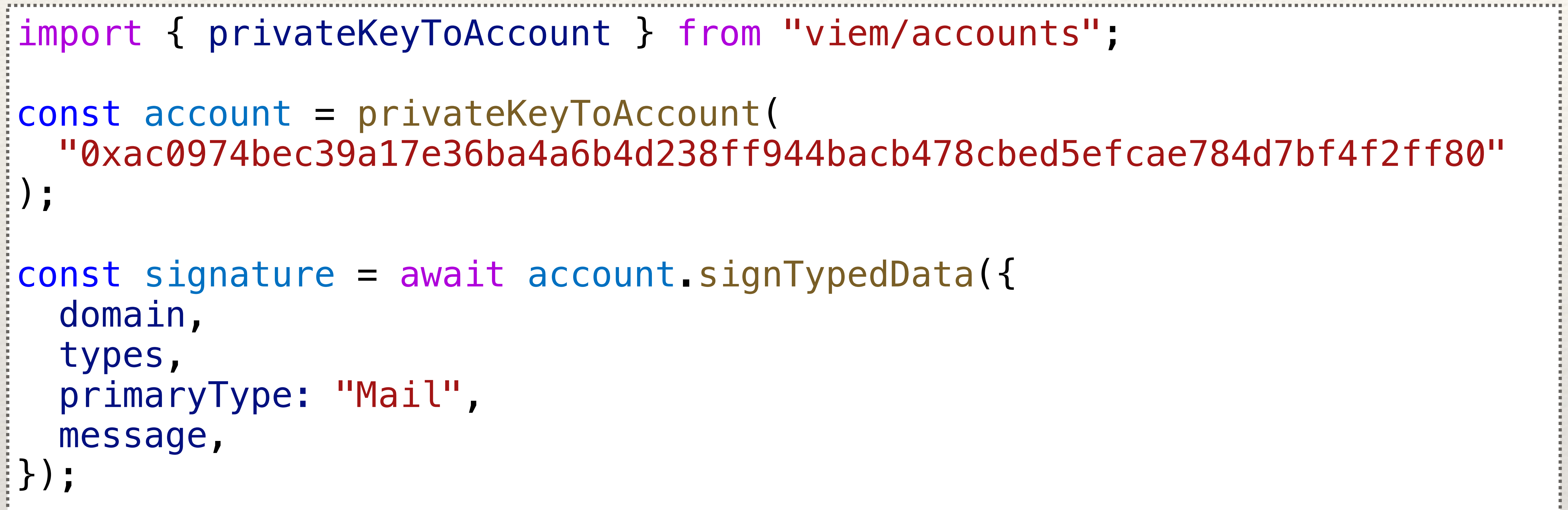
```
bytes32 constant MAIL_TYPEHASH = keccak256("Mail(Person from,Person to,string contents)Person(string name,address wallet)");
```

```
function hash(EIP712Domain eip712Domain) internal pure returns (bytes32) {
    return keccak256(abi.encode(
        EIP712DOMAIN_TYPEHASH,
        keccak256(bytes(eip712Domain.name)),
        keccak256(bytes(eip712Domain.version)),
        eip712Domain.chainId,
        eip712Domain.verifyingContract
    ));
}
```

```
function hash(Mail mail) internal pure returns (bytes32) {
        return keccak256(abi.encode(
            MAIL_TYPEHASH,
            hash(mail.from),
            hash(mail.to),
            keccak256(bytes(mail.contents))
        ));
    }
```

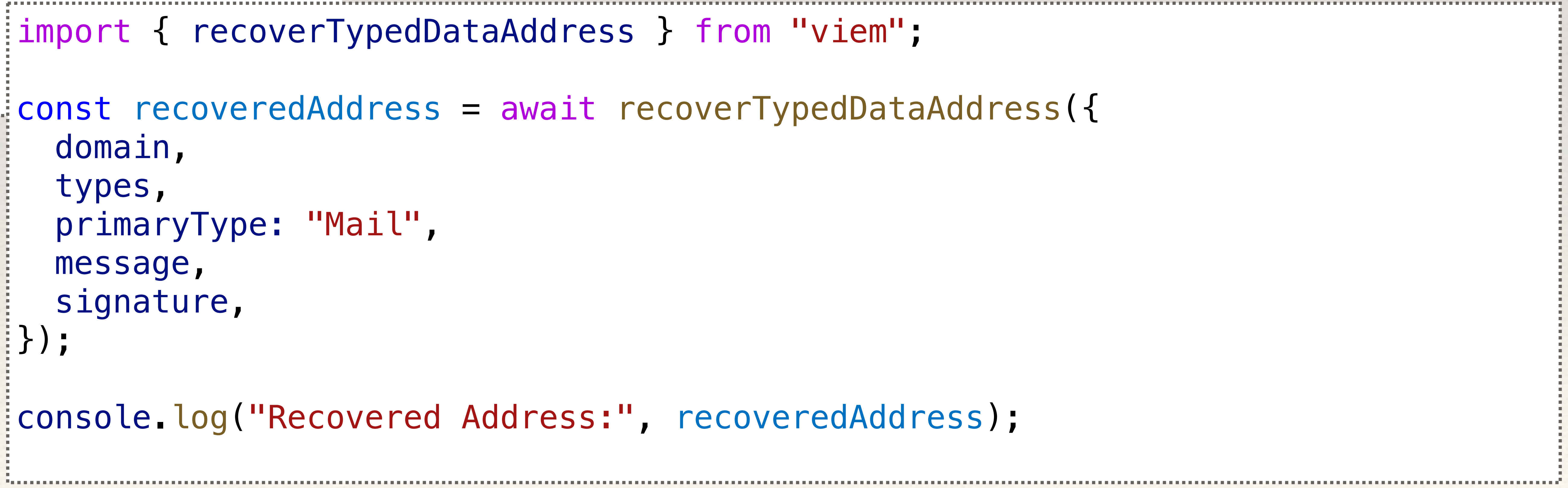
EIP-712结构化数据签名

签名



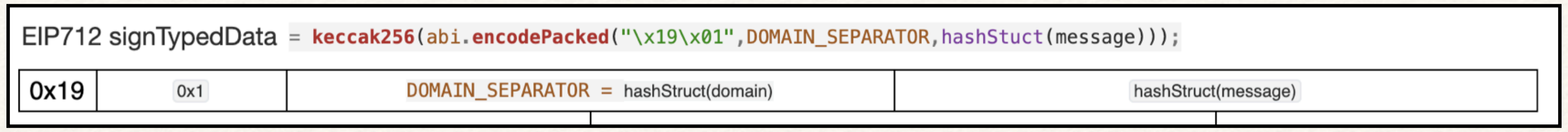
```
import { privateKeyToAccount } from "viem/accounts";

const account = privateKeyToAccount(
  "0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80"
);

const signature = await account.signTypedData({
  domain,
  types,
  primaryType: "Mail",
  message,
});

console.log("Signature:", signature);
```

验签



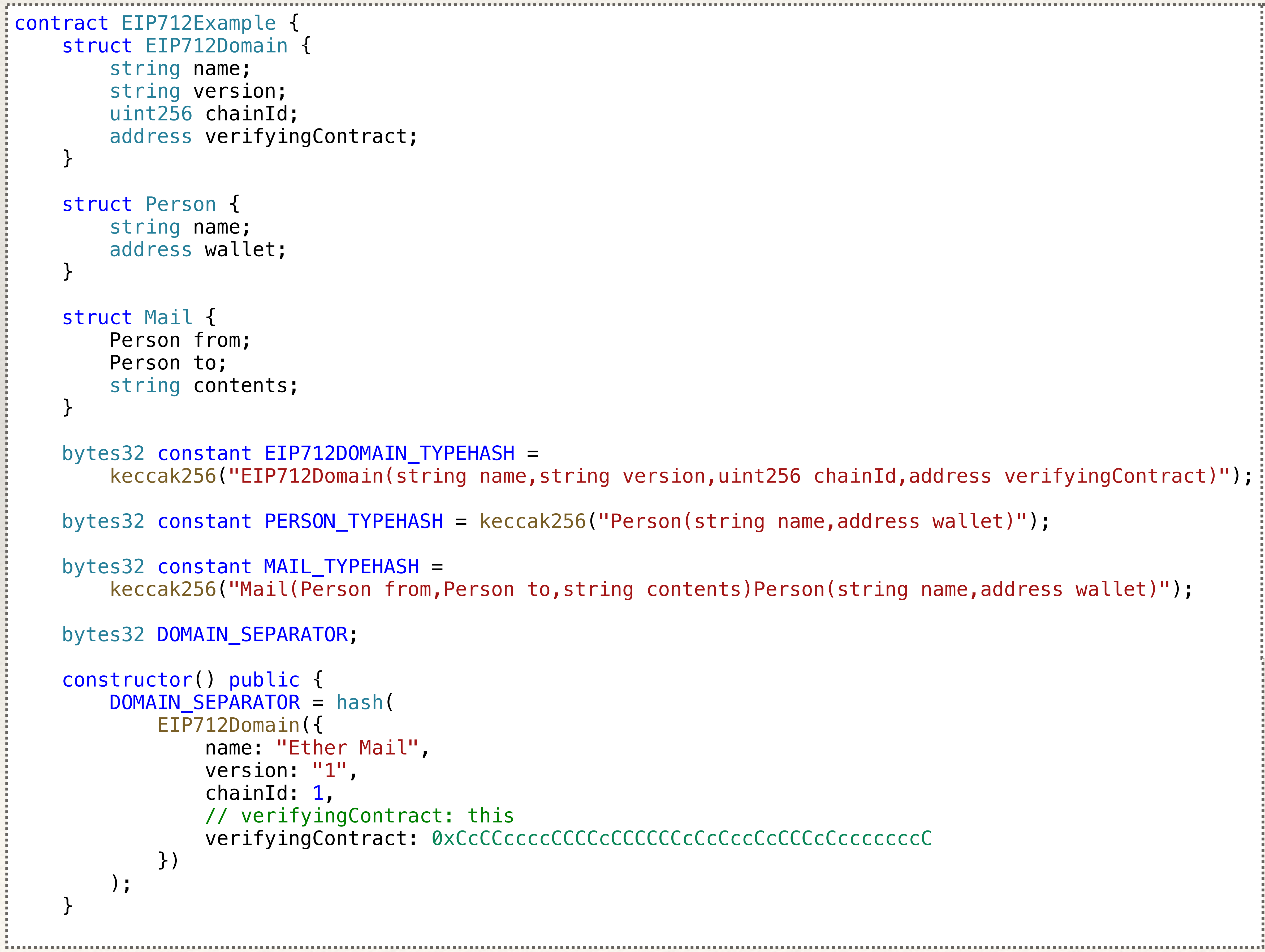
```
import { recoverTypedDataAddress } from "viem";

const recoveredAddress = await recoverTypedDataAddress({
  domain,
  types,
  primaryType: "Mail",
  message,
  signature,
});

console.log("Recovered Address:", recoveredAddress);
```

EIP-712结构化数据签名

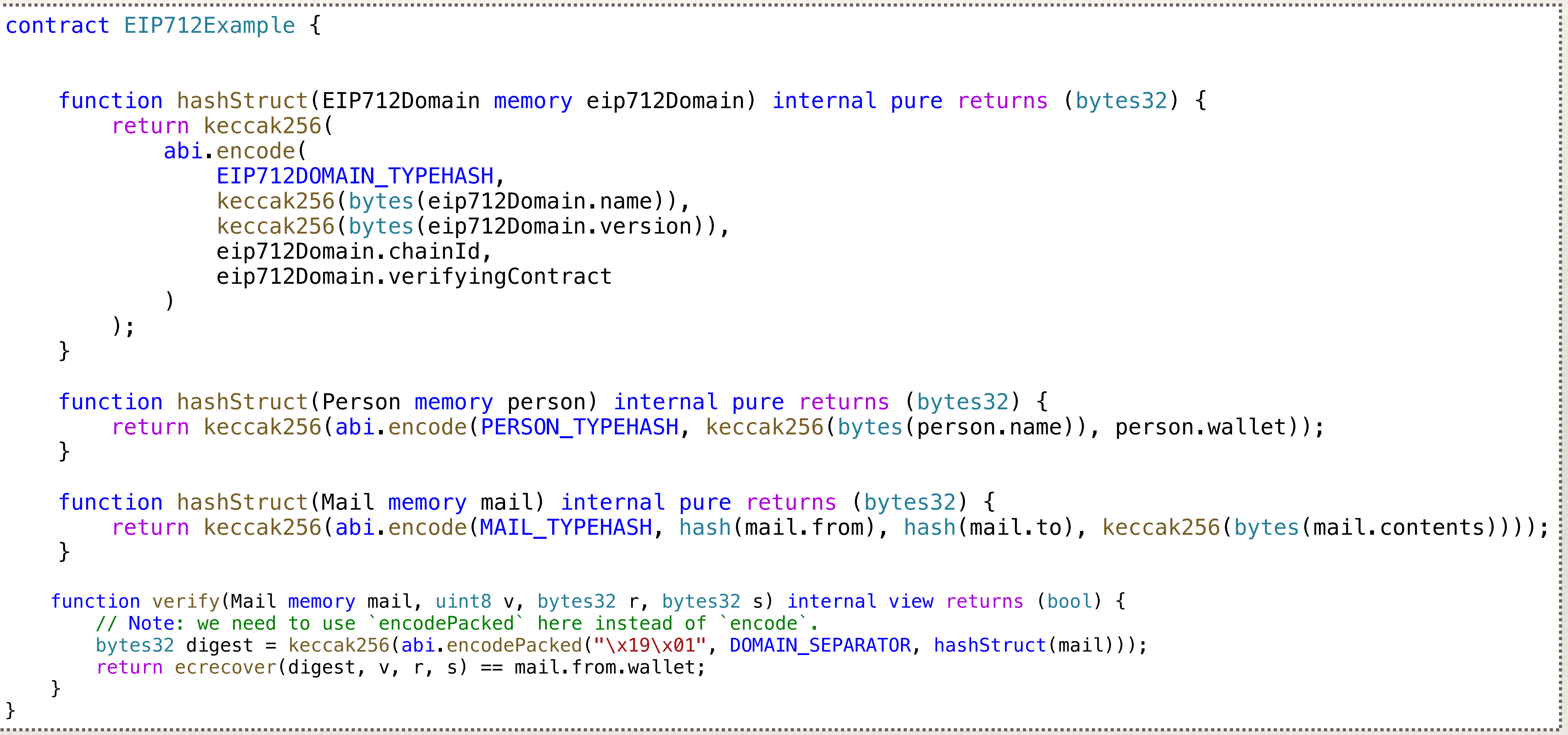


EIP712 signTypedData = `keccak256(abi.encodePacked("\x19\x01",DOMAIN_SEPARATOR,hashStuct(message)));`

0x19	0x1	DOMAIN_SEPARATOR = hashStruct(domain)	hashStruct(message)



```
contract EIP712Example {
    struct EIP712Domain {
        string name;
        string version;
        uint256 chainId;
        address verifyingContract;
    }

    struct Person {
        string name;
        address wallet;
    }

    struct Mail {
        Person from;
        Person to;
        string contents;
    }

    bytes32 constant EIP712DOMAIN_TYPEHASH =
        keccak256("EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)");

    bytes32 constant PERSON_TYPEHASH = keccak256("Person(string name,address wallet)");

    bytes32 constant MAIL_TYPEHASH =
        keccak256("Mail(Person from,Person to,string contents)Person(string name,address wallet)");

    bytes32 DOMAIN_SEPARATOR;

    constructor() public {
        DOMAIN_SEPARATOR = hash(
            EIP712Domain({
                name: "Ether Mail",
                version: "1",
                chainId: 1,
                // verifyingContract: this
                verifyingContract: 0xCcCCccccCCCCcCCCCCCcCcCccCcCCCcCcccccccC
            })
        );
    }
```



```
contract EIP712Example {


    function hashStruct(EIP712Domain memory eip712Domain) internal pure returns (bytes32) {
        return keccak256(
            abi.encode(
                EIP712DOMAIN_TYPEHASH,
                keccak256(bytes(eip712Domain.name)),
                keccak256(bytes(eip712Domain.version)),
                eip712Domain.chainId,
                eip712Domain.verifyingContract
            )
        );
    }

    function hashStruct(Person memory person) internal pure returns (bytes32) {
        return keccak256(abi.encode(PERSON_TYPEHASH, keccak256(bytes(person.name)), person.wallet));
    }

    function hashStruct(Mail memory mail) internal pure returns (bytes32) {
        return keccak256(abi.encode(MAIL_TYPEHASH, hash(mail.from), hash(mail.to), keccak256(bytes(mail.contents))));
    }

    function verify(Mail memory mail, uint8 v, bytes32 r, bytes32 s) internal view returns (bool) {
        // Note: we need to use `encodePacked` here instead of `encode`.
        bytes32 digest = keccak256(abi.encodePacked("\x19\x01", DOMAIN_SEPARATOR, hashStruct(mail)));
        return ecrecover(digest, v, r, s) == mail.from.wallet;
    }
}
```

ERC20支付方式

如何允许第三方消费我的Token?

BuyNFT时如何支付Token?

Transfer

Approve + TransferFrom

Transfer +CallBack

Signature + TransferFrom

ERC20离线签名

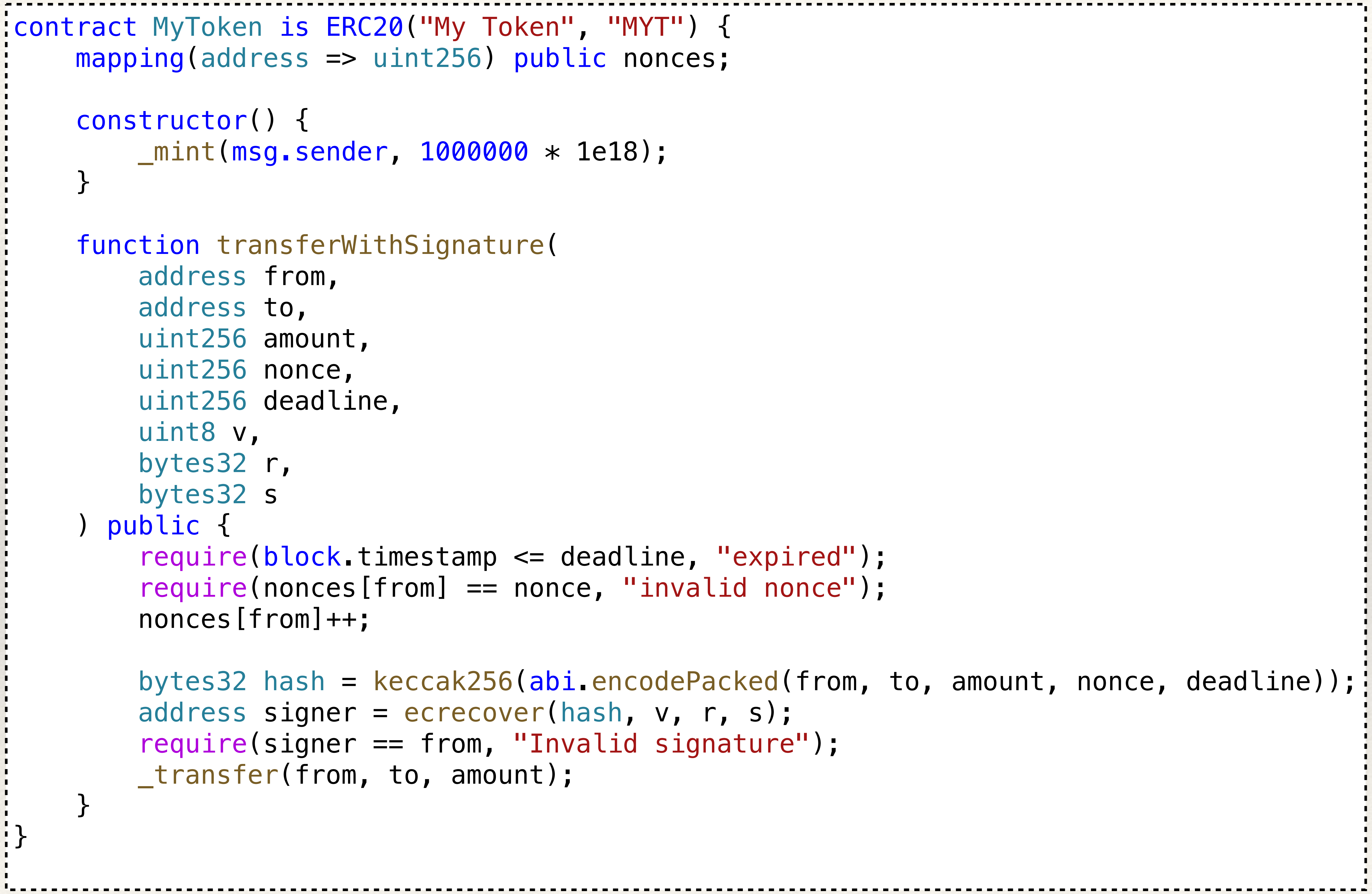
思路：

先离线签名允许使用花费

携带签名信息发送交易来花费Token

https://eips.ethereum.org/EIPS/eip-2612

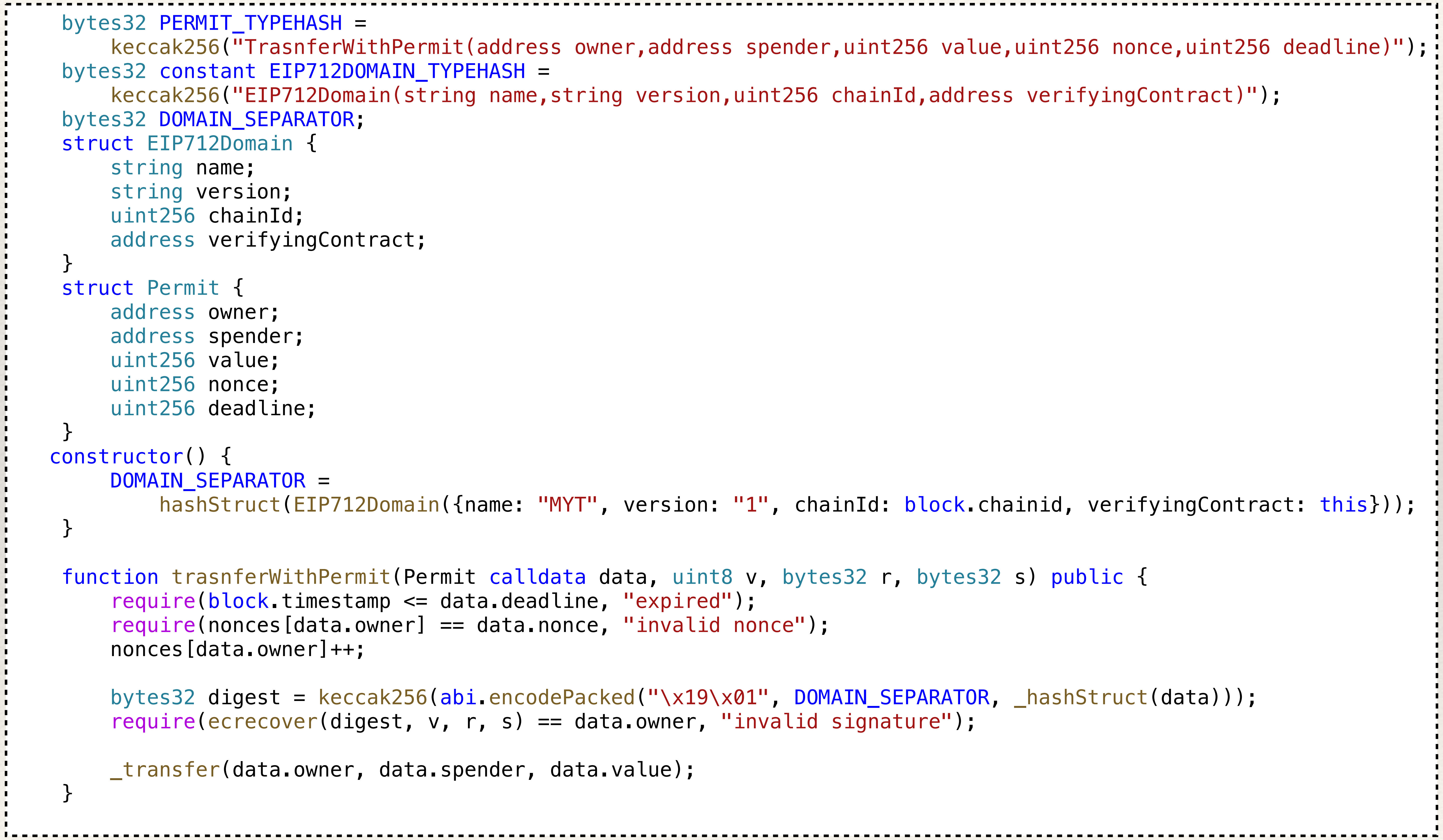
ERC20离线签名



```solidity
contract MyToken is ERC20("My Token", "MYT") {
    mapping(address => uint256) public nonces;

    constructor() {
        _mint(msg.sender, 1000000 * 1e18);
    }

    function transferWithSignature(
        address from,
        address to,
        uint256 amount,
        uint256 nonce,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) public {
        require(block.timestamp <= deadline, "expired");
        require(nonces[from] == nonce, "invalid nonce");
        nonces[from]++;

        bytes32 hash = keccak256(abi.encodePacked(from, to, amount, nonce, deadline));
        address signer = ecrecover(hash, v, r, s);
        require(signer == from, "Invalid signature");
        _transfer(from, to, amount);
    }
}
```

改进ERC20离线签名

使用ERC721（格式化签名）改进，EIP2612 则是一种标准实现



```
    bytes32 PERMIT_TYPEHASH =
        keccak256("TrasnferWithPermit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)");
    bytes32 constant EIP712DOMAIN_TYPEHASH =
        keccak256("EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)");
    bytes32 DOMAIN_SEPARATOR;
    struct EIP712Domain {
        string name;
        string version;
        uint256 chainId;
        address verifyingContract;
    }
    struct Permit {
        address owner;
        address spender;
        uint256 value;
        uint256 nonce;
        uint256 deadline;
    }
   constructor() {
        DOMAIN_SEPARATOR =
            hashStruct(EIP712Domain({name: "MYT", version: "1", chainId: block.chainid, verifyingContract: this}));
    }

    function trasnferWithPermit(Permit calldata data, uint8 v, bytes32 r, bytes32 s) public {
        require(block.timestamp <= data.deadline, "expired");
        require(nonces[data.owner] == data.nonce, "invalid nonce");
        nonces[data.owner]++;

        bytes32 digest = keccak256(abi.encodePacked("\x19\x01", DOMAIN_SEPARATOR, _hashStruct(data)));
        require(ecrecover(digest, v, r, s) == data.owner, "invalid signature");

        _transfer(data.owner, data.spender, data.value);
    }
```

作业说明

- 代码在自己的 github 提交
- 在 decert.me 提交领取证书
- **不可抄袭作业**，一经发现将不再检查抄袭者作业！

作业

完成挑战： https://decert.me/challenge/fc66ef6c-35db-4ee7-b11d-c3b2d3fa356a

1. 使用 EIP2612 标准（可基于 Openzepplin 库）编写一个自己名称的 Token 合约。
2. 修改 TokenBank 存款合约 ,添加一个函数 `permitDeposit` 以支持离线签名授权（permit）进行存款。
3. 修改Token 购买 NFT NTFMarket 合约，添加功能 `permitBuy()` 实现只有离线授权的白名单地址才可以购买 NFT （用自己的名称发行 NFT，再上架） 。白名单具体实现逻辑为：项目方给白名单地址签名，白名单用户拿到签名信息后，传给 `permitBuy()` 函数，在 `permitBuy()`中判断时候是经过许可的白名单用户，如果是，才可以进行后续购买，否则 revert 。

要求：

1. 有 Token 存款及 NFT 购买成功的测试用例
2. 有测试用例运行日志或截图，能够看到 Token 及 NFT 转移。

谢谢