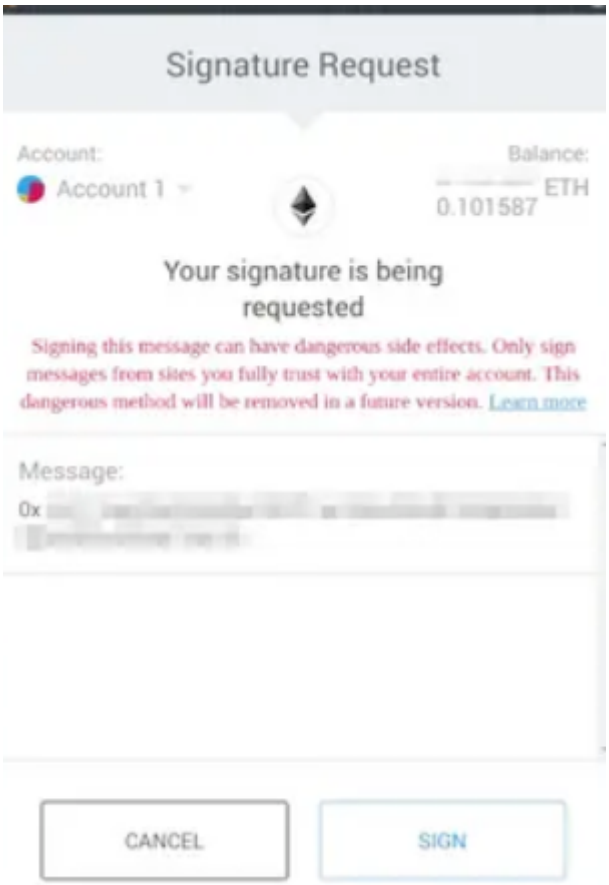
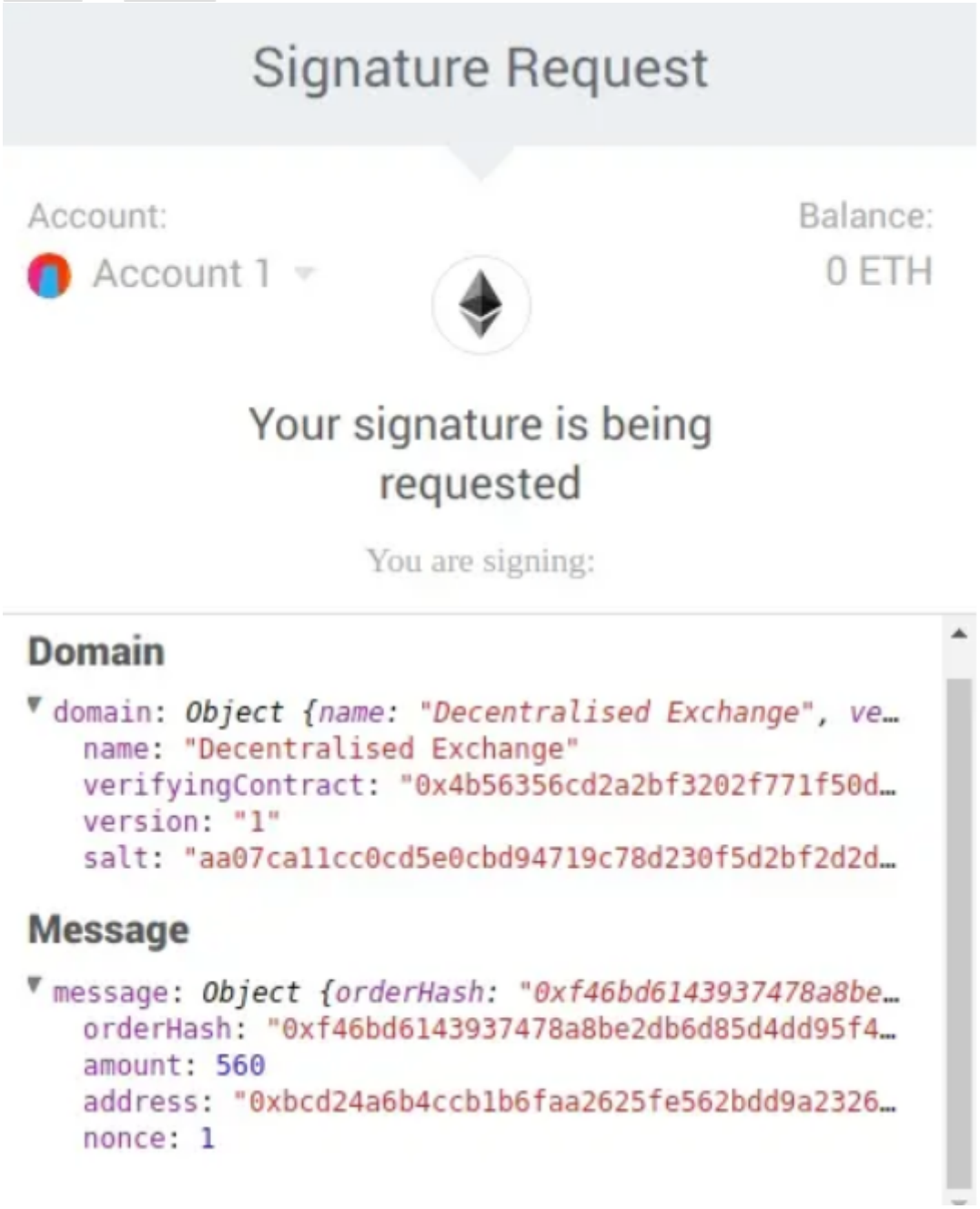
PersonalMessage & CIP23

conflux-sdk fluen`EIP712` `CIP23` `CIP23` `CIP23` (typedData)



`CIP23` `EIP712` metamask 1-2



`CIP23` `java-conflux-sdk` `CIP23`

`CIP23` `EIP712` https://github.com/Conflux-Chain/CIPs/blob/master/CIPs/cip-23.md

`CIP23` `personal_sign` `cfx_signTypedData_v4` "v0G9u7huK4mJb2K1",

```
{
  "types": {
```

```
    "CIP23Domain": [
      {
        "name": "name",
        "type": "string"
      },
      {
        "name": "version",
        "type": "string"
      },
      {
        "name": "chainId",
        "type": "uint256"
      },
      {
        "name": "verifyingContract",
        "type": "address"
      }
    ],
    "Person": [
      {
        "name": "name",
        "type": "string"
      },
      {
        "name": "wallet",
        "type": "address"
      }
    ],
    "Mail": [
      {
        "name": "from",
        "type": "Person"
      },
      {
        "name": "to",
        "type": "Person"
      },
      {
        "name": "contents",
        "type": "string"
```

```
      }
    ]
  },
  "primaryType": "Mail",
  "domain": {
    "name": "Ether Mail",
    "version": "1",
    "chainId": 1,
    "verifyingContract": "0xCcCCccccCCCCcCCCCCCcCcCccCcCCCcCcccccccC"
  },
  "message": {
    "from": {
      "name": "Cow",
      "wallet": "0xCD2a3d9F938E13CD947Ec05AbC7FE734Df8DD826"
    },
    "to": {
      "name": "Bob",
      "wallet": "0xbBbBBBBbbBBBbbbBbbBbbbbBBbBbbbbBbBbbBBbB"
    },
    "contents": "Hello, Bob!"
  }
}
```

typed

https://github.com/Conflux-Chain/CIPs/blob/master/CIPs/cip-23.md#:~:text=include%20chainId%20field.-,Encoding%20method,-The%20set%20of

`msg` `\x19Conflux Signed Message:\n` hash

fluent

fluent web3 https://confluxrpc.github.io/fluent-wallet-doc/docs/provider-rpc/

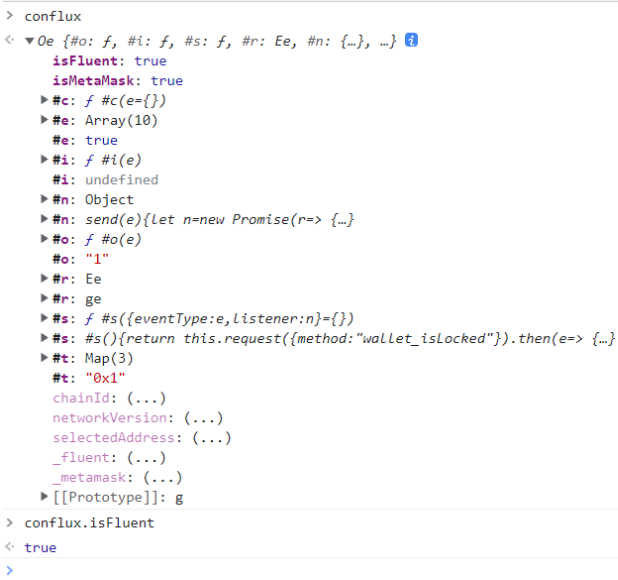
fluent https://fluentwallet.zendesk.com/hc/zh-cn fluent

`console`

```
> conflux

> conflux.isFluent
```

fluent js



fluent

personal_sign

console

```
conflux
  .request({
    method: 'personal_sign',
    params: [
 'v0G9u7huK4mJb2K1', '<your_address>']})
```

fluent


```
> conflux
    .request({
      method: 'personal_sign',
      params: [
  'v0G9u7huK4mJb2K1', 'cfxtest:aajb342mw5kzad6pjjkdz0wxx0tr54nfwpbu6yaj49']})
◂ ▾Promise {<pending>}
    ▸[[Prototype]]: Promise
     [[PromiseState]]: "fulfilled"
     [[PromiseResult]]: "0x7e4216720b40f8d7f2cda70433d4a94f3926635517cd5691c778e38eaed87236759ddf7c53ec55c5463f8e966ebd5a32c2dbe1061e95afcbe64ef3a8187badcb00"
>
```

cfx_signTypedData_v4

console

```
conflux
  .request({
    method: 'cfx_signTypedData_v4',
    params: [
 '<your_address>',
        `{
    "types": {
        "CIP23Domain": [
            {
                "name": "name",
                "type": "string"
            },
            {
                "name": "version",
                "type": "string"
            },
            {
                "name": "chainId",
                "type": "uint256"
            },
            {
                "name": "verifyingContract",
                "type": "address"
            }
        ],
        "Person": [
            {
                "name": "name",
                "type": "string"
            },
            {
                "name": "wallet",
                "type": "address"
            }
        ],
```

```
        "Mail": [
            {
                "name": "from",
                "type": "Person"
            },
            {
                "name": "to",
                "type": "Person"
            },
            {
                "name": "contents",
                "type": "string"
            }
        ]
    },
    "primaryType": "Mail",
    "domain": {
        "name": "Ether Mail",
        "version": "1",
        "chainId": 1,
        "verifyingContract": "0xCcCCccccCCCCcCCCCCCcCcCccCcCCCcCccccccccC"
    },
    "message": {
        "from": {
            "name": "Cow",
            "wallet": "0xCD2a3d9F938E13CD947Ec05AbC7FE734Df8DD826"
        },
        "to": {
            "name": "Bob",
            "wallet": "0xbBbBBBBbbBBBbbbBbbBbbbbBBbBbbbbBbBbbBBbB"
        },
        "contents": "Hello, Bob!"
    }
}`]
  })
```

签署这个信息?

Ether Mail

信息

from:

name: Cow

wallet: 0xCD2a3d9F938E13CD947Ec05AbC7FE734Df8DD826

to:

name: Bob

wallet: 0xbBbBBBBbbBBBbbbBbbBbbbbBBbBbbbBbBbbBBbB

contents: Hello, Bob!

取消

签名

```
            `
                "name": "contents",
                "type": "string"
            }
        ]
    },
    "primaryType": "Mail",
    "domain": {
        "name": "Ether Mail",
        "version": "1",
        "chainId": 1,
        "verifyingContract": "0xCcCCccccCCCCcCCCCCCcCcCccCcCCCcCcccccccC"
    },
    "message": {
        "from": {
            "name": "Cow",
            "wallet": "0xCD2a3d9F938E13CD947Ec05AbC7FE734Df8DD826"
        },
        "to": {
            "name": "Bob",
            "wallet": "0xbBbBBBBbbBBBbbbBbbBbbbbBBbBbbbbBbBbbBBbB"
        },
        "contents": "Hello, Bob!"
    }
}`]
  })
```

```
Promise {<pending>}
  [[Prototype]]: Promise
  [[PromiseState]]: "fulfilled"
  [[PromiseResult]]: "0x7f28d98e75cdcaee68354d6ad0b9a2e8c4a3d365fb10fb70a1bc03a72bdb70de5b6d6587c7af57994c494ca3a1672e17d3f8f013e20641a7299f0d427a39a39001"
```

Java-conflux-sdk

`java-conflux-sdk` `personal_sign` `cfx_signTypedData_v4`

```
package conflux.web3j.crypto;

import org.junit.jupiter.api.Test;
import org.web3j.utils.Numeric;

import java.io.IOException;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class SignDataTests {

    //cfx_signTypedData_v4
    @Test
    public void testSignValidStructure() throws IOException {
        StructuredDataTests t = new StructuredDataTests();
```

```
      //  TypedData
        String msg = t.getResource(
                "build/resources/test/"
                        + "structured_data_json_files/ValidStructuredData.json");
      //  msg
        StructuredDataEncoder dataEncoder = new StructuredDataEncoder(msg);
      //        msghash
        org.web3j.crypto.Sign.SignatureData sign =
org.web3j.crypto.Sign.signMessage(dataEncoder.hashStructuredData(), SampleKeys.KEY_PAIR,
false);
        assertEquals(

"0x371ef48d63082d3875fee13b392c5b6a7449aa638921cb9f3d419f5b6a817ba754d085965fb3a041c3b178d3ae3
798ea322ae74cb687dd699b5f6045c7fe47a91c",
                Numeric.toHexString(sign.getR()) + Numeric.toHexStringNoPrefix(sign.getS()) +
Numeric.toHexStringNoPrefix(sign.getV()));
    }

    //personal_sign
    @Test
    public void testSignAnyMessage() throws IOException {
        String message = "v0G9u7huK4mJb2K1";
      // msg          msghash
        org.web3j.crypto.Sign.SignatureData sign =
Sign.signPrefixedMessage(message.getBytes(), SampleKeys.KEY_PAIR);
        assertEquals(

"0xbb0ee8492623f2ef6ed461ea638f8b5060b191a1c8830c93d84245f3fb27e20a755e24ff60fe76482dd4377a0ae
f036937ef88537b2d0fdd834a54e76ecafadc1c",
                Numeric.toHexString(sign.getR()) + Numeric.toHexStringNoPrefix(sign.getS()) +
Numeric.toHexStringNoPrefix(sign.getV()));
    }
}
```

sdk

java-conflux-sdk ecrecover

```
package conflux.web3j.crypto;

import org.junit.jupiter.api.Test;
import org.web3j.crypto.*;
import org.web3j.crypto.Sign;
import org.web3j.utils.Numeric;

import java.io.IOException;
import java.util.Arrays;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class ECRecoverTest {
    private String getAddress() {
        return Numeric.prependHexPrefix(Keys.getAddress(getPubKey()));
    }

    private String getPubKey() {
        return SampleKeys.KEY_PAIR.getPublicKey().toString();
    }



    @Test
    public void testSignAndRecoverMessage() {
        String message = "v0G9u7huK4mJb2K1";

        byte[] msgHash = conflux.web3j.crypto.Sign.getConfluxMessageHash(message.getBytes());

        Sign.SignatureData sign =
conflux.web3j.crypto.Sign.signPrefixedMessage(message.getBytes(), SampleKeys.KEY_PAIR);
□□□//        recover,
        String recoverAddress = conflux.web3j.crypto.Sign.recoverSignature(sign, msgHash,
getAddress());
        assertEquals(recoverAddress, getAddress());
    }

  // fluent                   recover
    @Test
```

```
    public void testRecoverTyped() throws IOException {
        StructuredDataTests t = new StructuredDataTests();
        String msg = t.getResource(
                "build/resources/test/"
                        + "structured_data_json_files/ValidStructuredData.json");
        StructuredDataEncoder dataEncoder = new StructuredDataEncoder(msg);

        //           fluent
        String signature =

"0x371ef48d63082d3875fee13b392c5b6a7449aa638921cb9f3d419f5b6a817ba754d085965fb3a041c3b178d3ae3
798ea322ae74cb687dd699b5f6045c7fe47a91c";

      □ //
        byte[] signatureBytes = Numeric.hexStringToByteArray(signature);
        byte v = signatureBytes[64];
        if (v < 27) {
            v += 27;
        }

        Sign.SignatureData sd =
                new Sign.SignatureData(
                        v,
                        (byte[]) Arrays.copyOfRange(signatureBytes, 0, 32),
                        (byte[]) Arrays.copyOfRange(signatureBytes, 32, 64));
□□□//getAddress()       fluent
        String recoverAddress = conflux.web3j.crypto.Sign.recoverSignature(sd,
dataEncoder.hashStructuredData(), getAddress());
        assertEquals(recoverAddress, getAddress());
    }

  //    testSignAndRecoverMessage()
    @Test
    public void testSignAndRecoverTyped() throws IOException {
        StructuredDataTests t = new StructuredDataTests();
        String msg = t.getResource(
                "build/resources/test/"
                        + "structured_data_json_files/ValidStructuredData.json");
        StructuredDataEncoder dataEncoder = new StructuredDataEncoder(msg);
```

```
        Sign.SignatureData sign = Sign.signMessage(dataEncoder.hashStructuredData(),
SampleKeys.KEY_PAIR, false);

        String recoverAddress = conflux.web3j.crypto.Sign.recoverSignature(sign,
dataEncoder.hashStructuredData(), getAddress());
        assertEquals(recoverAddress, getAddress());
    }
}
```

```
// file: CIP23DomainExample.sol
pragma solidity ^0.4.24;

contract Example {
    struct Person {
        string name;
        address wallet;
    }

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

    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)"
    );

    struct CIP23Domain {
        string  name;
        string  version;
```

```
    uint256 chainId;
}

struct VerifyClaim{
    address userAddress;
    uint256 randNo;
    uint256 amount;
}

bytes32 constant CIP23DOMAIN_TYPEHASH = keccak256(
    "CIP23Domain(string name,string version,uint256 chainId)"
);

bytes32 constant VERIFYCLAIM_TYPEHASH = keccak256(
    "VerifyClaim(address userAddress,uint256 randNo,uint256 amount)"
);

bytes32 DOMAIN_SEPARATOR;

constructor () public {
    DOMAIN_SEPARATOR = hash(CIP23Domain({
        name: "VerifyClaim",
        version: '1',
        chainId: 97
    }));
}

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

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

```
            keccak256(bytes(mail.contents))
        ));
    }

    function hash(CIP23Domain cip23Domain) internal pure returns (bytes32) {
        return keccak256(abi.encode(
            CIP23DOMAIN_TYPEHASH,
            keccak256(bytes(cip23Domain.name)),
            keccak256(bytes(cip23Domain.version)),
            cip23Domain.chainId
        ));
    }

    function hash(VerifyClaim verifyclaim) internal pure returns (bytes32) {
        return keccak256(abi.encode(
            VERIFYCLAIM_TYPEHASH,
            verifyclaim.userAddress,
            verifyclaim.randNo,
            verifyclaim.amount
        ));
    }

    function verify(VerifyClaim verifyclaim, 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,
            hash(verifyclaim)
        ));
        return ecrecover(digest, v, r, s) == 0x53dE6A872435F5286BEFd0b6fB3bC06742aF8C8F;
    }

    function test(address _userAddress, uint256 _randNO, uint256 _amount, uint8 _v, bytes32
_r, bytes32 _s) public view returns (bool) {
        // Example signed message
        VerifyClaim memory verifyclaim = VerifyClaim({
            userAddress: _userAddress,
            randNo: _randNO,
            amount: _amount
```

```
        });
        assert(verify(verifyclaim, _v, _r, _s));
        return true;
    }
}
```

sdk sdk

1. fluent

```
> conflux
Uncaught ReferenceError: conflux is not defined
    at <anonymous>:1:1
```

:

2. fluent

```
conflux
  .request({
    method: 'personal_sign',
    params: [
'v0G9u7huK4mJb2K1', 'cfxtest:aajb342mw5kzad6pjjkdz0wxx0tr54nfwpbu6yaj49']})
Promise {<pending>}
- [wallet_validateAppPermissions] [Unauthorized 4100]
RPC Stack:
-> wallet_validateAppPermissions
-> personal_sign
{code: 4100, message: ' - [wallet_validateAppPermissions] [Unauthorized 4…> wallet_validateAppPermissions\n-> personal_sign\n', data: {…}}
Uncaught (in promise) {code: 4100, message: ' - [wallet_validateAppPermissions] [Unauthorized 4…> wallet_validateAppPermissions\n-> personal_sign\n', data: {…}}
```

fluent

https://fluent-wallet.zendesk.com/hc/zh-cn,

java-conflux-sdk-tests

Revision #7
Created 5 July 2022 02:35:12 by Pana
Updated 9 August 2022 09:08:07 by Xianqi