SealBlock 钱包 API 文档

更新日期：**2018 年 9 月 8 日**

SealBlock 硬件热钱包将钱包私钥存储在 SealBlock 硬件中，使用英特尔 SGX 创造完全加密的可信执行环境进行保护，加上安全规则如多签等达到最高安全强度以保护用户数字资产。SealBlock 钱包 API 是一套标准化的 API 接口，可用于调用 SealBlock 硬件热钱包上的功能如创建钱包、转账等。

APP 钱包集成 SealBlock API

需要开发的代码分为两个部分，APP 端和 APP 服务器端。在 APP 端需要生成一个私钥（称为控制私钥），该控制私钥对应一个地址。该控制私钥和地址将用于控制硬件钱包转账。用户注册时 APP 服务器端需要从客户端获得用户的账号 ID（邮件地址或手机号）和控制私钥对应的地址（钱包控制地址）。APP 服务器端将账号 ID 和控制地址绑定，以后创建钱包和转账都由该地址和它对应的私钥控制。

以下实例代码，APP 端为 Java，APP 服务器端为 NodeJS。

a. 用户注册

整体流程：app 创建本地私钥和地址，利用本地私钥对验证数据进行签名，将签名数据和被签名数据发送至服务器，服务器对数据进行验证，验证成功后将账号 ID 和钱包控制地址存入数据库用户表。

app：

1、 app 生成本地私钥和地址

```
import org.web3j:core:3.3.1-android
ECKeyPair ecKeyPair = Keys.createEcKeyPair();
BigInteger privateKeyInDec = ecKeyPair.getPrivateKey();
String sPrivatekeyInHex = privateKeyInDec.toString(16);
WalletFile aWallet = Wallet.createLight(pass, ecKeyPair);
String sAddress = aWallet.getAddress();
```

2、创建签名数据

```
String mm1 ="Ethereum Signed Message:\n";
String mm2 = "Welcome to SealBlock. Please sign to login. " + new Date().getTime();
String params = mm1 + mm2.length() + mm2;
byte[] paramsByte = params.toString().getBytes();
byte[] x = Tool.hexStringToByteArray("19");
byte[] px = new byte[x.length+paramsByte.length];
System.arraycopy(x, 0, px, 0, x.length);
System.arraycopy(paramsByte, 0, px, x.length, paramsByte.length);
```

3、使用本地私钥签名数据

```
Credentials credentials = Credentials.create(privateKey);
Sign.SignatureData signature = Sign.signMessage( px, credentials.getEcKeyPair());
```

4、拼接 RSV 签名数据

```
byte[]    = new byte[signature.getR().length+signature.getS().length+1];
System.arraycopy(signature.getR(), 0, bytes, 0, signature.getR().length);
System.arraycopy(signature.getS(), 0, bytes, signature.getR().length, signature.getS().length);
System.arraycopy(signature.getV(), 0, bytes, signature.getR().length+signature.getS().length, 1);
String sig = Numeric.toHexString(bytes);
```

5、向 APP 服务器发送数据：

```
FirstName -> 名
LastName -> 姓
Email ->      邮箱
ApprovalAddress -> sAddress
```

Message -> mm2

Signature -> sig

6、返回注册是否成功的结果

服务器：

Url -> 服务器地址/app/signup

1、获取参数：

1)、FirstName

2)、LastName

3)、Email

4)、ApprovalAddress

5)、Message

6)、Signature

2、拼接验证数据

```
const sig = ethUtil.fromRpcSig(Signature);
const prefix = new Buffer("\x19Ethereum Signed Message:\n");
const prefixedMsg = ethUtil.sha3(
        Buffer.concat([prefix, new Buffer(String(Message.length)), ethUtil.toBuffer(Message)])
);
```

3、利用验证数据和签名信息获取 app 钱包地址

```
const publicKey = ethUtil.ecrecover(prefixedMsg, sig.v, sig.r, sig.s);
const addrBuf = ethUtil.pubToAddress(publicKey);
const address = ethUtil.bufferToHex(addrBuf);
if (ApprovalAddress == address) { //verification succeeded
    //save email and ApprovalAddress to user table
    //report success to APP
} else {
    //report failure to APP
}
```

b. 创建钱包

整体流程：服务器端对硬件钱包 APP 发起钱包创建请求，SealBlock 硬件创建钱包并返回钱包地址。

服务器：

1、从数据库获取用户的 ApprovalAddress

调用硬件热钱包链接接口(url -> 硬件热钱包/create_wallet_mobile)

```
var client = new zerorpc.Client();
var rpcEndPoint = 'tcp://67.134.36.67:6665';     //测试硬件包 IP 及端口
client.connect(rpcEndPoint);
 const method = 'create_wallet_mobile';
 const data = [ApprovalAddress + "||999000000000000000000||", false];
 client.invoke(method, data, function (error, result, more) {
      //result is wallet address
      //save wallet address to db
 }
```

c. 转账

整体流程：APP 端拼接交易信息数据，利用本地私钥对数据进行签名，将数据和签名拼接成的数据发送至 APP 服务器，APP 服务器对数据进行解析，创建交易，再次进行拼接，发送至硬件热钱包，硬件热钱包校验交易请求，使用钱包私钥对交易进行数字签名，返回给 APP 服务器，APP 服务器将交易数据和数字签名合成以后上链，获得交易 ID，返回给 APP 端。

app:

1、拼接交易信息

```
String mm1 ="Ethereum Signed Message:\n";
String txtype = “00000000000000000000000000000000000000000000”; //全 0 代表 ETH
```

```
String txid = String.valueOf(new Date().getTime());
while (txid.length() < 16) {txid = txid + (int)(Math.random()*9);}
String mm2 = txtype + ':' + fromAddr +':'+ toAddr +':'+ amount+':'+ txid;
String params = mm1 + mm2.length() + mm2;
byte[] paramsByte = params.getBytes();
byte[] x = Tool.hexStringToByteArray("19");
byte[] px = new byte[x.length+paramsByte.length];
System.arraycopy(x, 0, px, 0, x.length);
System.arraycopy(paramsByte, 0, px, x.length, paramsByte.length);
```

2、利用本地私钥签名

```
Credentials credentials = Credentials.create(localPrivateKey);
Sign.SignatureData signature = Sign.signMessage(px, credentials.getEcKeyPair());
```

3、拼接发送数据

```
byte[] bytes = new byte[signature.getR().length + signature.getS().length + 1];
System.arraycopy(signature.getR(), 0, bytes, 0, signature.getR().length);
System.arraycopy(signature.getS(), 0, bytes, signature.getR().length, signature.getS().length);
System.arraycopy(signature.getV(), 0, bytes, signature.getR().length+signature.getS().length, 1);
String sig = Numeric.toHexString(bytes);
String data = txtype+'|'+fromAddr+'|'+toAddr+'|'+amount+'|'+ txid +'|'+ApprovalAddress+'|'+ sig;
```

txtype -> 交易类型

fromAddr -> 发送方

toAddr -> 接收方

amount -> 数量

txid -> 使用时间戳生成，部位至 16 位

ApprovalAddress -> 钱包控制地址

sig -> 钱包控制私钥生成的数字签名

4、将数据发送至服务器

服务器：

Url -> 服务器地址/transfer

1、传递参数：

1. datas

2、创建交易数据

```
//parse datas and get fromAddr, toAddr, amount
web3.eth.getTransactionCount('0x' + fromAddr, web3.eth.defaultBlock.pending, function (err,result) {
    const nonce = web3.toHex(result);
    const txParams = {
      nonce: nonce,
      gasPrice: web3.toHex(5000000000),
      gasLimit: web3.toHex(91000),
      from: '0x' + fromAddr,
      to: '0x' + toAddr,
      value: web3.toHex(amount),
      // EIP 155 chainId - mainnet: 1, ropsten: 3, Rinkeby 4
      chainId: '0x04'
    };
    const tx = new ethereumTx.Tx(txParams);
    const rlpEncoded = tx.hash(false, true);
    const rlpHex = rlpEncoded.toString('hex')
})
```

3、拼接硬件热钱包服务器新参数

```
const new_datas = datas[0] + '|' + datas[1] + '|' + datas[2] + '|' + datas[3] + '|' +rlpHex + '|' + datas[4] + '|' +
                  datas[5] + '|' + datas[6];
```

datas[0] -> 交易类型

datas[1] -> 发送方

datas[2] -> 接收方

datas[3] -> 交易额

rlpHex -> 交易原始数据

datas[4] -> 使用时间戳生成，部位至 16 位

datas[5] -> 钱包控制地址

datas[6] -> 使用控制私钥生成的签名

4、将 new_datas 发送至硬件热钱包

```
const method = 'create_wallet_mobile';
client.invoke(method, new_datas, function (error, result, more) {
    //if succeed, result is the signature signed by wallet private key
    //if fail, result is an error code
}
```

5、硬件热钱包返回签名信息，合成签名的数据和交易的数据

```
var tt = ethUtil.toBuffer("0x" + result)
const r = tt.slice(0, 32)
const s = tt.slice(32, 64)
const v = tt.slice(64, 65)[0]
tx.setRSV(ethUtil. bufferToHex (r), ethUtil. bufferToHex (s), v)
```

6、将交易信息广播

```
web3.eth.sendRawTransaction('0x' + tx.serialize().toString('hex'), function (err, hash) {
    //if succeed, hash is the transaction ID
    //return transaction ID to APP
}
```