问题汇总

问题描述

现在要进行的签名验签操作是，在用户登录时

1. 首先在前端对登录的表单数据进行 sm2 签名
2. 而后将表单数据、签名结果打在后端接口上
3. 后端解析表单数据，从数据库中读出用户对应的证书 cert
4. 通过该 cert 以及传来的原文（表单数据）、签名结果进行验签

在验签时报错

```
2024-06-20 11:27:47.603 ERROR 4599 --- [nio-8090-exec-1]
c.x.b.s.signverify.SignVerifyUtil        : verifySignedData failed,响应为空
---------->验签出错
com.xidian.bankdemo.security.signverify.exception.SVSResponseException:
verifySignedData failed,响应为空
        at
com.xidian.bankdemo.security.signverify.SignVerifyUtil.verifySignedData(SignVeri
fyUtil.java:106)
        at
com.xidian.bankdemo.security.signverify.OlymSignature.verify(OlymSignature.java:
103)
        at
com.xidian.bankdemo.security.signverify.OlymSignature.verifySignature(OlymSignat
ure.java:113)
        at
com.xidian.bankdemo.service.impl.UserServiceImpl.doLogin(UserServiceImpl.java:60
)
        at
com.xidian.bankdemo.controller.UserController.doLogin(UserController.java:88)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.jav
a:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at
org.springframework.web.method.support.InvocableHandlerMethod.doInvoke(Invocable
HandlerMethod.java:197)
```

详细过程

> 一次登录过程

前端签名结果

```
钥匙证书为:
MIIBYzCCAQmgAwIBAgIJAK33NOW5vW/DMAoGCCqBHM9VAYN1MDkxCzAJBgNVBAYTAkNOMRIwEAYDVQQI
DAlHdWFuZ0RvbmcxFjAUBgNVBAoMDU9seW0gVGVjaCBMdGQwHhcNMjQwNjE4MDgzOTI0WhcNMjcwMzE1
MDgzOTI0WjAZMRcwFQYDVQQDDA53dWZ5QG15aWJjLm5ldDBZMBMGByqGSM49AgEGCCqBHM9VAYItA0IA
BGl7U9JSlgD/Ig1/XSBqbhfxFa/h3JaMpF8HJMRofHULBSgAE8FQqmfzzVlcvtWSE5f7TEIDXsA6S4xy
PtnZU9OjGjAYMAkGA1UdEwQCMAAwCwYDVR0PBAQDAgXgMAoGCCqBHM9VAYN1A0gAMEUCIQCUEC6CG4ZB
dfcpPyZZJFTKlg5C8GwbUplZtwiITCipVwIgC7+SpcFB9BcOpbOHEvPJ/u84//46Ntz4w62X+NTr4Go=

前端签名原文为:
eyJ1c2VybmFtZSI6Inh6dDEiLCJwYXNzd29yZCI6Ilh6dDAxMTAyNiIsImtleSI6IjEyMzQ1NiIsImNv
ZGUiOiI3NmdLIiwic2lnbmF0dXJlIjpudWxsLCJpbmlEYXRhIjoie1widXNlcm5hbWVcIjpcInh6dDFc
IixcInBhc3N3b3JkXCI6XCJYenQwMTEwMjZcIixcImtleVwiOlwiMTIzNDU2XCIsXCJjb2RlXCI6XCJc
IixcInNpZ25hdHVyZVwiOm51bGwsXCJpbmlEYXRhXCI6bnVsbH0ifQ==

前端签名所得密文为:
zUYS8qAGQVsythDvLU/WvESw/3aY/OAZJwnN1MUlSERPF6WRMIibmQKzkgtmaNN0QoF0HZhsQDmjcwFX
hzXcoQ==
```

后端验签过程

```
验签所使用的证书为:
MIIBYzCCAQmgAwIBAgIJAK33NOW5vW/DMAoGCCqBHM9VAYN1MDkxCzAJBgNVBAYTAkNOMRIwEAYDVQQI
DAlHdWFuZ0RvbmcxFjAUBgNVBAoMDU9seW0gVGVjaCBMdGQwHhcNMjQwNjE4MDgzOTI0WhcNMjcwMzE1
MDgzOTI0WjAZMRcwFQYDVQQDDA53dWZ5QG15aWJjLm5ldDBZMBMGByqGSM49AgEGCCqBHM9VAYItA0IA
BGl7U9JSlgD/Ig1/XSBqbhfxFa/h3JaMpF8HJMRofHULBSgAE8FQqmfzzVlcvtWSE5f7TEIDXsA6S4xy
PtnZU9OjGjAYMAkGA1UdEwQCMAAwCwYDVR0PBAQDAgXgMAoGCCqBHM9VAYN1A0gAMEUCIQCUEC6CG4ZB
dfcpPyZZJFTKlg5C8GwbUplZtwiITCipVwIgC7+SpcFB9BcOpbOHEvPJ/u84//46Ntz4w62X+NTr4Go=

验签所使用的原文为:
eyJ1c2VybmFtZSI6Inh6dDEiLCJwYXNzd29yZCI6Ilh6dDAxMTAyNiIsImtleSI6IjEyMzQ1NiIsImNv
ZGUiOiJIdFFTIiwic2lnbmF0dXJlIjpudWxsLCJpbmlEYXRhIjoie1widXNlcm5hbWVcIjpcInh6dDFc
IixcInBhc3N3b3JkXCI6XCJYenQwMTEwMjZcIixcImtleVwiOlwiMTIzNDU2XCIsXCJjb2RlXCI6XCJI
dFFTXCIsXCJzaWduYXR1cmVcIjpudWxsLFwiaW5pRGF0YVwiOm51bGx9In0=

验签所使用的签名密文为:
YStwtM/vUaz5ZXLRr5iCTiCM51kjhWf+HLAqLV1SzFeVFlrpxndadK7yJAmZZdB6grjx4OPTCrg4rhfq
z8hvXw==

验签所使用的签名ID为:
MTIzNDU2NzgxMjM0NTY3OA==
```

另外，我在后端还对相同的原文用另一个用户的私钥进行签名（这里传的是私钥的索引值 101），得到的结果是

```
MEYCIQDsEdVM1jsQQOS5idaqgKzzerTTla17sdLYCTHgD3ZzpQIhAM1oSG6flXIlrhIyPch3jFJOjM1o
wWHXh6k3pgk3yfnX
```

这个签名是 96 个字符，而前端的签名为 88

问题排查

后端单元测试

首先我对后端功能进行了单元测试，提供了一个测试接口，对表单数据先进行加密，而后验证，即单独在后端进行签名验签操作（而非前端签名，后端验签），这个过程能够正常返回结果 true

```
@CrossOrigin
@GetMapping("/hello") // 在config.SecurityConfig设置白名单访问，便于curl测试
public Object sayHello() throws Exception{
    Map<String, Object> map = new HashMap<>();
    String iniData = "
{\"username\":\"wx\",\"password\":\"123456xW\",\"key\":\"12345678\",\"code\":\"f
iN2\",\"signature\":null,\"iniData\":null}";
    String signature = OlymSignature.signature(iniData);
    // cert为null时调用写死的证书

    return OlymSignature.verifySignature(null, iniData, signature);
}
```

调用的是一个已经上传的证书，签名请求传参如下

```
// 单包数字签名请求封装
SignDataReq signDataReq = new SignDataReq();
signDataReq.setSignMethod(SGD_SM3_SM2); // SGD_SM3_SM2 标识: 131585
signDataReq.setKeyIndex(KeyIndex); // 私钥索引: 101
signDataReq.setKeyValue(keyValue); // Base64("123456")
signDataReq.setSignerID(signerID); // Base64("1234567812345678")
signDataReq.setSignerIDLen(signerIDLen);
signDataReq.setInData(inData); // 原文字符串的 Base64 编码
signDataReq.setInDataLen(inDataLen);
String signature = SignVerifyUtil.signData(signDataReq);
```

验签请求传参如下

```
VerifySignedDataReq verifySignedDataReq = new VerifySignedDataReq();
verifySignedDataReq.setSignMethod(SGD_SM3_SM2);
verifySignedDataReq.setType(Type); // 为 1，表示使用证书验证，而非 certSN
verifySignedDataReq.setCert(cert); // 证书字符串
verifySignedDataReq.setInData(inData); // Base64 编码过的原文字符串
verifySignedDataReq.setInDataLen(inDataLen); // 原文字符串长度
verifySignedDataReq.setSignerID(signerID); // "1234567812345678"
verifySignedDataReq.setSignerIDLen(signerIDLen);
verifySignedDataReq.setSignature(signature); // 签名密文
verifySignedDataReq.setVerifyLevel(VerifyLevel); // 验证等级: 0
System.out.println(verifySignedDataReq);
Integer verifySignedDataResult =
SignVerifyUtil.verifySignedData(verifySignedDataReq);
```

证书是否一致

而后我检查了验签时所用证书是否一致，因为前端是从钥匙里读的证书，后端是从数据库中取出的证书（在注册过程写进去的），发现是一致的，前后端均为

```
MIIB7jCCAZKgAwIBAgIIHSSy3+yV4HQwDAYIKoEcz1UBg3UFADBDMQswCQYDVQQGEwJDTjEaMBgGA1UE
CgwRWGlkaWFuIFVuaXZlcnNpdHkxGDAWBgNVBAMMD1hJRElBTiBDWUJFUiBDQTAeFw0yNDAzMjgwNzI1
MzNaFw0yNzAzMjgwNzI1MzNaMGIxCzAJBgNVBAYTAkNOMQ8wDQYDVQQIDAZTaGFuWGkxGjAYBgNVBAoM
EVhpZGlhbiB1bml2ZXJzaXR5MSYwJAYDVQQDDB1YaWRpYW5fU2lnbmF0dXJlIHZlcmlmaWNhdGlvbjBZ
MBMGByqGSM49AgEGCCqBHM9VAYItA0IABPVXtOQJu0NT6r8BPFDQsvOqNJcWXKp7w5dGL8WAQ+ZX3rIN
l41wfSirYd35url2cqbhdQR9iXw9UY2DGdAhhCejTzBNMB8GA1UdIwQYMBaAFD4j5ykzXr4eK1uIBAyK
s/S5TFhPMAsGA1UdDwQEAwIHgDAdBgNVHQ4EFgQUuF6F6xLUEaPnKMMYG6WypZfdnaEwDAYIKoEcz1UB
g3UFAANIADBFAiAMOLcQ948pZUpujQjx7peUGQi2187TvlKPgj5Elkxy6gIhAJpOW5tVpw5sqFMud+UD
1zEoFsrPVUrT3OOcFYkLh7bt
```

其实他原来是这样的

```
-----BEGIN CERTIFICATE-----
MIIBYzCCAQmgAwIBAgIJAK33NOW5vW/DMAoGCCqBHM9VAYN1MDkxCzAJBgNVBAYT
AkNOMRIwEAYDVQQIDAlHdWFuZ0RvbmcxFjAUBgNVBAoMDU9seW0gVGVjaCBMdGQw
HhcNMjQwNjE4MDgzOTI0WhcNMjcwMzE1MDgzOTI0WjAZMRcwFQYDVQQDDA53dWZ5
QG15aWJjLm5ldDBZMBMGByqGSM49AgEGCCqBHM9VAYItA0IABGl7U9JSlgD/Ig1/
XSBqbhfxFa/h3JaMpF8HJMRofHULBSgAE8FQqmfzzVlcvtWSE5f7TEIDXsA6S4xy
PtnZU9OjGjAYMAkGA1UdEwQCMAAwCwYDVR0PBAQDAgXgMAoGCCqBHM9VAYN1A0gA
MEUCIQCUEC6CG4ZBdfcpPyZZJFTKlg5C8GwbUplZtwiITCipVwIgC7+SpcFB9BcO
pbOHEvPJ/u84//46Ntz4w62X+NTr4Go=
-----END CERTIFICATE-----
```

为了完全排除证书问题，我去掉了`-----BEGIN CERTIFICATE-----`和换行`\n`，没用

编码问题

而后，我仔细看了前端签名的接口文档

字段名	字段类型	是否必须	字段描述
action	string	是	sm2_sign
action_id	string	否	用于异步消息响应匹配，由用户管理自定义数据内容
keyindex	string	是	设备索引
pin	string	是	PIN码，需web调用“gen_and_getpubkey”对该字段加密
open_cipher_pin	string	否	若为”1”，则表示pin受保护，会先对pin做解密；若为”0”或者本字段缺省，则表示pin未受保护
container	string	是	容器索引
message	base64	是	需要签名的消息
hashtype			表明当前数据是否已经摘要处理过 none：表明为原始数据，需要websocket服务计算摘要 sm3:表明数据已经进行过SM3摘要，不需要websocket服务计算摘要，此值时，数据长度必须为32字节
format			签名后数据的格式 none：表明签名后的数据为裸的签名结果 asn.1: 表明签名后的数据进行ASN.1的编码

以及给我提供的函数接口

```
data_sm2_signature: function (keyindex, container, pass, challenge, hashtype,
format, callback) {
    //sm2 签名
    if(typeof hashtype == undefined || hashtype == ''){
        hashtype = 'none';
    }
    if(typeof format == undefined || format == ''){
        format = 'none';
    }
    var json = {
        "action": "sm2_sign",
        "pin": pass,
        "keyindex": keyindex,
        "container": container,
        "message": challenge,
        "hashtype": hashtype,
        "format": format
    };
    /*if (this._events.hasOwnProperty(json.action)) {
      return;
    }*/
    // 存储回调
    this.addEvent(json.action, callback);
    this.sendMessage(json);
    return this;
}
```

这个接口是缺省了`open_cipher_pin`字段的，故 pin 码直接作为原字符串传进来就行，我这里是 "123456"，另外 message 字段需要 Base64 编码，我前端的处理如下

```
encodeBase64(str){
    return Buffer.from(str, 'utf8').toString('base64');
};
```

我特地检查了一下这里的 Base64 编码是否正确，后端的编码方式如下

```
Base64.getEncoder().encodeToString(iniData.getBytes());
```

编码结果保持一致

代码附录

前端

```
// 登录
handleLogin() {
    this.$refs.loginForm.validate(valid => {
        valid = valid && this.ret
        if (valid) {
            this.loading = true
            this.loginForm.iniData = JSON.stringify(this.loginForm);
            console.log(this.loginForm.iniData);
            let inData = this.encodeBase64(this.loginForm.iniData); // Base64编码
```

```
            let ukey_val = this.keyindex;
            let identity_val = this.container;
            let pass = this.loginForm.key;
            let hashtype = "";
            let format = "";
            ntlsUtil.func.data_sm2_signature(ukey_val, identity_val, pass,
inData, hashtype, format, this.doLogin);
        } else {
            alert("error submit!")
            return false
        }
    })
};
// 回调函数
doLogin(message){
    if(!message || !message.data){
        alert("签名失败，请检查UKey是否插入或PIN码是否正确，或PIN码是否被锁定");
        this.loading = false;
        return false;
    }
    this.loginForm.signature = message.data;
    // 打请求
    this.$store.dispatch('user/login', this.loginForm).then(() => {
        this.$router.push({ path: this.redirect || '/' })
        this.loading = false
    }).catch(() => {
        this.loading = false
    })
};
```

后端

```
package com.xidian.bankdemo.security.signverify;

import com.xidian.bankdemo.security.signverify.properties.SignVerifyProperties;
import com.xidian.bankdemo.security.signverify.pojo.*;

import java.util.Base64;
import java.util.Map;
import java.util.Objects;

public class OlymSignature {
    private static final Integer KeyIndex = 101;
    private static final String Password = "123456";
    private static final Integer Type = 1;
    private static final Integer VerifyLevel = 0;
    private static final String SignerID = "1234567812345678";
    private static final Integer SGD_SM3_SM2 = 131585;
```

```
    private static final String Cert =
"MIIB7jCCAZKgAwIBAgIIHSSy3+yV4HQwDAYIKoEcz1UBg3UFADBDMQswCQYDVQQGEwJDTjEaMBgGA1U
ECgwRWGlkaWFuIFVuaXZlcnNpdHkxGDAWBgNVBAMMD1hJRElBTiBDWUJFUiBDQTAeFw0yNDAzMjgwNzI
1MzNaFw0yNzAzMjgwNzI1MzNaMGIxCzAJBgNVBAYTAkNOMQ8wDQYDVQQIDAZTaGFuWGkxGjAYBgNVBAo
MEVhpZGlhbiB1bml2ZXJzaXR5MSYwJAYDVQQDDB1YaWRpYW5fU2lnbmF0dXJlIHZlcmlmaWNhdGlvbjB
ZMBMGByqGSM49AgEGCCqBHM9VAYItA0IABPVXtOQJu0NT6r8BPFDQsvOqNJcWXKp7w5dGL8WAQ+ZX3rI
Nl41WfSirYd35url2cqbhdQR9iXW9UY2DGdAhhCejTzBNMB8GA1UdIwQYMBaAFD4j5ykzXr4eK1uIBAy
Ks/S5TFhPMAsGA1UdDwQEAwIHgDAdBgNVHQ4EFgQUuF6F6xLUEaPnKMMYG6WypZfdnaEwDAYIKoEcz1U
Bg3UFAANIADBFAiAMOLcQ948pZUpujQjx7peUGQi2187TvlKPgj5Elkxy6gIhAJpOW5tVpw5sqFMud+U
D1zEoFsrPVUrT3OOcFYkLh7bt";
    private static final String CertCN = "";
    public static final Integer SVSRESPONSE_RESPVALUE_SUCCESS = 0;

    // 单包数字签名
    public static String signature(String in) throws Exception {
        byte[] signerIDBytes = SignerID.getBytes();
        Integer signerIDLen = signerIDBytes.length; // 签名者长度
        String signerID = Base64.getEncoder().encodeToString(signerIDBytes);

        byte[] inDataBytes = in.getBytes();
        Integer inDataLen = inDataBytes.length;
        System.out.println(inDataLen);
        // 尝试不编码
        String inData = Base64.getEncoder().encodeToString(inDataBytes); // 对原文
Base64编码
        String keyValue =
Base64.getEncoder().encodeToString(Password.getBytes()); // PIN码Base64编码


        // 单包数字签名请求封装
        SignDataReq signDataReq = new SignDataReq();
        signDataReq.setSignMethod(SGD_SM3_SM2); // 131585
        signDataReq.setKeyIndex(KeyIndex); // 101
        signDataReq.setKeyValue(keyValue); // Base64(123456)
        signDataReq.setSignerID(signerID); // Base64(1234567812345678)
        signDataReq.setSignerIDLen(signerIDLen);
        signDataReq.setInData(inData); // Base64(原文字符串数据)
        signDataReq.setInDataLen(inDataLen);
        String signature = SignVerifyUtil.signData(signDataReq);

        System.out.println(signature);

        return signature;
    }

    public static String generateSignature(Map<String, Object> map) {
        String signed = "";
        try{
            signed = signature(map.toString());
        }catch (Exception e){
            System.out.println("---------->签名出错");
            e.printStackTrace();
        }
        return signed;
    }


    // 单包数字签名验证
```

```
    public static boolean verify(String cert, String inData, String signature)
throws Exception {
        if(cert == null){
            cert = Cert;
        }
        // 编码过了
        // cert = Base64.getEncoder().encodeToString(cert.getBytes());
        // signature = Base64.getEncoder().encodeToString(signature.getBytes());
        byte[] signerIDBytes = SignerID.getBytes();
        Integer signerIDLen = signerIDBytes.length;
        String signerID = Base64.getEncoder().encodeToString(signerIDBytes);

        byte[] inDataBytes = inData.getBytes();
        Integer inDataLen = inDataBytes.length;
        System.out.println(inDataLen);
        inData = Base64.getEncoder().encodeToString(inDataBytes);

        System.out.println("验签所使用的证书为: " + cert);
        System.out.println("验签所使用的原文为: " + inData);
        System.out.println("验签所使用的签名密文为: " + signature);
        System.out.println("验签所使用的签名ID为: " + signerID);

        VerifySignedDataReq verifySignedDataReq = new VerifySignedDataReq();
        verifySignedDataReq.setSignMethod(SGD_SM3_SM2);
        verifySignedDataReq.setType(Type);
        verifySignedDataReq.setCert(cert);
        // Type为1时certSN没用，将使用cert
        // verifySignedDataReq.setCertSN(certSN);
        verifySignedDataReq.setInData(inData);
        verifySignedDataReq.setInDataLen(inDataLen);
        // 这个ID默认为"1234567812345678"
        verifySignedDataReq.setSignerID(signerID);
        verifySignedDataReq.setSignerIDLen(signerIDLen);
        verifySignedDataReq.setSignature(signature);
        verifySignedDataReq.setVerifyLevel(VerifyLevel);
        System.out.println(verifySignedDataReq);
        Integer verifySignedDataResult =
SignVerifyUtil.verifySignedData(verifySignedDataReq);
        System.out.println(
                "单包验证数字签名结果：" +
Objects.equals(SVSRESPONSE_RESPVALUE_SUCCESS, verifySignedDataResult));
        return SignVerifyUtil.verifySignedData(verifySignedDataReq) == 0;
    }


    public static boolean verifySignature(String cert, String inData, String
signed){
        boolean flag = false;
        try{
            flag = verify(cert, inData, signed);
        }catch (Exception e){
            System.out.println("---------->验签出错");
            e.printStackTrace();
        }
        return flag;
    }
}
```