信息安全

HTTPS是通信层的，作用是保护通信不被窃听，服务器不被他人仿冒。
SHA和MD5是不可逆的，只能用作签名。MD5和HTTPS的区别在于，MD5是应用层（相对HTTPS）的，防止收到有业务数据在中途被人篡改，防止被未授权的人调用
业务层还有RSA、DES的对称和非对称加密。这个才是真正的加密（encrypt, and decrypt）。md5只是hash摘要，不是加密。
https, hash, des三种是可以同时存在的，也是有必要同时存在的。我们的运维平台就是js端调服务端的open api，同时使用了这三种安全技术：https://www.ops.best/#/product/advantages/security
总结一下
1.https防止攻击者通过公共wifi窃听客户的账号密码等信息。
2.api签名防止攻击者cc，csrf，replay attack。
3.des防止内部员工直接查看客户敏感信息，万一我们被拖库了，客户也是安全的。
4.RSA、DES是防止数据被劫持，从而泄漏用户数据

RSA加密

用RSA非对称加密方式实现。后台生成rsa密钥对，然后在页面设置rsa公钥，提交时用公钥加密密码，生成的密文传到后台，后台再用私钥解密，获取密码明文。这样客户端只需要知道rsa加密方式和公钥，前台不知道私钥是无法解密的。
不过RSA效率不是特别高，只适合低数据量的情况，高数据量使用DES
使用
1、需要到http://www.bouncycastle.org 下载bcprov-jdk16-140.jar文件
2、前端引入Barrett.js、BigInt.js和RSA.js文件

```
encryptedString : (function(paramStr, rsaKey){
    setMaxDigits(130);
    //第一个参数为加密指数、第二个参数为解密参数、第三个参数为加密系数
    key = new RSAKeyPair("10001", "", rsaKey);
    //返回加密后的字符串
    return encryptedString(key, encodeURIComponent(paramStr));
        })
```

3、后台加密解密

```
import java.io.ByteArrayOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
```

```
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.math.BigInteger;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.SecureRandom;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.RSAPrivateKeySpec;
import java.security.spec.RSAPublicKeySpec;

import javax.crypto.Cipher;

import org.apache.commons.lang.StringUtils;

import com.jd.uwp.common.Constants;

/**
* RSA 工具类。提供加密，解密，生成密钥对等方法。
* 需要bcprov-jdk16-140.jar包。
*
*/
public class RSAUtil {

        private static String RSAKeyStore = "RSAKey.txt";

        /**
         * * 生成密钥对 *
         * @return KeyPair *
         * @throws EncryptException
         */
        public static KeyPair generateKeyPair(String basePath) throws Exception
{
    try {
```

```
        KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("RSA",
        new org.bouncycastle.jce.provider.BouncyCastleProvider());
        //大小
        final int KEY_SIZE = 1024;
        keyPairGen.initialize(KEY_SIZE, new SecureRandom());
        KeyPair keyPair = keyPairGen.generateKeyPair();
        saveKeyPair(keyPair, basePath);
        return keyPair;
    } catch (Exception e) {
        throw new Exception(e.getMessage());
    }
        }

        /**
         * 获取密钥对
         * @return
         * @throws Exception
         */
        public static KeyPair getKeyPair(String basePath) throws Exception {
    FileInputStream fis = new
FileInputStream(StringUtils.isNotBlank(basePath) ? (basePath + RSAKeyStore) :
RSAKeyStore);
    ObjectInputStream oos = new ObjectInputStream(fis);
    KeyPair kp = (KeyPair) oos.readObject();
    oos.close();
    fis.close();
    return kp;
        }

        /**
         * 保存密钥
         * @param kp
         * @throws Exception
         */
        public static void saveKeyPair(KeyPair kp, String basePath) throws
Exception {
    FileOutputStream fos = new
FileOutputStream(StringUtils.isNotBlank(basePath) ? (basePath + RSAKeyStore) :
RSAKeyStore);
```

```
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    // 生成密钥
    oos.writeObject(kp);
    oos.close();
    fos.close();
        }

        /**
         * * 生成公钥 *
         * @param modulus *
         * @param publicExponent *
         * @return RSAPublicKey *
         * @throws Exception
         */
        public static RSAPublicKey generateRSAPublicKey(byte[] modulus,
    byte[] publicExponent) throws Exception {
    KeyFactory keyFac = null;
    try {
        keyFac = KeyFactory.getInstance("RSA",
        new org.bouncycastle.jce.provider.BouncyCastleProvider());
    } catch (NoSuchAlgorithmException ex) {
        throw new Exception(ex.getMessage());
    }
    RSAPublicKeySpec pubKeySpec = new RSAPublicKeySpec(new BigInteger(
        modulus), new BigInteger(publicExponent));
    try {
        return (RSAPublicKey) keyFac.generatePublic(pubKeySpec);
    } catch (InvalidKeySpecException ex) {
        throw new Exception(ex.getMessage());
    }
        }

        /**
         * * 生成私钥 *
         * @param modulus *
         * @param privateExponent *
         * @return RSAPrivateKey *
         * @throws Exception
         */
```

```
        public static RSAPrivateKey generateRSAPrivateKey(byte[] modulus,
    byte[] privateExponent) throws Exception {
    KeyFactory keyFac = null;
    try {
        keyFac = KeyFactory.getInstance("RSA",
        new org.bouncycastle.jce.provider.BouncyCastleProvider());
    } catch (NoSuchAlgorithmException ex) {
        throw new Exception(ex.getMessage());
    }
    RSAPrivateKeySpec priKeySpec = new RSAPrivateKeySpec(new BigInteger(
        modulus), new BigInteger(privateExponent));
    try {
        return (RSAPrivateKey) keyFac.generatePrivate(priKeySpec);
    } catch (InvalidKeySpecException ex) {
        throw new Exception(ex.getMessage());
                }
        }

        /**
         * * 加密 *
         * @param key
         *            加密的密钥 *
         * @param data
         *            待加密的明文数据 *
         * @return 加密后的数据 *
         * @throws Exception
         */
        public static byte[] encrypt(PublicKey pk, byte[] data) throws
Exception {
    try {
        Cipher cipher = Cipher.getInstance("RSA",
        new org.bouncycastle.jce.provider.BouncyCastleProvider());
        cipher.init(Cipher.ENCRYPT_MODE, pk);
        // 获得加密块大小，如：加密前数据为128个byte，而key_size=1024
        int blockSize = cipher.getBlockSize();
        // 加密块大小为127
        // byte,加密后为128个byte;因此共有2个加密块，第一个127
        // byte第二个为1个byte
        int outputSize = cipher.getOutputSize(data.length);// 获得加密块加密后块大
```

```
小
        int leavedSize = data.length % blockSize;
        int blocksSize = leavedSize != 0 ? data.length / blockSize + 1 :
data.length / blockSize;
        byte[] raw = new byte[outputSize * blocksSize];
        int i = 0;
        while (data.length - i * blockSize > 0) {
                if (data.length - i * blockSize > blockSize) {
                    cipher.doFinal(data, i * blockSize, blockSize, raw, i *
outputSize);
                } else {
                    cipher.doFinal(data, i * blockSize, data.length - i *
blockSize, raw, i * outputSize);
                }
                i++;
        }
        return raw;
    } catch (Exception e) {
        throw new Exception(e.getMessage());
    }
        }

        /**
         * * 解密 *
         *
         * @param key
         *            解密的密钥 *
         * @param raw
         *            已经加密的数据 *
         * @return 解密后的明文 *
         * @throws Exception
         */
        @SuppressWarnings("static-access")
        public static byte[] decrypt(PrivateKey pk, byte[] raw) throws
Exception {
    try {
        Cipher cipher = Cipher.getInstance("RSA", new
org.bouncycastle.jce.provider.BouncyCastleProvider());
        cipher.init(cipher.DECRYPT_MODE, pk);
```

```
            int blockSize = cipher.getBlockSize();
            ByteArrayOutputStream bout = new ByteArrayOutputStream(64);
            int j = 0;
            while (raw.length - j * blockSize > 0) {
                bout.write(cipher.doFinal(raw, j * blockSize, blockSize));
                j++;
            }
            return bout.toByteArray();
        } catch (Exception e) {
            throw new Exception(e.getMessage());
        }
            }

/**
* 解密方法
* paramStr    ->密文
* basePath    ->RSAKey.txt所在的文件夹路径
**/
        public static String decryptStr(String paramStr, String basePath)
throws Exception{
    byte[] en_result = new BigInteger(paramStr, 16).toByteArray();
    byte[] de_result = decrypt(getKeyPair(basePath).getPrivate(), en_result);
    StringBuffer sb = new StringBuffer();
    sb.append(new String(de_result));
    //返回解密的字符串
    return sb.reverse().toString();
}
}
```

4、前后端调用方法

```
//前端 表单提交
$.ajax({
    url : contextPath + "test.action",
    //加密传输
    data : {pwd:encryptedString ($("#pwd").val(),
"adasdasdasdasdadsasdasdasdasd")},
    type  : "post",
    datatype : "json",
```

```
    success : function(retData){
    }
});

//后端解密代码
RSAUtil.decryptStr(paramMap.getString("pwd"),
request.getSession().getServletContext().getRealPath("/"));
```