Android 端加解密实现

详细的实现键 testtoolbar 中 RSAEncryptionUtil 类。

编码格式：aes 加解密后，js 和 android 端的加解密后的信息都应该是 Base64 格式的，

两者在交互时需要留意编码格式。

Java 中有专门用来公私钥加解密的 package：java.security.*

一、在整个结构中，Android 端作为私钥的拥有者，首先要做的便是生成公私钥对：

私钥：openssl genrsa -out privatekey.pem 1024

公钥：openssl rsa -in privatekey.pem -pubout -out publickey.pem

二、Android 端从 privatekey.pem 文件中读取私钥

privatekey.pem 文件存放在 assets 文件夹下，读取并实例化 RSAPrivateKey：

```
<java-code>
RSAPrivateKey privateKey = loadPrivateKeyFromFile(“privatekey.pem”);

public RSAPrivateKey loadPrivateKeyFromFile(String fileName){
InputStream inputStream = getResources().getAssets()
                                        .open(fileName);
try {
                BufferedReader br=
                    new BufferedReader(new InputStreamReader(inputStream));
                String readLine= null;
                StringBuilder sb= new StringBuilder();
                while((readLine= br.readLine())!=null){
                    if(readLine.charAt(0)=='-'){
                        continue;
                    }else{
                        sb.append(readLine);
                        sb.append('\r');
                    }
                }
                return loadPrivateKey(sb.toString());
```

```
        } catch (IOException e) {
            throw new Exception("私钥数据读取错误");
        } catch (NullPointerException e) {
            throw new Exception("私钥输入流为空");
        }
}

public RSAPrivateKey loadPrivateKey(String privateKeyStr) throws Exception{
        try {
            byte[] buffer= Base64.decode(privateKeyStr, Base64.DEFAULT);
            PKCS8EncodedKeySpec keySpec=
                                        new PKCS8EncodedKeySpec(buffer);
            KeyFactory keyFactory= KeyFactory.getInstance("RSA");
            return (RSAPrivateKey) keyFactory.generatePrivate(keySpec);
        } catch (NoSuchAlgorithmException e) {
            throw new Exception("无此算法");
        } catch (InvalidKeySpecException e) {
            throw new Exception("私钥非法");
        } catch (Exception e) {
            throw e;
        }
    }
</java-code>
```

三、rsa 解密 js 端发送过来的 AESKey(AESKey 已被 publickey 加密)

Js 端发送过来的数据应该是 base64 编码，以 string 形式发送过来。所以 Android 端在接

受到 String 后应该进行 Base64 解码，具体实现如下：

```
<java-code>
    public static final String ALGORITHM = "RSA/ECB/PKCS1Padding";

    public String decrypt(String ciphertext){
        byte[] text = Base64.decode(ciphertext,Base64.DEFAULT);
        byte[] dectyptedText = null;
        try{
            final Cipher cipher = Cipher.getInstance(ALGORITHM);
            cipher.init(Cipher.DECRYPT_MODE,privateKey);
            dectyptedText = cipher.doFinal(text);
```

```
        }catch (Exception e){
            e.printStackTrace();
        }
        return new String(dectyptedText);
    }
</java-code>
```

注意，解密后中文可能是乱码，需要将结果转为 utf8，即 new String(dectypedText,”UTF-8”).

四、使用 AESKey 加密消息发送给 js

aes 加密需要注意的时 google 提供的 crypto-js 的 aes 加密方案中编码补全方式只有 ZeroPadding，以及 NoPadding。但是作者在使用 java 中的 Cipher 中的 AES/CBC/NoPadding 与 crypto-js 中的 Nopadding 时没有调通，而据 oracle 给出的 API 文档中 ,并没有 ZeroPadding 模式。所以 js 采用的 padding 为 ZeroPadding,Android 端采用 NoPadding,这里就有一个中间的转换过程，具体方式见下面的加密过程，代码中的 key 即为 AESKey：

```
private String key;

public String encrypt(String msg){

    if(key == null)return null;
    try {
        Cipher cipher = Cipher.getInstance(‘AES/CBC/NoPadding‘);


        int blockSize = cipher.getBlockSize();


        byte[] dataBytes = msg.getBytes();
        int plaintextLength = dataBytes.length;
        if(plaintextLength % blockSize != 0){
```

```
                plaintextLength = plaintextLength + (blockSize /  (plaintextLength %
blockSize));
            }
            byte[] plaintext = new byte[plaintextLength];
            System.arraycopy(dataBytes, 0, plaintext, 0, dataBytes.length);
            SecretKeySpec keyspec = new SecretKeySpec(key.getBytes(), 'AES');
            IvParameterSpec ivspec = new IvParameterSpec(key.getBytes());
            cipher.init(Cipher.ENCRYPT_MODE, keyspec, ivspec);
            byte[] encrypted = cipher.doFinal(plaintext);
            return Base64.encodeToString(encrypted,Base64.DEFAULT);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (NoSuchPaddingException e) {
            e.printStackTrace();
        } catch (InvalidAlgorithmParameterException e) {
            e.printStackTrace();
        } catch (InvalidKeyException e) {
            e.printStackTrace();
        } catch (BadPaddingException e) {
            e.printStackTrace();
        } catch (IllegalBlockSizeException e) {
            e.printStackTrace();
        }
```

```
    return null;
}
```

五、Android 端 aes 解密 js 发送来的消息

Js 发送来的消息的编码格式也应为 base64，只是以 string 发送过来，所以需要进行

Base64 解码，具体见如下代码，其中的 key 即为 AESKey：

```
<java-code>
    public String aesDecrypt(String ciphertext) throws Exception {
        byte[] enc = Base64.decode(ciphertext, Base64.DEFAULT);
        SecretKeySpec keyspec = new SecretKeySpec(key.getBytes(),"AES");
        IvParameterSpec ivspec = new IvParameterSpec(key.getBytes());
        Cipher cipher = Cipher.getInstance("AES/CBC/NoPadding");


        cipher.init(Cipher.DECRYPT_MODE,keyspec,ivspec);
        byte[] plainText = cipher.doFinal(enc);

        return new String(plainText,"UTF-8");
    }
</java-code>
```

最后也需要将解密后的 plainText 转换为 UTF-8 格式，以免中文乱码。