Cơ chế bảo mật dữ liệu trong ứng dụng gamerid

1. **Server**:

- Sử dụng spring-security

- Oauth2 server

- Protocol: HTTPs

2. **Client**:

3. **Trao đổi dữ liệu client/server:**

a. *Bảo mật dữ liệu (Mã hóa dữ liệu truyền nhận)*:

- Protocol: HTTPs.

- Sử dụng cặp public/private key để mã hóa/giải mã.

- Publickey: sẽ là key chung dùng để mã hóa

- Privatekey: mỗi client sẽ có 1 private key riêng dùng để mã hóa. Có 2 cách sinh ra private key:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Mô tả | Ưu điểm | Nhược điểm |
| Client/Server dùng chung 1 thuật toán gennerate ra privatekey | Sẽ có 1 thuật toán gennerate ra privatekey để ở dưới client/server | Không phải trao đổi privatekey → an toàn | Dễ bị lộ thuật toán do để dưới client |
| privatekey sẽ được server gennerate và gửi xuống client | Chỉ có server mới có thể gennerate privatekey | Thuật toán không bị lộ do không để dưới client | Phải trao đổi privatekey → kém an toàn hơn |

→ Sử dụng cách 2. Nhưng với request đầu tiên trao đổi client/server sẽ sử dụng cách 1 để mã hóa. Response của request đầu tiên server sẽ trả privatekey mới và từ request thứ 2 trở đi client/server sẽ sử dụng cách 2.

b. *Toàn vẹn dữ liệu*:

- Sử dụng thuật toán signature của amazon

- Ref: [http://docs.aws.amazon.com/general/latest/gr/signature-v4-examples.html#signature-v4-examples-java](http://docs.aws.amazon.com/general/latest/gr/signature-v4-examples.html" \l "signature-v4-examples-java)

4. Ví dụ về gửi dữ liệu:

Giả sử client muốn gửi đến API login request body:

{

"username": "test01",

"password": "123456"

}

- Sau khi kí ta được: signature: “N5ske9cBVjlVfgmQ9VpFzSwzLLOD/2THg2iDrW3NGQZfz3SSWviwCe7G”  
→ Request body hiện tại:

{

"username": "test01",

"password": "123456",

"signature": "N5ske9cBVjlVfgmQ9VpFzSwzLLOD/2THg2iDrW3NGQZfz3SSWviwCe7G"

}

- Sau khi mã hóa dữ liệu(**không mã hóa trường signature**) ta được: data: “wUwDPglyJu9LOnkBAf4vxSpQgQZltcEquhdm5kSQIkQlZtfxtSTsmaw”

→ Request body gửi lên server lúc này:

{

"data": "wUwDPglyJu9LOnkBAf4vxSpQgQZltcEquhdm5kSQIkQlZtfxtSTsmaw",

"signature": "N5ske9cBVjlVfgmQ9VpFzSwzLLOD/2THg2iDrW3NGQZfz3SSWviwCe7G"

}

Example:

1. Signature (Code nodejs):

*/\*\**

*\* Created by phucnguyen on 12/05/2017.*

*\*/*

**var** crypto = require("crypto-js");

**var** config = require("./my\_modules/config/ConfigLog4JS"); //module Log4JS

**var** logger = config.getLogger("callAPI.js"); //Get logger for this file

**function** getSignatureKey(Crypto, kSecret, dataRequest) {

//Sort json by alpha b

**var** dataSort = JSON.stringify(dataRequest);

**var** kSigning = Crypto.HmacSHA256(dataSort, kSecret);

**var** kEnc = crypto.enc.Base64.stringify(kSigning);

**return** kEnc;

}

**var** dataRequest = {

"offset": 0,

"limit": 10

};

//kSecret must read by file

**var** kSecret = "vtcmobile";

**var** kSign = getSignatureKey(crypto, kSecret, dataRequest);

dataRequest.signature = kSign;

logger.info("\n"+JSON.stringify(dataRequest));

2. Mã hóa (Code java):

a. Đọc publickey từ file certificate.cer (gửi trong tệp đính kèm)

private static byte[] readPublicKey(){

try {

FileInputStream fin = new FileInputStream("certificate.cer");

CertificateFactory f = CertificateFactory.*getInstance*("X.509");

X509Certificate certificate = (X509Certificate)f.generateCertificate(fin);

PublicKey pk = certificate.getPublicKey();

return pk.getEncoded();

} catch (Exception e) {

*logger*.info(e.toString());

return null;

}

}

b. Gennerate private key (từ publickey + access token)

private static byte[] generatePrivateKey(String access\_token) throws NoSuchAlgorithmException, UnsupportedEncodingException {

MessageDigest sha = MessageDigest.*getInstance*("MD5");

byte[] keyBytes = sha.digest((access\_token).getBytes("UTF-8"));

System.*arraycopy*(keyBytes, 0, *readPublicKey*(), 0, *KEY\_LENGTH*/2);

keyBytes = Arrays.*copyOf*(keyBytes, *KEY\_LENGTH*);

return keyBytes;

}

c. Mã hóa dữ liệu (privatekey + dataRequest)

public static String encrypt(String plainText, String access\_token) throws Exception {

byte[] keyBytes = Arrays.*copyOf*(*generatePrivateKey*(access\_token), *KEY\_LENGTH*);

IvParameterSpec iv = new IvParameterSpec(keyBytes);

SecretKeySpec skeySpec = new SecretKeySpec(keyBytes, "AES");

Cipher cipher = Cipher.*getInstance*("AES/CBC/PKCS5Padding");

cipher.init(Cipher.*ENCRYPT\_MODE*, skeySpec, iv);

byte[] encrypted = cipher.doFinal(plainText.getBytes("UTF-8"));

return Base64.*encodeBase64String*(encrypted);

}

d. Giải mã dữ liệu (privatekey + dataResponse)

public static String decrypt(String encrypted, String access\_token) throws Exception {

byte[] keyBytes = Arrays.*copyOf*(*generatePrivateKey*(access\_token), *KEY\_LENGTH*);

IvParameterSpec iv = new IvParameterSpec(keyBytes);

SecretKeySpec skeySpec = new SecretKeySpec(keyBytes, "AES");

Cipher cipher = Cipher.*getInstance*("AES/CBC/PKCS5PADDING");

cipher.init(Cipher.*DECRYPT\_MODE*, skeySpec, iv);

byte[] original = cipher.doFinal(Base64.*decodeBase64*(encrypted));

return new String(original);

}