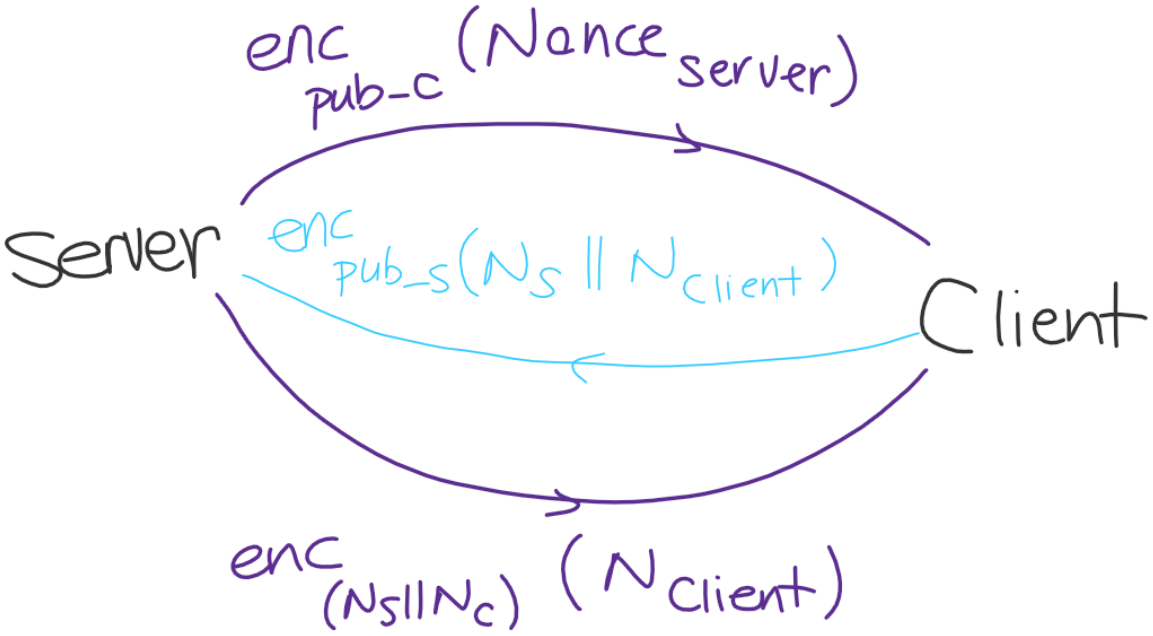
گزارش پروژه امنیت شبکه

نگار سخایی

۹۵۲۸۰۰۳

برای پروتکل تبادل کلید، طبق مباحثی که سر کلاس مطرح شد، از نسخه اصلاح‌شده Needham-Schroeder استفاده خواهیم کرد. این پروتکل، به شکل زیر کار می‌کند:



با توجه به تمام قابلیت‌های اتکر، بررسی می‌کنیم چرا مشکلی برای این پروتکل به وجود نخواهد آمد:

- کاربر مخرب کلید عمومی همه کاربران را می‌داند. این نکته، مشکلی به وجود نخواهد آورد. زیرا ابتدا فرض می‌کنیم که به دست آوردن کلید خصوصی از عمومی ممکن نیست، و این پروتکل برای به دست آوردن کلید جلسه نیاز دارد که کاربران پیام اول سرور را رمزگشایی کنند.
- کاربر مخرب می‌تواند پیام‌ها را ببیند و تغییر دهد. از آن جایی که پیام‌ها همه رمزشده هستند؛ مشاهده آن‌ها برای اتکر تفاوتی ایجاد نخواهد کرد. به همین دلیل نیز تغییر معناداری در آن‌ها ممکن نیست.

- کاربر مخرب می‌تواند پیام‌هایی تزریق کند. این پروتکل اصلاح‌شده، نسبت به حمله‌های MITM و Replay امن است. حمله MITM به خاطر پیام آخر فرستاده‌شده از طرف کاربر از بین می‌رود و در اصل اصلاح اصلی نسبت به پروتکل نیدهام–شرودر اولیه است. (چون کاربر با باز کردن پیام آخر، با کلید جلسه رمز آخر را انجام می‌دهد.) حمله تکرار نیز ممکن نیست، چون نانس طرف مقابل تصادفی‌ست و می‌تواند متفاوت باشد.
- کاربر مخرب الگوریتم را می‌شناسد. این فرض همیشگی امنیت است و همانطور که در بالا مشخص شد، مشکلی ایجاد نخواهد کرد.
- حمله دیگری که ممکن است پیش بیاید، این است که یک اتکر خود را به جای سرور جا بزند. در مکانیزم کلی این برنامه، از این بحث جلوگیری شده است. چون همه کاربران کلید عمومی سرور را می‌دانند و هیچ حمله‌کننده‌ای نمی‌تواند خود را به جای سرور جا بزند.

حال، به پیاده‌سازی این پروتکل می‌پردازیم. نکته اولی که باید به آن توجه شود، hardcode کردن کلید پابلیک سرور برای تمام کاربران است. برای این کار از دو فایل serverkey که برای سرور است و serverpub که برای کلاینت‌هاست استفاده می‌کنیم. از آنجایی که کلاس Config در برنامه وجود دارد، از آن کلاس برای خواندن خصوصیات Pub و Pri از فایل‌ها استفاده می‌کنیم.

```
String serverpub = Config.getAsString( configFileName: "serverpub.txt",  name: "Pub");
SecureSocket sock = new SecureSocket(serverHost, serverPort,
        clientPrivateKey, serverpub.getBytes());
```

```
RSA my_serverRSA = new RSA(Config.getAsString( configFileName: "serverkey.txt",  name: "Pub"),
        Config.getAsString( configFileName: "serverkey.txt",  name: "Pri"));
```

برای پیاده‌سازی مکانیزم تبادل کلید، نیاز به پیاده‌سازی رمزنگاری نامتقارن هستیم. (چون پروتکل تبادل کلید ما بر اساس رمزنگاری نامتقارن عمل می‌کند.) پروتکل رمزنگاری انتخاب شده، برای سادگی پیاده‌سازی RSA است که در فایل RSA.java قرار دارد.

در این کلاس، دو نوع constructor وجود دارد. نوع اول، یک جفت کلید ایجاد می‌کند. نوع دوم، با دریافت کلید پابلیک یک entity دیگر کلاس را تشخیص می‌دهد. در ابتدا نوع اول را بررسی می‌کنیم.

Constructor کلاس BigInteger قابلیت ایجاد عدد اول تصادفی را دارد و با کمک آن، دو عدد اول p و q را ایجاد می‌کنیم. بقیه اعداد مورد نیاز، به کمک این دو عدد توسط توابع موجود در کلاس BigInteger محاسبه خواهند شد.

```
RSA(){
    // BigInteger constructor generates a random prime
    SecureRandom rand = new SecureRandom();
    BigInteger p = new BigInteger( bitLength: 2048,  certainty: 100, rand);
    BigInteger q = new BigInteger( bitLength: 2048,  certainty: 100, rand);

    BigInteger n = p.multiply(q); // n = pq
    // m = (p-1)(q-1)
    BigInteger m = (p.subtract(new BigInteger( val: "1"))).multiply(q.subtract(new BigInteger( val: "1")));
    BigInteger e = new BigInteger( val: "65537");

    BigInteger d = e.modInverse(m);

    this.publicKey = e;
    this.privateKey = d;
    this.n = n;
}
```

و در روش دوم نیز، دو رشته که دریافت می‌شوند، پردازش شده و به عنوان کلیدها ذخیره می‌شوند:

```
RSA(String pubkey, String prikey){
    this.publicKey = new BigInteger(pubkey.split( regex: ",")[0]);
    this.n = new BigInteger(pubkey.split( regex: ",")[1]);
    this.publicKey = new BigInteger(prikey);
}

RSA(String pubkey){
    this.publicKey = new BigInteger(pubkey.split( regex: ",")[0]);
    this.n = new BigInteger(pubkey.split( regex: ",")[1]);
}
```

دو حالت برای زمان‌هایی که کلاس برای کاربر دیگری ساخته می‌شود (کلید خصوصی را نداریم) و برای خود کاربر ساخته می‌شوند ایجاد شده اند.

کلیدهای عمومی نیز فرمت زیر را دارند:

```
String getPublicKey() {
    return publicKey.toString() + "," + n.toString();
}
```

و انجام عملیات رمزگذاری و رمزگشایی نیز به کمک به توان رساندن انجام می‌شود:

```
byte[] encryptWithPublic(byte[] mes){
    BigInteger enc = new BigInteger(mes).modPow(this.publicKey, this.n);
    return enc.toByteArray();
}

byte[] decryptWithPrivate(byte[] mes){
    BigInteger enc = new BigInteger(mes).modPow(this.privateKey, this.n);
    return enc.toByteArray();
}
```

بعد از بررسی این کلاس، به خود پروتکل می‌پردازیم:

۱. کاربر، جفت کلید تولید کرده و کلید عمومی خود را به سرور ارسال می‌کند :

```
}else{ // client
    RSA myRSA = new RSA(); // generate a key pair
    RSA serverRSA = new RSA(Arrays.toString(hisPublicKey));

    byte[] mypub = myRSA.getPublicKey().getBytes();
    outstream.write(mypub,  off: 0, mypub.length); // client sends its pub key
    outstream.flush();
```

۲. سرور، کلید عمومی را دریافت می‌کند و کلاس متناظر با کاربر را می‌سازد.

```
if(iAmClient){ // server
    RSA my_serverRSA = new RSA(Config.getAsString( configFileName: "serverkey.txt",  name: "Pub"),
            Config.getAsString( configFileName: "serverkey.txt",  name: "Pri"));

    byte[] clientpub = new byte[5000];
    int num = instream.read(clientpub,  off: 0, clientpub.length); // server gets client pub key
    if(num != clientpub.length)    throw new RuntimeException();

    RSA clientRSA = new RSA(Arrays.toString(clientpub));
```

۳. سرور، نانس تولید کرده و آن را با کلید عمومی کاربر رمزکرده، برای کاربر ارسال می‌کند.

```
byte[] nonceServer = Util.getRandomByteArray( num: 16);
byte[] enNonce = clientRSA.encryptWithPublic(nonceServer);
outstream.write(enNonce,  off: 0, enNonce.length); // Server sends its nonce, encrypted with client pub key
outstream.flush();
```

۴. کاربر نانس را به دست آورده، با مقدار نانس خودش و رمزشده با کلید عمومی سرور، ارسال می‌کند: (هر دو نانس ۱۶ تایی هستند، کاربر در این مرحله ۳۲ تایی را ارسال خواهد کرد.)

```
byte[] serverNonce_rec = new byte[16];
int num = instream.read(serverNonce_rec,  off: 0, serverNonce_rec.length); // client gets server nonce
if(num != serverNonce_rec.length)    throw new RuntimeException();
byte[] nonceServer_dec = myRSA.decryptWithPrivate(serverNonce_rec);

byte[] nonceClient = Util.getRandomByteArray( num: 16);
byte[] key = new byte[32];
System.arraycopy(nonceServer_dec,  i: 0, key,  i1: 0,  i2: 16);
System.arraycopy(nonceClient,  i: 0, key, i1: 16,  i2: 16);

byte[] enNonce = serverRSA.encryptWithPublic(key);
outstream.write(enNonce,  off: 0, enNonce.length); // client sends its nonce
outstream.flush();
```

۵. سرور آن را باز کرده، کلید جلسه را به دست می‌آورد و نانس کلاینت را نیز جدا می‌کند:

```
byte[] clientNonce_rec = new byte[32];
num = instream.read(clientNonce_rec,  off: 0, clientNonce_rec.length);
if(num != clientNonce_rec.length)    throw new RuntimeException();
byte[] key = my_serverRSA.decryptWithPrivate(clientNonce_rec);

byte[] nonceClient_dec = new byte[16];
System.arraycopy(key,  i: 0, nonceClient_dec, i1: 16,  i2: 16);
```

۶. سرور، نانس کلاینت را با کلید جلسه رمز می‌کند (رمزمتقارن) و آن را ارسال می‌کند:

```
BlockCipher cipher = new BlockCipher(key);
byte[] nonceClient_dec_enc = new byte[16];
cipher.encrypt(nonceClient_dec,  inOffset: 0, nonceClient_dec_enc,  outOffset: 0);

// server sends nonce client encrypted with shared key
outstream.write(nonceClient_dec_enc,  off: 0, nonceClient_dec_enc.length);
outstream.flush();
return key;
```

۷. کلاینت، آن را رمزگشایی کرده و با نانس خود مقایسه می‌کند:

```
BlockCipher cipher = new BlockCipher(key);
byte[] rec_nonce = new byte[16];
byte[] rec_nonce_dec = new byte[16];
num = instream.read(rec_nonce,  off: 0, rec_nonce.length);
if(num != rec_nonce.length)    throw new RuntimeException();
cipher.decrypt(rec_nonce,  inOffset: 0, rec_nonce_dec,  outOffset: 0);

if (Arrays.equals(rec_nonce_dec, nonceClient)){
    return key;
}
else{
    throw new RuntimeException("Not valid!");
}
```

+ فایل کلید سرور به شکل زیر است:

serverkey.txt - Notepad

File Edit Format View Help

```
Pub:
65537,4954700925284459157114132044137810746381138810043990082202842991072313768223226434896785593863111300114190680416557373471948
7670200119222201223450589506110939915366225061449817501244759154109975586716422435500613366495087102943269097330043204656382629729
0295589341460655000595089474072811168455304875989393177979145994176136912410313809332831710222324534075541382802623595521933701677
7672422638515898042316613020461506414021507627575270928681090162844471681860440963097565758598785167307673239857650148372770421916
8531965115035277588154160360400042237488038059362184289196002239320311588329463578621640436330459252854866280436756154415197538297
6231234411945146479491772738605489909843338389796564447449846134944326799283153709323025066103763979275150359760088666365005636832
0723125322389949710263897069053703408663105414044689300667290832453429501130579817761962676750276044467395533807714697482699616133
8969995935777835838481293846387527341143163955687370760921065910456480029050731272413323513370843830999439316457247534888021964171
3171664961463805996006265198943495475298127730167280139288935048844331786667326623243706032870628618198742982374411188777065409164
7776681095984387962773807753209940874248476696474964294908424287055 61
Pri:
1102271075738093207054397442719826673211117442825295259143956098232057230887813623156310694089204003168666556608837855031829547021
5568888714677783688527077795809482270494467832509393908608372727528790232069054105968895408242075665096648794267724950161610224010
3950417265817914710843218053722875297085492551574453147818714437282496030873248282440988166863097508176469204449741516904503743631
7226445546690525599551983288272524430187680965716162935358259798237965198731128408513669687165102856033036674537486839384646042992
5864430998477734526421023736451509461752986872254397938762062501227670530829551299534029603062492267170074849062796799025742176882
7516988437876905768152071909974407612675097143888712925670445228218873018391815177044044948271317026832086230151047485703076671742
0619105329711132943420808092422724820301533638385415656910383099769919368969106040483307182557082830952336585787521418232941404929
0161675505922198535266704372350735356731010565738847167834094565665445214837644662749227484273301251382040018521904840108032438931
6674879655558271418995528663997865076295120913950673255972332649529673374128926347829531245656438077847831788808878127613091948209
9500540198560194238669265271653513635735104466339425047699359 53
```

اجرای نهایی پروژه، در تصویر run_screenshot آمده است.