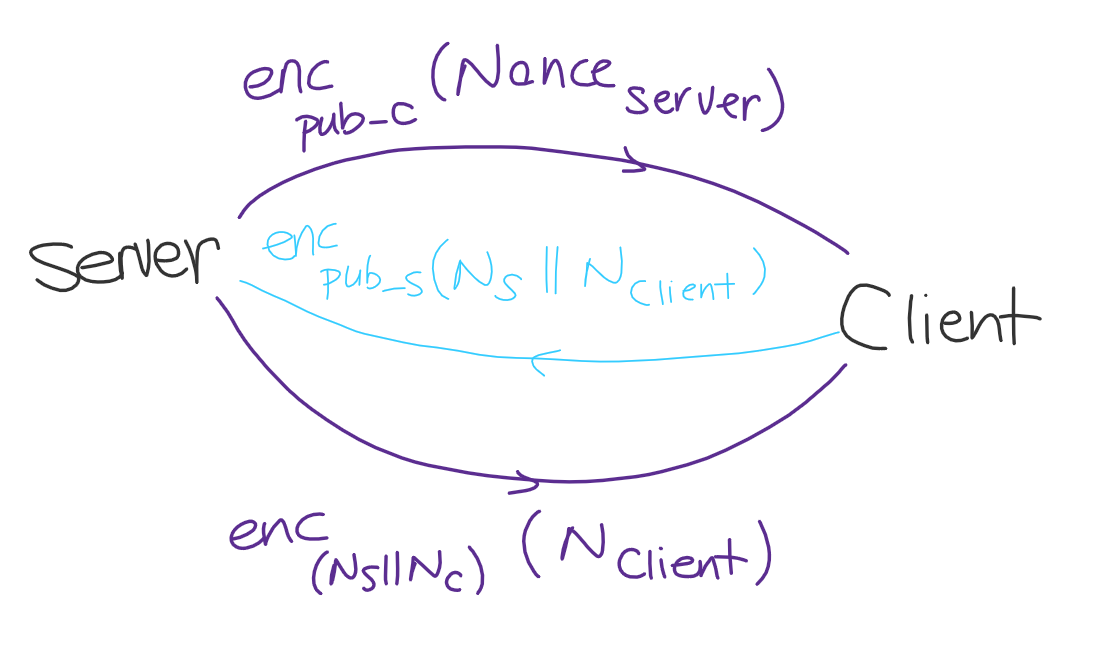
گزارش پروژه امنیت شبکه

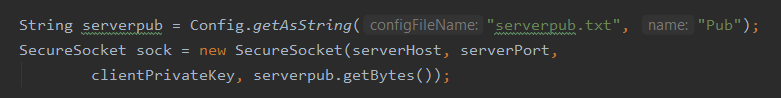
نگار سخایی

9528003

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:

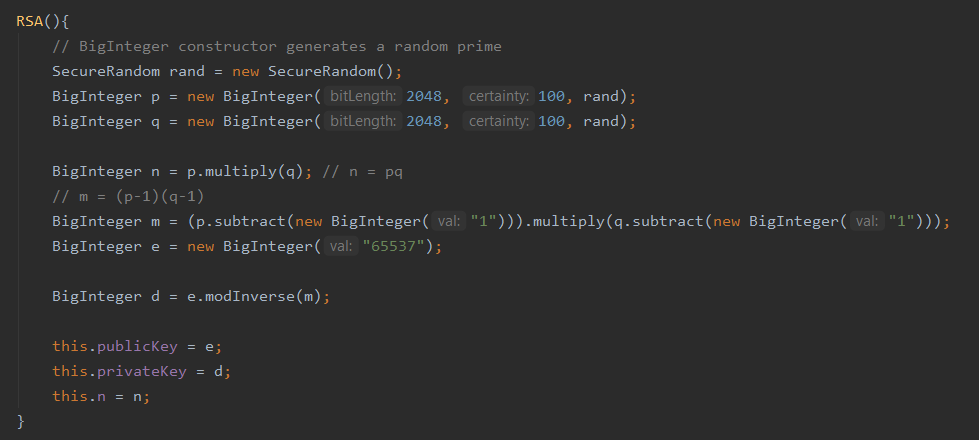
* کاربر مخرب کلید عمومی همه کاربران را می‌داند. این نکته، مشکلی به وجود نخواهد آورد. زیرا ابتدا فرض می‌کنیم که به دست آوردن کلید خصوصی از عمومی ممکن نیست، و این پروتکل برای به دست آوردن کلید جلسه نیاز دارد که کاربران پیام اول سرور را رمزگشایی کنند.
* کاربر مخرب می‌تواند پیام‌ها را ببیند و تغییر دهد. 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.

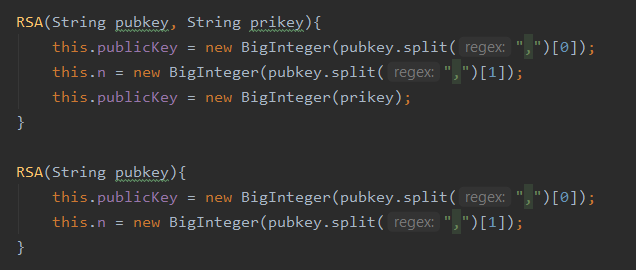
حال، به پیاده‌سازی این پروتکل می‌پردازیم. نکته اولی که باید به آن توجه شود، hardcode کردن کلید پابلیک سرور برای تمام کاربران است. برای این کار از دو فایل serverkey که برای سرور است و serverpub که برای کلاینت‌هاست استفاده می‌کنیم. از آنجایی که کلاس Config در برنامه وجود دارد، از آن کلاس برای خواندن خصوصیات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

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.

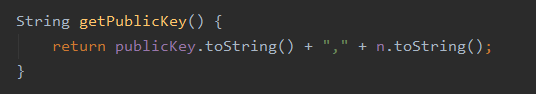
در این کلاس، دو نوع constructor وجود دارد. نوع اول، یک جفت کلید ایجاد می‌کند. نوع دوم، با دریافت کلید پابلیک یک entity دیگر کلاس را تشخیص می‌دهد. در ابتدا نوع اول را بررسی می‌کنیم.

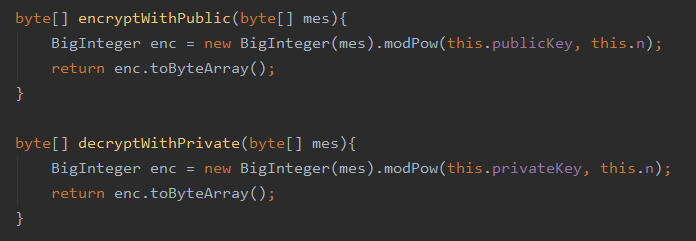
Constructor کلاس BigInteger قابلیت ایجاد عدد اول تصادفی را دارد و با کمک آن، دو عدد اول p و q را ایجاد می‌کنیم. بقیه اعداد مورد نیاز، به کمک این دو عدد توسط توابع موجود در کلاس BigInteger محاسبه خواهند شد.



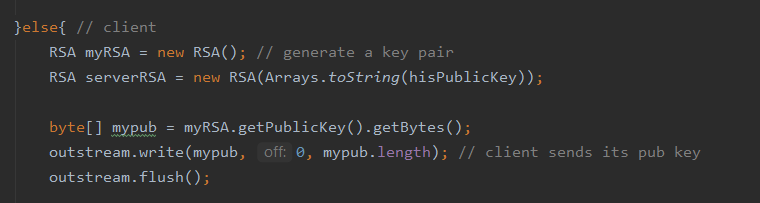
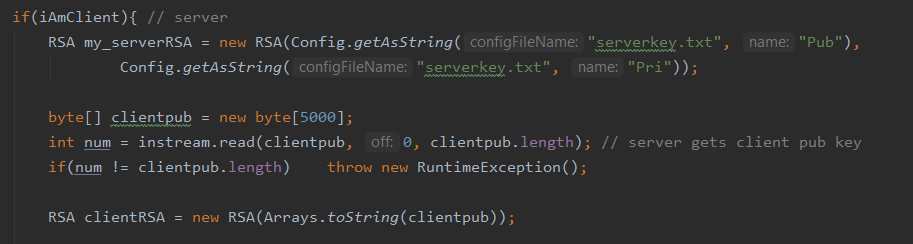
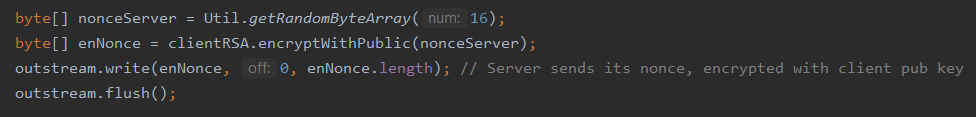
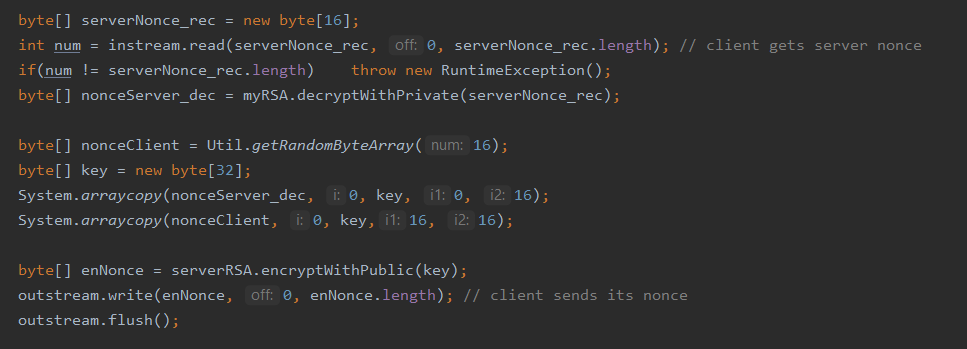
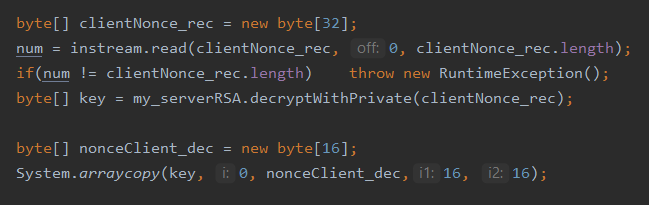
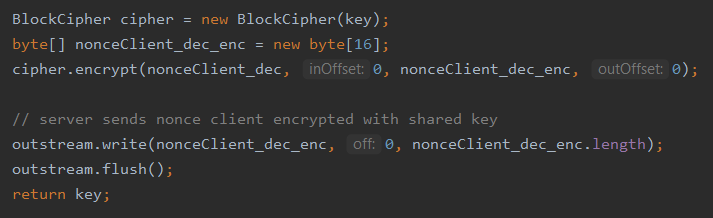
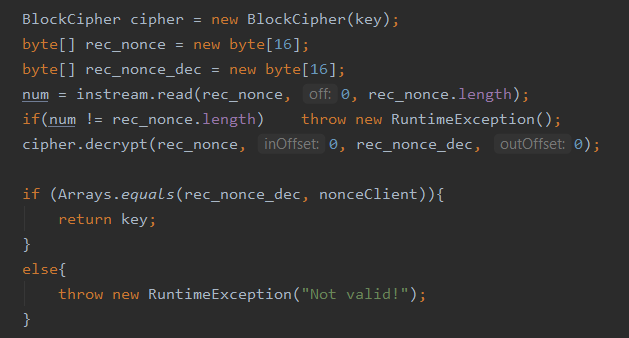
و در روش دوم نیز، دو رشته که دریافت می‌شوند، پردازش شده و به عنوان کلید‌ها ذخیره می‌شوند:

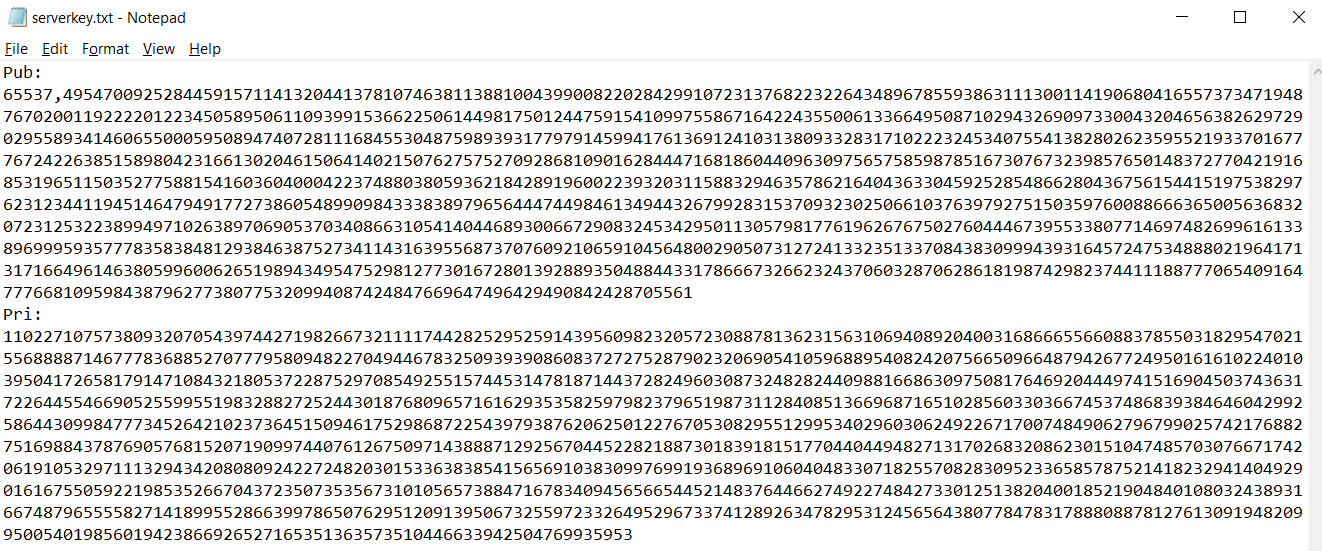
دو حالت برای زمان‌هایی که کلاس برای کاربر دیگری ساخته می‌شود (کلید خصوصی را نداریم) و برای خود کاربر ساخته می‌شوند ایجاد شده اند.

کلید‌های عمومی نیز فرمت زیر را دارند:

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

بعد از بررسی این کلاس، به خود پروتکل می‌پردازیم:

1. کاربر، جفت کلید تولید کرده و کلید عمومی خود را به سرور ارسال می‌کند :
2. سرور، کلید عمومی را دریافت می‌کند و کلاس متناظر با کاربر را می‌سازد.
3. سرور، نانس تولید کرده و آن را با کلید عمومی کاربر رمزکرده، برای کاربر ارسال می‌کند.
4. کاربر نانس را به دست آورده، با مقدار نانس خودش و رمزشده با کلید عمومی سرور، ارسال می‌کند: (هر دو نانس 16 تایی هستند، کاربر در این مرحله 32 تایی را ارسال خواهد کرد.)
5. سرور آن را باز کرده، کلید جلسه‌ را به دست می‌آورد و نانس کلاینت را نیز جدا می‌کند:
6. سرور، نانس کلاینت را با کلید جلسه رمز می‌کند (رمزمتقارن) و آن را ارسال می‌کند:
7. کلاینت، آن را رمزگشایی کرده و با نانس خود مقایسه می‌کند:

+ فایل کلید سرور به شکل زیر است:

اجرای نهایی پروژه، در تصویر run\_screenshot آمده است.

