دانشکده‌ی علوم ریاضی

مقدمه‌ای بر رمزنگاری	نیم‌سال دوم ۱۳۹۹
پروژه برنامه‌نویسی	
مدرّس: دکتر شهرام خزائی	مهلت تحویل نهایی: ۲۰ تیر

نکات مهم

لطفا ابتدا به نکات زیر توجه کنید:

- برای پیاده‌سازی این تمرین از JAVA نسخه 8 استفاده نمایید.
- پاسخ نهایی خود را به صورت یک فایل ZIP فقط در سامانه CW در بخش مربوط به پروژه برنامه‌نویسی بارگذاری کنید. ارسال پاسخ‌ها از طریق ایمیل یا پیام‌رسان‌ها قابل قبول نیست.
- به اشتراک گذاشتن کدهای مرتبط با پروژه‌تان با هر کسی به جز استاد و دستیاران آموزشی (حتی برای دی‌باگ) مجاز نیست. هم‌چنین مجاز به استفاده از کدهای موجود در اینترنت، و یا کدهای دیگران نیستید.
- تمام پاسخ‌هایی که برای این پروژه در سامانه ثبت می‌کنید، از لحظه‌ی ارسال، مالکیت معنوی و حق نشر آن را به این درس واگذار می‌کنید و بدون اجازه‌ی مستقیم استاد درس حق نشر پاسخ‌های خود را در هیچ کجا ندارید (حتی به دوستان خود نمی‌توانید بفرستید).
- سوالات خود پیرامون پروژه را از طریق amir77nadiri@gmail.com مطرح کنید. نکات اعلام شده در مورد ایمیل زدن در فایل اصلی توضیحات درس، در اینجا نیز صادق است.

موفق باشید.

مقدمه

در این درس به عنوان پروژه برنامه‌نویسی، پیاده‌سازی برخی از الگوریتم‌ها و سیستم‌هایی که در درس با آن‌ها آشنا شدید، در قالب یک پیام‌رسان در نظر گرفته شده است. بخش‌های گرافیک و شبکه این پیام‌رسان از قبل طراحی و به صورت کد اولیه در اختیار شما قرار داده‌شده‌اند. بخش‌های اصلی که انتظار می‌رود شما در پروژه پیاده‌سازی کنید، به شرح زیر است:

۱. سیستم رمز کلیدعمومی RSA
۲. سیستم رمزنگاری متقارن AES256
۳. پروتکل تبادل کلید Diffie–Hellman
۴. مولد اعداد شبه تصادفی بر پایه RC4
۵. تابع چکیده‌ساز SHA256 (نسخه ۲۵۶ بیتی SHA-2)

برای پیاده‌سازی برخی موارد عنوان‌شده (مانند سیستم RSA) نیاز به پیاده‌سازی توابع کار با اعداد بزرگ و بررسی اول بودن آن‌ها نیز دارید. بررسی اول بودن اعداد را بر پایه آزمون Miller-Rabin پیاده‌سازی کنید. برای تجزیه اعداد به دو عامل اول نیز الگوریتم Pollard’s Rho را پیاده‌سازی کنید.

ویژگی‌ها و قابلیت‌های این پیام‌رسان به شرح زیر است:

- این پیام‌رسان تنها یک سرور دارد.
- هر کاربر یک نام‌کاربری یکتا و رمزعبور دارد.
- هر فرد می‌تواند با داشتن برنامه پیام‌رسان و انتخاب نام کاربری و رمزعبور در پیام‌رسان عضو شود.
- هر کاربر می‌تواند به صورت هم‌زمان حداکثر از طریق یک دستگاه به پیام‌رسان متصل باشد.
- کاربران می‌توانند با داشتن نام‌کاربری هر کاربر دیگری، او را به لیست مخاطبان خود اضافه کنند.
- هر کاربر می‌تواند به هر کدام از کاربران موجود در لیست مخاطبانش پیام ارسال کند.
- برای ارسال پیام به کاربر دیگری، پیام‌ها ابتدا به سرور فرستاده می‌شوند و سرور آن‌ها را به مخاطب ارسال می‌کند.
- ارتباط میان کاربر و سرور دارای ویژگی رمزنگاری سرتاسر [1] است.
- اطلاعات کاربر در سرور به صورت امن نگه‌داری می‌شود.

بخشی از پروژه، شامل برخی از مقدمات رمزنگاری مورد نیاز برای پروژه، باید تا موعد زمانی مشخص به عنوان تحویل میانی، انجام شده باشند. فهرست موارد مورد نظر در ادامه آمده است که ویژگی‌های آن‌ها باید مطابق موارد خواسته شده در ادامه پروژه باشد.

- سیستم رمزنگاری متقارن AES256 با مود CBC شامل توابع رمزنگاری، رمزگشایی و تولید کلید
- توابع مورد نیاز برای کار با اعداد بزرگ (شامل تفریق، ضرب، توان پیمانه‌ای، وارون ضربی پیمانه‌ای و ب.م.م)
- آزمون Miller-Rabin برای اعداد بزرگ
- الگوریتم Pollard’s Rho برای تجزیه اعداد اول

[1] End-to-end encryption

توضیحات

پروتکل ایجاد ارتباط با سرور

در این پروتکل از دو عدد تصادفی ۱۲۸ بیتی که یکی توسط سرور و دیگری توسط کاربر انتخاب می‌شود برای تمییز ارتباط سرور با کاربرهای متفاوت استفاده می‌شود. از این دو عدد با عنوان nonce و serverNonce یاد می‌شود و در تمامی پیام‌ها به‌جز پیام اول که هنوز serverNonce انتخاب نشده است، به صورت رمز نشده نیز قرار خواهند داشت.

در این پروتکل ارتباطات به صورت زیر خواهد بود:

۱. کاربری درخواست ارتباط به همراه یک عدد تصادفی ۱۲۸ بیتی به سرور ارسال می‌کند.

$$\mathcal{M} = \{\text{nonce} : \text{int128}\}$$

۲. سرور پیامی حاوی عدد انتخابی کاربر، عدد ۱۲۸ بیتی تصادفی دیگر به انتخاب سرور، چکیده کلید عمومی سرور و عدد چالش به کاربر ارسال می‌کند. عدد چالش حاصل‌ضرب دو عدد اول ۳۰ یا ۳۱ بیتی است که به منظور مقابله با حمله منع سرویس [۲] وجود دارد و کاربر باید برای اثبات صداقت درخواست خود، آن را به عوامل اول تجزیه کند. تولید عدد چالش باید تصادفی و برای هر پیام متفاوت باشد. همچنین چکیده کلید عمومی سرور نیز برای بررسی تطابق با کلید از پیش موجود در برنامه کاربر، ارسال می‌شود.

$$\mathcal{M} = \{\text{nonce} : \text{int128}, \text{serverNonce} : \text{int128}, \text{publicKeyFingerprint} : \text{string}, \text{pq} : \text{string}\}$$

۳. کاربر عدد چالش را به دو عامل اول خود تجزیه کرده و به همراه اعداد تصادفی گام قبلی و یک محتوای رمز شده به سرور ارسال می‌کند. محتوای رمز شده برابر نام کاربری و رمز عبور و یک عدد تصادفی ۲۵۶ بیتی جدید به همراه اعداد تصادفی گام قبلی است که با کلید عمومی سرور که کاربر چکیده آن را در گام قبلی دریافت کرده است، رمز شده است. رمزنگاری به روش RSA انجام می‌شود.

$$\text{data} = \{\text{nonce} : \text{int128}, \text{serverNonce} : \text{int128}, \text{newNonce} : \text{int256}, \text{username} : \text{string}, \text{password} : \text{string}\}$$

$$\text{encryptedData} = \text{RSA}((\text{SHA256}(\text{data}) + \text{data} + \text{padding}), \text{serverPublicKey})$$

$$\mathcal{M} = \{\text{nonce} : \text{int128}, \text{serverNonce} : \text{int128}, \text{p} : \text{string}, \text{q} : \text{string}, \text{encryptedData}\}$$

۴. سرور با استفاده از اعداد تصادفی تبادل شده، کلید و بردار اولیه موقتی AES را محاسبه می‌کند. یک عدد تصادفی a و یک مولد g به صورت تصادفی انتخاب می‌کند و پیامی شامل مقدار g و g^a پروتکل Diffie–Hellman را به صورت رمزشده با کلید محاسبه‌شده، ارسال می‌کند.

$$\text{tmpAESKey} = \text{SHA256}(\text{newNonce} + \text{serverNonce})[0..15]$$

[2] Denial-of-service attack

$$+\text{SHA256}(\text{serverNonce} + \text{newNonce})[16..31]$$

$$\text{tmpAESIV} = \text{SHA256}(\text{serverNonce} + \text{newNonce})[0..15] \\ \oplus \text{SHA256}(\text{newNonce} + \text{newNonce})[0..15]$$

$$\text{answer} = \{\text{nonce} : \text{int128}, \text{serverNonce} : \text{int128}, \text{g} : \text{string}, \text{gA} : \text{string}\}$$

$$\text{answerHashed} = \text{SHA256}(\text{answer}) + \text{answer} + \text{padding}$$

$$\text{encryptedAnswer} = \text{AES256CBC}(\text{answerHashed}, \text{tmpAESKey}, \text{tmpAESIV})$$

$$\mathcal{M} = \{\text{nonce} : \text{int128}, \text{serverNonce} : \text{int128}, \text{encryptedAnswer} : \text{string}\}$$

۵. کاربر نیز کلید و بردار اولیه موقتی AES را محاسبه می‌کند. یک عدد تصادفی b انتخاب می‌کند و g^b پروتکل Diffie–Hellman را به صورت رمزشده با کلید محاسبه‌شده، ارسال می‌کند.

$$\text{answer} = \{\text{nonce} : \text{int128}, \text{serverNonce} : \text{int128}, \text{gB} : \text{string}\}$$

$$\text{answerHashed} = \text{SHA256}(\text{answer}) + \text{answer} + \text{padding}$$

$$\text{encryptedAnswer} = \text{AES256CBC}(\text{answerHashed}, \text{tmpAESKey}, \text{tmpAESIV})$$

$$\mathcal{M} = \{\text{nonce} : \text{int128}, \text{serverNonce} : \text{int128}, \text{encryptedAnswer} : \text{string}\}$$

۶. در نهایت سرور نیز پیامی برای تایید پایان‌یافتن پروتکل ارسال می‌کند.

$$\text{responseHash} = \text{SHA256}(\text{newNonce} + \text{SHA256}(\text{gAB})[0..7])$$

$$\mathcal{M} = \{\text{nonce} : \text{int128}, \text{serverNonce} : \text{int128}, \text{responseHash} : \text{string}\}$$

نماد $X[a..b]$ به معنی بایت‌های aـام تا bـام از متغیر X با شروع از صفر و نماد $a + b$ به معنی الحاق [۳] دو رشته a و b و نماد $a \oplus b$ به معنی عملگر بیتی XOR بین دو رشته a و b است.

از این پس برای کلید تبادل شده از نام authKey استفاده می‌کنیم.

در نهایت طرفین دو متغیر زیر را نیز محاسبه می‌کنند و از آن‌ها نیز در ارسال پیام‌ها استفاده می‌کنند.

$$\text{salt} = \text{newNonce}[0..7] \oplus \text{serverNonce}[0..7]$$
$$\text{authKeyId} = \text{SHA256}(\text{authKey})[0..7]$$

پروتکل ارسال پیام

زمانی که می‌خواهیم پیامی با محتوا messageData را از سرور به کاربر یا بالعکس انتقال دهیم، از این پروتکل پیروی می‌کنیم. ابتدا به معرفی متغیرها و توابع مورد استفاده در این پروتکل می‌پردازیم و سپس بقیه محاسبات تا به دست آوردن بسته ارسالی $\mathcal{M}$ را عنوان می‌کنیم. در تصویر ۱ نیز شماتیک این پروتکل آورده شده است.

- تابع unixtime در هر لحظه زمان فعلی را به فرمت مهر زمانی یونیکس [۴] خروجی می‌دهد. واحد این فرمت ثانیه بوده و

[3] concatenate
[4] Unix Timestamp

مقدار آن برابر تعداد ثانیه‌های گذشته از آغاز سال ۱۹۷۰ می‌باشد. برای محاسبه آن می‌توانید از تابع System.nanoTime کمک بگیرید.

- متغیر messageId به منظور مشخص‌کننده یکتای پیام بین دو کاربر استفاده می‌شود و مقدار آن 2^{32} برابر مقدار unixtime به علاوه مقداری تصادفی است. این مقدار تصادفی عددی بین 2 تا 2^{30} می‌تواند باشد به صورتی که لزوما عدد نهایی به دست آمده از messageId پیام‌های ارسال شده پیشین بیش‌تر باشد و برای پیام‌های ارسالی کاربر زوج و برای پیام‌های ارسالی سرور فرد باشد.

$$\text{messageId} = \text{unixtime}() \times 2^{32} + \text{randomNumber}$$

- متغیر sequenceId برابر شماره پیام ارسالی است. این شماره در ابتدا تبادل کلید با سرور برای کاربر برابر ۰ و برای سرور برابر ۱ قرار داده می‌شود و پس از هر پیام ارسالی ۲ واحد برای ارسال‌کننده افزایش داده می‌شود. در نتیجه این عدد همیشه برای پیام‌های ارسالی سرور فرد و برای کاربر زوج است.
- متغیر messageDataLength برابر طول messageData با واحد بایت است.
- مقدار x در پیام‌های ارسالی از کاربر برابر 0 و در پیام‌های ارسالی از سرور برابر 8 است.

۱. محاسبات بخش مربوط به ساختار و کلید اختصاصی پیام:

$$\begin{aligned}\text{data} = \{&\text{salt} : \text{int64}, \text{messageId} : \text{int64}, \text{sequenceId} : \text{int64},\\ &\text{messageDataLength} : \text{int32}, \text{messageData} : \text{bytes}, \text{padding} : \text{bytes}\}\end{aligned}$$

$$\text{msgKey} = \text{SHA256}(\text{authKey}[88 + \text{x}..119 + \text{x}] + \text{data})[8..23]$$

۲. محاسبات بخش مربوط به تولید مقادیر مورد نیاز برای رمزکردن پیام:

$$\text{sha256A} = \text{SHA256}(\text{msgKey} + \text{authKey}[\text{x}..\text{x} + 35])$$

$$\text{sha256B} = \text{SHA256}(\text{authKey}[40 + \text{x}..75 + \text{x}] + \text{msgKey})$$

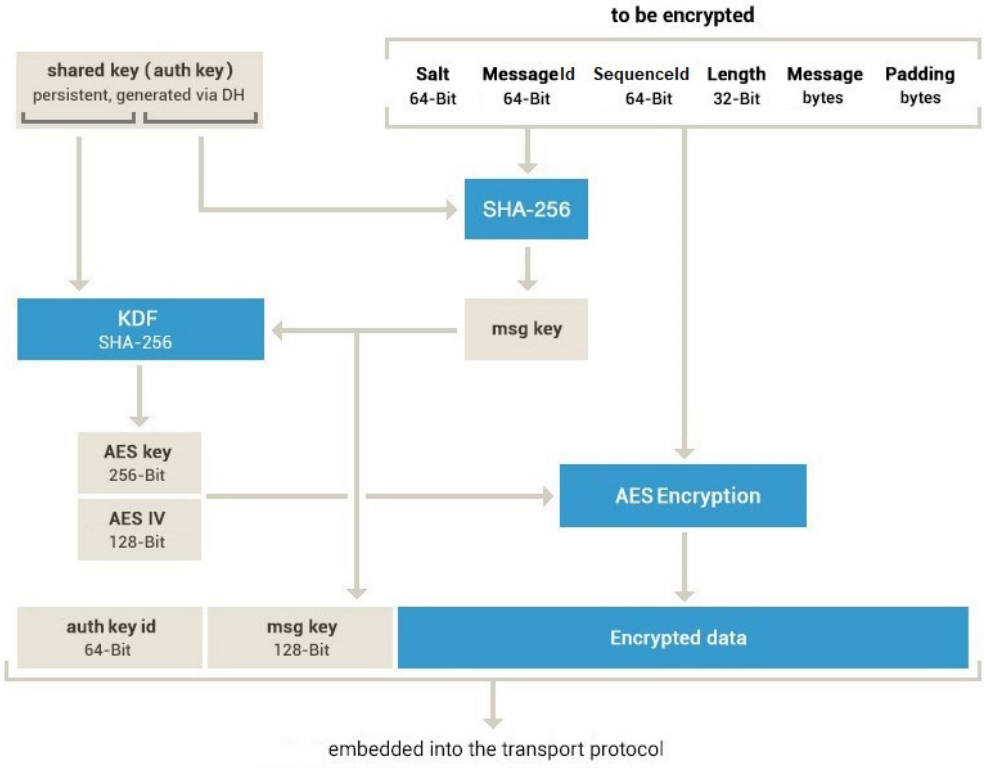
$$\text{AESKey} = \text{sha256A}[0..7] + \text{sha256B}[8..23] + \text{sha256A}[24..31]$$

$$\text{AESIV} = (\text{sha256B}[0..7] + \text{sha256B}[24..31]) \oplus \text{sha256A}[8..23]$$

۳. محاسبات بخش مربوط به تولید نهایی پیام ارسالی:

$$\text{encryptedData} = \text{AES256CBC}(\text{data}, \text{AESKey}, \text{AESIV})$$

$$\mathcal{M} = \{\text{authKeyId} : \text{int64}, \text{msgKey} : \text{int128}, \text{encryptedData} : \text{bytes}\}$$



شکل ۱: پروتکل ارسال پیام

ذخیره‌سازی امن

سرور در هنگام تبادل کلید نیاز به بررسی صحت رمز عبور کاربر دارد و برای این کار لازم دارد تا اطلاعاتی از رمز کاربر را ذخیره داشته باشد. برای امنیت بیشتر کاربران، رمز کاربران به صورت متن ساده [۵] ذخیره نمی‌شود و صرفا چکیده‌ای از آن ذخیره شده و سرور نیز با دریافت رمز از کاربر، از آن چکیده می‌گیرد و برابری آن با چکیده ذخیره‌شده را بررسی می‌کند. با استفاده از این روش اگر اطلاعات مرتبط با رمز کاربران که در سرور ذخیره شده در معرض خطر قرار گیرد، باز هم مشکلی برای کاربران ایجاد نخواهد شد. بدیهی است روشی که برای چکیده‌سازی استفاده می‌شود اهمیت بسیاری در امنیت دارد. شما در این پیاده‌سازی از الگوریتم چکیده‌سازی SHA256 با تغییراتی استفاده خواهید کرد. در این پیاده‌سازی برای مقابله با حمله جدول رنگین‌کمانی [۶] که بر پایه استفاده از نگاشت از پیش محاسبه‌شده چکیده به متن اصلی است، از افزودن salt استفاده کنید.

برای این کار اگر قصد ذخیره‌کردن رمزی که در متغیر x است را دارید، ابتدا یک متغیر تصادفی y با طول ۱۲۸ بیت تولید کنید و آن را به متغیر x الحاق کنید و سپس از حاصل ۱۰۲۴ بار به صورت متوالی چکیده بگیرید. در هنگام ذخیره‌سازی مقدار به‌دست‌آمده به عنوان چکیده، متغیر y را نیز ذخیره کنید تا در آینده با استفاده از آن بتوانید تطابق رمز کاربر با چکیده ذخیره‌شده را بررسی کنید. توجه کنید که تعداد بار چکیده گرفتن مقدار لزوما ثابتی نیست و یکی از پارامترهای امنیتی سیستم است و مقدار ۱۰۲۴ تنها یک مقدار پیشنهادی است.

همچنین سرور لیست مخاطبان هر کاربر را نیز نباید در حافظه (RAM) خود نگه‌داری کند و باید به صورت یک فایل در

[5] plain text
[6] Rainbow table attack

فضای ذخیره‌سازی خارجی (Storage) ذخیره کند. این ذخیره‌سازی نیز نباید به صورت متن ساده باشد اما بر خلاف رمز عبور نباید یک‌طرفه نیز باشد و باید امکان بازیابی محتوا داشته باشد. تصمیم برای سیستم رمزنگاری و فرمت ذخیره‌سازی در فایل به اختیار شما است اما باید همه موارد امنیتی ذکر شده، در آن رعایت شده باشند.

جزئیات

- اگر کاربری فعال باشد و درخواست ورودی با نام کاربری وی ارسال شود، درخواست نادیده گرفته می‌شود. کاربر فعال کاربری است که بعد از آخرین ورود، پیام خروج به سرور ارسال نکرده باشد و کمتر از ۲۰ دقیقه از ارسال آخرین پیامش به سرور گذشته باشد. اگر از آخرین پیام کاربری به سرور ۲۰ دقیقه گذشته باشد، کلید هویتی وی باطل می‌گردد و برای ارسال پیام نیاز به ورود مجدد دارد.
- سرور تاریخچه‌ای از پیام‌ها نگه‌داری نمی‌کند. در صروتی که کاربری غیرفعال باشد پیام‌هایی که به او ارسال می‌شوند، نادیده گرفته می‌شوند.
- در تمام قسمت‌هایی که padding به متغیری افزوده شده است، اندازه آن بین ۱ تا ۱۶ بایت به صورتی که در انتها، اندازه متغیر به دست آمده ضریبی از ۱۶ شود، می‌باشد. برای افزودن padding از روش ارائه شده در PKCS#7 استفاده کنید.
- در مرحله سوم ایجاد ارتباط، پس از دریافت پیام توسط سرور اگر کاربری با این نام از قبل وجود داشته باشد، سرور باید صحت رمز عبور او را بررسی کند و در صورت عدم وجود این نام کاربری، این کاربر را با رمزعبور دریافت کرده ایجاد کند و به ادامه پروتکل بپردازد.
- در هر گام از پروتکل‌ها هر طرف بعد از دریافت پیام از طرف مقابل باید صحت تمام موارد مورد نیاز را بسته به پیام دریافتی (از جمله صحت رمز عبور یا صحت تجزیه عدد چالش در مرحله سوم پروتکل ایجاد ارتباط) بررسی کند و در صورت عدم صحت باید پس از ارسال پیام زیر به ارتباط خود پایان دهد و اگر کلید هویتی در این ارتباط استفاده شده است، آن را باطل کند.

$$\mathcal{M} = \{\text{data} = \text{'error'}\}$$

- می‌توانید برای راحتی، در ابتدای همه پیام‌ها ($\mathcal{M}$) یک رشته ۴ بایتی الحاق کنید که نوع پیام را نشان می‌دهد. (نوع پیام می‌تواند شروع ارتباط، درخواست ورود، درخواست خروج و ... باشد)

پیاده‌سازی

برنامه سمت کاربر در پوشه Client و برنامه سمت سرور در پوشه Server قرار دارد.
برنامه کاربر دارای کدهای گرافیک و شبکه و برنامه سمت سرور دارای کدهای اجرا در خط فرمان [۷] و شبکه است.

در هر کدام از دو پوشه فایلی با نام Core.java قرار دارد که رابط کدهای اولیه با بخشی است که شما پیاده‌سازی می‌کنید. شما اجازه افزودن فایل‌های دیگر به پوشه‌ها را دارید اما نمی‌توانید در کدهای اولیه داده‌شده به‌جز فایل‌های Core.java تغییری

[7] command-line

ایجاد کنید. همچنین می‌توانید توابع دیگری به این فایل اضافه کنید اما اجازه تغییر نام، ورودی‌ها و شکل خروجی تابع‌های از پیش موجود در این فایل را ندارید. توابعی که در این فایل خروجی‌شان از نوع boolean است، باید مقدار بازگردانده‌شده‌شان نشان‌دهنده موفقیت‌آمیز بودن یا نبودن فرمان باشد. اطلاعات دقیق‌تر در رابطه با این توابع در ادامه به شما داده می‌شود.

سمت کاربر

برای ارسال پیام به سرور از تابع ChatClient.connection.sendData که ورودی آن آرایه‌ای از بایت‌ها است و ورودی آن عینا به تابع receiveData سرور داده می‌شود، استفاده کنید.

برای نمایش پیام‌ها به کاربر تابعی با نام receiveMessage در کلاس ChatClient قرار دارد که برای هر پیامی که کاربر در زمان فعال‌بودن دریافت می‌کند، باید آن را با ورودی صحیح از جنس Message صدا بزنید.

در فایل Core.java سمت کاربر نیز هفت تابع قرار دارد که شما باید بدنه آن‌ها را به شکل صحیحی کامل کنید. عملکرد این توابع باید مطابق پروتکل‌های گفته شده باشد. زمان صداشدن هر کدام از آن‌ها به شرح زیر است:

- receiveData: زمانی که پیامی از سمت سرور دریافت شود، این تابع با ورودی محتوای دریافت‌شده صدا زده می‌شود.
- loginOrRegister: زمانی که در محیط گرافیکی کاربری اقدام به ورود یا ثبت نام می‌کند، این تابع با مقادیر نام کاربری و رمزعبور صدا زده می‌شود.
- logOut: زمانی که کاربر در محیط گرافیکی اقدام به خروج از برنامه می‌کند، قبل از خروج این تابع صدا زده می‌شود تا شما عملیات‌های لازم از جمله باطل‌کردن کلید هویتی فعلی را انجام دهید.
- getMessages: زمانی که کاربر صفحه ارسال پیام به مخاطبی را باز می‌کند، این تابع با ورودی نام کاربری مخاطب صدا زده می‌شود و باید ۱۰۰ پیام اخیر میان کاربر فعلی و مخاطب به ترتیب messageId را برگرداند. (در صورتی که تعداد پیام‌ها تبادل شده کمتر از ۱۰۰ بود، آرایه تنها شامل همه پیام‌های موجود می‌شود)
- getContacts: زمانی که کاربر صفحه فهرست مخاطبان را باز می‌کند، این تابع صدا زده می‌شود و باید لیست مخاطبان فعلی کاربر به صورت آرایه‌ای از نام‌های کاربری برگردانده شود. (آرایه می‌تواند خالی باشد)
- addContact: زمانی که کاربر در محیط گرافیکی اقدام به افزودن مخاطب می‌کند، این تابع با ورودی آی‌دی کاربری که می‌خواهد به مخاطبان خود اضافه کند صدا زده می‌شود.
- sendMessage: زمانی که کاربر در محیط گرافیکی اقدام به ارسال پیام به کاربر دیگری می‌کند، این تابع با ورودی متن و نام‌کاربری مخاطب صدا زده می‌شود.

سمت سرور

برای ارسال پیام به کاربران از تابع ChatServer.connection.sendData که ورودی آن آرایه‌ای از بایت‌ها و شماره اختصاصی هر کاربر است و ورودی آن عینا به تابع receiveData آن کاربر داده می‌شود، استفاده کنید. شماره اختصاصی کاربران وابسته به ارتباط است و برای یک کاربر در ارتباطات مختلف می‌تواند تغییر کند.

در فایل Core.java سمت سرور دو تابع قرار دارد که شما باید بدنه آن‌ها را به شکل صحیحی کامل کنید. عملکرد این توابع باید به شکل زیر باشد:

- receiveData: زمانی که اطلاعاتی از سمت کاربر دریافت شود، این تابع با پیام ارسال شده و شماره اختصاصی کاربر ارسال‌کننده صدا زده می‌شود. تمییز کاربر ارسال‌کننده این پیام، تنها با استفاده از اطلاعات موجود در پیام همان‌طور که در پروتکل‌ها گفته شده است باید انجام شود. این تابع باید مطابق پروتکل عملیات‌های مورد نیاز را انجام دهد.
- generateRSAKeyPair: یک جفت کلید عمومی و خصوصی RSA ایجاد می‌کند و به صورت آرایه‌ای از String خروجی می‌دهد که عضو اول آن کلید خصوصی و عضو دوم آن کلید عمومی است.

نکات

- شما در کدهای خود مجاز به استفاده از کتاب‌خانه‌ی java.io تنها برای نوشتن در فایل هستید و مجاز به استفاده از هیچ‌کدام از دیگر کتاب‌خانه‌های JAVA از جمله تمام کلاس‌های موجود در java.math, java.util, javax.crypto, java.security و . . . نیستید. در صورت استفاده در هر بخش از برنامه، نمره آن بخش از شما کسر می‌شود.
- در پیاده‌سازی شما باید توابعی برای اجرای الگوریتم‌های RSAKeyGen، RSAEncrypt، RSADecrypt، AESEncrypt، AESDecrypt، RC4، SHA256، Miller-Rabin و Pollard's Rho به صورت مشخص و قابل استفاده به صورت مستقل وجود داشته باشند. ورودی و خروجی این توابع باید مطابق ساخت‌های ارائه شده در کتاب مرجع درس باشند و پارامترهای امنیتی تمام پیاده‌سازی‌ها باید به سادگی قابل تغییر باشند. ورودی توابع باید به صورت رشته متنی باشد. توابع AES مانند پیاده‌سازی آن‌ها در پروتکل، باید ۲۵۶ بیتی با مود CBC باشند.
- در پایان برنامه نهایی شما باید توانایی عملکرد صحیح به عنوان یک پیام‌رسان را داشته باشد و همه قابلیت‌های آن اجرایی باشند. برنامه باید در موارد ذکر شده، با مستند پروژه مطابقت داشته باشد اما در مورد موارد ذکر نشده (مانند فرمت ذخیره‌سازی در فایل) اختیار عمل دارید.
- سرعت اجرا توابع کار با اعداد بزرگی که پیاده می‌کنید، تاثیر بسیار زیادی در زمان ایجاد ارتباط با سرور دارد و باید از سریع‌ترین الگوریتم‌های موجود برای پیاده‌سازی این توابع بهره بگیرید. برای مثال، در پیاده‌سازی ضرب، توان و تقسیم می‌توانید از الگوریتم‌هایی مبتنی بر روش تقسیم و حل (مانند Karatsuba و Burnikel-Ziegler) و در پیاده‌سازی وارون ضربی از تعمیم الگوریتم اقلیدسی استفاده کنید.
- در پایان پیاده‌سازی پروژه به جهت آزمایش اجرا، یک جفت کلید RSA با پارامتر $n = 512$ برای استفاده در پروتکل ایجاد ارتباط، تولید کنید و در کد کاربر و سرور برای استفاده قرار دهید. این کلید باید به سادگی قابلیت تغییر با کلید تولیدشده توسط generateRSAKeyPair را داشته باشد.
- در ابتدای اجرا، مولد اعداد تصادفی RC4 که پیاده کردید را با یک seed اولیه آماده کنید و در هر بخشی از برنامه که به عدد تصادفی نیاز داشتید، از آن استفاده کنید.
- در توابع سمت کاربر که نیاز به دریافت پاسخ از سمت سرور برای محاسبه خروجی را دارند، می‌توانید از توابع wait و notify برای پیاده‌سازی انتظار پاسخ استفاده کنید.
- پروژه یک تحویل میانی و یک تحویل نهایی دارد که هر دو به صورت آنلاین هستند و زمان‌بندی آن‌ها متعاقبا اعلام خواهد شد.