

گزارش پروژه امنیت شبکه

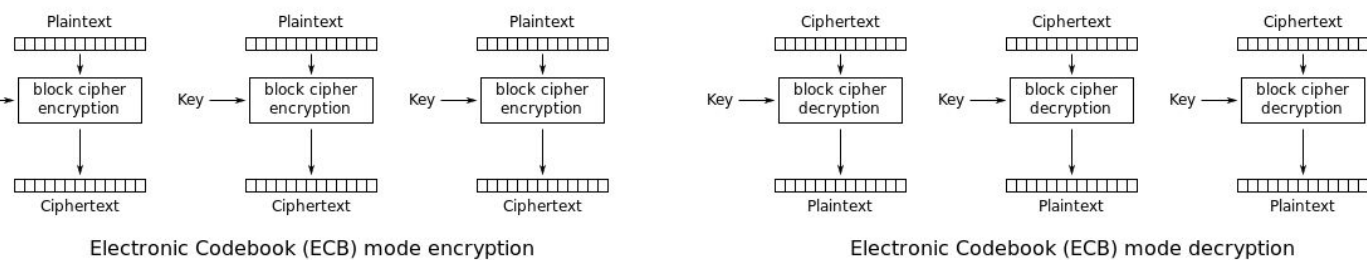
محیا جمشیدیان

9525133

فایل با نام StreamCipher

توابع Encrypt و Decrypt

در ادامه معرفی توابع رمزنگاری و رمزگشایی در فایل BlockCipher, در این فایل با بهره‌وری از همان الگوریتم رمزنگاری، یک رمزگذاری استریم را پیاده‌سازی کرده ایم که در مود ECB است.



استفاده از این توابع به آسانی فراخوانی توابع encrypt و decrypt است.

فایل با نام RSA:

توابع Encrypt و Decrypt:

برای پروتکل تبادل کلید بین افراد، لازم است که از یک الگوریتم رمزنگاری نامتقارن بهره ببریم. تعریف توابع این فایل هم بدین منظور است. توجه کنید که ورودی توابع encrypt و decrypt هر دو شامل کلید رمزگشایی هستند. که به صورت یک دوتایی از e یا d تنظیم شده اند. میدانیم با داشتن کلید، روند رمزگشایی و رمزنگاری یکی است و جداسازی این دو تابع به هدف افزایش سهولت در فهم کد است.

فایل با نام AuthenticationInfo:

تابع constructor:

در این تابع، ابتدا مقادیر ورودی که اطلاعات مربوط به کاربر شامل نام کاربری، کلمه عبور و حالت ورودی است در جای خود قرار میگیرد. همچنین، لیستی از تمامی کاربرهای ثبت نام شده که در یک فایل کنار کد قرار دارد لود میشود.

- تابع **checkStrength**:

- این تابع باید قدرت یک پسورد انتخاب شده را چک کند و در صورت تایید true برگرداند. مواردی که چک می شود شامل طول پسورد به اندازه حداقل 8 و شامل حرف کوچک، حرف بزرگ، علائم خاص و عدد است.

- تابع **isValid**:

- این تابع چک میکند که مقادیر نام کاربری و کلمه عبور در فایل کاربران موجود باشد و با هم مرتبط باشند.

- تابع **getUserName**:

- در این تابع، با توجه به حالت ورودی، که در وضعیت In یا Sign in باشیم و یا در وضعیت Up یا Sign Up باشیم، خروجیهای متفاوت برمیگرداند.
- اگر در وضعیت In باشیم، خروجی تابع isValid را نگاه میکنیم. اگر true بود نام کاربری را برمیگردانیم. در غیر اینصورت، مقدار SIE یا Sign In Error را برمیگردانیم.
- اگر در وضعیت Up باشیم:
- ابتدا چک میکنیم که نام کاربری از قبل وجود نداشته باشد. در صورت خطا مقدار UAE یا Username Already Exists را برمیگردانیم.
- سپس چک میکنیم که قدرت پسورد انتخاب شده کافی باشد. در غیر اینصورت مقدار PSE یا Password Strength Error را برمیگردانیم.
- در نهایت با عبور از این موانع، نام کاربری و رمز عبور متقاصی به فایل مربوطه اضافه میشود و پیغام SUS یا Sign Up Successful بازگردانیده میشود. توجه کنید که در این صورت کاربر باید با حالت In و با نام کاربری و رمز عبور خود، دوباره درخواست وارد شدن به سرور و ارتباط را بدهد.

این خروجی ها در سمت سرور و در فایل ChatServer به صورت زیر بررسی میشوند و پیغام مربوطه ارسال میگردد.

```

if(username.equals("SIE"))
{
    System.out.println("Authentication failed");
    send_error_message(out, E: "Authentication failed. Please try again later.\n");
    continue;
}
if(username.equals("UAE"))
{
    System.out.println("Username Error");
    send_error_message(out, E: "This username is already in use. Please try again with a new one or Sign In.\n");

    continue;
}
if(username.equals("PSE"))
{
    System.out.println("Password Error");
    send_error_message(out, E: "Your password is not strong enough. Please try again with new password\n");
    continue;
}
if(username.equals("SUS"))
{
    System.out.println("Sign Up Successful");
}
}

```

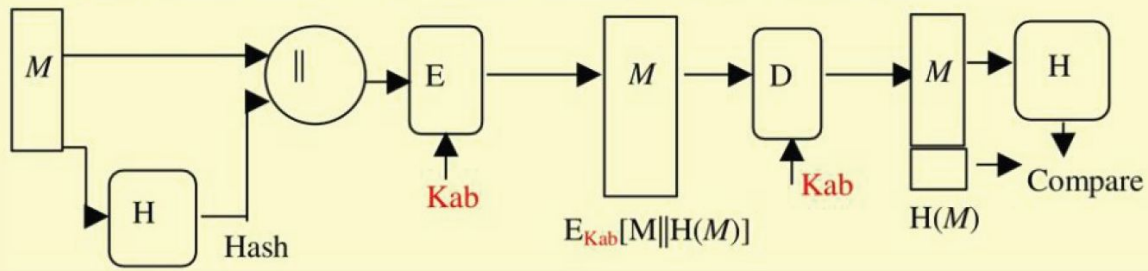
- فایل SecureOutputStream :

- مجموعه توابع **write**:
- با توجه به کلید جلسه به اشتراک گذاشته شده بین طرفین، تابع **write** برای ارسال اطلاعات محرمانه بین افراد مراحل زیر را طی میکند:
- مقدار هش هر بایت را در میآورد.
- هش و مقدار اصلی را در قالب یک قطعه 8 بایتی با استفاده از کلید جلسه رمز میکند. بایت اصلی همان بایت آخر و 7 بایت اول مقدار هش هستند.
- این مقدار هشت بایتی را روی کانال ارسال میکند.

- فایل SecureInputStream :

- مجموعه توابع **read**:
- با توجه به کلید جلسه به اشتراک گذاشته شده بین طرفین، تابع **read** برای دریافت اطلاعات محرمانه بین افراد مراحل زیر را طی میکند:
- به اندازه 8 بایت داده را دریافت میکند.
- با استفاده از کلید جلسه، مقدار 8 بایتی اصلی را دریافت میکند.
- از بایت آخر هش می‌گیرد و 7 بایت اول این مقدار هش را با 7 بایت اول ورودی رمزگشایی شده مقایسه میکند.
- در صورت برابر بودن، بایت آخر به عنوان داده استخراج شده کنار بقیه مقادیر قرار میگیرد.

Integrity, Authentication & Confidentiality provided with Hash.



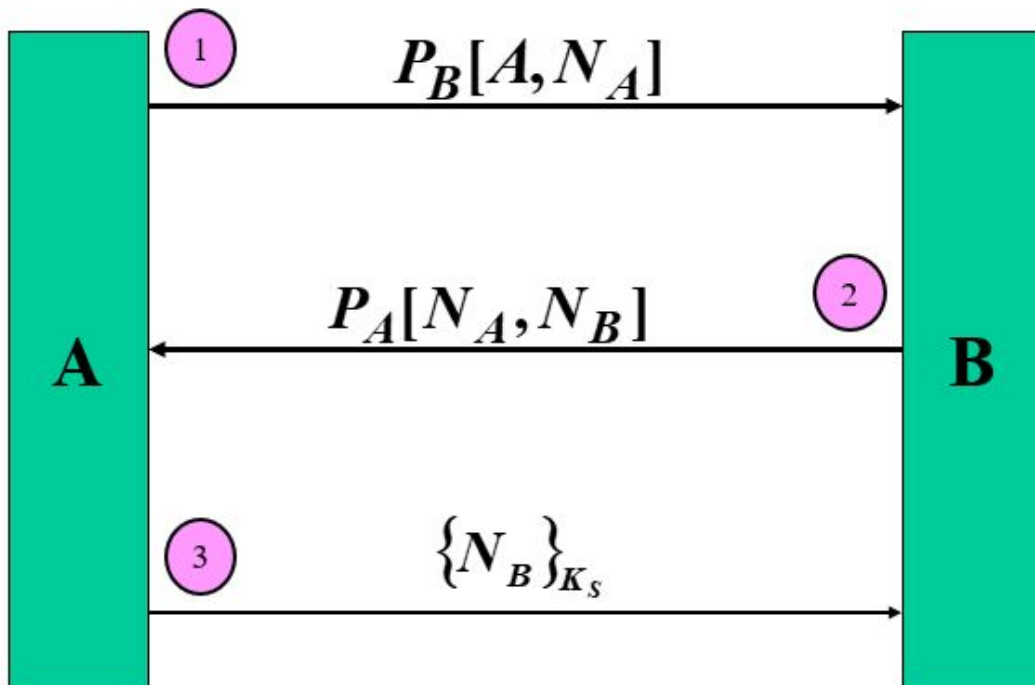
این دو کلاس به جای کلاسهای InputStream و OutputStream در تمامی فایلها قرار میگیرند.

توضیح مربوط به این پروژه در این قسمت قرار دارد!

- فایل SecureSocket:

- تابع keyExchange:

- در این تابع با استفاده از پروتکل نیدهام-شرودر، طرفین به توافق کلید جلسه میپردازند.



- به این منظور، مراحل زیر برای توافق کلید طی میشود.

- **[USER]** کاربر برای شروع ارتباط کلید عمومی خود را برای سرور میفرستد.

```
outstream.write(myPublicKey, off: 0, myPublicKey.length);
outstream.flush();
```

- **[USER]** سپس یک مقدار رندم به عنوان Nonce که آن را NC مینامیم به همراه یک قسمت از کلید به نام k1 را با استفاده از کلید عمومی سرور رمز میکند و برای وی میفرستد.

```
//---
byte[] k1 = Util.getRandomByteArray( num: 4);
byte[] NC = Util.getRandomByteArray( num: 4);
byte[] toenc = new byte[8];
System.arraycopy(k1, srcPos: 0, toenc, destPos: 0, length: 4);
System.arraycopy(NC, srcPos: 0, toenc, destPos: 4, length: 4);
byte[] toenc_b64 = Base64.getEncoder().encode(toenc);
byte[] enc1 = rsa.encryptMessage(toenc_b64, otherPubKey);

outstream.write(enc1, off: 0, enc1.length);
outstream.flush();
//---
```

- **[SERVER]** سرور با دریافت این مقدار و رمزگشایی آن، هر دو مقدار را ذخیره میکند. سپس یک پیغام شامل NC، یک مقدار Nonce برای خود سرور به نام NS و سپس قسمت دوم کلید k2 را با کلید عمومی کاربر رمز میکند و برای او میفرستد.

```

byte[] b2 = new byte[3000];
int read2 = instream.read(b2);

byte[] enc_recv = Arrays.copyOfRange(b2, from: 0, read2);
byte[] dec_recv_b64 = rsa.decryptMessage(enc_recv, myPrivateKey);
byte[] dec_recv = Base64.getDecoder().decode(dec_recv_b64);
byte[] k1 = Arrays.copyOfRange(dec_recv, from: 0, to: 4);
byte[] NC = Arrays.copyOfRange(dec_recv, from: 4, to: 8);
//---
byte[] k2 = Util.getRandomByteArray( num: 4);
byte[] NS = Util.getRandomByteArray( num: 4);

byte[] toencenc = new byte[12];
System.arraycopy(k2, srcPos: 0, toencenc, destPos: 0, length: 4);
System.arraycopy(NS, srcPos: 0, toencenc, destPos: 4, length: 4);
System.arraycopy(NC, srcPos: 0, toencenc, destPos: 8, length: 4);
byte[] toencenc_b64 = Base64.getEncoder().encode(toencenc);
byte[] enc1 = rsa.encryptMessage(toencenc_b64, otherPubKey);
outstream.write(enc1, off: 0, enc1.length);
outstream.flush();

```

- **[USER]** کاربر با دریافت این مقادیر و رمزگشایی با استفاده از کلید خصوصی اش، مقدار NC فرستاده شده را با مقدار اصلی مقایسه میکند. در صورت مغایرت، مخابره را قطع میکند.

```

byte[] b2 = new byte[3000];
int read2 = instream.read(b2);
byte[] servEnc = Arrays.copyOfRange(b2, from: 0, read2);
byte[] dec_serv_b64 = rsa.decryptMessage(servEnc, myPrivateKey);
byte[] dec_serv = Base64.getDecoder().decode(dec_serv_b64);
byte[] k2 = Arrays.copyOfRange(dec_serv, from: 0, to: 4);
byte[] NS = Arrays.copyOfRange(dec_serv, from: 4, to: 8);
byte[] NC_rec = Arrays.copyOfRange(dec_serv, from: 8, to: 12);

if(!Arrays.equals(NC, NC_rec)){
    throw new RuntimeException();
}

```

- **[USER]** در صورت برابری مقادیر فوق، کاربر با کنار هم قرار دادن مقادیر k1 و k2 کلید نشست را به دست می آورد. سپس با استفاده از این کلید و

الگوریتم رمزنگاری متقارن توافق شده، مقدار NS را رمز میکند و برای سرور میفرستد.

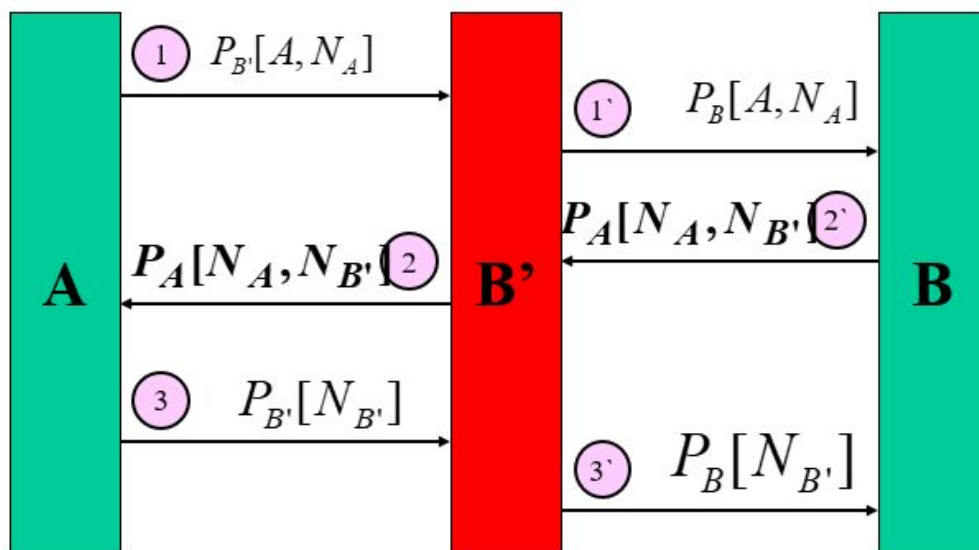
```
//---
byte[] key = new byte[k1.length + k2.length];
System.arraycopy(k1, srcPos: 0, key, destPos: 0, k1.length);
System.arraycopy(k2, srcPos: 0, key, k1.length, k2.length);
StreamCipher st = new StreamCipher(key);
byte [] NS_enc = st.encrypt(NS, inOffset: 0, outOffset: 0);
outstream.write(NS_enc, off: 0, NS_enc.length);
outstream.flush();
```

- **[SERVER]** سرور هم هر دو مقدار k1 و k2 را در دست دارد و کلید نشست را میسازد. همچنین با دریافت مقدار NS رمز شده از کاربر، بعد از رمزگشایی با استفاده از کلید نشست، مقدار NS استخراج شده با مقدار اصلی مقایسه میکند و در صورت مغایرت ارتباط را قطع میکند. در صورت برابری هر دو طرف به کلید جلسه دسترسی دارند.

```
//
byte[] b3 = new byte[3000];
int read3 = instream.read(b3);
byte[] resp = Arrays.copyOfRange(b3, from: 0, read3);
byte[] key = new byte[k1.length + k2.length];
System.arraycopy(k1, srcPos: 0, key, destPos: 0, k1.length);
System.arraycopy(k2, srcPos: 0, key, k1.length, k2.length);
StreamCipher st = new StreamCipher(key);
byte[] NS_rec = st.decrypt(resp, inOffset: 0, outOffset: 0);
byte[] NS_rec_ = Arrays.copyOfRange(NS_rec, from: 0, to: 4);
if(!Arrays.equals(NS, NS_rec_)){
    throw new RuntimeException();
}
```

1) استفاده از الگوریتم رمز نامتقارن جلوی این را میگیرد که کاربر بدخواه بتواند خودش را به جای هر کاربری جا بزند. چرا که اگر از کلید عمومی کاربر دیگری استفاده کند، نمیتواند پیامها را رمزگشایی کند.

2) حمله MITM هم به علت تشکیل کلید جلسه برای ارسال nonce سرور از طرف کاربر نیز منتفی است. چرا که از مقادیر توافق شده k1 و k2 نمیتواند با خبر شود، در صورتی که اگر در مرحله آخر، کاربر از کلید عمومی سرور برای ارسال پاسخ چالش نانس استفاده میکرد، حمله زیر رخ میداد.



(3) هر گونه تغییر در مقادیر ارسالی و یا تزریق اطلاعات بین داده‌های ارسالی برای هر دو طرف قابل تشخیص است. (Nonce)

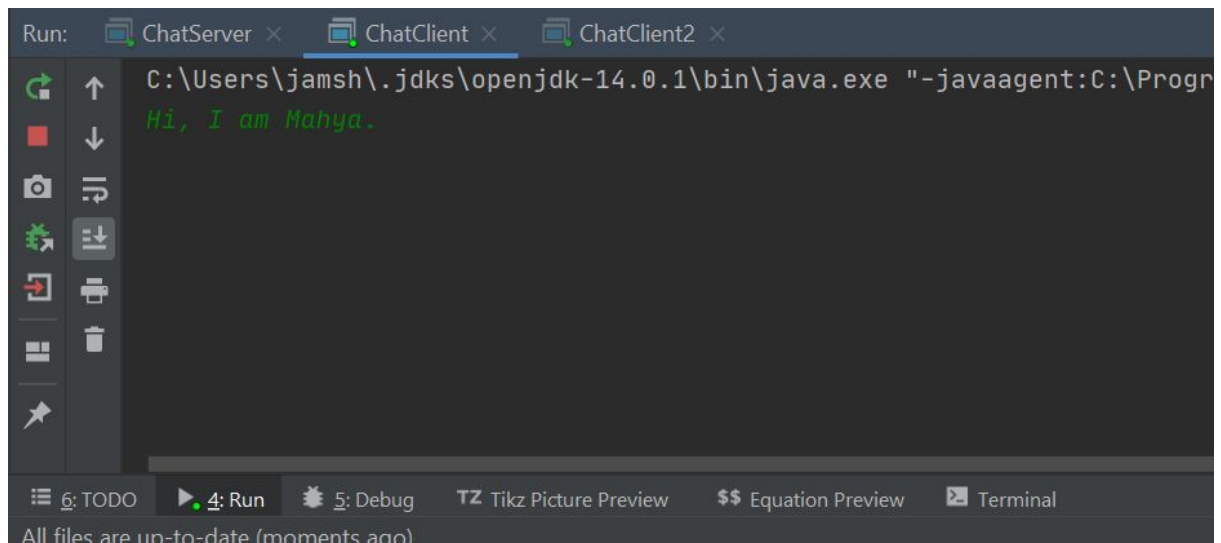
(4) خاصیت رندم بودن مقادیر $NS, NC, k1, k2$ از این که از سورس کد چیزی مشخص باشد جلوگیری میکند.

(5) توجه شود که هیچ کاربری نهایتاً بدون احراز هویت، از سرور نمیتواند استفاده کند.

```

ChatServer x ChatClient x ChatClient2 x
C:\Users\jamsh\.jdk\openjdk-14.0.1\bin\java.exe "-javaagent
I am listening on 3636
Got connection from Mahya
Server
Got connection from Dina
Server

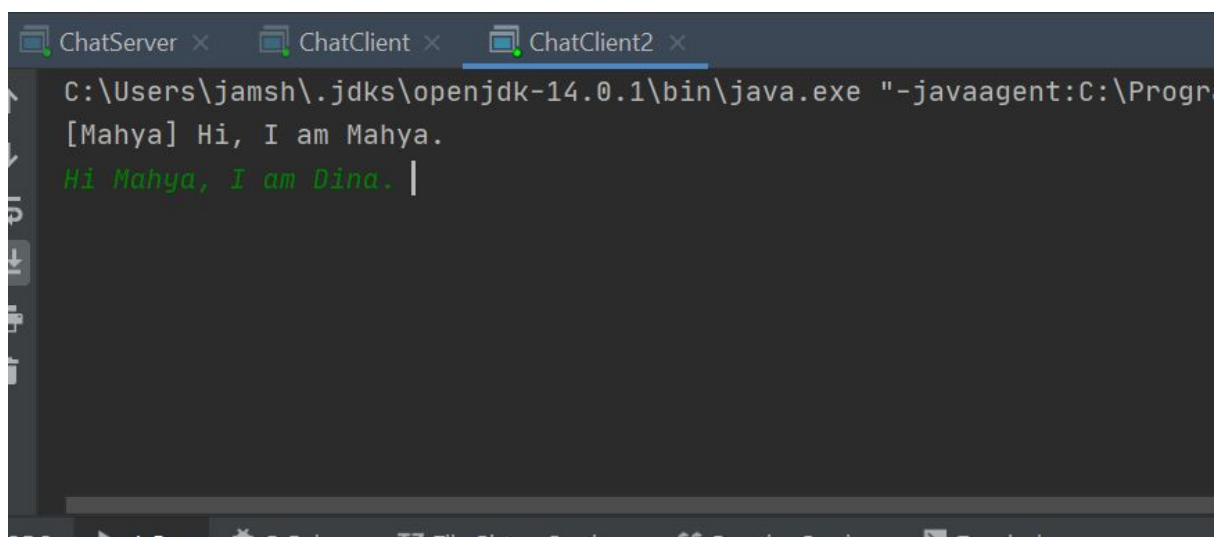
```

```
Run: ChatServer x ChatClient x ChatClient2 x
C:\Users\jamsh\.jdk\openjdk-14.0.1\bin\java.exe "-javaagent:C:\Progr
Hi, I am Mahya.
```

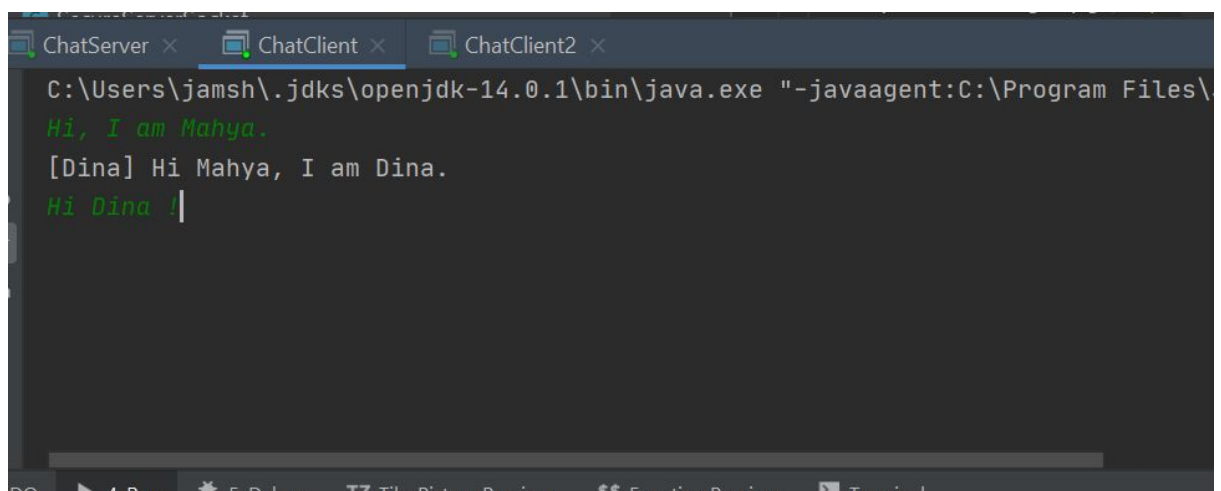
6: TODO 4: Run 5: Debug TZ Tikz Picture Preview \$\$ Equation Preview Terminal

All files are up-to-date (moments ago)



```
ChatServer x ChatClient x ChatClient2 x
C:\Users\jamsh\.jdk\openjdk-14.0.1\bin\java.exe "-javaagent:C:\Progr
[Mahya] Hi, I am Mahya.
Hi Mahya, I am Dina. |
```

4: Run 5: Debug TZ Tikz Picture Preview \$\$ Equation Preview Terminal



```
ChatServer x ChatClient x ChatClient2 x
C:\Users\jamsh\.jdk\openjdk-14.0.1\bin\java.exe "-javaagent:C:\Program Files\
Hi, I am Mahya.
[Dina] Hi Mahya, I am Dina.
Hi Dina :|
```

4: Run 5: Debug TZ Tikz Picture Preview \$\$ Equation Preview Terminal

```
ChatServer x ChatClient x ChatClient2 x
C:\Users\jamsh\.jdk\openjdk-14.0.1\bin\java.exe "-javaagent:C:\Pr
[Mahya] Hi, I am Mahya.
Hi Mahya, I am Dina.
[Mahya] Hi Dina !
D|
```