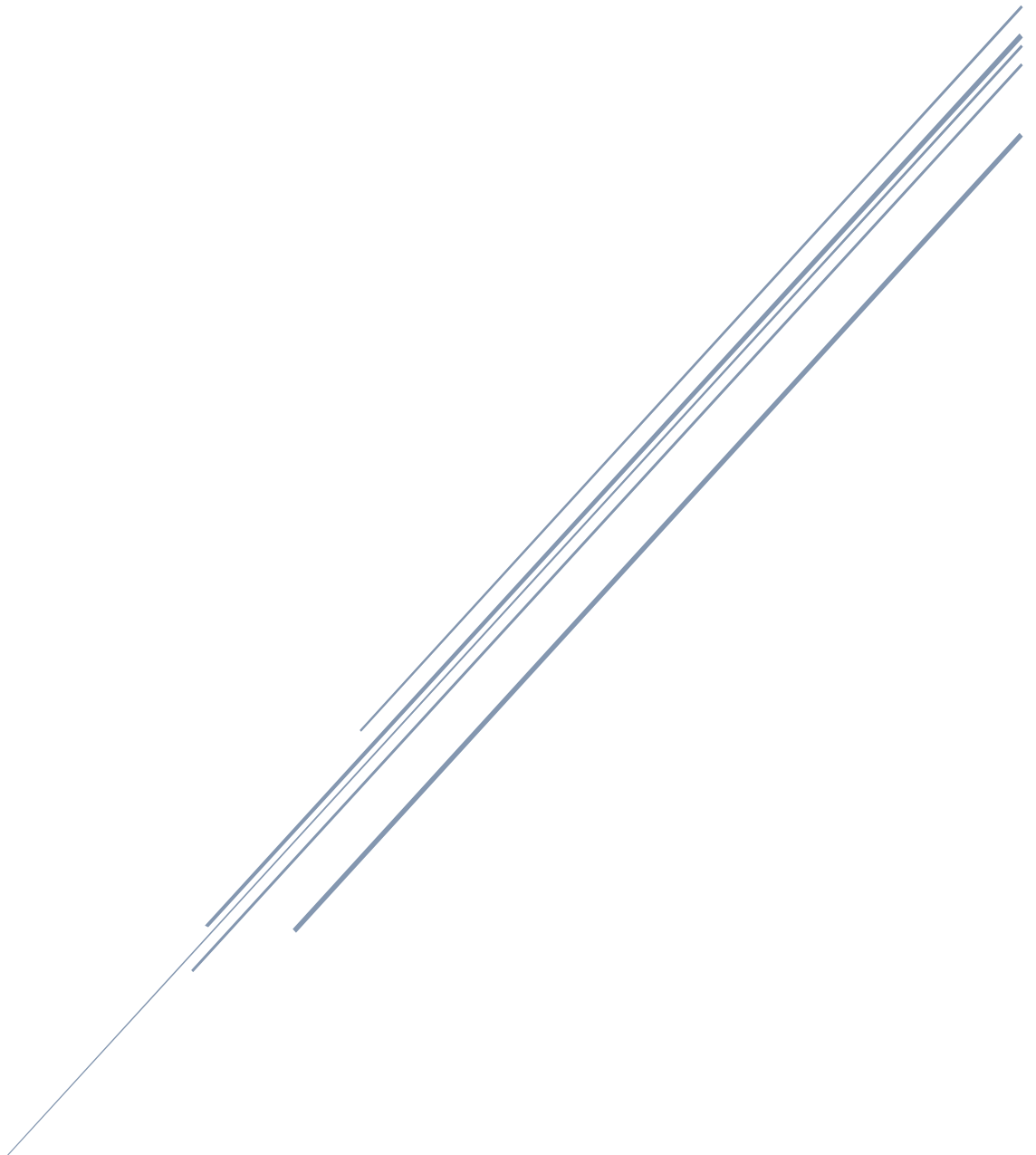


VERSCHLÜSSELUNG

DEZSYS06



Pitirut Stefan, Willinger Andreas
30.01.2015

Inhaltsverzeichnis

Aufgabenstellung	3
Designüberlegung:	4
Arbeitsaufteilung	6
Aufwandsschätzung	6
Arbeitsdurchführung	6
Erfolge:	6
Niederlagen:	6
Verwendete Technologien	7
AES	7
RSA	7
Testen des Programmes	8
Quellen	13

Aufgabenstellung

Kommunikation [12Pkt]

Programmieren Sie eine Kommunikationsschnittstelle zwischen zwei Programmen (Sockets; Übertragung von Strings). Implementieren Sie dabei eine unsichere (plainText) und eine sichere (secure-connection) Übertragung.

Bei der secure-connection sollen Sie eine hybride Übertragung nachbilden. D.h. generieren Sie auf einer Seite einen privaten sowie einen öffentlichen Schlüssel, die zur Sessionkey Generierung verwendet werden. Übertragen Sie den öffentlichen Schlüssel auf die andere Seite, wo ein gemeinsamer Schlüssel für eine synchrone Verschlüsselung erzeugt wird. Der gemeinsame Schlüssel wird mit dem öffentlichen Schlüssel verschlüsselt und übertragen. Die andere Seite kann mit Hilfe des privaten Schlüssels die Nachricht entschlüsseln und erhält den gemeinsamen Schlüssel.

Sniffer [4Pkt]

Schreiben Sie ein Sniffer-Programm (Bsp. mithilfe der jpcap-Library <http://jpcap.sourceforge.net> oder jNetPcap-Library <http://jnetpcap.com/>), welches die plainText-Übertragung abfangen und in einer Datei speichern kann. Versuchen Sie mit diesem Sniffer ebenfalls die secure-connection anzuzeigen.

Info

Gruppengröße: 2 Mitglieder

Punkte: 16

- Erzeugen von Schlüsseln: 4 Punkte
- Verschlüsselte Übertragung: 4 Punkte
- Entschlüsseln der Nachricht: 4 Punkte
- Sniffer: 4 Punkte

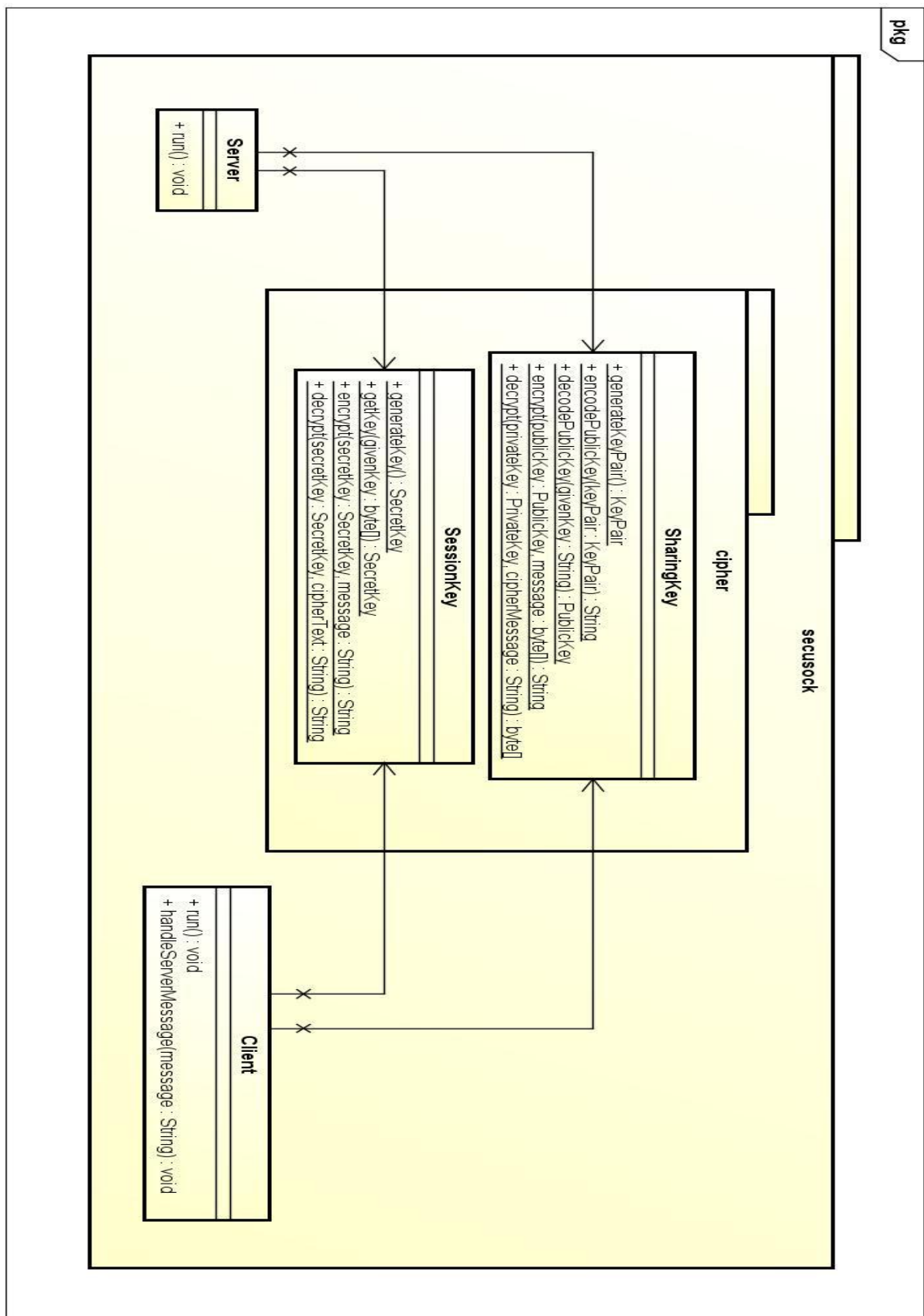
Designüberlegung:

Als erstes hatten wir die Idee, die jeweiligen Verschlüsselungsalgorithmen auszuprobieren, somit haben wir diese als Erstes versucht zu implementieren. Nachdem wir dies realisiert hatten, haben wir uns überlegt, die jeweiligen Schritte mittels einer „Start-Klasse“ darzustellen. Unser nächster Gedankengang war, wie wir die Übertragung der jeweiligen Schlüssel/Nachrichten realisieren sollten.

Den Entschluss, den wir gefasst haben war folgender:

- Client hat keinen sessionKey:
 - Der Befehl !keys muss eingegeben werden um die folgenden Schritte zu ermöglichen
 - erstellt den SharingKey basierend auf RSA
 - sendet den PublicKey via Sockets an den Server
 - → GENKEY | publicKey
 - Server erhält den verschlüsselten PublicKey
 - Server erstellt den AES secretKey und verschlüsselt ihn mithilfe des publicKeys
 - Server schickt diesen mithilfe von Sockets an den Client
 - → SETKEY | secretKey
 - Client erhält den secretKey und entschlüsselt ihn mithilfe des privateKeys und setzt ihn auch (local)
 - Wenn der !keys Befehl nicht verwendet wird, sendet der Client die Nachricht unverschlüsselt
 - → PLAINMSG | message
- Client hat sessionKey:
 - Client verschlüsselt Nachricht und sendet diese an den Server mittels Sockets
 - → ENCMMSG | message
 - Server entschlüsselt mithilfe des sessionKeys und zeigt die Nachricht an
- In beiden Fällen sendet der Server GOTMESSAGE | 0 zurück

Unser Konzept schaut wie folgt aus:



Arbeitsaufteilung

	Pitirut	Willinger
Dokumentation	X	X
Designüberlegung	X	X
Cipher		X
Client	X	X
Server	X	X
Sniffer	X	

Aufwandsschätzung

Pitirut	Geschätzte Zeit	Tatsächliche Zeit
Dokumentation	2 h	1 h 30 m
Designüberlegung	2 h	30 m
Client	1 h 30 m	1 h
Server	1 h 30 m	1 h
Sniffer	2 h	2 h 30 m
Gesamt:	7 h	6 h 30 m

Willinger	Geschätzte Zeit	Tatsächliche Zeit
Dokumentation	1 h	1 h
Designüberlegung	1 h	30 m
Cipher	2 h 30 m	3 h
Client	1 h 30 m	1 h
Server	1 h 30 m	1 h
Gesamt:	7 h 30 m	6 h 30 m

Arbeitsdurchführung

Erfolge:

- Sniffer war in Python einfacher als erwartet
- Encryption schneller realisiert als gedacht, jedoch Zeitverlust durch die Niederlage

Niederlagen:

- Beim Ausführen auf einer standard Java installation, ist das Problem aufgetaucht, dass ein „Invalid Key size or Parameters“ Fehler geworfen wird. Dieser entstand daraus, da wir nicht die JAR's hatten, um die Keysize limitation von AES (die Standardmäßig auf 16Bit limitiert ist) aufzuheben. Diese JAR's befinden sich in dem REQUIRED Ordner, diese MÜSSEN in das /lib/security Verzeichnis der JRE eingefügt werden.

Verwendete Technologien

AES

"The Advanced Encryption Standard is based on a design principle known as a substitutionpermutation network. Its predecessor DES used a Feistel network(a network of linked Feistel functions). AES's block size is fixed on 128 bits and a key size of 128, 192, or 256 bits. Based on the Rijndael specification the block and key sizes may be any multiple of 32 bits, both with a minimum of 128 and a maximum of 256 bits. Different from the last chapter DES, the number of rounds are specified as follows:

- *10 cycles of repetition for 128-bit keys*
- *12 cycles of repetition for 192-bit keys*
- *14 cycles of repetition for 256-bit keys*

Each round consists of 4 processing steps, some of them depend on the encryption key itself. To transform the ciphertext back into the original plaintext the same encryption key and a set of reverse rounds are applied." [1]

RSA

"RSA is by far the simplest public key encryption algorithm to understand, and to implement. It is named after the three inventors - Ron Rivest, Adi Shamir, and Leonard Adleman, and has also withstood years of extensive cryptanalysis (the study of cracking the ciphertexts)." [1]

Testen des Programmes

Um das Programm auszuführen, muss Apache Ant installiert sein.

Wenn das der Fall ist, wechselt man in das Verzeichnis, wo die build.xml liegt und führt folgenden Befehl aus:

```
ant jar
```

Dieser Befehl erzeugt eine JAR Datei, die im Ordner out/ gefunden werden kann.

Sobald der obige Befehl fertig ist, wechselt man in den Ordner out/ und das Programm kann mithilfe des folgenden Kommandos ausgeführt werden:

```
java -jar SecuSock_<nummer>_Pitirut-Willinger.jar
```

Wobei <nummer> das aktuelle Datum im Format YYYYMMdd ist, z.B. 20150129.

Im Folgenden wählt man dann den Typ der Ausführung aus (Server/Client) und gibt die benötigten Informationen ein (IP/port).

Den Sniffer startet man folgenderweise:

```
python3 sniffer.py <source_ip> <dest_ip>
```

Wobei <source_ip> <dest_ip> mit den jeweiligen IP's von Client/Server ersetzt werden. Des Weiteren muss in der sniffer.py Datei, die Variable HOST mit der IP des Interfaces ersetzt werden, über die die Kommunikation läuft (IP vom lokalen Rechner, bzw 127.0.0.1 wenn sie am selben Rechner sind)

Bei der Auswahl des Servers, muss das Folgende gemacht werden:

- 1) Nachdem man den Server ausgewählt hat, muss man den Port angeben, damit der Client sich über die IP des Servers und den angegebenen Port verbinden kann.

```

C:\Windows\system32\cmd.exe - java -jar SecuSock_20150129_Pitirut-Willinger.jar
C:\Users\Xhywheer\Documents\SecuSock>cd out
C:\Users\Xhywheer\Documents\SecuSock>ls
SecuSock_20150129_Pitirut-Willinger.jar  build
C:\Users\Xhywheer\Documents\SecuSock>java -jar SecuSock_20150129_Pitirut-Will
Welcome to SecuSock!
Please select one of the following implementations:
    [1] - Server
    [2] - Client
Your choice: 1
Please enter the port to listen on: 1234
Starting SecuSock server ..
SecuSock server started, waiting for connections ..
Connection accepted from /10.0.104.157:8752
Received plain-text message
-----
Message: test
Received plain-text message
-----
Message: Unencrypted message

```

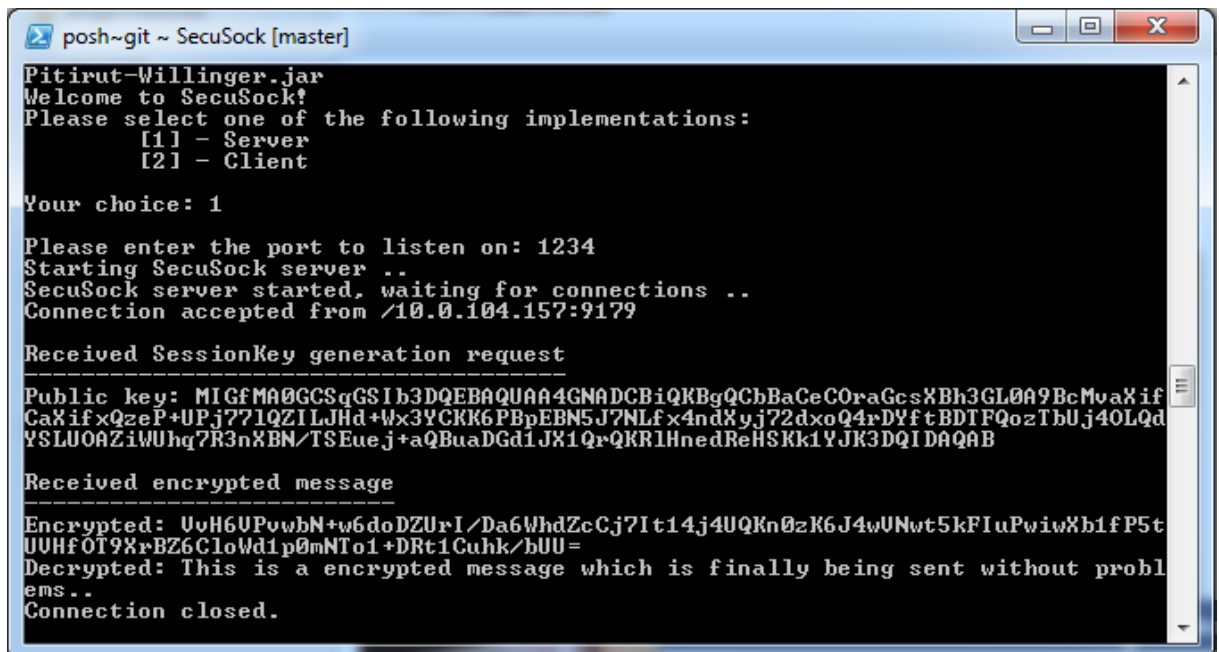
- 2) Man kann erkennen, dass der Server die Plain-text messages vom Client empfangen hat.
- 3) Als nächstes wird vom Client ein request für die Generierung eines SessionKeys an den Server geschickt, dieser erhält den PublicKey vom Client und erstellt den Sessionkey und schickt diesen dem Client zurück.

```

Your choice: 1
Please enter the port to listen on: 1234
Starting SecuSock server ..
SecuSock server started, waiting for connections ..
Connection accepted from /10.0.104.157:8752
Received plain-text message
-----
Message: test
Received plain-text message
-----
Message: Unencrypted message
Received plain-text message
-----
Message:
Received SessionKey generation request
-----
Public key: MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC2kUC6 +ATmqPKTfC03BYvvNEwv1YeA
5D6P83mY2Rro2nykc5FpbR34xZZQfmwQjrE9KWJ/8SExKt1ddUu9wrEJiuyOpqDY7PKr9PkbN/Ro0U88

```

- 4) Der letzte Schritt, ist das Erhalten der verschlüsselten Nachricht vom Client und er entschlüsselt sie zusätzlich. Zum Schluss erkennt man, dass der Client die Verbindung getrennt hat.



```

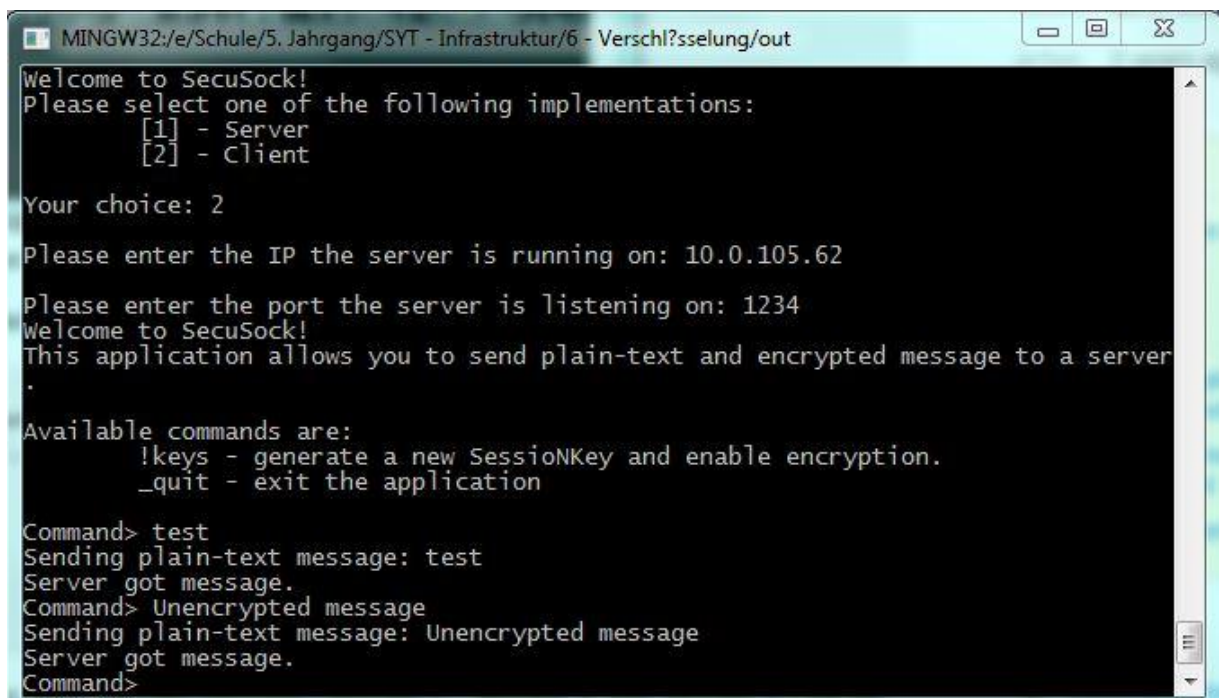
posh~git ~ SecuSock [master]
Pitirut-Willinger.jar
Welcome to SecuSock!
Please select one of the following implementations:
    [1] - Server
    [2] - Client
Your choice: 1
Please enter the port to listen on: 1234
Starting SecuSock server ..
SecuSock server started. waiting for connections ..
Connection accepted from /10.0.104.157:9179

Received SessionKey generation request
-----
Public key: MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCbBaCeC0raGcsXBh3GL0A9BcMvaXif
CaXifxQzeP+UPj77lQZILJHd+Wx3YCKK6PBpEBN5J7NLfx4ndXyj72dxoQ4rDYftBDTFQozTbUj40LQd
YSLU0AZiWUHQ7R3nXBN/TSEuej+aQBuaDgd1JX1QrQKR1HnedReHsKk1YJK3DQIDAQAB
-----
Received encrypted message
-----
Encrypted: UvH6UPvwbN+w6doDZUrI/Da6WhdZcCj7It14j4UQKn0zK6J4wUNwt5kFIuPwiwXb1fP5t
UvHf0T9XrBZ6C1oWd1p0mNT01+DRt1Cuhk/bUU=
Decrypted: This is a encrypted message which is finally being sent without probl
ems..
Connection closed.

```

Bei der Auswahl des Clients, muss das Folgenden gemacht werden:

- 1) Nachdem man den Client ausgewählt hat, muss man die IP des Servers angeben, und den Port auf welchem das Programm am Server läuft.



```

MINGW32:/e/Schule/5. Jahrgang/SYT - Infrastruktur/6 - Verschl?sslung/out
Welcome to SecuSock!
Please select one of the following implementations:
    [1] - Server
    [2] - Client
Your choice: 2
Please enter the IP the server is running on: 10.0.105.62
Please enter the port the server is listening on: 1234
Welcome to SecuSock!
This application allows you to send plain-text and encrypted message to a server
.

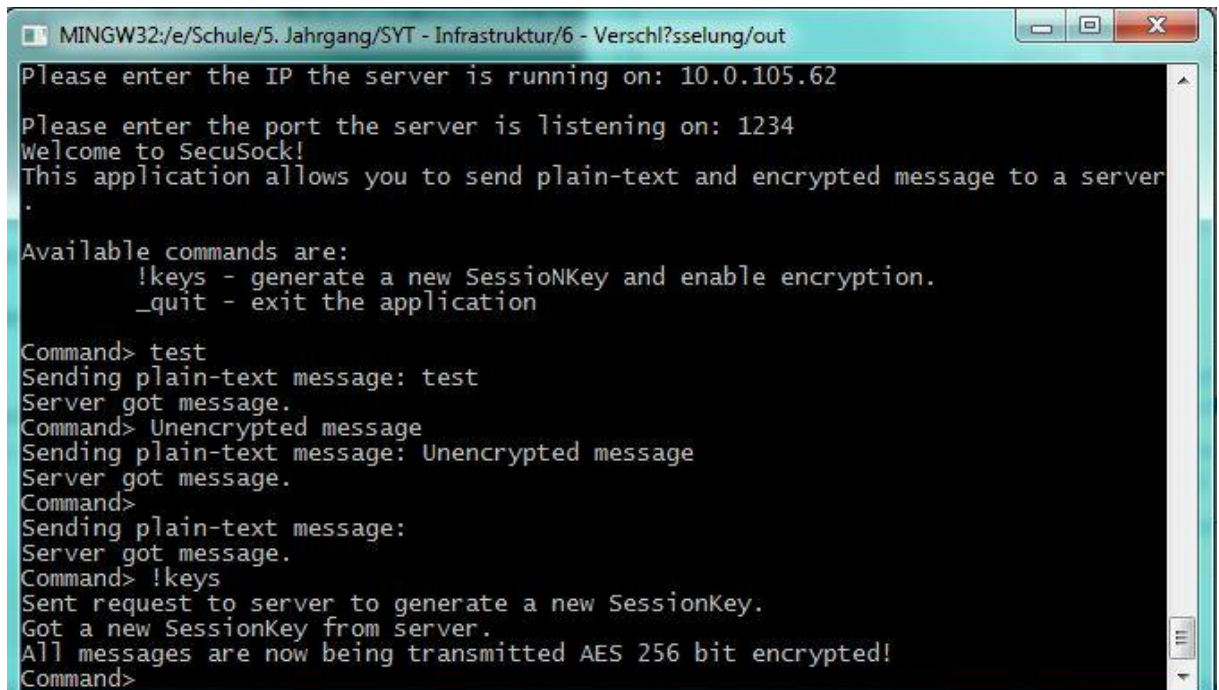
Available commands are:
    !keys - generate a new SessionKey and enable encryption.
    _quit - exit the application

Command> test
Sending plain-text message: test
Server got message.
Command> Unencrypted message
Sending plain-text message: Unencrypted message
Server got message.
Command>

```

- 2) Man kann erkennen, dass die Plain-text messages die der Client verschickt vom Server empfangen wurden.

- 3) Als nächstes wird durch den Befehl !keys eine Anfrage zur Erstellung eines SessionKeys an den Server geschickt, man kann auch erkennen, dass man einen erhalten hat.

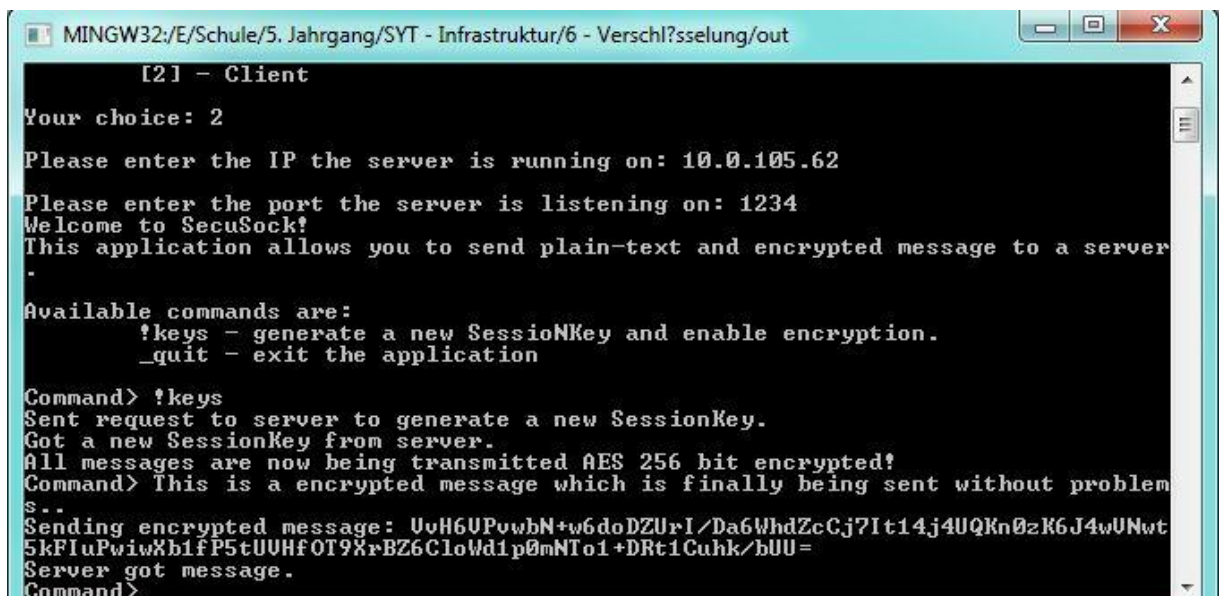


```

MINGW32:/e/Schule/5. Jahrgang/SYT - Infrastruktur/6 - Verschl?sslung/out
Please enter the IP the server is running on: 10.0.105.62
Please enter the port the server is listening on: 1234
Welcome to SecuSock!
This application allows you to send plain-text and encrypted message to a server
.
Available commands are:
    !keys - generate a new SessionKey and enable encryption.
    _quit - exit the application
Command> test
Sending plain-text message: test
Server got message.
Command> Unencrypted message
Sending plain-text message: Unencrypted message
Server got message.
Command>
Sending plain-text message:
Server got message.
Command> !keys
Sent request to server to generate a new SessionKey.
Got a new SessionKey from server.
All messages are now being transmitted AES 256 bit encrypted!
Command>

```

- 4) Nachdem man den SessionKey vom Server erhalten hat, kann man seine Nachricht verschicken, die wird dann Verschlüsselt beim Server ankommen. Man kann auch erkennen, dass der Server diese erhalten hat

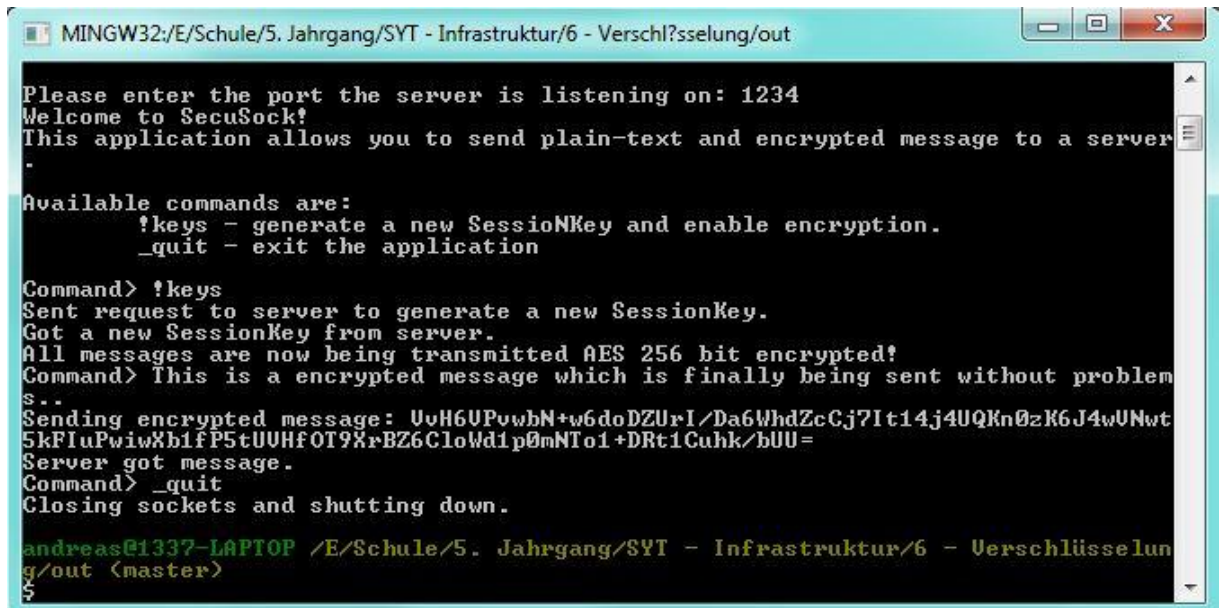


```

MINGW32:/E/Schule/5. Jahrgang/SYT - Infrastruktur/6 - Verschl?sslung/out
[2] - Client
Your choice: 2
Please enter the IP the server is running on: 10.0.105.62
Please enter the port the server is listening on: 1234
Welcome to SecuSock!
This application allows you to send plain-text and encrypted message to a server
.
Available commands are:
    !keys - generate a new SessionKey and enable encryption.
    _quit - exit the application
Command> !keys
Sent request to server to generate a new SessionKey.
Got a new SessionKey from server.
All messages are now being transmitted AES 256 bit encrypted!
Command> This is a encrypted message which is finally being sent without problem
s..
Sending encrypted message: UvH6UPowbN+w6doDZUxI/Da6WhdZcCj7It14j4UQKn0zK6J4wUNwt
5kFIuPwiwXb1fP5tUvHfOT9XrBZ6ClOWd1p0mNT01+DRt1Cuhk/bUU=
Server got message.
Command>

```

- 5) Zum Schluss, kann man die Verbindung vom Client aus trennen. Dies geschieht mit dem Befehl `_quit`



```
MINGW32:/E/Schule/5. Jahrgang/SYT - Infrastruktur/6 - Verschl?sslung/out

Please enter the port the server is listening on: 1234
Welcome to SecuSock!
This application allows you to send plain-text and encrypted message to a server
-
Available commands are:
    !keys - generate a new SessionKey and enable encryption.
    _quit - exit the application

Command> !keys
Sent request to server to generate a new SessionKey.
Got a new SessionKey from server.
All messages are now being transmitted AES 256 bit encrypted!
Command> This is a encrypted message which is finally being sent without problem
s..
Sending encrypted message: UvH6UPowbN+w6doDZUrI/Da6WhdZcCj7It14j4UQKn0zK6J4wUNwt
5kFluPwiwXbifP5tUUhFOT9XrBZ6CloWd1p0mNT01+DRt1Cuhk/bUU=
Server got message.
Command> _quit
Closing sockets and shutting down.

andreas@i337-LAPTOP /E/Schule/5. Jahrgang/SYT - Infrastruktur/6 - Verschlüsselun
g/out (master)
$
```

Quellen

[1] Gary Ye, Christian Janeczek, Cryptography, zuletzt bearbeitet: 19.01.15, zuletzt geöffnet: 29.01.15