Verschlüsselung

DEZSYS06

Pitirut Stefan, Willinger Andreas

28.01.2015

Inhaltsverzeichniss

[Aufgabenstellung 3](#_Toc410208067)

[Designüberlegung: 4](#_Toc410208068)

[Arbeitsaufteilung 7](#_Toc410208069)

[Aufwandsschätzung 7](#_Toc410208070)

# 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](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:

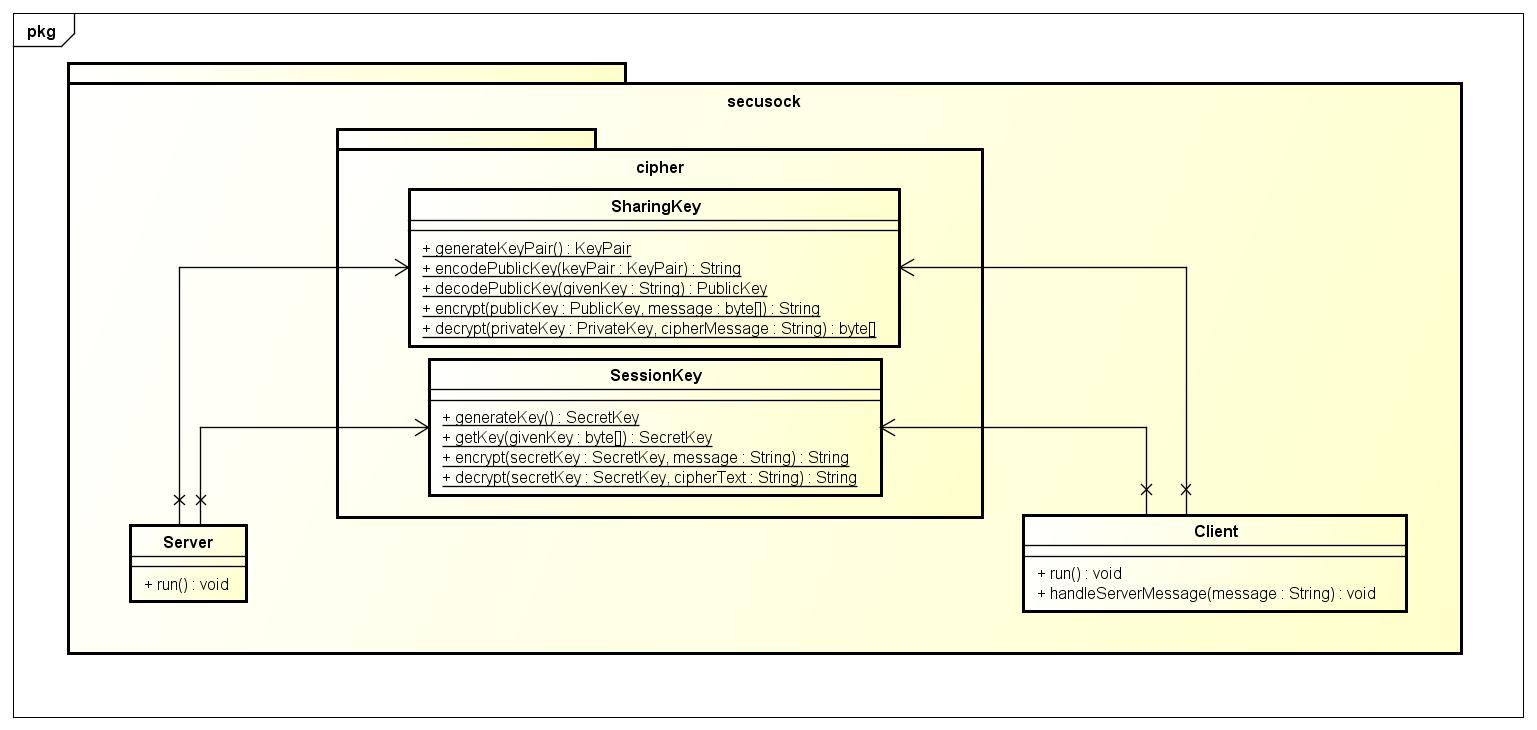
Als erstes hatten wir die Idee, die jeweiligen Verschlüsselungsalgorithmen auszuprobieren, somit habne wir diese als Erstes versucht zu implementieren. Nachdem wir dies realisiert hatten, haben wir uns überlegt, die jeweiligen Schritte mittels einer „Start-Klasse“ darzustellen.

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
    - 🡪 ENCMSG | 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 | 1h 30 m |
| **Gesamt:** | **9 h** | **5 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 | 1h 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

## RSA

## Sockets

# 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, kann das Programm 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).

# Quellen