Gruppe
Pawlowsky, Sochovsky

Verschlüsselung - Übung

VSDB 5BHITS 24.01.2012

Pawlowsky, Sochovsky

Inhalt

Aufgabenstellung

AUFGABE07 - VERSCHLÜSSELUNG VON NACHRICHTEN

Kommunikation [8Pkt]
Programmieren Sie eine Kommunikationsschnittstelle zwischen zwei Java-Programmen (Sockets; Übertragung von Strings). Implementieren Sie dabei eine unsichere (plainText) und eine sichere (secure) Übertragung. Bei der secure-connection sollen Sie eine hybride Übertragung nachbilden. D.h. generieren Sie auf beiden Seiten einen privaten sowie einen öffentlichen Schlüssel, die zur Sessionkey Generierung verwendet werden.

Sniffer [8Pkt]
Schreiben Sie mithilfe der jpcap-Library (http://jpcap.sourceforge.net/) oder jNetPcap-Library (http://jnetpcap.com/)ein Sniffer-Programm, welches die plainText-Übertragung abfangen und anzeigen kann. Versuchen Sie mit diesem Sniffer ebenfalls die secure-connection anzuzeigen. Optional können Sie bei unsicheren Algorithmen (bekannte Methodik, einfacher Schlüssel) versuchen, den asymmetrischen Teil der Übertragung zu entschlüsseln, um so den Sessionkey zu erhalten. Speichern Sie die abgefangenen Übertragungen ab, um eine Offline-Bruteforce Attacke durchführen zu können (z.B. als pcap-File).

Info
Gruppengröße: 2 Mitglieder
Punkte: 16 (+8 Extrapoints für SessionKey-Entschlüsselung beim Sniffer)

Inbetriebnahme der Applikation

Kommunikator

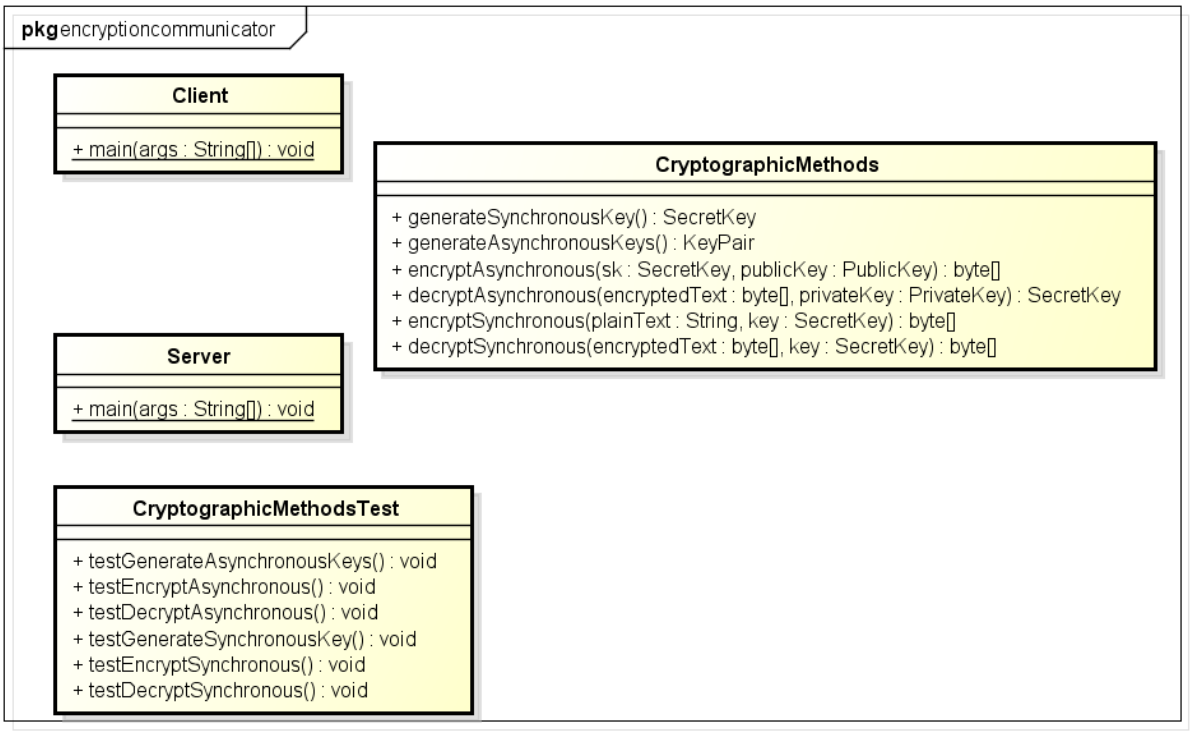
Dieser Teil der Applikation wird gestartet über Client und Server.(Es liegen .jar Dateien für den Aufruf bereit) Bei dem Server wird am Anfang ein Port abgefragt. Dieser Port wird für die TCP Kommunikation auf dem Server verwendet. Der Client verlangt beim Start die IP des Servers und den Port den man davor auch am Server eingestellt hat.
Danach folgt die Ausgabe der Tätigkeiten bevor die Verbindung vollständig aufgebaut wurde. Wenn dieser Prozess beendet ist, kann man mittels Eingabe von „crypt“, „plain“ oder „close“ zwischen Client und Server kommunizieren.

Sniffer

Zu allererst muss die beigelegte jpcap.dll in den bin-Ordner der JDK kopiert werden, die man zum Ausführen des Sniffers benutzen möchte. Danach muss die main-Methode des Sniffer (z.B. ebenfalls über die beiliegende .jar-Datei) gestartet werden. Dabei muss als Parameter die device-ID des Interfaces auf das man höhren will angegeben werden. Gibt man keinen Parameter an, so zeigt einem das Programm was für device-ID vorhanden sind und wie man diese benutzt. Anschließend wird man von der Applikation gefragt, auf welchen host und port man hören will und schon werden alle Pakete abgefangen.

Design

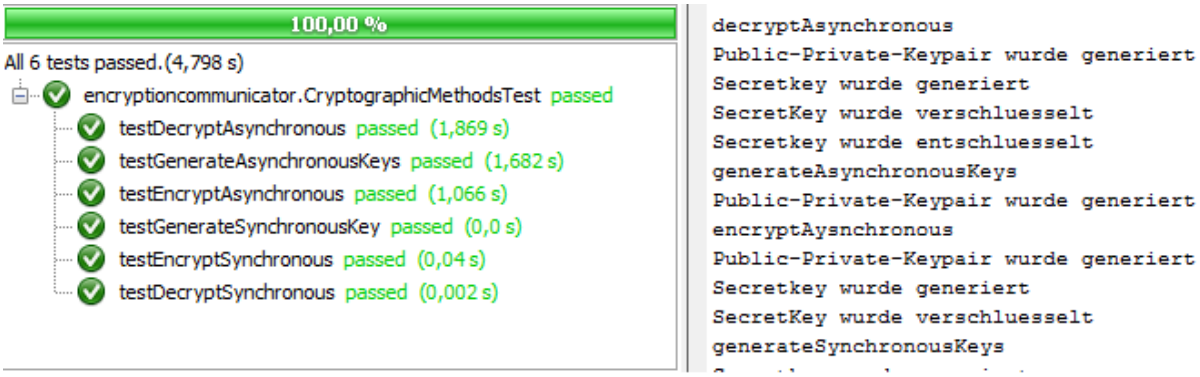
Kommunikator



Die Klasse CryptographicMethods ist das Kernelement für die Ver- und Entschlüsselung. Hier werden Keys generiert, ver- und entschlüsselt, ebenso wie man auch Text ver- und entschlüsselt.

Beim Start der Applikation wird am Server eine Public/Privatekeypair generiert. Beim Client ein Secretkey. Der Server versendet seinen Public Key als Plaintext. Der Client verschlüsselt den Secretkey mit dem Publickey, damit in der Server mit dem Privatekey entschlüsseln kann. Nach diesem Prozess kennen beide Applikationen den Secretkey und können verschlüsselt, aber auch unverschlüsselt miteinander kommunizieren.

In den Testfällen zu diesem Teil der Aufgabe wird jede Methode des Kernelements, also der CryptographicMethods Klasse, getestet. Das bedeutet es werden beide Keyarten generiert und überprüft ob das auch getan werden kann. Ebenso wird getestet ob Keys asynchron und Text synchron ver- und entschlüsselt werden kann.



Sniffer

Der Sniffer wurde mit Hilfe der jpcap-Library (JPCAP, 2002) realisiert. Dieser basiert auf WinPCAP auf dem auch schon Programme wie z.B. Wireshark aufsetzen. Leider ist diese JPCAP-Java-Library allerdings ein wenig veraltet und daher ist die letzte Kompilierung nur auf einem 32-bit System durchgeführt worden. Daher war sie auf unserem 64-bit System leider nicht nutzbar. Ich habe sie allerdings unter 64-bit neu kompiliert. Dies funktionierte wie folgt:

Kompilierung der JPCAP-Library

Da das letzte Release für JPCAP nur als 32-Bit-Version verfügbar ist, es sich bei JPCAP aber um ein OpenSource-Produkt handelt, mussten die Sources zur Verwendung auf einem 64-Bit-System neu kompiliert werden. Dazu muss man zuerst ein Visual Studio 2010 Projekt erstellen. Dieses muss ein ganz normales C/C++ WIN32 Projekt sein, dass zu einer DLL kompiliert werden soll. Anschließend muss man in den Project Settings den Konfigurations-Manager öffnen, bei „Aktive Projektmappenplattform“ auf neu klicken und x64 auswählen und dies speichern, um das Projekt für 64-Bit Systeme kompilieren zu lassen. Hat man dies geschafft muss man sich die Developer-Version (die normale sollte allerdings bereits installiert sein) von WinPCAP herunterladen und an einen gewünschten zugänglichen (nicht System32) Ort entpacken. Nun muss man in den Projekt-Eigenschaften unter Konfigurationseigenschaften und VC++-Verzeichnisse bei Includeverzeichnisse einige Ordner hinzufügen. Diese sind der Include-Ordner der entpackten WinPCAP Developer-Version, der Include Ordner einer JDK und den Include/Win32-Ordner derselben JDK. Bei sah dies zu ende so aus:

C:\Program Files (x86)\WinPcap\WpdPack\Include;C:\Program Files\Java\jdk1.7.0_07\include;C:\Program Files\Java\jdk1.7.0_07\include\win32;$(IncludePath)

Im selben Verzeichnis unter Bibliotheksverzeichnisse muss nur der lib/x64 Ordner der Developer-Version von WinPCAP hinzugefügt werden. Bei mir in etwa so:

C:\Program Files (x86)\WinPcap\WpdPack\Lib\x64;$(LibraryPath)

Nun muss wieder in den Projekt-Eigenschaften unter Konfigurationseigenschaften->Linker->Eingabe->Zusätzliche Abhängigkeiten folgendes hinzugefügt werden: Packet.lib;wpcap.lib;

Zu guter letzt muss noch unter Konfigurationseigenschaften->C/C++->Präprozessor->Präprozessordefinitionen „WPCAP“ hinzugefügt werden.

Nun können die Dateien jpcap.c, process.hpp und process.cpp aus dem Ordner src/java/net/sourceforge/jpcap/capture/ der heruntergeladenen JPCAP-Sources in das Projekt kopiert werden. Die Datei jpcap.c muss außerdem, um richtig kompiliert zu werden, im neuen Visual Studio Projekt in jpcap.cpp umbenannt werden. Danach muss im jpcap.cpp File noch folgendes hinzugefügt werden:

```
#ifndef WIN32
#define WIN32
#endif lines
```

Nun sollte die jpcap.dll erfolgreich gebuildet werden können. Ist dies der Fall kann man sie nun in den bin-Ordner der benutzten JDK oder JRE ziehen und sollte dann in der Java-DIE darauf zugreifen können.

Um nun eine Programmierung mit z.B. Netbeans zu ermöglichen muss das entsprechende JDK/JRE zum kompilieren gewählt werden und es müssen aus dem Sources-Archiv von JPCAP alle jars aus dem Ordner jars und alle aus thirdParty\jars in das Projekt integriert werden.

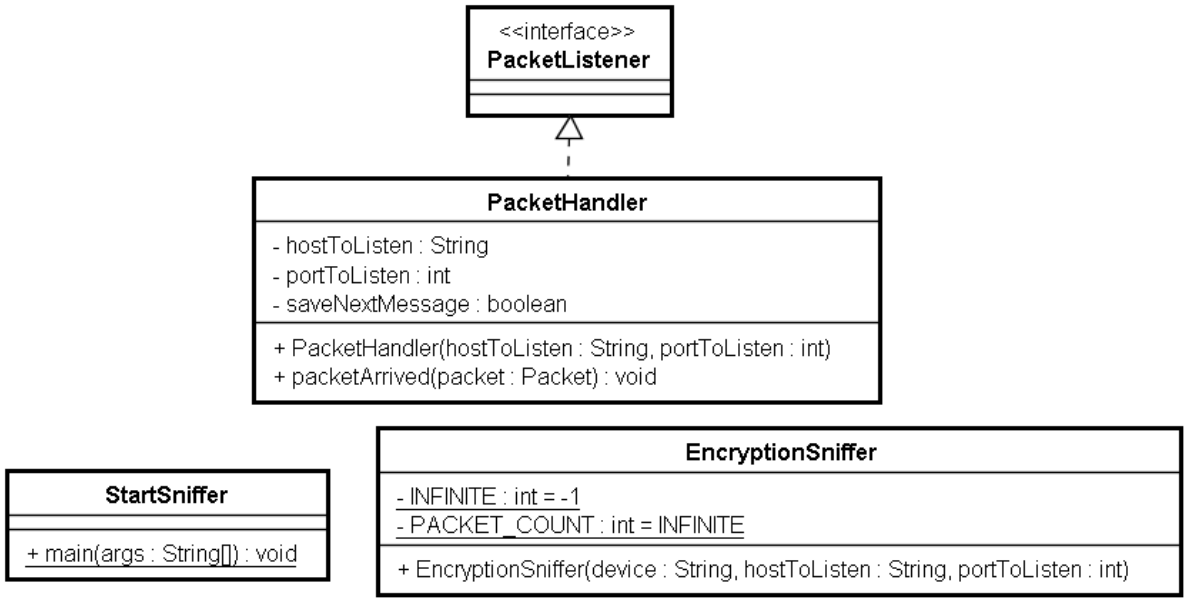
Dabei darf ich bei den Quellen auf die zwei folgenden Foren-Posts verweisen:

http://sourceforge.net/projects/jpcap/forums/forum/85831/topic/4587789

http://stackoverflow.com/questions/9790170/jpcap-compilation-installation-failing-on-gumstix

Design

Das Klassendesign sieht wie folgt aus:



Dabei haben wir uns dafür entschieden uns stark an das gegebene Sniffer-Beispiel von jpcap zu lehnen. Dieses snifft einfach den gesamten Netzwerk-Traffik auf einem Interfcae mit. Wir haben dieses Beispiel auf eine schöne Klassenstruktur aufgeteilt und eine Usereingabe hinzugefügt, sodass nur noch eine Kommunikation mit einer Hostadresse auf einem Port vom Sniffer angezeigt wird. Diese beiden können bei jedem Ausführen des Sniffers erneut gesetzt werden. Außerdem gab es eine zusätzliche Aufgabenstellung die besagt, dass bei verschlüsselten Nachrichten die herausgesnifft werden auch gleich versucht werden soll diese per brute force zu entschlüsseln. Wir haben uns dabei dafür entschieden, dass wir alle Nachrichten die verschlüsselt abgefangen werden in eine .txt-Datei gespeichert werden, damit diese dann extern entschlüsselt werden können.

Wir haben auch mit einer direkten Weiterleitung an einen Brute-forcer experimentiert. Dies hat auch eigentlich gut funktioniert, nur hatten wir das Problem, dass wir nicht wussten wann der Text erfolgreich entschlüsselt war, da die Software ja nicht automatisch erkennen kann ob es sich nach dem Entschlüsselungsversuch nun um den richtigen Text handelt.

Testfälle zum Sniffer konnten wir leider nicht durchführen, da er nur Kommunikationen über das Netzwerk und nicht am eigenen PC abfangen kann.

Arbeitsteilung

Task	Paw	Soc	Geplant:	Durchgeführt:
Kommunikator		x	2h	2h 11.01.13
		x	3h	3h 18.01.13
Sniffer	x		3h	2h 17.01.13
	x		2h	3h 18.01.13
	x		1h	1h 24.01.13

Gesamt		**x**	**5 h**	**5 h**
Gesamt	**x**		**6 h**	**6 h**
Gesamt	**x**	**x**	**11 h**	**11 h**

Resultate/Niederlagen

Pawlowsky:

> *Anfangs was es uns beim Sniffer nicht möglich die benötigte .ddl-Datei zur Verwendung von jpcap richtig in das Projekt zu importieren, da diese für Windows 32bit ausgelegt ist.*
>
> Wir haben nun mit Hr. Prof. Borko gesprochen, der uns verboten hat eine andere Library zu benutzen. Daraufhin haben wir wie im entsprechenden Punkt beschrieben die alte jpcap Library neu auf einem 64-bit Rechner mittel Visual Studio 2010 kompiliert. Danach hat es einwandfrei funktioniert.

Literaturverzeichnis

JPCAP, 2002. *JPCAP-Sourceforge.* [Online]
Available at: http://jpcap.sourceforge.net/
[Zugriff am 24 01 2013].

Oracle, 2010. *JavaTM Cryptography Extension (JCE).* [Online]
Available at: http://docs.oracle.com/javase/1.4.2/docs/guide/security/jce/JCERefGuide.html
[Zugriff am 11 01 2013].