Encrypted Chat Dokumentation

Informieren

Die Idee für dieses Projekt gab mir mein Vater. Ich fragte ihn, was ihm als Anwender interessant vorkommen würde. So entstand die Idee der Chat Applikation. Er legte ausserdem Wert auf den Aspekt, dass die Nachrichten verschlüsselt gesendet werden sollen. Demnach sin die Anforderungen an die Applikation wie folgt:

- Applikation mit welcher man einer Person im Netzwerk, die auch die Software auf ihrem Computer hat, Nachrichten senden kann.
- Die Nachrichten sollen beim Absenden verschlüsselt werden und beim Eintreffen direkt entschlüsselt.
- Fokus auf Anwenderfreundlichkeit -> Applikation muss über ein GUI und nicht über die Command Line angewendet werden können. Ausserdem soll die Verschlüsslung und der Handshake zwischen den Applikationen automatisch stattfinden

Planen

Diese Applikation schrieb ich in meiner Freizeit, ich legte also nicht allzu grossen Wert auf eine möglichst lupenreine Planung. Sollte ich die Applikation verwirklichen können ist das super, wenn nicht suche ich mir halt eine einfachere Aufgabe, dachte ich mir. Ich wusste im Vorhinein, dass ich JavaFX brauchen werde, darin hatte ich auch schon Erfahrungen gesammelt und es kam mir allgemein wie ein Lauf im Park vor, nachdem ich Swing programmiert habe. Es wird ausserdem etwas mit Netzwerk-Sockets auf mich zukommen, darin hatte ich noch sehr wenig Erfahrung. Das Ausmass der Verschlüsselung konnte ich auch noch nicht sehr gut bestimmen. Ich konnte mir aber vorstellen, dass das Bestimmen des Keys, welcher jedes Mal anders sein soll, nicht plaintext gesendet werden darf und in jedem Fall genau der gleiche auf beiden Maschinen sein muss, schwierig werden könnte.

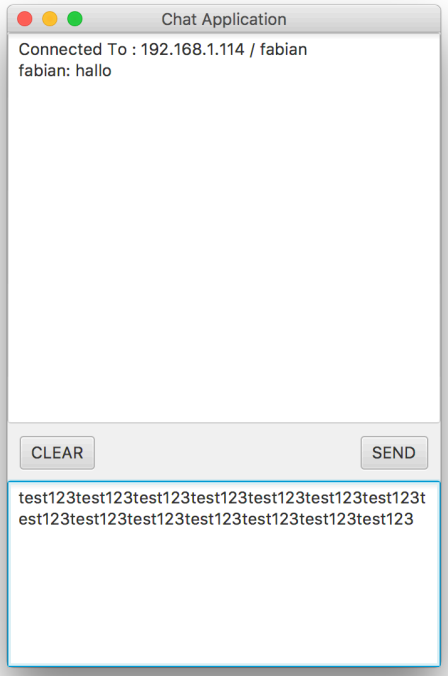
Erkennen

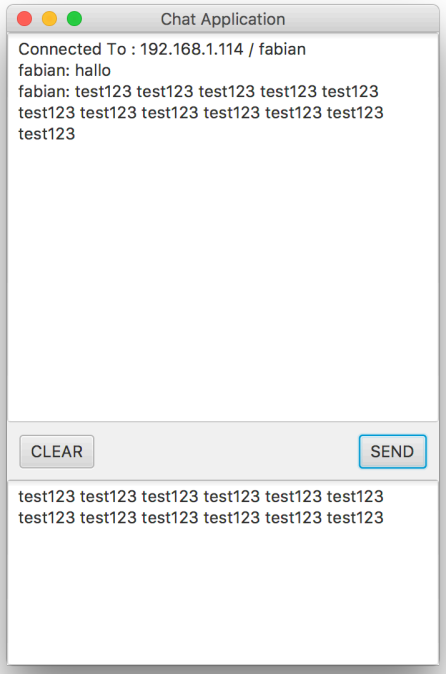
Wie schon in der Planung erwähnt, machte ich mir um die grundsätzliche Funktionalität Sorgen, sprich, wird es mir gelingen überhaupt irgendwelche Informationen über das Netzwerk zu schicken. Das Bestimmen des Keys machte mich auch ein wenig nervös.

Realisieren

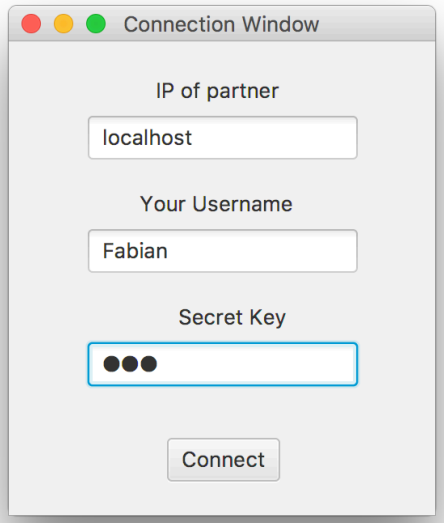
Ich begann im Flug nach Amsterdam mit dem Design des GUIs. Zu diesem Zeitpunkt machte ich nur das Chat GUI und noch nicht das Fenster, um die Verbindung herzustellen, weil ich noch keine Ahnung hatte, wie dieses aussehen soll (Muss man sich über die IP oder MAC Adresse verbinden, gibt man dem User die Möglichkeit, den Key zu bestimmen oder macht man einfach Etwas im Hintergrund usw.) Beim Design des Chat GUIs legte ich Wert, auf grundsätzliche Kompetenzen, wie man sie schon von Whatsapp u.a. kennt -> Es soll ein Feld zur Text Ein und Ausgabe geben, Textwrapping, das Feld soll scroll bar sein, bei längeren Konversationen und es soll in diesem Fall immer ans Ende der Konversation scrollen, sodass man nicht beim Eintreffen einer neuen Nachricht zuerst nach unten scrollen muss, um diese zu lesen. Den letzten Punkt konnte ich noch nicht im GUI abhaken, ich nahm mir vor, es im Code irgendwie zu bewerkstelligen.

Hier das GUI (zu diesem Zeitpunkt war das „Connected to...“ natürlich noch nicht implementiert):





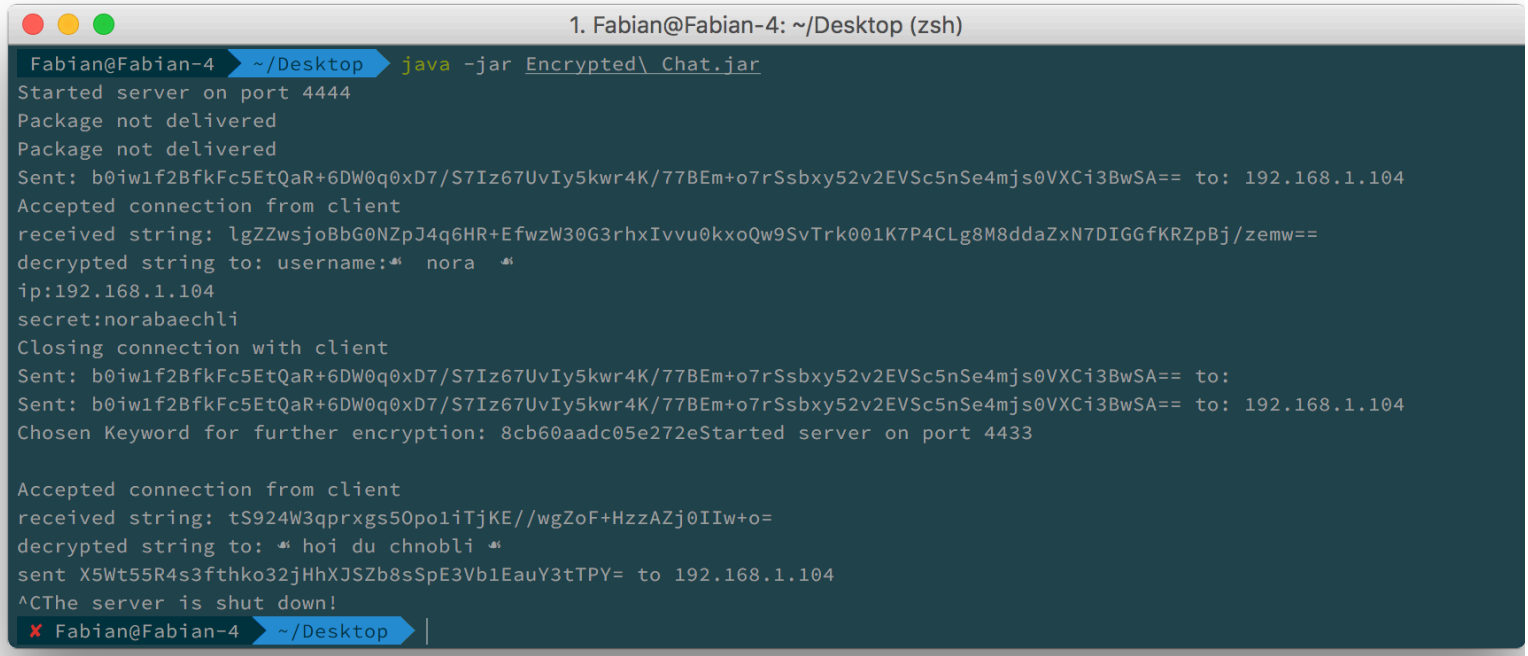
Danach genoss ich erst mal meine Ferien in Amsterdam. An einen regnerischen Tag dann aber, setzte ich mich ins Café und informierte mich über Netzwerk Sockets. Nach einiger Zeit, hatte ich eine Test-Applikation, welche den Port 4444 öffnet und eingehende Nachrichten ausgibt. Mit telnet localhost 4444 testete ich das dann. Als ich das hatte, begann ich das Fenster, welches helfen soll, die Verbindung herzustellen, zu machen. Das Endprodukt:



Als Nächstes, begann ich den Handshake-Socket besser auf meine Bedürfnisse abzustimmen. Ich beschriebe hier das Endprodukt, weil ich die genaue Reihenfolge nicht mehr im Kopf habe und es die ganze Sache überkomplizieren würde.
Der Handshake-Socket auf dem Port 4444, wird eingeschaltet, sobald alle Felder in der Connection Window ausgefüllt wurden und der „Connect“ Button gedrückt wurde. Er wartet auf ein Handshake-Package, welches die Informationen „username:<text>“, „ip:<text>“, „secret:<text>“ enthält. Dieses Handshake-Package wird jede Sekunde geschickt, sobald der „Connect“ Button gedrückt wurde. Sobald dieses Packet eingetroffen ist, wird ein Letztes Handshake-Package an den Zielort geschickt, sodass dieser auch ganz sicher ein Handshake-Package erhalten hat und eine variable namens canContinue wird auf true gesetzt und man gelangt in das eigentliche Chat Fenster. Das habe ich alles beinahe an einem Tag geschrieben (viel Zeit für Essen blieb aber nicht). Und ich machte hohe Luftsprünge als ich es endlich schaffte, das neue Fenster anzuzeigen (Das ist wirklich sehr umständlich gelöst in JavaFX).
Am nächsten Tag schrieb ich den Socket in der eigentlichen Chat Applikation, welcher jeglichen Input Stream auf dem Label für Nachrichten anzeigte.
Zum Schluss baute ich noch die Verschlüsslung ein. Auf dem Internet fand ich Code, welcher eine AES Ver- und Entschlüsselung implementierte. Ich probierte lange mit dem Code in einem Test-Projekt rum und fand, dass es das Richtige für meine Anwendungszwecke sei. Nun ging es an das Bestimmen des Keys. Das senden des Handshake-Package wird mit einem geheimen, 16-Stelligen Wort verschlüsselt, welches immer das Gleiche bleibt. (Über die Sicherheitslücke, dass man das Class File decompilen kann und somit das geheime Wort finden kann und in Folge vom dem das Handshake Package entschlüsseln kann, ist mir bewusst).
Das Verschlüsseln der Nachrichten, die danach kommen, funktioniert so: Es werden beide Secret Keys miteinander verglichen, ein String wird gebildet, welcher zuerst das längere, dann das kürzere Keyword enthält, wenn die Keywords identisch sind, ist die Reihenfolge egal. Wenn beide Keywords gleich lang sind, aber nicht identisch, kommt zuerst das Wort, welches als erstes einen Char mit höherem Wert hat. Sprich: Keyword1 = „abc“ und Keyword2 = „abd“ ergibt diesen String = „Keyword2 + Keyword1“. Danach wird auf den String ein MD5 Hash angewandt und die ersten 16 Characters des Hashs ergeben dann das Keyword für die folgende Ver und –Entschlüsselung.

Kontrollieren

Um die Kontrolle zu machen, werde ich den Command Line Output, welcher in einer wirklichen Anwendung mit meiner Schwester zustande kam, erläutern:



1. 1. Zeile: Der „Connect“ Button wurde in dem Verbindungsfenster gedrückt. Es startet der Handshake-Socket auf dem Port 4444, der auf das Handshake-Package wartet
2. 2. und 3. Zeile: Das Handshake-Package wird geschickt, jedoch ist der „Connect“ Button in der Applikation, welche auf dem Mac meiner Schwester läuft noch nicht gedrückt worden und somit kommt es nicht an.
3. 4. Zeile: Die Applikation bei meiner Schwester hat den Handshake-Socket freigegeben und somit kann ich mein verschlüsseltes Handshake-Package an diesen Socket schicken (Dieses Packet wird immer mit dem gleichen Schlüssel verschlüsselt)
4. 5. und 6. Zeile: Meine Schwester schickt mir wiederum ein verschlüsseltes Handshake-Package
5. 7., 8. und 9. Zeile: Ich entschlüssle das erhaltene Packet und bestimme aufgrund von dem, an welche IP ich in Zukunft Nachrichten schicke, wie der Benutzername meines Chat Partners ist und mit welchem Secret die später folgenden Nachrichten verschlüsseln werde.
6. 10. Zeile: Nach dem erhaltenen Packet schliesse ich meinen Socket
7. 11. und 12. Zeile: Nur zum sicher sein, schicke ich mein Handshake-Package nochmals
8. 13. Zeile: Aufgrund von dieser Logik bestimme ich das Keyword:

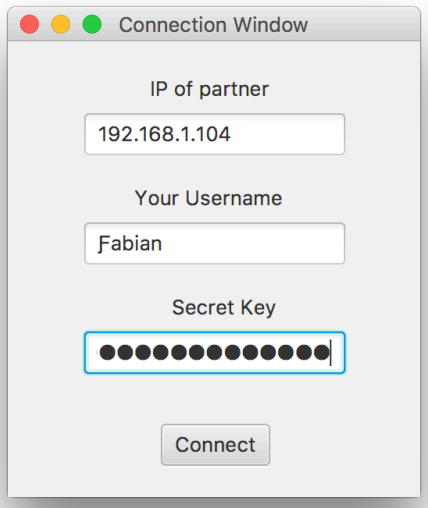
```
// Choosing the keyword and generating the hash
if (secretOfClient.length() ≥ ownSecret.length()) {
    communicationKeyWord = Encryption.sha256(secretOfClient + ownSecret).substring(0, 16);
} else if (secretOfClient.length() ≤ ownSecret.length()) {
    communicationKeyWord = Encryption.sha256(ownSecret + secretOfClient).substring(0, 16);
} else if (ownSecret.matches(secretOfClient)) {
    communicationKeyWord = Encryption.sha256(ownSecret + secretOfClient).substring(0, 16);
} else {
    for (int i = 0; i ≤ ownSecret.length(); i++) {
        if (ownSecret.charAt(i) ≥ secretOfClient.charAt(i)) {
            communicationKeyWord = Encryption.sha256(ownSecret + secretOfClient).substring(0, 16);
            break;
        } else if (ownSecret.charAt(i) ≤ secretOfClient.charAt(i)) {
            communicationKeyWord = Encryption.sha256(secretOfClient + ownSecret).substring(0, 16);
            break;
        }
    }
}
System.out.println("Chosen Keyword for further encryption: " + communicationKeyWord);
```

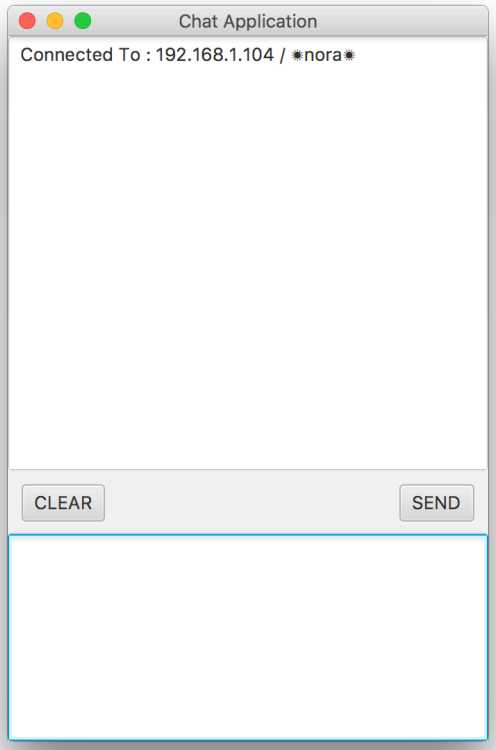
9. 15. Zeile: Auf beiden Maschinen ist jetzt der Port 4433 offen für Nachrichten
10. 16. und 17. Zeile: Ich erhalte einen verschlüsselten Text von meiner Schwester, welchen ich zu „hoi du Chnobli“ entschlüssle (Ich zeigte meiner Schwester an diesem Tag die Unicode Tabelle, darum sind überall Sonderzeichen)
11. 18. Zeile: Ich sende eine verschlüsselte Nachricht an meine Schwester
12. 19. Zeile: Ich beende die Applikation und der Thread, welcher den Socket offenhält, endet.

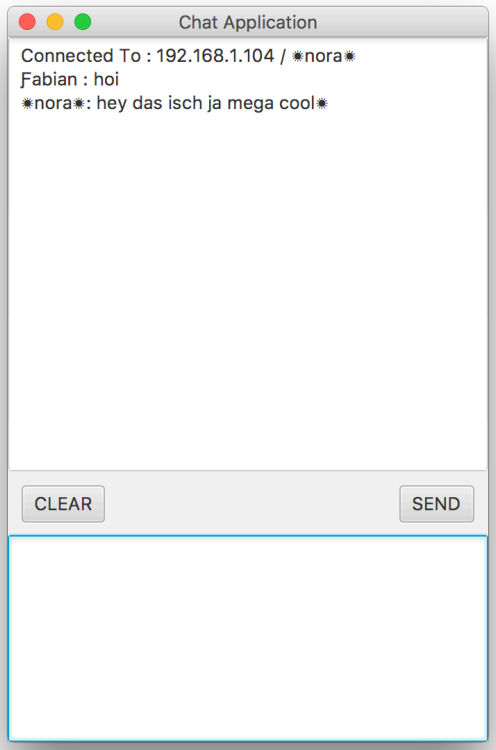
Meine Schwester und ich testeten auch verschiedene Konstellationen von Keywords, um zu schauen ob in einem Fall nicht der Gleiche Key zur Verschlüsselung oder Entschlüsselung angewandt werde. Meine Applikation hielt diesem Test aber stand und wir erhielten in jedem Fall das gleiche Keyword.
Ich testete ausserdem, ob ich mit localhost als Partner kommunizieren kann, das hat auch funktioniert.

Analysieren

Das Endprodukt:







Ich persönlich bin sehr zufrieden mit dem Endprodukt. Ich bin stolz, dass ich die gesamte Funktionalität, die ich mir am Anfang vornahm, realisieren konnte. Ich habe während dem Projekt viel über Ports und Threads gelernt. Ich finde, dass ich auch sehr objektorientier gearbeitet habe. Ich benutze zwar überall, wo ich konnte statische Variablen und Methoden, an einigen Orten aber musste ich mit Getter und Setter arbeiten. So gibt der Handshake Socket nach dem Eintreffen eines Handshake-Package ein Objekt zurück, welches die Methoden getIp, getUsername und getSecret hat. Mit statischen Variablen wäre das schwerer zu lösen gewesen.