Hochschule Düsseldorf
University of Applied Sciences

Prof. Dr. Holger Schmidt
Professur für Informatik, insb. IT-Sicherheit
Fachbereich Medien – Hochschule Düsseldorf

Fachbereich Medien
Faculty of Media

Lastenheft „Secure Text Editor“

Ziel ist die Implementierung eines *Secure Text Editors* (STE), der Textdateien unter Verwendung verschiedener kryptografischer Verfahren schützen kann.

Funktionale Anforderungen

Der STE MUSS über grundsätzliche Funktionen eines Editors verfügen und MUSS das Erstellen und Bearbeiten von Texten ermöglichen. Die Texte MÜSSEN beim Abspeichern und Laden ver- bzw. entschlüsselt werden können. Folgende kryptografische Verfahren MÜSSEN zur Auswahl stehen:

- Symmetrische Verschlüsselung:
  - `AES` mit Schlüssellängen 128, 192, 256 Bit
- Passwort-basierte Verschlüsselung:
  - AES 256 Bit, GCM, SCRYPT
  - `PBEWithSHA256And128BitAES-CBC-BC`
  - `PBEWithSHAAnd40BitRC4`
- Digitale Signatur (Erzeugung und Verifikation):
  - `SHA256withDSA`

In Abhängigkeit des gewählten Verfahrens, MÜSSEN die Padding Mechanismen `NoPadding`, `PKCS7Padding` und `ZeroBytePadding`, sowie als Blockmodi `ECB`, `CBC`, `OFB`, `CTS`[1] und `GCM` zur Verfügung stehen. Der STE MUSS die benötigten Schlüssel generieren können.
Der STE MUSS Manipulationen an abgespeicherten Texten erkennen können, indem ein Hashwert mit abgespeichert und beim Laden des Textes verifiziert wird. Dazu MUSS der Nutzer auf folgende Hashfunktion und MACs zurückgreifen können:

- `SHA-256`
- `AESCMAC`
- `HMACSHA256`

Das Speichern der Schlüssel, Texte, Hashwerte und MACs mit entsprechenden Metadaten MUSS (ggf. jeweils separat) in einem strukturierten Dateiformat (z. B. basierend auf XML oder JSON) erfolgen.

[1] CTS wird manchmal auch als Padding bezeichnet.

Nicht-funktionale Anforderungen

Der STE SOLL in Java und MUSS mithilfe der aktuellsten Version von `The Legion of the Bouncy Castle`[2] implementiert werden. Die Nutzerinteraktion SOLL über eine grafische Benutzeroberfläche erfolgen. Die Softwarearchitektur MUSS eine einfache Erweiterbarkeit erlauben, sodass zusätzliche Kryptografiekomponenten leicht ergänzt werden können.
Der STE MUSS inline dokumentiert werden. Dazu MUSS für Java `Javadoc` und `Doxygen`[3] verwendet werden. Weiterhin MUSS eine Validierung basierend auf dem Unit-Test-Framework implementiert werden Dazu MUSS für Java `JUnit`[4] verwendet werden. Weiterhin MUSS eine Analyse der Testabdeckung hinsichtlich Anweisungs-, Zweig- und Bedingungsabdeckung mithilfe eines Werkzeugs wie z. B. `JaCoCo`[5] durchgeführt werden.
Idealerweise arbeiten Sie mit `IntelliJ IDEA`[6] oder `Eclipse`[7], `Git`[8] und `Maven`[9].
`IntelliJ IDEA Ultimate` ist als Teil des `JetBrains Product Pack for Students` kostenlos `for educational use only` durch Registrierung auf https://account.jetbrains.com/login erhältlich.

[2] https://www.bouncycastle.org, aufgerufen am 31.01.2019
[3] http://www.stack.nl/~dimitri/doxygen/, aufgerufen am 31.01.2019
[4] http://junit.org/, aufgerufen am 31.01.2019
[5] http://www.eclemma.org/jacoco/, aufgerufen am 31.01.2019
[6] https://www.jetbrains.com/idea/, aufgerufen am 31.01.2019
[7] https://eclipse.org/, aufgerufen am 31.01.2019
[8] https://git-scm.com/, aufgerufen am 31.01.2019
[9] https://maven.apache.org/, aufgerufen am 31.01.2019