Anleitung

Passwörter hashen und salten in Java

M214

19. Dezember 2021

Von Manuel Schumacher, Joaquin Koller und Yannick Basler

1 Inhaltsverzeichnis

1. Einleitung

In dieser Anleitung wird gezeigt, wie Passwörter gesaltet und gehasht werden können. Am Ende hat man eine Klasse, welche Passwörter verschlüsselt. Dies wird in der Programmiersprache Java umgesetzt, kann jedoch in ähnlichem Format in andere Programmiersprachen umgewandelt werden.

Wichtig

Um dieser Anleitung folgen zu können, wird ein Grundverständnis von Kryptografie, Java und Passwortsicherheit benötigt.

1.1 Hashing

Hashing wird typischerweise dann verwendet, wenn die Verwaltung des Originalstrings zu aufwendig oder riskant wäre und wenn man diesen Wert mit einem anderen vergleichen möchte. Ein solcher Anwendungsfall wäre die Verwaltung von Passwörtern, da vertrauliche Daten nicht für jeden einsehbar sein dürfen. Über hashing wird das Passwort in eine Richtung verschlüsselt, wodurch man nie zurück auf das Originalpasswort kommen kann. Man kann lediglich alle Möglichkeiten ausprobieren und so auf den Originalwert kommen.

1.2 Salting

Salting ist eine Methode, um auch bei identischen Passwörtern unterschiedliche Hashes zu generieren. Dabei wird ein Zufallswert generiert, um dem Passwort anzubinden, bevor dieser zu einem Hash transformiert wird. Dadurch ist man gegen sogenannte *Rainbowtable Angriffe* sicher. Diese können nur mit ungehashten Passwörtern umgehen. Zudem muss ein Angreifer jeden einzelnen Hash entschlüsseln, da auch bei gleichem Passwort ein anderer herauskommt. Dies verlängert die Dauer und den Aufwand, welcher gebraucht wird, um alle Passwörter zu bekommen.

2 Hinweise

2.1 Symbole

In dieser Anleitung werden folgende Symbole[1] für die Verdeutlichung verwendet:

Dieses Symbol weist darauf hin, dass ein Abschnitt genau zu beachten und durchzulesen ist. Somit sollte dieser nicht übersprungen, damit es zu keinen Fehlern führt.

Dieses Symbol weist auf einen Tipp hin, welcher als Hilfestellung gilt. Dieser Schritt ist meist optional für die Vollständigkeit und Verständigkeit notwendig.

Dieses Symbol weist auf die Herkunft der genannten Information hin. Diese Quellen können ihren Inhalt ohne Vorwarnung ändern, aus diesem Grund gibt es keine Gewährleistung der Richtigkeit und Vollständigkeit.

Dieses Symbol zeigt, dass Kontrollfragen kommen. Diese helfen zur Überprüfung, um sicherzugehen, dass die Schritte richtig durchgeführt wurden und es funktioniert.

Dieses Symbol weist auf eine Leitfrage hin. Diese sollten zum Nachdenken und Vertiefen anregen. Es gibt keine Komplettlösungen auf diese Fragen.

Grüne Schrift z.B Beispiel

Begriffe die Kursiv und in grüner Farbe dargestellt werden sind Begriffe, die im Glossar erklärt werden auf der letzten Seite.

2.2 RITA

Diese Anleitung ist nach RITA[2] aufgeteilt. RITA ist ein Lernprozessmodell zum Verarbeiten neuer Informationen, zur Transferanbahnung und zum Auswerten des Lernprozesses.

Ressourcen Aktivieren

Informationen verarbeiten

Transfer Anbahnen

Auswerten

[1] *https://moodle.bztf.ch/pluginfile.php/111787/mod_folder/content/0/SPc420n_Bedienungsanleitung__Software.pdf*

[2] *https://www.methodenwuerfel.ch/lernprozessmodell/*

3 Ablauf

Der Ablauf von Hashing mit Salt ist von der Grundstruktur aus sehr einfach. Der Benutzer gibt sein Passwort ein, welches dann von einer Klasse angenommen wird. Über die eingebaut Java Klasse MessageDigest, welche später noch genauer erklärt wird, wird ein Algorithmus ausgewählt. Die Hashing Klasse generiert einen zufälligen Salt, welcher jedes Mal einen anderen Wert beinhaltet. Der Salt und das Passwort werden der Klasse übergeben, welcher die ganze Arbeit übernimmt und einen Hash generiert. Der Hash wird dann in ein hexadezimales Format umgewandelt und schlussendlich zurückgegeben oder gespeichert. Der Ablauf ist in folgendem Sequenzdiagramm dargestellt:

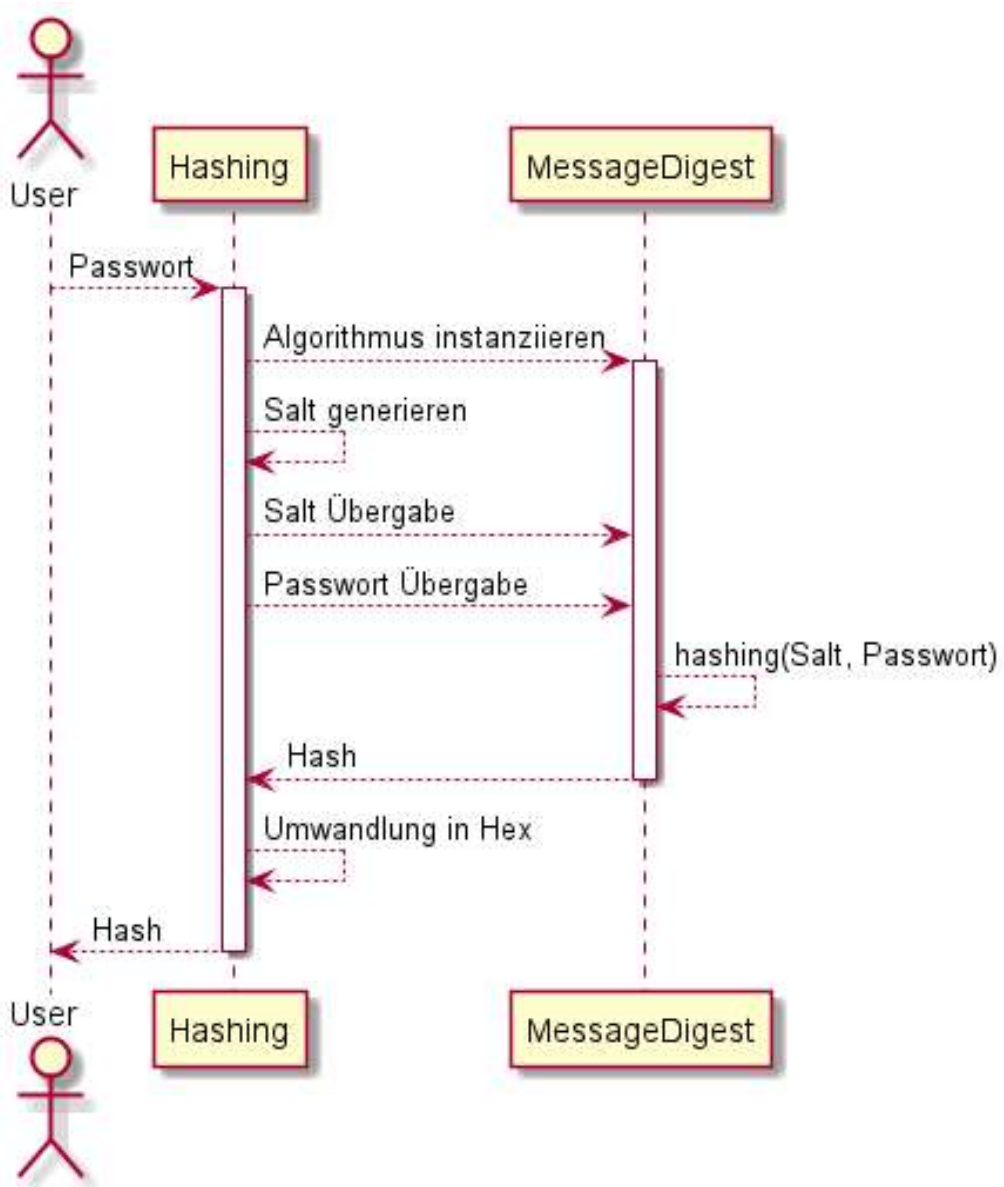

Abbildung 1 Sequenzdiagramm Hashing Ablauf

4 Anleitung

4.1 Vorlage

Für diese Anleitung gibt es eine Vorlage mit Musterlösung, welche sich auf *GitHub* befindet. Die Musterlösung beinhaltet Erklärungen in Form von Kommentaren, welche die einzelnen Schritte erklären. Es wird empfohlen, dass man diese Vorlage verwendet, da Bibliotheken schon importiert wurden. Zudem befindet sich eine vorgegebene Methode darin, welche somit nicht mehr selbst geschrieben werden muss.

4.1.1 Repository klonen

1 Option Webbrowser

1. Geben Sie die URL *https://github.com/21r8390/m214 in Ihrem Browser ein.*
2. Downloaden Sie die Zipdatei (siehe Bild).

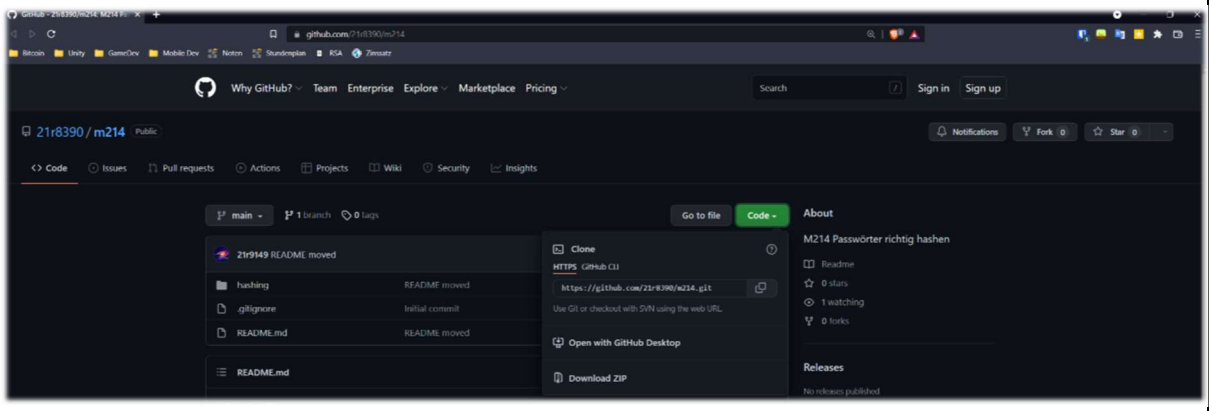

Abbildung 2 GitHub klonen

2 Option Command Line Interface

1. Öffnen Sie ihr Terminal.

Abbildung 3 Windows Git CMD App

2. Navigieren Sie mit «cd» zum gewünschten Verzeichnis.
3. Klonen Sie das Repository mit folgendem Befehl:
   `git clone https://github.com/21r8390/m214`

Wichtig

Damit diese Option funktioniert müssen Sie *Git* bereites installiert haben. Ansonsten können Sie dies unter https://git-scm.com/downloads nachholen.

Kontrollfrage

- Sind alle Dateien und Ordner wie in GitHub vorhanden?
- Keine Fehlermeldung erhalten beim Klonen?

4.2 Ordnerstruktur

Der Arbeitsbereich enthält standardmässig drei Ordner, und zwar:

- **src**: der Ordner zur Verwaltung des Codes
    - **Musterlösung**: der Ordner mit der Musterlösung am Ende
- **bin**: der Ordner mit den kompilierten Java Dateien

4.2.1 Dateistruktur

Es gibt zwei Hauptdateien, welche verwendet werden:

- **App.java**: die Klasse mit dem Main / nimmt Passwort an
- **Hasher.java**: die Klasse, welche das Passwort hasht und salted

Es gibt in der Musterlösung noch eine weitere Datei, welche die vollständige Klasse samt Erklärungen beinhaltet. Diese wird aber für die Anleitung nicht gebraucht und stellt das Endprodukt dar.

4.3 Imports

Für diese Anleitung werden folgende Bibliotheken verwendet, welche alle standardmässig in Java beinhaltet sind. Alle Bibliotheken stammen aus java.security. Diese sollten in der Klasse, welche für das Hashen zuständig ist, hinzugefügt werden.

```
import java.security.MessageDigest;

import java.security.NoSuchAlgorithmExeception;

import java.security.SecureRandom;
```

MessageDigest: Ist die Bibliothek, welche die Hashfunktion enthält, ohne diese können wir nicht den Hash generieren.

NoSuchAlgorithmExeception: Diese Bibliothek wird verwendet, um Fehler beim Algorithmus abzufangen. So können nicht schlimmere Dinge geschehen.

SecureRandom: Ist unsere Wertegenerator, welcher den Salt mit zufälligen Werten befüllt.

Referenz

Weiter Informationen über java.security kann unter folgender Quelle gefunden werden:
https://www.baeldung.com/java-security-overview

Kontrollfrage

- Wurden die Imports gefunden und nichts ist mit Rot markiert?

Leitfragen

- Was für Eigenschaften besitzt die Bibliothek java.security?
- Was ist MessageDigest und was kann diese Klasse?

4.4 Umwandlung zu Hex

Wenn die GitHub-Vorlage verwendet wird, dann ist Methode *„binToHex“* bereits vorhanden. Allenfalls kann sie ohne Probleme selbst geschrieben werden. Dafür werden keine weiteren Bibliotheken gebraucht. Diese Methode wird später verwendet, um aus dem binären Passwort ein *Hexadezimales* zu machen. Dadurch wird es leserlich und ist einfacher zum Speichern.

```java
private static String binToHex(byte[] binaryBytes) {
    // StringBuilder wird verwendet um mehrere Zeichen effizient
    // aneinander zu ketten
    StringBuilder sb = new StringBuilder();

    // Jedes Byte als Hex (%02x) formatieren
    for (byte binByte : binaryBytes) {
        sb.append(String.format("%02x", binByte));
    }

    // Hash als HexString zurück geben
    return sb.toString();
}
```

Abbildung 4 binToHex-Methode

Als *Übergabeparameter* wird ein Byte Array übergeben. Dies ist zum Beispiel ein Hash- oder Saltwert. Die Methode wurde als statisch markiert, da kein Aufruf auf eine aussenstehende Methode gemacht wird. Zudem gibt sie am Ende einen String zurück. Deswegen muss der Rückgabetyp auch einen String sein. Mithilfe eines *StringBuilders* wird jedes Byte in ein hexadezimales Zeichen umgewandelt. Dies passiert über die Formatierung von „%02x“, welche in Java bereits eingebaut ist. Zum Schluss wird aus dem StringBuilder ein einziger String erstellt.

Im folgenden Bild wird dieser Vorgang grafisch veranschaulicht. Dafür wird der Hash, welcher zuerst binär, ist durch diese Methode in eine hexadezimale Zeichenkette umgewandelt. Dies ist die übliche Variante, wie Hashes aussehen und abgespeichert werden. Das Hex-System geht von 0-9 und A-F. Zudem werden die Buchstaben grossgeschrieben, damit man besser sieht, dass Zeichen darin enthalten sind.

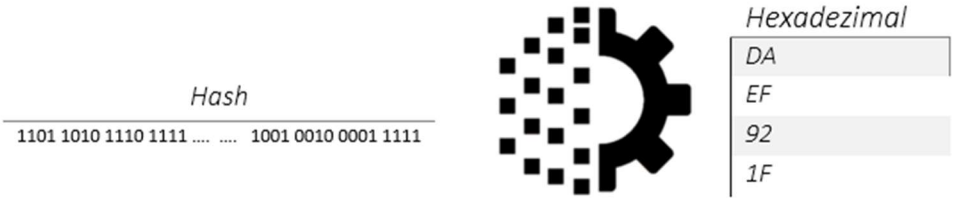

Abbildung 5 Hash Binär zu Hexadezimal

Kontrollfrage

- Wird ein Byte Array in Hex umgewandelt? (Methode temporär als public markieren)

Leitfragen

- Was für Vorteile hat es Hexadezimal anstelle von Binär abzuspeichern?
- Was macht das Formatierungszeichen «%02x»?

4.5 Hauptmethode

Zu Beginn wird eine neue Methode erstellt, welche als Übergabeparameter ein Byte Array annimmt. Dieses Array beinhaltet die Zeichen des Passwortes. In diesem Beispiel wird der Hash am Schluss nicht zurückgegeben, sondern in die Konsole ausgegeben. Deswegen ist der Rückgabetyp auf «void» definiert. Da kein Aufruf auf eine aussenstehende Methode gemacht wird, kann sie als statisch markiert werden.

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;

public class Hasher {

    private static String binToHex(byte[] binaryBytes) {…

    public static void hashPasswordMitSalt(byte[] passwort) {

    }

}
```

Abbildung 6 Neue Methode hashPasswordMitSalt erstellen

Referenz

Passwörter sollten immer als Byte oder Char Array verwaltet werden, da Strings immutable sind. Das bedeutet, dass die Zeichen unveränderlich sind. Sobald man ein Zeichen ändert, wird eine neue Zeichenkette generiert. Somit ist es nicht möglich, das Passwort im Speicher zu überschreiben, damit es nicht von anderen Programmen ausgelesen werden kann. Zudem kann ein Array schlecht in eine Logdatei geschrieben werden. Somit wird verhindert, dass aus Versehen das Passwort als Klartext irgendwo vorhanden ist.
Quelle: https://www.geeksforgeeks.org/use-char-array-string-storing-passwords-java/

Wichtig

Damit die gleichen Methoden vorhanden sind, ist es notwendig, dass das zuvor genannte Kapitel «Import» und «Umwandlung zu Hex» durchgeführt und in die Klasse aufgenommen wurde. Ansonsten sind die Methoden nicht aufrufbar und es wird einen Fehler beim Ausführen geben.

Leitfragen

- Wieso wird das Passwort nicht als String, sondern als Byte Array verwaltet?
- Wieso wird die Methode als statisch markiert?

4.5.1 Try Catch einfügen

Damit bei einem Fehler das Programm nicht abstürzt, wird ein **tryCatch** verwendet, welchen den NoSuchAlgorithm-Fehler abfängt. Dieser Fehler wird ausgegeben, wenn der eingegebene Hashing-Algorithmus nicht vorhanden oder falsch geschrieben ist. Im Moment zeigt es noch eine Fehlermeldung an, da keine Methode diesen Fehler auswirft. Dies kann aber temporär ignoriert werden, da es im nächsten Schritt behoben wird.

```
public static void hashPasswordMitSalt(byte[] passwort) {
    try {

    } catch (NoSuchAlgorithmException e) {
        System.out.println(e.getMessage());
    }
}
```

Abbildung 7 tryCatch

Hinweis

Der Code wird im Try geschrieben, sollte es zu einem Fehler kommen, wird der Code im Catch ausgeführt. In unserem Fall wird die Fehlermeldung im Terminal ausgegeben. Zur Fehlerverwaltung wäre es besser, wenn ein Logger wie Log4J[3] verwendet wird. Dies würde aber für diese Anleitung zu weitgehen.

Leitfragen

- Was für andere Möglichkeiten gibt es, um den Fehler zu Verwalten oder zu Speichern?

[3] *https://logging.apache.org/log4j/2.x/*

4.5.2 Algorithmus instanziieren

Als nächsten Schritt erstellt man eine Instanz der MessageDigest Klasse, welche wir bereits in den vorherigen Kapiteln importiert wurde. Beim Konstruktor übergibt man den gewünschten Algorithmus. In unserem Fall verwenden wir SHA-256, da dieser sehr sicher und zuverlässig ist. Die Instanz wird dann in die Variable «md» vom Datentype MessageDigest gespeichert.

MessageDigest Instanz erstellen:
«**MessageDigest md = MessageDigest.getInstance("SHA-256");**»

```
public static void hashPasswordMitSalt(byte[] passwort) {
    try {

        MessageDigest md = MessageDigest.getInstance("SHA-256");

    } catch (NoSuchAlgorithmException e) {
        System.out.println(e.getMessage());
    }
}
```

Abbildung 8 MessageDigest Instanz

Referenz

Mehr zu MessageDigest und dessen möglicher Algorithmen finden Sie unter:
https://docs.oracle.com/javase/7/docs/api/java/security/MessageDigest.html

Kontrollfrage

- Sieht man mittels Debugger die neue Instanz *md* und wurde diese auch gesetzt?

Leitfragen

- Welche Algorithmen wären noch möglich zu instanziieren?
- Wieso wird sehr oft für das Hashen von Passwörtern den Algorithmus SHA-256 verwendet?

4.5.3 SecureRandom erstellen

Im Vergleich zum herkömmlichen Random verwendet SecureRandom 128bit und nicht bloss 48bit. Dadurch ist die Wahrscheinlichkeit nochmals geringer, dass ein wiederholter Wert auftritt. Zudem ist diese Klasse auch schwieriger zu knacken und geht anders mit dem Speicherbereich um. Durch den SecureRandom befüllen wir später den Salt mit zufälligen Werten.

SecureRandom Instanz erstellen:
«**SecureRandom random = new SecureRandom();**»

```
MessageDigest md = MessageDigest.getInstance("SHA-256");

SecureRandom random = new SecureRandom();
```

Abbildung 9 SecureRandom

Referenz

Mehr zum Unterschied zwischen Random und SecureRandom in Bezug zu Java finden Sie unter: https://www.geeksforgeeks.org/random-vs-secure-random-numbers-java/

Kontrollfrage

- Sieht man mittels Debugger die neue Instanz random und wurde diese auch gesetzt?

Leitfragen

- Worin besteht der Vorteil bei der Verwendung von SecureRandom zum herkömmlichen Random?

4.5.4 Salt generieren

Es wird ein neues Byte Array (*byte[]*) erstellt, in welchem später der Salt gespeichert wird. In diesem Beispiel wurde die Grösse des Salts auf 16 Bytes definiert. Wahlweise kann diese länge auch vergrössert werden, wodurch ein noch sicherer Salt generiert wird. Durch den Salt wird auch bei identischen Passwörtern einen unterschiedlichen Hash generiert.

Byte Array erstellen:
«**byte[] salt = new byte[16];**»

Über die Instanz «random», welche wir zuvor erstellten, rufen wird nun die Methode **nextBytes()** auf. Diese befüllt den Salt mit zufälligen Werten, damit nie denselben Salt generiert wird.

Array befüllen:
«**random.nextBytes(salt);**»

```
MessageDigest md = MessageDigest.getInstance("SHA-256");

SecureRandom random = new SecureRandom();

byte[] salt = new byte[16];
random.nextBytes(salt);
```

Abbildung 10 Salt erstellen

Kontrollfrage

- Enthält der *salt* zufällige Werte zwischen -128 und 127?

4.5.5 Salt Übergabe

Wie man vom Sequenzdiagramm ablesen kann, wird der Salt an MessageDigest übergeben, da MessageDigest das Passwort in Kombination mit dem Salt hasht. Dies muss zuerst gemacht werden, bevor man den Hash generiert, da ansonsten der Salt nicht verwendet wird.

Salt an MessageDigest übergeben:
«**md.update(salt);**»

```
MessageDigest md = MessageDigest.getInstance("SHA-256");

SecureRandom random = new SecureRandom();

byte[] salt = new byte[16];
random.nextBytes(salt);

md.update(salt);
```

Abbildung 11 Salt Übergabe

Kontrollfrage

- Hat sich der Status von der Instanz *md* verändert?

Leitfragen

- Wieso muss man den Salt zuerst übergeben, bevor man hasht?
- Was für einen Vorteil bietet es, wenn man den Salt vergrössert?

4.5.6 Passwort Übergabe / Hash erstellen

Jetzt, wo wir den Saltwert übergeben haben, können wir das Passwort auch übergeben und Hashen. MessageDigest übernimmt dann die ganze Arbeit und erstellt einen Hash. Dieser wird in Form von einem Byte Array zurückgegeben. Deswegen wird dieser Wert temporär zwischengespeichert.

Passwort übergeben und hashen:
«**byte[] hashedPasswordBytes = md.digest(passwort);**»

```
MessageDigest md = MessageDigest.getInstance("SHA-256");

SecureRandom random = new SecureRandom();

byte[] salt = new byte[16];
random.nextBytes(salt);

md.update(salt);

byte[] hashedPasswordBytes = md.digest(passwort);
```

Abbildung 12 Passwort Hash generieren (Code)

In der folgenden Grafik sieht man, wie der Vorgang von Hashing mit Salt funktioniert. Zuerst wird der Salt übergeben, dann das Passwort. Dies kommt in eine *Blackbox*, welche aus diesen Kombinationen einen Hash generiert.

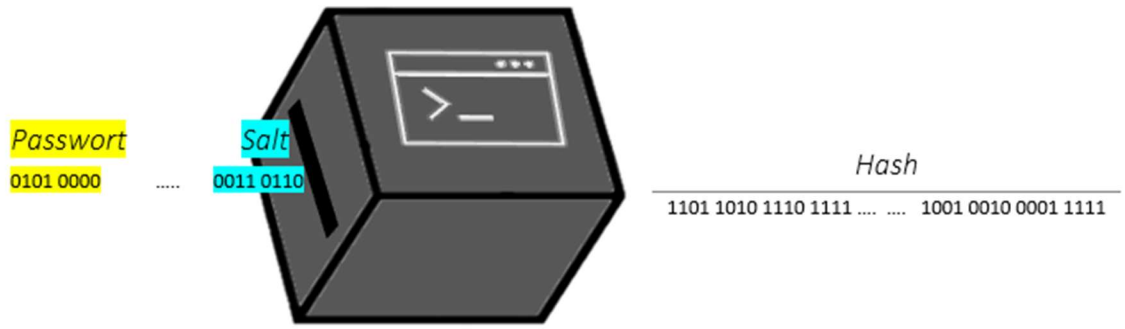

Abbildung 13 Passwort Hash generieren (Visualisierung)

Kontrollfrage

- Enthält der *hashedPasswordBytes* einen Hash?

Leitfragen

- Wieso werden Passwörter während dem hashen mit einem Salt versehen?

4.5.7 Umwandlung zu Hex

Zur besseren Darstellung und Übersicht wird der noch binäre Hash ins hexadezimale Format umgewandelt. Dazu verwenden wir die binToHex() Methode, welche wir ganz am Anfang erstellten. Das machen wir je für den Passwort-Hash und für den Salt.

Wichtig

Es wird immer das gesaltete und gehashte Passwort und der Saltwert selbst abgespeichert (beispielsweise in die Datenbank). Das macht man, damit man später wieder von der Passworteingabe zum abgespeicherten Hash kommt. Somit muss beides gespeichert werden.

```
MessageDigest md = MessageDigest.getInstance("SHA-256");

SecureRandom random = new SecureRandom();

byte[] salt = new byte[16];
random.nextBytes(salt);

md.update(salt);

byte[] hashedPasswordBytes = md.digest(passwort);

String hashedPasswd = binToHex(hashedPasswordBytes);
String hashedSalt = binToHex(salt);
```

Abbildung 14 Umwandlung zu Hex

Leitfragen

- Weshalb werden Passwörter gehasht abgespeichert?

4.6 Mögliche Fehler

Fehler Meldung	Erklärung
NoSuchAlgorithmException	Wenn dem Programm der Angegebene Algorithmus nicht bekannt ist oder falsch angeben ist z.B. SHA-254 (existiert nicht).
… cannot be resolved	Der Import der Bibliothek fehlt oder wurde nicht richtig angeben (Tippfehler).
… is not applicable for the arguments (String)	Es wird ein String verwendet anstatt byte[]. Lösung des Problems ist es den Befehl *.getBytes()* zu verwenden

5 Endergebnis

```java
public class HasherML {

    private static String binToHex(byte[] binaryBytes) {
        // StringBuilder wird verwendet um mehrere Zeichen effizient aneinander zu ketten
        StringBuilder sb = new StringBuilder();

        // Jedes Byte als Hex (%02x) formatieren
        for (byte binByte : binaryBytes) {
            sb.append(String.format("%02x", binByte));
        }

        return sb.toString(); // Hash als HexString zurück geben
    }

    public static void HashPasswortMitSalt(byte[] passwort) {
        try {
            // MessageDigests ist eine sichere unidirektionale Hashfunktion, die Daten beliebiger Größe annimmt und einen Hashwert fester Länge ausgiebt.
            // Dem Konstruktor übergibt man den Algorihtmus (SHA-256) und gibt die dementsprechende Instanz.
            MessageDigest md = MessageDigest.getInstance("SHA-256");

            // Randome generiert eine zufähllige Zahl
            SecureRandom random = new SecureRandom();

            // Erstellung eines Arrays von bytes für das Salting
            // Wahlweise Zahl auch erweiterbar für noch mehr Sicherheit
            byte[] salt = new byte[16];
            // Geht beim Array von byte zu byte und setzt eine zufällige Zahl hinein.
            random.nextBytes(salt);
            // Übergeben des salt an den Digest für die Berechnung
            md.update(salt);

            // Nun kann mit md ein salted Hash generiert werden
            byte[] hashedPasswordBytes = md.digest(passwort);

            // Aus dem Binär wird nun ein Hex generiert
            String hashedPasswd = binToHex(hashedPasswordBytes);
            String hashedSalt = binToHex(salt);

            // Beides Speichern, damit gleicher Hash generiert werden kann
            System.out.println("Password Hash:" + hashedPasswd);
            System.out.println("Salt als Hex: " + hashedSalt);
        } catch (NoSuchAlgorithmException e) {
            // Ausgewählter Algorithmus existiert nicht
            System.out.println(e.getMessage());
        }
    }
}
```

Abbildung 15 Endergebniss

6 Abschluss

Wenn Sie dieser Anleitung gefolgt sind, haben Sie nun eine Java-Klasse, welche ein Passwort annimmt und dieses mittels eines Salts, einen Hash generiert. Diese Klasse können Sie so mit leichten Anpassungen in Ihr eigenes Projekt implementieren.

Selbst wenn dasselbe Passwort eingegeben wird, wird nicht derselbe Hash generiert. Womit man gegen Rainbowtable Angriffe sicher ist.

Jetzt sollten Sie wissen, was ein Hash ist und wieso man ihn saltet. Zudem, wie man dieses Wissen in Java anwendet und eine Methode programmiert, welche das Passwort annimmt und in einen Hash umwandelt. Zukünftig sollten Sie ohne Schwierigkeiten Passwörter hashen können, was für mehr Sicherheit der Benutzer führt.

Wenn man das Passwort und den Salt hat, dann kann man beliebig weiterfahren. Ab dort kommt es auf die Anwendung an, wo man diese ganze Klasse verbaut hat. Mögliche Speicherorte wären Dateien oder Datenbanken. Zur Veranschaulichung geben wir in unserem Beispiel den Passwort-Hash und den Salt im Terminal aus.

```java
public static void HashPasswordMitSalt(byte[] passwort) {
    try {

        MessageDigest md = MessageDigest.getInstance("SHA-256");

        SecureRandom random = new SecureRandom();

        byte[] salt = new byte[16];
        random.nextBytes(salt);

        md.update(salt);

        byte[] hashedPasswordBytes = md.digest(passwort);

        String hashedPasswd = binToHex(hashedPasswordBytes);
        String hashedSalt = binToHex(salt);

        System.out.println("Passwort Hash:" + hashedPasswd);
        System.out.println("Salt als Hex: " + hashedSalt);

    } catch (NoSuchAlgorithmException e) {
        System.out.println(e.getMessage());
    }
}
```

Abbildung 16 HashedPasswd und Salt Ausgabe

Kontrollfrage

- Wenn zweimal dasselbe Passwort eingegeben wird, kommen unterschiedliche Hashes heraus?

7 Glossar

B

Blackbox
Teil eines kybernetischen Systems, dessen Aufbau und innerer Ablauf erst aus den Reaktionen auf eingegebene Signale erschlossen werden können 16

G

Git Git ist ein kostenloses, verteiltes Versionskontrollsystem für Softwareprojekte 6
GitHub GitHub ist ein netzbasierter Dienst zur Versionsverwaltung für Software-Entwicklungsprojekte 6

H

Hexadezimal
ist ein Zahlensystem, dass 16 Zeichen besitzt. *8*

R

Rainbowtable Angriffe
Sammlung von bekannten Hashes mit deren Eingabewerten 3

S

StringBuilders
Methode, um effizient Zeichen aneinander zu ketten 8

U

Übergabeparameter
sind Variablen, welche einer Methode übergeben werden zur Verarbeitung *8*

8 Abbildungsverzeichnis