

AES Keyscheduler

kry Projekt

Moritz Roth

4. April 2019

Einleitung

Die Aufgabe war es, den Keyschedule-Algorithmus der AES Verschlüsselung für 128 bit Keys zu implementieren, so wie dieser im Standardisierungsdokument beschrieben ist.

Für die Implementation habe ich die Sprache Java gewählt, da ich mit dieser am besten vertraut bin. Ausserdem habe ich nach dem Keyscheduler noch versucht den kompletten Verschlüsselungsalgorithmus zu implementieren, dafür hat mir allerdings dann die Zeit gefehlt.

Keyschduler Implementation

Das Projekt ist ein Maven-Projekt, da ich so im Code sehr einfach Testfälle definieren kann, die dann ausgeführt werden können. Der Implementationscode befindet sich im Verzeichnis `src/main/java/ch/fhnw/kry/keyscheduler`.

Für den Keyscheduler wichtig sind die Klassen `KeySchedule.java`, `SubBytes.java`, `Byte.java` und `Word.java`, sowie die Enums `AESConfig.java` und `Base.java`.

Klasse/Enum	Beschreibung
KeySchedule.java	Enthält den eigentlichen Keyexpansion Alorithmus, der den Key von 16 Byte auf 44 Byte expandiert.
SubBytes.java	Enthält Methoden, die als Parameter ein Byte erwarten (in verschiedenen Basen) und das entsprechende SubByte aus der S-Box Tabelle zurückgibt.
Byte.java	Ein Byte Objekt, das das Byte intern als Integer speichert. Es kann ein Byte aus dezimal-, binär- und hexadezimal-Zahlen erstellen und dieses auch in diesen Basen zurückgeben.
Word.java	Enthält ein Array aus 4 Bytes.
AESConfig.java	Dieses Enum enthält die Konfigurationen für die verschiedenen KeyLängen.
Base.java	Dieses Enum enthält Details zu den Basen HEX (hexadezimal), DEZ (dezimal) und BIN (binär).

Die Klasse `KeySchedule.java` ist die Hauptklasse für das Projekt, sie enthält die Methode `public static Word[] keyExpansion(Byte[] key, AESConfig conf)`. Dieser kann man einen Key in form eines Bytearrays mitgeben, sowie eine `AESConfig` (also `AES_128`, `AES_192` oder `AES_256`) und sie gibt ein `Wordarray` zurück, das den expanded Key enthält. Wobei `AES_192` und `AES_256` noch nicht funktionieren, weil das round constant Array noch nicht dafür angepasst ist.

Bei der Implementation habe ich mich sehr genau an den Pseudocode aus der Präsentation gehalten.

Tests

Die Tests für den Keyscheduler befinden sich in der Testklasse `src/test/java/ch/fhnw/kry/keyscheduler/KeyScheduleTest.java`. Die Testfälle habe ich von der Webseite <https://www.samiam.org/key-schedule.html>. Es gibt jeweils ein Test für die edge cases (alles 00 und alles ff), einen bei dem die 16 Bytes im Key durchgezählt sind (00 bis 0f) und einen Key dazwischen, ausserdem habe ich einen Test erstellt, der fehlschlägt, weil ein Bit in der Ausgabe verändert wurde und zwei Tests, die falsche Schlüssel übergeben.

`testKeyExpansion1():`

INPUT

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

EXPECTED OUTPUT

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
62 63 63 63 62 63 63 63 62 63 63 63 62 63 63 63
9b 98 98 c9 f9 fb fb aa 9b 98 98 c9 f9 fb fb aa
90 97 34 50 69 6c cf fa f2 f4 57 33 0b 0f ac 99
ee 06 da 7b 87 6a 15 81 75 9e 42 b2 7e 91 ee 2b
7f 2e 2b 88 f8 44 3e 09 8d da 7c bb f3 4b 92 90
ec 61 4b 85 14 25 75 8c 99 ff 09 37 6a b4 9b a7
21 75 17 87 35 50 62 0b ac af 6b 3c c6 1b f0 9b
0e f9 03 33 3b a9 61 38 97 06 0a 04 51 1d fa 9f
b1 d4 d8 e2 8a 7d b9 da 1d 7b b3 de 4c 66 49 41
b4 ef 5b cb 3e 92 e2 11 23 e9 51 cf 6f 8f 18 8e

RESULTAT: **Erfolgreich**

`testKeyExpansion2():`

INPUT

ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

EXPECTED OUTPUT

```
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
e8 e9 e9 e9 17 16 16 16 e8 e9 e9 e9 17 16 16 16
ad ae ae 19 ba b8 b8 0f 52 51 51 e6 45 47 47 f0
09 0e 22 77 b3 b6 9a 78 e1 e7 cb 9e a4 a0 8c 6e
e1 6a bd 3e 52 dc 27 46 b3 3b ec d8 17 9b 60 b6
e5 ba f3 ce b7 66 d4 88 04 5d 38 50 13 c6 58 e6
71 d0 7d b3 c6 b6 a9 3b c2 eb 91 6b d1 2d c9 8d
e9 0d 20 8d 2f bb 89 b6 ed 50 18 dd 3c 7d d1 50
96 33 73 66 b9 88 fa d0 54 d8 e2 0d 68 a5 33 5d
8b f0 3f 23 32 78 c5 f3 66 a0 27 fe 0e 05 14 a3
d6 0a 35 88 e4 72 f0 7b 82 d2 d7 85 8c d7 c3 26
```

RESULTAT: **Erfolgreich**

testKeyExpansion3():

INPUT

```
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
```

EXPECTED OUTPUT

```
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
d6 aa 74 fd d2 af 72 fa da a6 78 f1 d6 ab 76 fe
b6 92 cf 0b 64 3d bd f1 be 9b c5 00 68 30 b3 fe
b6 ff 74 4e d2 c2 c9 bf 6c 59 0c bf 04 69 bf 41
47 f7 f7 bc 95 35 3e 03 f9 6c 32 bc fd 05 8d fd
3c aa a3 e8 a9 9f 9d eb 50 f3 af 57 ad f6 22 aa
5e 39 0f 7d f7 a6 92 96 a7 55 3d c1 0a a3 1f 6b
14 f9 70 1a e3 5f e2 8c 44 0a df 4d 4e a9 c0 26
47 43 87 35 a4 1c 65 b9 e0 16 ba f4 ae bf 7a d2
54 99 32 d1 f0 85 57 68 10 93 ed 9c be 2c 97 4e
13 11 1d 7f e3 94 4a 17 f3 07 a7 8b 4d 2b 30 c5
```

RESULTAT: **Erfolgreich**

testKeyExpansion4():

INPUT

69 20 e2 99 a5 20 2a 6d 65 6e 63 68 69 74 6f 2a

EXPECTED OUTPUT

69 20 e2 99 a5 20 2a 6d 65 6e 63 68 69 74 6f 2a
fa 88 07 60 5f a8 2d 0d 3a c6 4e 65 53 b2 21 4f
cf 75 83 8d 90 dd ae 80 aa 1b e0 e5 f9 a9 c1 aa
18 0d 2f 14 88 d0 81 94 22 cb 61 71 db 62 a0 db
ba ed 96 ad 32 3d 17 39 10 f6 76 48 cb 94 d6 93
88 1b 4a b2 ba 26 5d 8b aa d0 2b c3 61 44 fd 50
b3 4f 19 5d 09 69 44 d6 a3 b9 6f 15 c2 fd 92 45
a7 00 77 78 ae 69 33 ae 0d d0 5c bb cf 2d ce fe
ff 8b cc f2 51 e2 ff 5c 5c 32 a3 e7 93 1f 6d 19
24 b7 18 2e 75 55 e7 72 29 67 44 95 ba 78 29 8c
ae 12 7c da db 47 9b a8 f2 20 df 3d 48 58 f6 b1

RESULTAT: **Erfolgreich**

testKeyExpansionFail():

In diesem Test ist der EXPECTED OUTPUT falsch (die rote 9 in der letzten Zeile).
Dieser Test soll zeigen, dass das erwartete Resultat auch wirklich mit dem tatsächlichen
Output verglichen wird.

INPUT

69 20 e2 99 a5 20 2a 6d 65 6e 63 68 69 74 6f 2a

EXPECTED OUTPUT

69 20 e2 99 a5 20 2a 6d 65 6e 63 68 69 74 6f 2a
fa 88 07 60 5f a8 2d 0d 3a c6 4e 65 53 b2 21 4f
cf 75 83 8d 90 dd ae 80 aa 1b e0 e5 f9 a9 c1 aa
18 0d 2f 14 88 d0 81 94 22 cb 61 71 db 62 a0 db
ba ed 96 ad 32 3d 17 39 10 f6 76 48 cb 94 d6 93
88 1b 4a b2 ba 26 5d 8b aa d0 2b c3 61 44 fd 50
b3 4f 19 5d 09 69 44 d6 a3 b9 6f 15 c2 fd 92 45
a7 00 77 78 ae 69 33 ae 0d d0 5c bb cf 2d ce fe
ff 8b cc f2 51 e2 ff 5c 5c 32 a3 e7 93 1f 6d 19
24 b7 18 2e 75 55 e7 72 29 67 44 95 ba 78 29 8c
ae 12 7c da db 47 99 a8 f2 20 df 3d 48 58 f6 b1

RESULTAT: **Erfolgreich**

→ heisst, dass der Vergleich zwischen tatsächlichem und erwartetem Output `false` zurückgibt. Darum auch `assertFalse()`.

`testKeyExpansionException1()` / `testKeyExpansionException2()`:

Diese beiden Tests failen mit einer `IllegalArgumentException`, weil der Key zu lange bzw. zu kurz ist. Die Tests sind **erfolgreich** im Sinne, dass die Exception geworfen wird.

Weitere Tests:

Ansonsten habe ich noch einige Tests für Byte, SubBytes und Word geschrieben, um zu testen, ob sie sich wie erwartet verhalten. Dazu gehören auch Tests von `xor()` und `rotWord()`. Diese sind alle **erfolgreich**.

Weitere Implementation AES

Nach der Implementation habe ich mich noch an der Implementation weiterer Teile des AES versucht. Die Verschlüsselung habe ich auch beendet, die Entschlüsselung allerdings nicht. In der Klasse `AES.java` befinden sich die Methoden `cypher()` für die Verschlüsselung und `eqInvCipher()` für die Entschlüsselung. Man kann jeweils den Plaintext als `ByteArray` und den expanded Key als `Wordarray` mitgeben, sowie die Schlüssellänge (die könnte auch anhand des Keys berechnet werden).

Das rcon-Array in der Keyexpansion müsste noch für die grösseren Keysizes erweitert (oder generiert) werden.

Die anderen Klassen sind jeweils selbstredend: `AddRoundKey.java` enthält die `addRoundKey` Methode (und Inverse), `MixColumns.java` enthält die `mixColumns` Methode (und Inverse) und `ShiftRows.java` enthält die `shiftRows` Methode (und Inverse).

Zudem habe ich in der Klasse `MixColumns.java` noch die Galois-Multiplikation implementiert (`gmul()`).

→ Alle diese Klassen und Methoden sind allerdings ungetestet!