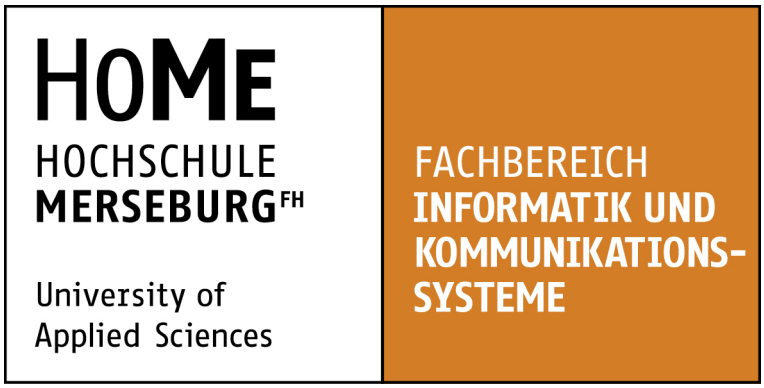


Zur Erlangung des Grades

eines

Bachelor of Science (B. Sc.)

von Herrn Daniel Kube

Geboren am: 13.08.1988

in: Halle (Saale)

Vorgelegte Abschlussarbeit: Bachelorarbeit

Thema: Visuelle Implementierung von AES

Erstprüfer Herr Prof. Dr. phil. Dr. rer. nat. habil. Michael Schenke
Zweitprüfer Herr Prof. Dr. rer. nat. Hartmut Kröner

Merseburg, 24. April 2017

1. Inhaltsverzeichnis

1. Einleitung

Diese Arbeit ist der Abschluss einer längeren Bemühung, welche darauf abzielte, ein Programm zu entwickeln, welches vom Dozenten während der Vorlesung eingesetzt wird. Das Programm sollte dazu dienen, dem Studenten direkt während der Vorlesung vor Augen zu führen, wie die mathematischen Konzepte hinter Verschlüsselungsverfahren in der Praxis algorithmisch umgesetzt werden. Da theoretisch die Möglichkeit bestand, dass das Programm auch außerhalb der Hochschule aufgeführt werden könnte, um mehr Personen für diese zu interessieren, sollten die Animationen des Programms schlüssig wirken, ohne Vorkenntnisse beim Betrachter vorauszusetzen. Im Laufe dieser Arbeit wird die visuelle Implementierung von AES behandelt, geht auf die Vorüberlegungen ein und zeigt diverse Instanzen auf, in denen es unerwartet zu Problemen kam. Es wird kurz eine Aussage über den Sinn von Animationen in der Vorlesung vor dem Hintergrund sogenannter Lernstile (genaugenommen dem VARK Lernstil) getroffen. Am Ende findet eine Auswertung statt, und es werden verschiedene Möglichkeiten aufgeführt, wie das Programm erweitert werden kann.

2. Aufgabenbegründung

2.1. Bedeutung der Arbeit für den Anwender

Die Bedeutung dieser Arbeit für den Anwender liegt nicht in der Schaffung neuen Wissens zum Thema AES. Die AES Chiffre wird für dieses Programm als gegeben angenommen. Vielmehr soll dieses Programm bei der Vermittlung des AES helfen. Dazu muss zuerst die Frage beantwortet werden, ob es überhaupt Sinn macht, in einer Vorlesung ein solches Programm zu verwenden. Zu diesem Zweck wird kurz auf VARK eingegangen.

2.1.1. VARK

VARK ist ein Lernmodell welches 1992 in einer Arbeit von Neil Fleming und Colleen Mills erwähnt wird /FM92/. In /NF17/ wird auf die im Modell festgelegten unterschiedlichen Lerntypen/Lernpräferenzen Visual (Sehen), Aural (Hören), Read/Write (Lesen/Schreiben), Kinesthetic (Tun) und Multimodal (Mischung mehrerer Typen) eingegangen. Beinhaltet eine Vorlesung die Elemente „Vortrag des Dozenten“, sowie „Notizen an der Tafel anbringen“, so werden hauptsächlich die Präferenzen Aural und Read/Write abgedeckt. Hier soll das Programm ansetzen, indem es die Vorlesung um eine visuelle Komponente erweitert. Da die Animation automatisch ablaufen kann, kann der Dozent theoretisch weiterhin die beiden Typen Aural und Read/Write abdecken, wodurch hoffentlich Studenten mit einer gewissen Neigung zum Visual Typ besser in der Lage sind das Gelehrte zu lernen.

2.1.2. Kritik

VARK ist nicht unumstritten. Es gibt Publikationen wie /OA10/ welche dem Modell sehr wohlgesonnen sind, aber auch solche wie /LSS10/, welche im Abstrakt über Vorbehalte gegenüber der Nutzung von VARK in der Forschung sprechen.
Lernmodelle an sich sind ein großes Gebiet, jedoch ist es nicht Sinn dieser Arbeit, mehr als nötig auf diese einzugehen.

2.2. Aufteilung der Implementierung

Die Implementierung soll in 2 Schritten erfolgen. Im ersten Schritt wird zuallererst einmal der AES Algorithmus an sich implementiert. Hierbei wird der Fokus auf die korrekte Implementierung des Algorithmus an sich gelegt, während das Nutzerinterface selbst ersteinmal nicht wichtig ist. Ist der Algorithmus implementiert und getestet, wrird im Anschluss der Fokus auf das Nutzerinterface und die Animation gelegt. Hierbei gilt zu beachten, dass die Animation nicht vorgegeben wird, wie bei einem Film oder ähnlichem, sondern dass die Animation auf beliebige Eingaben funktioniert.

3. Wissensstand

Es gibt diverse recht gute Werkzeuge, mit denen man heutzutage Texte mittels AES de-/chiffrieren kann, wie zum Beispiel das Tool Cryptool. Es gibt auch vergleichsbar leicht erreichbare Animationen zum AES (auch in Cryptool enthalten), jedoch gehen diese in der Regel von einem vorgegebenen Beispieltext aus, was nur bedingt von Nutzen ist. Dieses Programm vereint beides. Es bietet zum einen die Möglichkeit, Texte mittels AES zu de-/chiffrieren (was von Vorteil ist, um während der Vorlesung Beispiele auf Korrektheit zu überprüfen), sowie die Chiffrierung eines Textes Schritt für Schritt anzusehen.

4. Lösungsansatz

Das Programm soll den AES Algorithmus animiert darstellen. Dafür ist es wichtig zu verstehen, wie der AES Algorithmus funktioniert. Um ein besseres Gefühl dafür zu entwickeln (und im Anschluss eine Möglichkeit zu haben, die Animation darauf zu überprüfen, dass es korrekt den AES Algorithmus abarbeitet) macht es Sinn, den Algorithmus zuerst selbst zu implementieren. Um im Anschluss herauszufinden, wie man den AES animieren kann, gilt es zu ermitteln, was vom AES animiert werden muss. Dazu wird der AES Algorithmus in seine einzelnen Subalgorithmen aufgespalten. Dies wird erreicht indem man eine möglichst kompakte Zusammenfassung auf Begriffe untersucht, welche ihrerseits Algorithmen (oder Produkte solcher) sind, diese markiert, und im Anschluss selbst aufspaltet. Dieser Vorgang wird wiederholt, bis alle Algorithmen nicht weiter aufgespalten werden können. Dadurch wird ein Baum aufgebaut, der die Algorithmen und Subalgorithmen sortiert. Die Blätter des Baumes können hier als atomare Algorithmen des AES angesehen werden, die nicht in weitere Subalgorithmen zerlegt werden können. Hierbei ist es wichtig zu bedenken, dass die Operationen in einem Galoiskörper mit 2^8 Elementen ausgeführt werden, wodurch gewisse Grundrechenarten sich unter Umständen anders verhalten (Multiplikation und Addition). Diese Besonderheiten dürfen nicht außer Acht gelassen werden. Taucht ein Algorithmus mehrmals auf, kann zugunsten eines kompakteren Baumes darauf verzichtet werden, diesen mehr als einmal aufzuspalten.

4.1. Aufspaltung des AES

4.1.1. AES

Nach /NIST01/ s.14 ist der AES ist ein Algorithmus, welcher chiffriert indem auf einen 128 Bit langen Block an Eingabedaten mittels eines Schlüssels über eine gewisse Anzahl an Runden hinweg die Funktionen AddRoundKey, MixColumns, ShiftRows und SubBytes ausgeführt werden. Hierzu werden die Eingabedaten als ein zweidimensionales Array mit 4 Spalten und 4 Zeilen angesehen, worin jede Zelle ein Byte enthält. Die Anzahl der Runden richtet sich wie folgt an der Schlüssellänge:

128 Bit - 10 Runden
192 Bit - 12 Runden
256 Bit - 14 Runden

4.2. Aufspaltung der Subalgorithmen - Ebene 1

4.2.1. AddRoundkey

Ist laut /NIST01 s.18/ ein Algorithmus, bei welchem die Eingabedaten mit einem mittels Schlüsselexpansion erstellten Rundenschlüssels XOR-Verknüpft werden.

4.2.2. MixColumns

Ist ein in /NIST01/ s.17-18 beschriebener Algorithmus, welcher die Eingabedaten

spaltenweise mit der folgenden Polynom (hier in Matrixdarstellung) multipliziert:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

4.2.3.ShiftRows

In /NIST01 s.17/ beschriebener Algorithmus, welcher die Eingabedaten zeilenweise um n-1 Positionen nach links verschiebt. Dabei entspricht n der Zeile in der die Daten stehen (Dementsprechend werden die Elemente in der ersten Zeile um 0 Positionen nach Links verschoben und in der vierten Zeile um 3 Elemente).

4.2.4.SubBytes

Ist laut /NIST01/ s.15-16 ein Algorithmus, der alle Bytes der Eingabe in Halbbytes teilt, die wiederum die Position des Elements in der S-Box angeben, welches dann das Byte der Eingabe substituiert.

4.3. Aufspaltung der Subalgorithmen - Ebene 2

4.3.1.Schlüsselexpansion

Ist nach /NIST01 s.19/ ein Algorithmus, welcher aus dem Schlüssel mittels diverser Wortsubstitutionen und Wortrotationen sowie XOR-Verknüpfungen mit Einträgen aus der Liste der Rundenkonstanten Rcon eine Liste mit allen Rundenschlüsseln erstellt.

4.3.2.XOR-Verknüpfung

Ist ein Algorithmus, welcher zwei Eingabewerte bitweise miteinander vergleicht. Sind beide Bits gleich, ist das resultierende Bit 0, unterschiedliche Bits dagegen resultieren in einem Bit mit dem Wert 1. XOR ist eine Kurzform für eXclusive OR (exklusives Oder). In /NIST01 s.10/ wird festgestellt, dass im Rahmen des AES die XOR-Verknüpfung, Addition und Subtraktion dasselbe sind.

4.3.3.Multiplikation

Multiplikationen im AES werden laut /NIST01 s.10-11/ in einem Galoiskörper mit 2^8 Elementen ausgeführt und müssen im Anschluss modulo des folgenden irreduziblen Polynoms gerechnet werden:

$$m(x)=x^8+x^4+x^3+x+1.$$

4.3.4.S-Box

Ist in /NIST01 s. 15/ als das Ergebnis eines Algorithmus beschrieben, welcher zuerst zu allen Werten das Multiplikative Inverse bildet, und im Anschluss jeweils mit dem Byte {01100011} XOR-Verknüpft.

4.4. Aufspaltung der Subalgorithmen - Ebene 3

4.4.1.Wortsubstitution

Laut /NIST01 s.19/ ist die Wortsubstitution (subword()) ein Algorithmus, der von einem word alle 4 Bytes durch Werte aus der S-Box ersetzt.

4.4.2.Wortrotation

Dem /NIST01 s.19/ folgend werden bei der Wortrotation (rotword()) vom übergebenen word, welches auch als ein Array mit 4 Byte interpretiert werden kann, die Position aller Elemente im Array gemäß der folgenden Formel geändert:

$$Index_{neu} = Index_{alt}+3 \bmod 4.$$

4.4.3.Rcon

In /NIST01 s.19/ wird das Rcon als ein Array aus Rundenkonstanten beschrieben, worin jedes Element ein word bzw. 4-Byte Array ist deren Elemente alle den Wert {00} haben, bis auf das erste Element, welches eine Potenz von 2 ist. Genaugenommen hat in Runde i das Element den Wert 2^{i-1}, wobei zu beachten ist, dass es sich dabei letzten Endes um einen Multiplikation im Galoisfeld 2^8 handelt.

4.4.4.Multiplikative Inverse

Das Multiplikative Inverse einer Zahl x ist dem /NIST01 s.11/ zufolge die Zahl, welche mit x multipliziert das neutrale Element ergibt, welches in diesem Fall als „multiplicative identity“ mit dem Wert {01} beschrieben wird.

Dies ergibt den folgenden Baum:

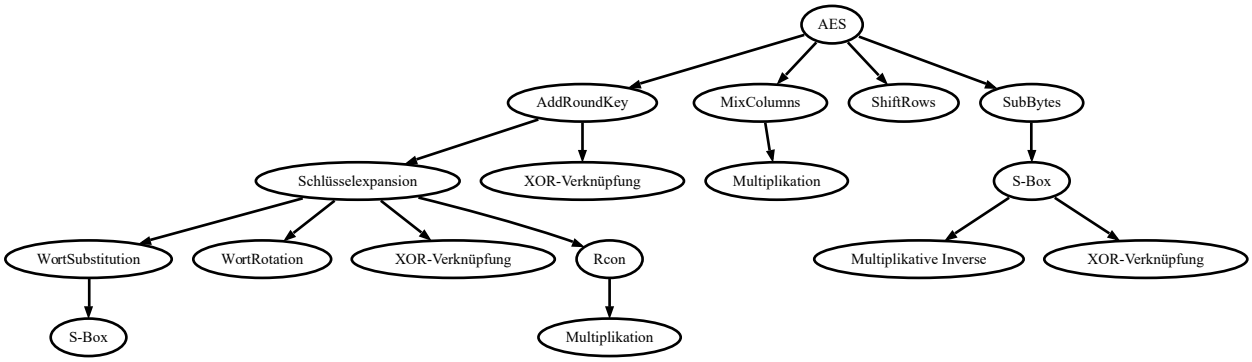


Abb.1: Aufgespaltener AES Baum

Mit diesem Baum kann theoretisch eine Abschätzung getroffen werden. Von der AES Wurzel ausgehend steigt die Komplexität/Dauer der Animation mit jeder neuen Ebene, die in die Animation eingeschlossen wird. Gleichzeitig steigt aber auch die Genauigkeit. Ein guter Mittelweg scheint die Animation bis einschließlich Ebene 1. Zwar würde bei Inklusion der zweiten Ebene sowohl die Entstehung der S-Box als auch die Schlüsselexpansion ausführlich dargestellt werden, jedoch würde die Länge der Animation der Schlüsselexpansion in etwa der Länge der Animation des eigentlichen AES bis inklusive 1 entsprechen, ohne einen entsprechenden Mehrwert zu bieten.

5. Implementierung

5.1. AES Implementierung:

Zu Beginn der praktischen Arbeit lag der Fokus auf einer korrekten Implementierung des AES-Algorithmus. Während dieser Phase galt es, folgende Unterziele zu erfüllen:

- Funktionsbasierte Erstellung der einzelnen Teilschritte
- Aufbau des Verständnisses für den Algorithmus
- Kontrolle der Korrektheit der AES Implementierung

5.1.1. Funktionsbasierte Erstellung der einzelnen Teilschritte

AES chiffriert den Text, indem es über mehrere Runden hinweg die gleichen 4 Schritte hintereinander auf ein 4x4 Byte Array ausführt. Da die Rundenanzahl variabel ist (abhängig davon, ob man einen 128. 192, oder 256 Bit langen Schlüssel verwendet), machte es Sinn, den Algorithmus in Funktionsaufrufe zu unterteilen, welche dann in einer Schleife entsprechend der verwendeten Schlüssellänge oft aufgerufen wurden. Dies hat den praktischen Nebeneffekt, dass die Anzahl der möglichen Programmierfehler reduziert werden, da gleiche Funktionen nicht mehrmals implementiert werden müssen.

5.1.2. Aufbau des Verständnisses für den Algorithmus

Hier galt es, die speziellen Bewegungsabläufe der Animation für die separaten Funktionen zu konzeptualisieren. Es galt, einen Bewegungsablauf zu finden, der dem Betrachter ein Gefühl von Kohärenz der Daten vermittelte, und gleichzeitig Besonderheiten des aktuellen Schritts verdeutlichte. Dazu gehörte auch die Festlegung einer einheitlichen kontrastreichen Farbpalette, die nach Möglichkeit beim Einsatz über einen Beamer als unterschiedliche Farben zu erkennen sein sollte.

5.1.3. Kontrolle der Korrektheit der AES Implementierung

Wahrscheinlich der wichtigste Punkt war die Kontrolle des Algorithmus auf Korrektheit. Ein Indikator für die Korrektheit der Implementierung fand sich in den Testdaten des AES Standards, welche jede Konfiguration des AES State während der Chiffrierung beinhaltete. Aufgrund der modularen Natur des Algorithmus konnte jede Funktion pro ursprünglich eingegebenen Testdaten mehrmals auf die Erstellung korrekter Ergebnisse hin überprüft werden. Des weiteren bestand sowohl die Mög-

lichkeit, unter Zuhilfenahme von verschiedenen Programmen mit AES Implementation, beliebig weitere Daten zur Überprüfung zu erzeugen. Natürlich reicht dies nicht aus um mit hundertprozentiger Sicherheit Fehler in der Implementation auszuschließen, dafür müsste man theoretisch für alle 2^{128} möglichen Belegungen der 128 Bit großen Eingabedaten (was allein bereits einer Datenmenge von rund $5{,}445 \times 10^{15}$ Yottabytes entspricht) alle 2^{128} möglichen Schlüssel auf Korrektheit überprüfen, und hätte dann gerade erst einmal sichergestellt, dass die 128 Bit langen Schlüssel eine korrekte Ausgabe liefern, ohne eine Aussage über 192 Bit oder 256 Bit lange Schlüssel zu treffen. Es diente lediglich als schneller Indikator, ob es grobe Fehler in der implementierung gab.

5.2. GUI Implementierung und Animation:

Die GUI Implementierung startete parallel zur Implementierung des Algorithmus. Dies bedeutete am Anfang auch, Am Anfang bestand das GUI aus wenig mehr als einem blanken Feld mit ein paar Textzeilen und Knöpfen, die mittels Javas GUI Designer grob zusammengestellt wurden. Der Fokus lag zu dieser Zeit in der richtigen Implementierung des Algorithmus, und da diese Verschlüsselung ohne Animation innerhalb des GUI darzustellen, gab es auch kein Konzept für diese. Dementsprechend wuchs die Oberfläche zunächst mit den fertiggestellten Teilen. Das Konzept entwickelte sich erst mit der fortschreitenden Implementierung.
Relativ früh galt es, sich für die Animationen zu entscheiden, ob diese mittels eines sogenannten Canvas realisiert werden sollten, oder mittels einer knotenbasierten Herangehensweise. Bei einem Canvas werden die Elemente gemäß ihren Attributen in ein Bild gezeichnet, welches angezeigt wird. Dies bedeutet zum einen, dass man die Elemente nur ein mal zeichnen müsste, wenn sie sich nicht ändern, auf der anderen aber auch, dass bei Translationen Elemente aktiv aus dem Bild entfernt werden müssen, sofern man nicht will, dass diese einen Schweif hinter sich herziehen. Die knotenbasierte Variante wiederum ist nicht wirklich auf komplexe Animationen ausgelegt, erlaubt es aber, Elemente in einer baumartigen Hierarchie zu gruppieren. Letzten Endes wurde die knotenbasierte Herangehensweise gewählt.
Javas objektorientierte Natur erstreckt sich auch auf das GUI. Jedes Fenster entspricht einem einzelnen Objekt, dessen Funktionalität mittels Klasse definiert wird. Um eine bessere Trennung der Animation von der AES De-/Chiffrierung zu erhalten, wurde die Oberfläche auf 3 Klassen aufgeteilt:

- Eine Klasse, welche die Animation beinhaltet
- Eine Klasse, welche die anfängliche AES Implementation beinhaltet
- Eine Klasse, die zur sichtbaren Trennung beider dient

Der größte Vorteil lag hierin, dass ein Fehler bei der Oberflächenerstellung im Laufe der Implementierung nicht auf bereits abgeschlossene Teile des Programms über-

ging, des weiteren verringerte es die Gefahr, dass eine einzelne Klasse zu groß und damit unübersichtlich würde. Ein weiteres Anliegen war die Fortführung des modularen Ansatzes des Programms. Dies sollte es in der Zukunft vereinfachen, das Programm um weitere Funktionalitäten wie zum Beispiel Algorithmen zu erweitern. Da es nicht möglich ist, vorherzusehen, welche Steuerungselemente für zukünftige Algorithmen vonnöten sein werden (auch wenn es sich empfiehlt dem Nutzer ein möglich kohärentes Interface zu bieten, und deshalb nach Möglichkeit Steuerungselemente ähnlich zu anderen sein sollten), macht es Sinn, das Programm in abgeschlossene Einheiten zu unterteilen, welche bei Bedarf aufgerufen werden können. Dies bedeutet aber auch, dass es zu einer gewissen Redundanz in der Implementierung von Funktionen kommen wird, wenn diese sich nicht in externe Klassen auslagern lassen können.
Nachdem die Implementierung des AES Algorithmus abgeschlossen, und ein Konzept für die Darstellung der Bewegungsabläufe gefunden wurde, begann die Implementierung jener.

5.3. Probleme bei der Implementierung

Während der Implementierung traten mehrere Probleme auf. Nachfolgend sollen einige wenige der Probleme näher beleuchtet werden, welche unterschiedliche Aspekte des Implementierungsvorganges hervorheben

5.3.1. Javas Byte Datentyp

Das Problem ergab sich aus dem Bedürfnis, die Bytebelegung als Hexadezimalzahlen (Zwei Hexadezimale Zeichen zur Darstellung von einem Byte, im Bereich von 00 bis FF) auszugeben. Dies wird im Programm erreicht, indem die byte Eingabe über eine Funktion in ein Integer Array überträgt (auf welchem dann gearbeitet wird), welche den Eingabewert des Bytes zum einen ganzzahlig durch 16 dividiert um das höherwertige Halb-Byte zu bekommen, zum anderen Modulo 16 rechnet, um das niederwertige Halb-Byte zu bekommen. Beide werden im Anschluss jeder für sich durch eine weitere Operation durch ein Objekt vom Typ char zwischen 0 und F ersetzt, um diese dann in einem String auszugeben. Für die Umrechnung wird das Byte in einen Integer umgewandelt. Hier greift eine Eigenart von Java, was den Byte Datentyp angeht. In sonstigen geläufigen Programmiersprachen wird dieser als Ganzzahl in einem Wertebereich von 0 bis 255 angegeben, Java allerdings definiert diesen allerdings in seiner Dokumentation in /OroD/ als eine vorzeichenbehaftete Ganzzahl im Zweierkomplement, welche von -128 bis 127 geht. Dadurch werden alle Zeichen, deren UTF-8-kodierte Darstellungen einem höheren Wert als 127 entsprechen bei der Umwandlung in einen Integer verfälscht.

Der erste Lösungsansatz sah hier eine simple Operation vor, bei welchem die einzelnen Bits des Bytes durch ein bitweise-Oder auf die niederwertigsten Bits des Integers übertragen werden sollten (was eventuell nicht die eleganteste Form gewesen wäre, aber zweckmäßig und zeiteffizient). Jedoch führte dies nicht zum erwarteten Ergebnis, weshalb letzten Endes jedes Byte mittels einer if-Anweisung überprüft werden muss, um gegebenenfalls durch eine Addition auf den gewünschten Wert zu kommen. Da im Programm auch die Möglichkeit besteht, Daten im Hexadezimalzahlformat einzugeben, wurde auch die Gegenrichtung implementiert.

5.3.2.Schrift

Im Programm werden die Dateninhalte mittels Texten dargestellt. Da zu erwarten ist, dass das Programm im Rahmen einer Vorlesung über einen Beamer an die Wand geworfen werden, muss sichergestellt sein, dass die Buchstaben groß genug sind, dass diese von Personen weiter hinten in einem Vorlesungssaal noch gut gelesen werden zu können. Dem Programm stehen für die Darstellung aller Inhalte nur eine begrenzte Anzahl an Bildpunkten zur Verfügung.
Das Problem ist, herauszufinden, wieviel Bildpunkte ein Buchstabe benötigt. Java erlaubt es nicht ohne weiteres, die Breite und Höhe von Schriftarten pixelgenau festzulegen. Es ist nur möglich, die Höhe und Breite eines Buchstaben zu ermitteln, nachdem dieser im Programm gerendert wurde. Um die Lesbarkeit für alle im Saal zu gewährleisten, müssen die einzelnen Zeichen der Schriftart also möglichst groß dargestellt werden, ohne dabei mit anderen Zeichen zu überlappen, oder gar die Begrenzungen des Programmes zu sprengen.
Die Lösung des Problems bestand aus zwei Teilen. Zuerst galt es, in allen Teilen der Animation den Teilbereich zu finden, bei dem die meisten Zeichen nebeneinander sowie übereinander standen, sowie wieviel Platz ihnen auf dem GUI zugestanden werden konnte, nachdem Elemente fester Größe (Trennstriche, XOR-Operatoren etc) sowie nötige Freiräume vom zur Verfügung stehenden Raum abgezogen wurden. Dadurch war es möglich die maximale Höhe/Breite zu bestimmen, mit der die Symbole noch im Rahmen der Vorgaben in das GUI reinpassten. Anschließend wurde die Schriftgröße experimentell bestimmt, deren Zeichen am größten sind, ohne die maximale Höhe oder Breite zu überschreiten. Da als Schriftart eine sogenannte Monospace Schriftart verwendet wurde, war sichergestellt, dass alle Zeichen gleich viel Platz benötigen.

5.3.3.Auslagerung von Animationen in separate Prozesse

Ursprünglich sollten die Animationen in separate Prozesse ausgelagert werden, um die Ansprechbarkeit des Programms zur Laufzeit zu gewährleisten. Jedoch erlaubt Java es nicht, dass separate Prozesse Änderungen wie Animationen an dem Interface

durchführen. Dies ist ein Problem, da der automatische Ablauf der Animation ein wesentlicher Bestandteil des Programms ist, und es Teil der Animation ist, zwischen den einzelnen Animationen eine gewisse Menge an Zeit verstreichen zu lassen. Dies wird mit einer Funktion erreicht, welche den gesamten Prozess blockiert, soll heißen, wird diese Funktion auf dem Thread des Interface ausgeführt, wäre dieses während der Prozess blockiert ist nicht ansprechbar.
Jedoch erlaubt Java keine Änderungen an dem Interface aus anderen Threads als dem Hauptthread.

5.3.4.Die Animation von XOR

Die Animation von XOR stellte sich als besonders schwierig heraus. Jeder einzelne Schritt in der Animation verbraucht eine gewisse Menge an Zeit, weshalb es nötig ist, mit möglichst wenig Schritten die Zusammenhänge darzustellen. Dies ist beim Nachschlagen in der S-Box Tabelle als auch beim Verschieben der Reihen gut möglich. Will man dagegen XOR gut darstellen muss man beachten, dass XOR eine bitweise Operation ist, während die Darstellung des States aber in hexadezimaler Schreibweise erfolgt. Dementsprechend müsste für eine leichter verständliche Darstellung der Operation zuerst beide Operanden aus der hexadezimalen Darstellung in die Binäre übertragen werden, zusätzlich zum Rückübersetzen vor dem speichern des Ergebnisses in den neuen State. Dann ist aber noch immer nicht die Operation an sich dargestellt. Eine Möglichkeit dafür bestünde darin, die Operation mittels einer schematisch dargestellten Schaltung zu animieren, was aber nicht zweckmäßig ist, da sie eine recht spezifische Grundkenntnis des Betrachters voraussetzt. Die offensichtlichste Form wäre über eine 2x2 Tabelle. Der Vorgang würde animiert werden wie beim Nachschlagen der Werte in der S-Box Tabelle, was bedeutet, dass die Animation des XOR 8-Mal so lang wäre (da jedes Bit nachgeschlagen werden muss). Wenn man den bitweisen Charakter ignoriert und direkt die hexadezimalen Werte in einer Tabelle nachschlägt, bräuchte man für das Nachschlagen des gesamten Bytes eine Tabelle, die aufgrund ihrer Größe (128x128 Elemente) so nicht auf dem Bildschirm dargestellt werden kann. Allerdings wäre es möglich, das höherwertige und niederwertige Halbbyte getrennt voneinander nachzuschlagen, was dazu führt, dass die Animation zwei Mal so lang wäre wie die von SubBytes (bei gleich großer Tabelle). In Anbetracht dessen, dass während einer Runde im AES XOR mehrmals auftritt, sind diese Varianten nicht sehr empfehlenswert.
Letzten Endes musste hier abgewogen werden zwischen der Laufzeit der Animation und der benötigten Komplexität. Letzten Endes ist XOR an sich ein simpler Vorgang, der innerhalb von ein bis zwei Sätzen erklärt werden kann. Es ist also unnötig diesen noch weiter zu vereinfachen.

5.3.5.Speicherleck

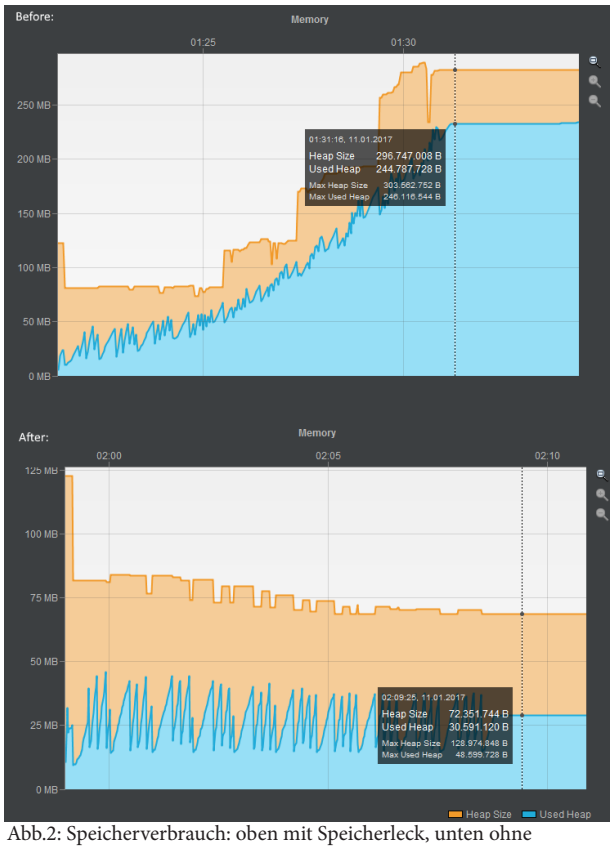


Abb.2: Speicherverbrauch: oben mit Speicherleck, unten ohne

Das mit Abstand größte Problem im gesamten Implementierungsprozess stellte ein Speicherleck dar. Um einen Text beliebiger Länge mit AES zu verschlüsseln, wird dieser vorher in Blöcke fester Länge aufgeteilt, welche dann mit AES chiffriert werden können. Als es daran ging, mehrere Blöcke mittels Animation zu verschlüsseln, stellte sich heraus, dass während des Chiffrierungsvorganges der Speicherbedarf stetig anwuchs, und kurz nach Beginn der Chiffrierung des zweiten Blocks aufgrund von zu hohem Speicherbedarf abstürzte.
Der erste Verdacht lag hier in der Art, wie Attribute der einzelnen Elemente gespeichert werden. Jedes dargestellte Objekt bekommt zur Laufzeit seine eigene CSS (Cascading Style Sheet) Datei, in welcher Attribute wie etwa Schriftgröße oder Hintergrundfarbe gespeichert werden. Während des Programmablaufs wurden die geänderten Daten einfach in die Datei gespeichert, weshalb die Vermutung nahelag, dass es die CSS Dateien durch wiederholtes Speichern der geänderten Attribute immer größer wurden. Dieser Verdacht wurde jedoch fürs erste zerstreut, als eine Ausgabe der CSS Dateien auf die Konsole zur Laufzeit keine Änderung im Umfang der CSS Dateien erahnen ließ.
Bei einem erneuten Durchgang des Debuggers fiel die scheinbar ungewöhnlich hohe Menge an Objekten älterer Generationen auf. In der Regel sind die meisten Objekte in Java kurzlebig (also junger Generation), weshalb eine hohe Menge an langlebigen Objekten ein Hinweis auf ein Fehler in der Programmierung sein kann. Dementsprechend lag der nächste Schritt darin, die Erstellung von neuen Objekten (bis dahin entsprach jede Art einer Animation eines Bytes, egal ob Translation oder Farbänderung, der Erstellung eines neuen Objekts) zu minimieren. Leider führte auch dies nicht zum gewünschten Erfolg.
Letzten Endes stellte sich heraus, dass der Fehler sehr wohl in den CSS Dateien lag, wenn auch nicht durch die Ausgabe des Inhalts der CSS Datei auf die Konsole. Es war erst möglich, die CSS Dateien als Fehlerquelle festzulegen, als die programmatische Änderung dieser im Rahmen der allgemeinen Fehlersuche testweise komplett entfernt wurde, was sofort zu einem deutlich merkbaren Rückgang des Speicherbedarfs führte. Nachdem die betroffenen Funktionen angepasst wurden, war auch das Speicherleck nicht mehr vorhanden.

6. Auswertung

Das Programm wurde in einer Vorlesung eingesetzt. Die Rückmeldung durch die Studenten war positiv.

Geht man rein von dem aus, was das Programm bietet, lässt sich sagen: Die Animation an sich bietet nicht genug Informationen, um allein daraus die Funktionsweise des AES zu erschließen. Deshalb kann das Programm in seiner jetzigen Form allerhöchstens genutzt werden, wenn eine Person nebenbei erklärt was die Animationen zu bedeuten haben. Jedoch genügt das Programm dahingehend, zu zeigen dass es möglich ist, Animationen mit einem Knotenbasierten GUI zu erstellen. Des weiteren wurden im Rahmen der Bachelorarbeit Funktionen erstellt, die es erleichtern sollten, weitere Algorithmen dem Programm hinzuzufügen.

Als unerwartet nützlich erwies sich der Aufbau des Baumes bei der Aufspaltung des AES Algorithmus. Es erleichterte vor allem Dingen die Abschätzung des benötigten Aufwands für die Animation.

7. Ausblick

Ein Fokus dieser Arbeit war die Schaffung einer Basis und von Animationen, die es erlauben, das Programm mit relativ wenig Aufwand um weitere Algorithmen zu erweitern.
Je nachdem worauf der Fokus einer Weiterentwicklung liegt, könnten dem Programm weitere Informationen während der Animation hinzugefügt werden (damit das Programm irgendwann auch zum Selbststudium von Algorithmen geeignet ist), oder aber durch das Hinzufügen anderer Algorithmen zu einem Tool ausgebaut werden, was irgendwann idealerweise den gesamten Kurs begleitet.
Es könnte des weiteren auch Sinn machen, die Oberfläche zu überarbeiten. So könnte man von dem Knotenbasierten GUI zu einem Canvas basierten GUI übergehen, um einige der Animationen eventuell besser zu realisieren.
Ein weiteres nicht vollends erfasstes Gebiet stellt die Speicherung der Abläufe dar. Eine weitere Möglichkeit das Programm zu erweitern könnte also auch darin liegen, ein allgemein gültiges System zu kreieren, mit dem Animationen so abgespeichert werden, dass man direkt jeden Einzelschritt in dem Algorithmus anspringen kann.

8. Literaturverzeichnis

/FM92/	Fleming, N. D., & Mills, C. (1992, 1. Januar). Not Another Inventory, Rather a Catalyst for Reflection. Abgerufen von http://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1245&context=podimproveacad
/LSS10/	Leite, W. L., Svinicki, M., & Shi, Y. (2010). Attempted Validation of the Scores of the VARK: Learning Styles Inventory With Multitrait–Multimethod Confirmatory Factor Analysis Models. Educational and Psychological Measurement, 70(2), 323-339. doi:http://journals.sagepub.com/doi/pdf/10.1177/0013164409344507
/NIST01/	National Institute of Standards and Technologies. (2001, 26. November). FIPS 197, Advanced Encryption Standard (AES). Abgerufen von https://doi.org/10.6028/NIST.FIPS.197
/OA10/	Othman, N., & Amiruddin, M. H. (2010). Different Perspectives of Learning Styles from VARK Model. Procedia - Social and Behavioral Sciences, 7, 652-660. doi:https://doi.org/10.1016/j.sbspro.2010.10.088
/OroD/	Oracle. (o.D.). Primitive Datatypes. Abgerufen von https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html
/VLL17/	VARK Learn Limited. (o.D.). The VARK Modalities. Abgerufen 17. April, 2017, von http://vark-learn.com/introduction-to-vark/the-vark-modalities/

9. Glossar

Algorithmus	Eine Abfolge von Befehlen, welche in einem Programm ein vorhandenes Problem abarbeitet/löst
Subalgorithmus	Ein Algorithmus, welcher Teil eines Algorithmus ist
GUI	Englische Kurzform für Grafisches Benutzerinterface. Das grafische Benutzerinterface stellt alle für den Nutzer sichtbaren Komponenten des Programms dar (Programmfenster, Knopf, Text etc).
Atomarer Algorithmus	Ein Algorithmus, der keine Subalgorithmen besitzt

10. Abbildungsverzeichnis

11. Anhangsverzeichnis

Anhangsnummer	Titel	Seitenzahl
1	VAES Tool Handbuch	7

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit eigenständig und nur unter Zuhilfenahme der angegebenen Quellen erstellt habe.

Merseburg, den 24. April 2017

Unterschrift

Anhang 1

VAES Tool Handbuch

Inhalt

Voraussetzungen

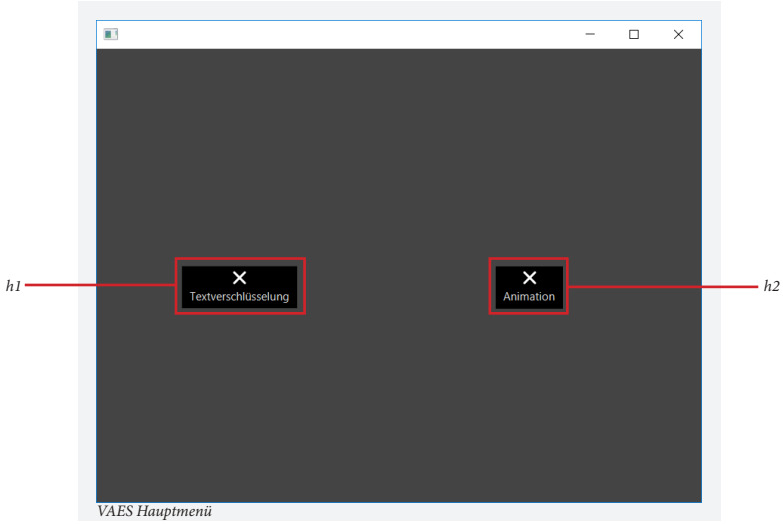
Zum Betrieb des Programms wird eine installierte Version der Java Runtine Environment der Version 8 oder neuer von Oracle vorausgesetzt. Von der Verwendung von OpenJRE wird abgeraten, da es hierbei zu Problemen kam.

Installation von JRE

Oracle Webseite mit Java Download

Die JRE können sie direkt bei Oracle runterladen. Gehen sie dazu auf die Seite:
https://java.com/de/download/
Klicken sie anschließend auf den Kostenlosen Java-Download Button und folgen sie den Anweisungen zum herunterladen. Anschließend können sie die JRE nach Belieben installieren.

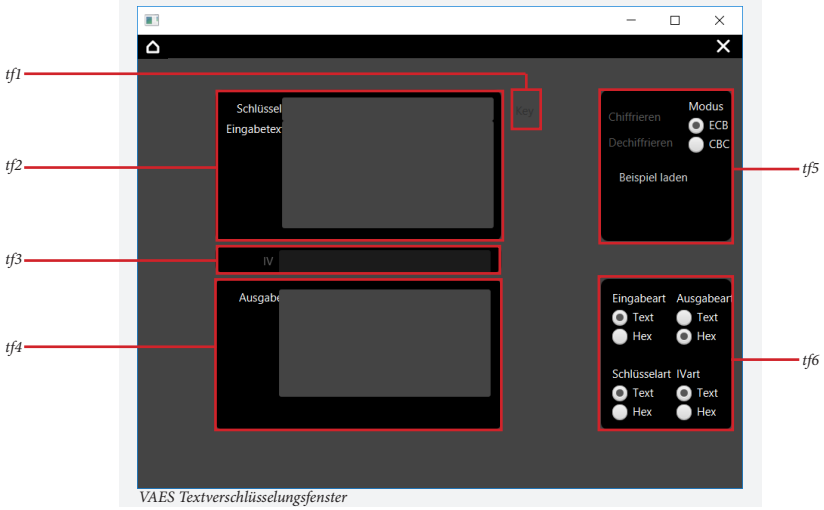
Hauptmenü



VAES Hauptmenü

Im Hauptmenü stehen ihnen derzeit zwei Auswahlmöglichkeiten zur Verfügung.
h1. Textverschlüsselung führt zu dem Teil des Programms, mit welchem man beliebige Strings mittels AES verschlüsseln und entschlüsseln kann.
h2. Animation führt zu dem Teil des Programms, welches die Verschlüsselung eines selbstgewählten Textes animiert.

Textverschlüsselung



VAES Textverschlüsselungsfenster

tf1. Schlüsselkontrollsymbol tf2. Eingabemaske tf3. Initialisierungsvektor Eingabefenster
tf4. Ausgabefenster tf5. Kontrollfenster f6. Einstellungsfenster

In diesem Fenster kann man beliebige Texte in den Betriebsmodi ECB und CBC ver-/ oder entschlüsseln. Sowohl Klartext als auch hexadezimale Byte-Darstellung werden unterstützt, und können für alle Eingaben unabhängig voneinander eingestellt werden. Standardmäßig wird davon ausgegangen, dass man einen Text im ECB Modus verschlüsselt, und dabei alle Eingaben (zu chiffrierender Text, Schlüssel) in Klartext tätigt, während die Ausgabe in Hexadezimaler Byte-Darstellung erfolgen soll.

Bitte beachten:
Es wird derzeit davon ausgegangen, dass der verwendete Schlüssel genau 128/192/256 Bit, bzw 16/24/32 Byte lang ist. Aufgrund dessen, dass alle Klartexteingaben mittels UTF-8 kodiert werden, entsprechen nicht-standard-englische Zeichen (wie zum Beispiel ä,ß,あ) mehr als einem Byte.

Anleitung zur Benutzung

Text verschlüsseln:

1. Geben sie den Text und Schlüssel in die dazugehörigen Felder in tf2 ein
    - Kontrollierung der Korrektheit des Schlüssels mittels tf1

Optional: Wählen sie in tf5 den CBC Modus und ergänzen die Eingabe in tf3 um den IV

2. Wählen bzw. kontrollieren sie die gewünschten Darstellungsparameter in tf6
3. Klicken sie in tf5 auf Chiffrieren

Text entschlüsseln:

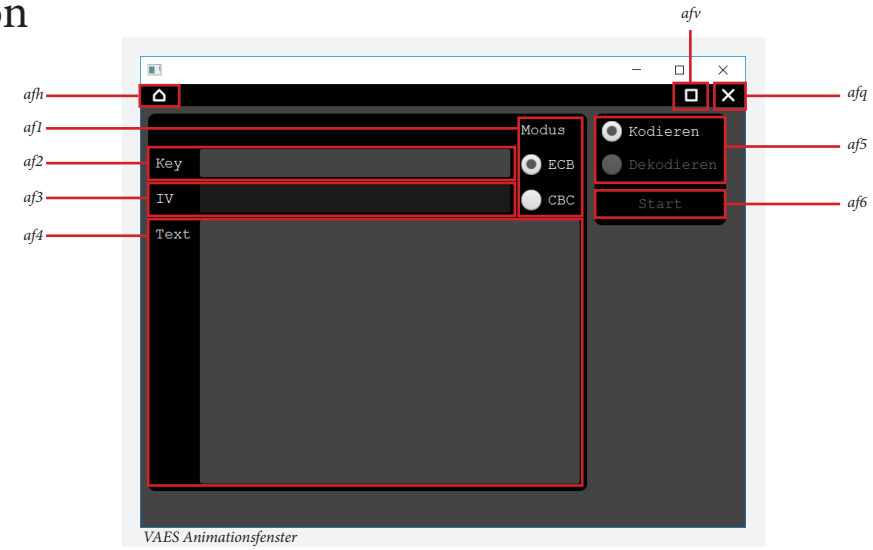
1. Kopieren sie den verschlüsselten Text in den Bereich für Eingabetext in tf2, und schreiben sie dazu noch den zu verwendenden Schlüssel in das dafür vorgesehene Feld
    - Der Schlüssel wird automatisch in tf1 auf Korrektheit überprüft

Optional: Wählen sie in tf5 den CBC Modus und ergänzen sie die Eingabe in tf3 um den IV

2. Wählen bzw. kontrollieren sie die gewünschten Darstellungsparameter in tf6
3. Klicken sie in tf5 auf Dechiffrieren

Nützlich:
Die Auswahl der Darstellung (Klartext oder Hexadezimal) der Ausgabe wird sofort übernommen, ohne erneut den Text de-/chiffrieren zu müssen.

Animation



VAES Animationsfenster

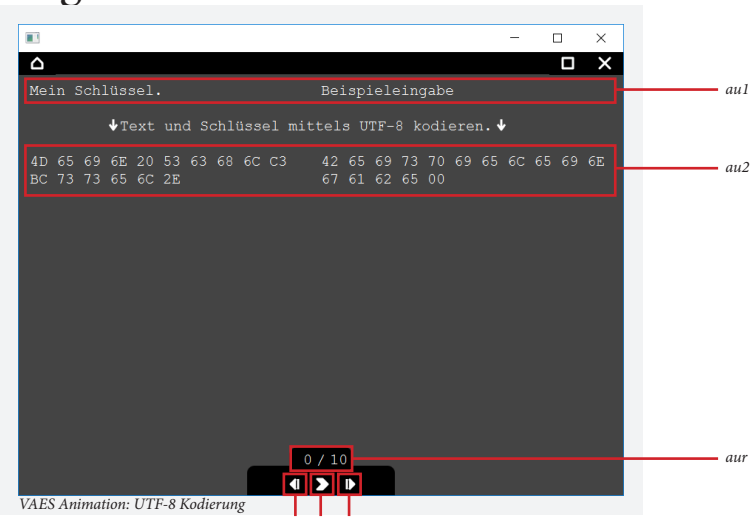
af1. Betriebsmodusauswahl
af2. Schlüsselfeld
af3. Initialisierungsvektorfeld
af4. Eingabetextfeld
af5. Animationsauswahl
af6. Animationsstartknopf
afh. Hauptmenüknopf
afv. Vollbildknopf
afq. Programm beenden

In diesem Fenster kann man sich die Chiffrierung eines beliebigen Textes animieren lassen. Im Moment wird nur der ECB Modus unterstützt, af3 ist funktionslos. Es wird angenommen, dass alle Daten als Klartext eingegeben werden.

Um eine Animation zu starten, gibt der Nutzer einen zu verschlüsselnden Text in af4 und einen gültigen Schlüssel in af1 an. Sobald ein gültiger Schlüssel (wie auf der vorigen Seite erklärt) vorliegt, kann der Animationsstartknopf bei af6 betätigt werden.

Erwähnenswert:
Dieser Teil des Programms erlaubt es in den Vollbildmodus zu gehen, indem man auf den Vollbild-Knopf drückt.

UTF-8 Kodierung



VAES Animation: UTF-8 Kodierung

au1. Nutzereingaben
auz. Schritt-zurück-Knopf
aur. AES-Rundenanzeige
aus. Autolauf Start/Stop
au2. UTF-8 kodierte Darstellung
auv. Schritt-vorwärts-Knopf

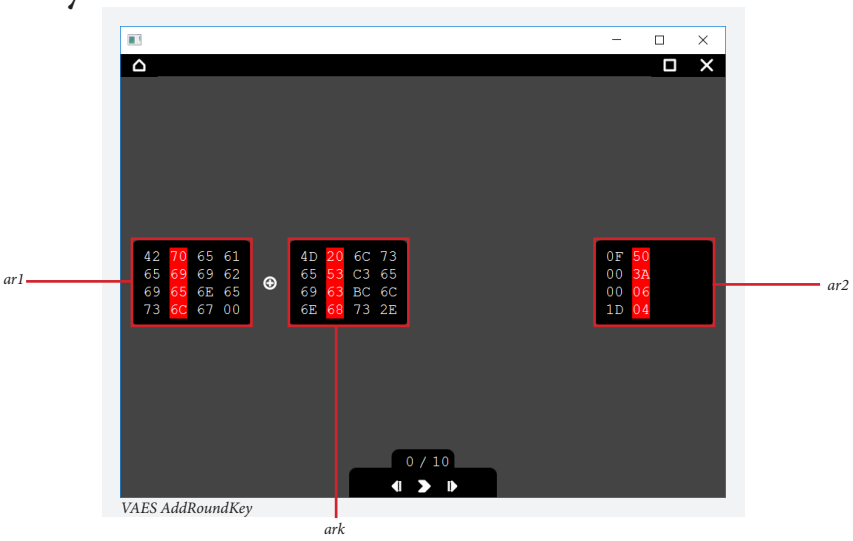
Die Animation beginnt mit der UTF-8 Kodierung des eingegebenen Textes und Schlüssels aus ihrer Klartextform in au1 in die entsprechende Hexadezimale Byte-Darstellung. Da die Kodierung streng genommen nicht Teil des AES Algorithmus ist, wird diese in einem Schritt für beide Texte durchgeführt.
Mit dem Start der Animation hat sich auch die Bedienoberfläche geändert. Zuoberst befindet sich bei *aur* die AES-Rundenanzeige, welche stets die aktuelle Runde im AES Algorithmus anzeigt. Darunter befinden sich die eigentlichen Bedienelemente. Mit dem mittleren Element bei *aus* wird der automatische Ablauf der Animation gestartet bzw. gestoppt. Links davon befindet sich das Element *auz*, mit welchem man einen einzelnen Schritt in der Animation zurück gehen kann. Rechts befindet sich analog dazu das Element *auv*, mit welchem man die Animation einen einzelnen Schritt vorwärts gehen kann.

Ein Hinweis:
Sowohl der einzelne Schritt zurück, als auch der einzelne Schritt vorwärts beenden den automatischen Ablauf der Animation.

Im Anschluss wird, wie unten zu sehen, noch die Hexadezimale Darstellung des Eingabetextes in die vom AES genutzte 4x4 Array Darstellungsform übertragen.

VAES Übertragung von ECB auf state vorher
VAES Übertragung von ECB auf state nachher

AddRoundKey

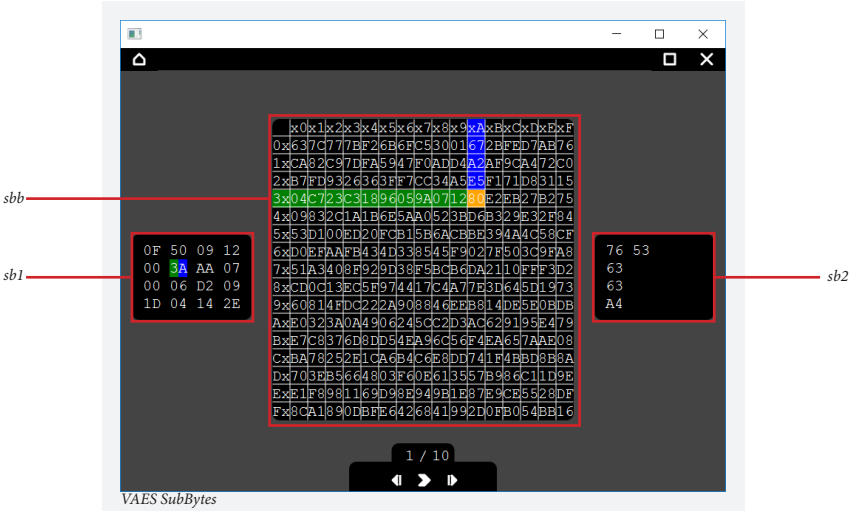


ar1. AES State vor der Operation ar2. AES state nach der Operation ark. Rundenschlüssel

In diesem Schritt wird der State aus ar1 mit dem Rundenschlüssel in ark verXORt. Zur Orientierungshilfe werden die zu verXORenden Spalten farbig eingefärbt.

Beachte:
Wie die Rundenschlüssel errechnet werden wird hier nicht dargestellt. Dies muss unbedingt kommuniziert werden, um Unklarheiten beim Studenten zu beseitigen.

SubBytes

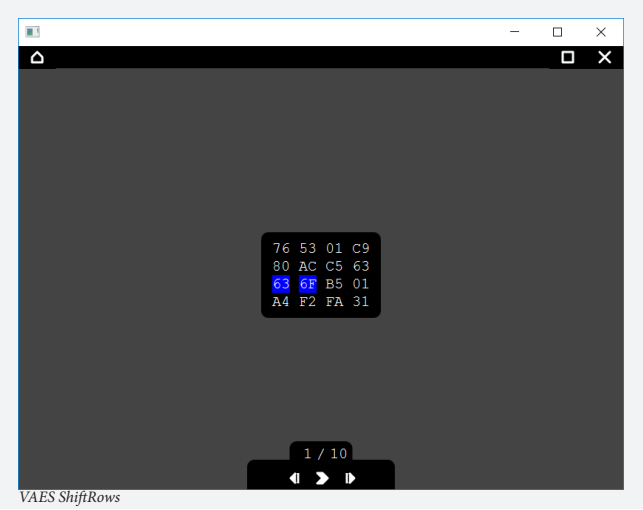


sb1. AESstate vor Operation sbb. Substitutionsbox sb2. AESstate nach Operation

In diesem Schritt der Animation wird gezeigt, wie man in SubBytes aus der S-Box in sbb die Bytes findet, mit denen man die Bytes aus sb1 im Ergebnisstate (sb2) ersetzt.

Bedenke:
Die Animation geht aus Zeitgründen nicht darauf ein, wie die S-Box gebildet wird. Sollte ein Student Wert darauf legen, wie diese Substitutionsbox gebildet wird, muss dies so erläutert werden.

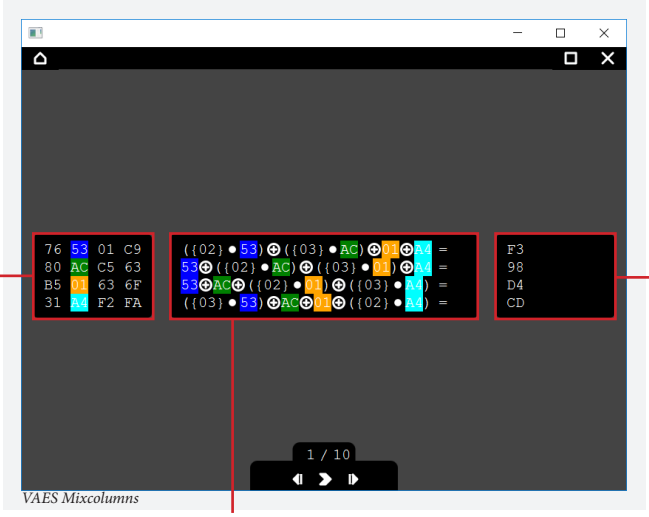
ShiftRows



VAES ShiftRows

ShiftRows ist der wahrscheinlich am leichtesten zu verstehende Teil im gesamten AES Algorithmus. Dementsprechend ist auch die Animation so gehalten, dass er keine Erklärungen benötigt.

MixColumns



mc1 *mc2* *mcm*

VAES Mixcolumns

mc1. AESstate vor der Operation mcm. Gleichungssystem mc2. AESState nach der Operation

In diesem Schritt wird dargestellt, wie der state mc1 durch affine Transformationen in mcm zum State mc2 übergeht. Alternativ zu dieser Art hätte man die Matrizenmultiplikation auch eindeutiger als solche darstellen können, allerdings wurde darauf verzichtet, da dies zu mehr Animationsschritten, und dementsprechend zu einer längeren Animationszeit geführt hätte.