AES

-

Symmetrische Verschlüsselung in TASM umgesetzt

erstellt von

Tim Michael Kretzschmar

Konstantin Blechschmidt

Abschlussprojekt “Assemblerprogrammierung“ - 19-INB1

Inhaltsverzeichnis

Kapitel 1

Einleitung

AES wurde von `Joan Daemen` und `Vincent Rijmen` (unter dem Alias Rijndael) entwickelt. Es basiert auf dem Prinzip der symmetrischen Verschlüsselung und gilt mit seinen verschiedenen Standards als eines der besten Verschlüsselungsverfahren der Welt. Dies ist bedingt durch die Tatsache, dass bis heute kein effektiver Algorithmus gefunden wurde, welcher die Brute-Force-Methode zum Entschlüsseln des Chiffretextes umgeht. Da in absehbarer Zeit die Prozessorleistung keinen sonderlichen Leistungsschub mehr erhalten wird, ist die Schlüssellänge für die Sicherheit ausschlaggebend; also die Dauer der Entschlüsselung.

Die hier als Assemblerprogramm umgesetzte Variante beruht auf dem Grundgedanken von AES. Sie erfolgt über eine Schlüssellänge von 64 Zeichen a 8 Bit (also 256 mögliche Werte (0..255), was der erweiterten `ASCII Tabelle` entspricht).
Somit erfolgt die Verschlüsselung nach folgender Grafik:

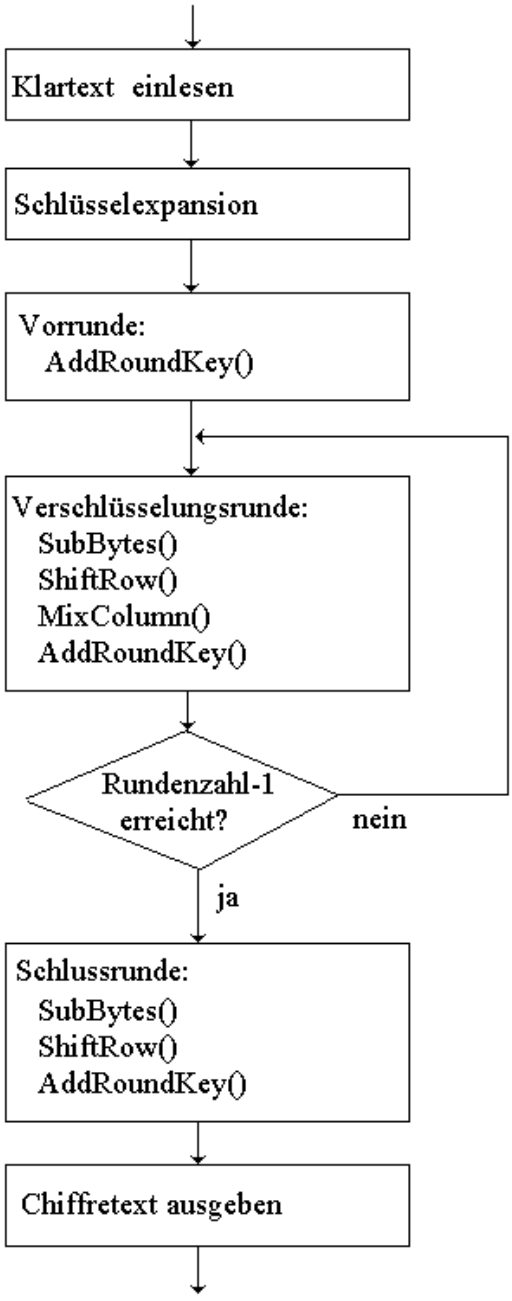


Abbildung 1.1: Schema Verschlüsselung mit AES - `Quelle im Anhang`

Und die Entschlüsselung nach diesem Muster:

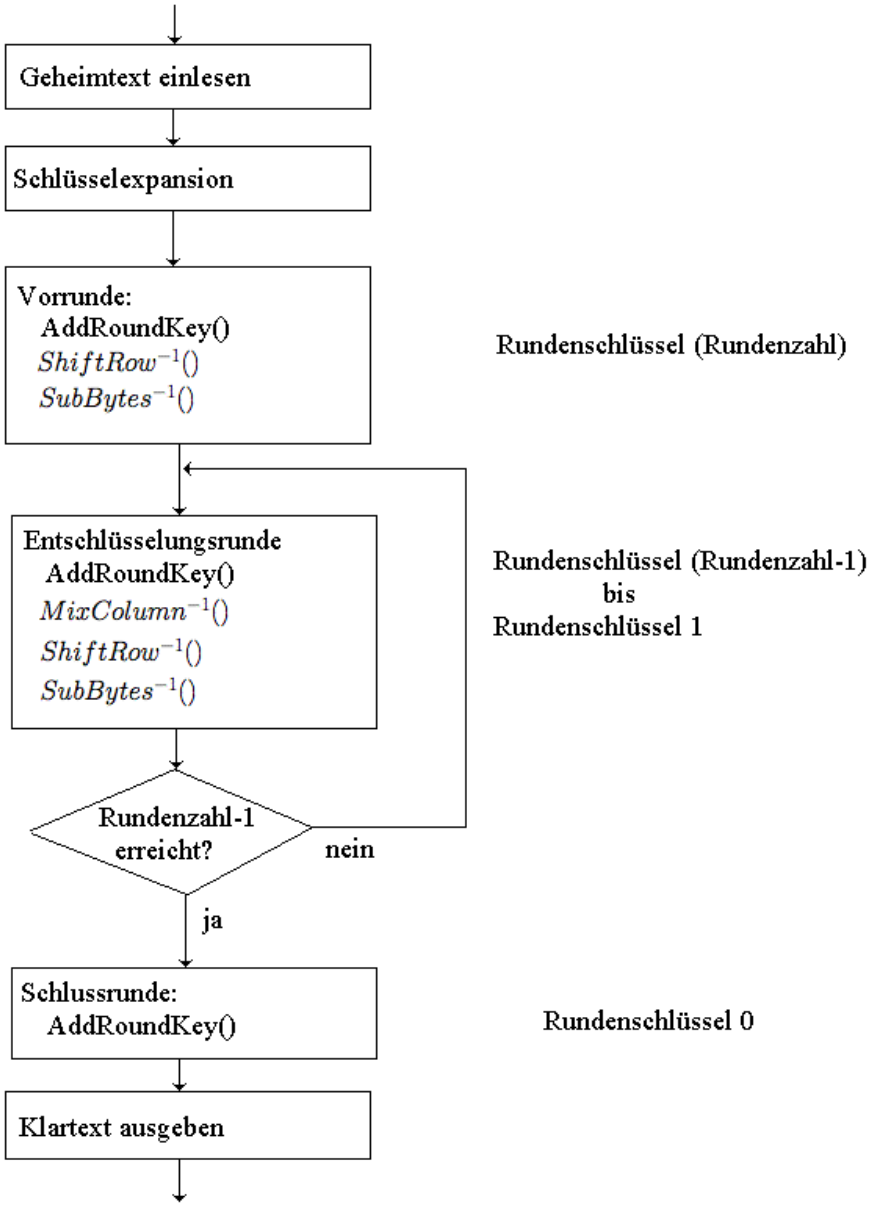


Abbildung 1.2: Schema Verschlüsselung mit AES - `Quelle im Anhang`

Kapitel 2

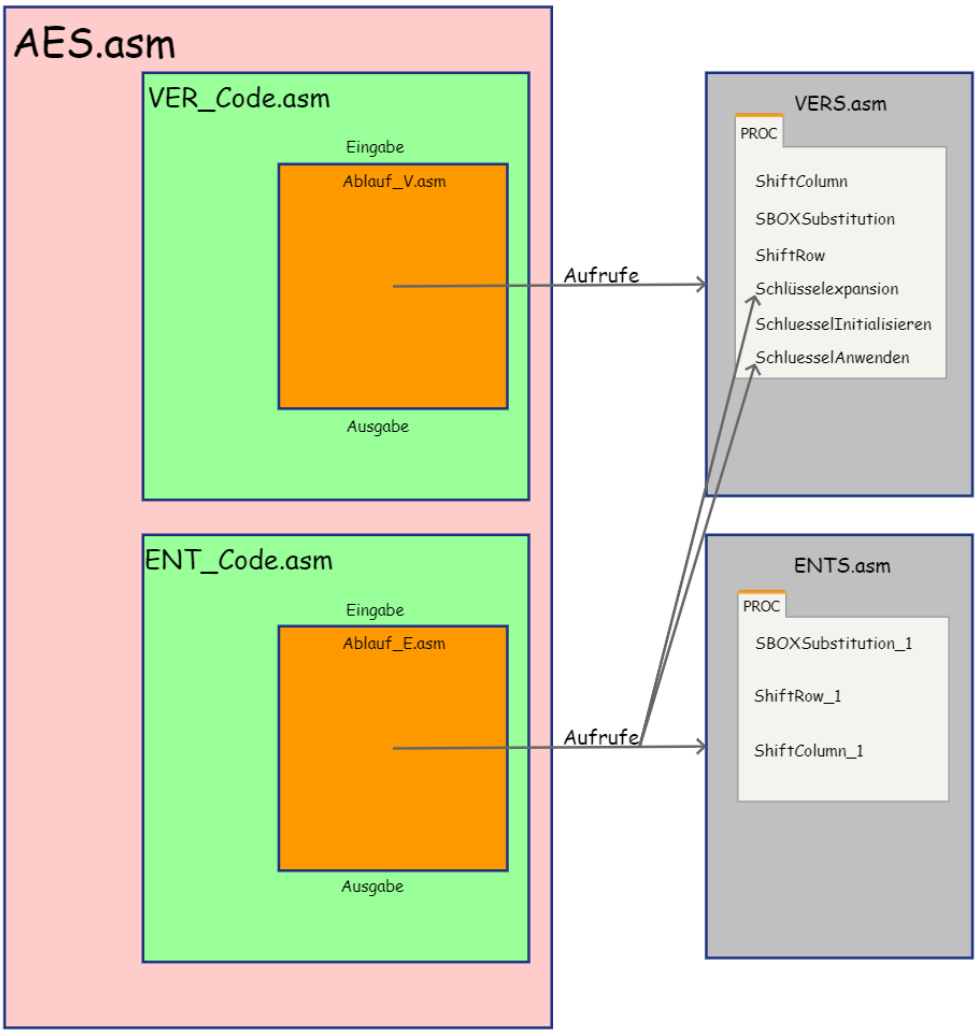
Die ASM-Dateien und ihr Inhalt

HINWEIS: Zu jeder Datei findet hier eine kurze Beschreibung statt. Meist ist jedoch die genauere Erklärung im Quelltext als Kommentar hinter den jeweiligen Befehlszeilen zu finden. Zusätzlich gibt es im Kopf jeder Datei eine Beschreibung, die diesem Werk ähnelt. Es soll lediglich dazu dienen, diese Kopf-Kommentare zu erweitern, noch besser zu strukturieren und einen allgemeinen Überblick zu erhalten.

2.1 Grafische Übersicht über alle ASM-Dateien

Hier sind alle ASM-Dateien aufgeführt. Gleichzeitig soll das Zusammenspiel der `AES.asm`, der `VER_Code.asm` und `ENT_Code.asm`, sowie den `Ablauf_V.asm` und `Ablauf_E.asm` graphisch demonstriert werden, was im späteren Verlauf schriftlich erklärt wird.


SBOX.asm
MAUS.asm
PROC
MausPosiProc
ISR.asm
config.asm
BildPROC.asm
PROC
HauptbildschirmProc
BildschirmProcX


AES.asm
VER_Code.asm
Eingabe
Ablauf_V.asm
Ausgabe
Aufrufe
VERS.asm
PROC
ShiftColumn
SBOXSubstitution
ShiftRow
Schlüsselexpansion
SchluesselInitialisieren
SchluesselAnwenden
ENT_Code.asm
Eingabe
Ablauf_E.asm
Ausgabe
Aufrufe
ENTS.asm
PROC
SBOXSubstitution_1
ShiftRow_1
ShiftColumn_1

2.2 AES.asm

Hier findet die grundlegende Verwaltung des Ablaufes statt. Diese Datei kann man auch als `main` bezeichnen. Im Folgenden wird auf die einzelnen `Label` und ihre Funktionalitäten näher eingegangen.

2.2.1 Beginn (Label)

Laden des Data-Segmentes in DS. Im Data-Segment sind 2 INCLUDES vorhanden:

1 `SBOX.asm`

2 `config.asm`

2.2.2 Hauptbildschirm darstellen (Label)

Wie der Name schon erwarten lässt, wird hier der Hauptbildschirm gezeichnet. Dies geschieht über den Aufruf einer `PROC HauptbildschirmProc`. Davor wird jedoch ebenfalls noch in diesem `Labels` die Maus-Click-Funtkion belegt. Dies geschieht durch das Freisetzten der beiden Primärmaustasten links und rechts, sowie dem Definieren des OFFSETS der `PROC MausPosiProc`.

2.2.3 Endlosschleife (Label)

In der Endlosschleife wird auf einen Tastaturdruck gewartet (`INT 16h, Unterfunktion 1h`). Nachdem dieser erfolgt ist, folgt ein `CMP` mit dem Ziel, die ESC-Taste zu detektieren. Bei einer Übereinstimmung springt das Programm zu dem `ESCEnde` (Label). Andernfalls wird eine Überprüfung des Inhalts der Varible *bool_Ent_Ver* vorgenommen. Hier können 3 Fälle eintreten.

```
Wert = 1  Springe zum EntschluesselungBeginn (Label)
Wert = 2  Springe zum VerschluesselungBeginn (Label)
Wert = 3  Springe zurück zur Endlosschleife (Label)
```

2.2.4 VerschluesselungBeginn (Label)

Hier wird über ein `INCLUDE` der Code für das Verschlüsseln aus der Ver_Code.asm eingefügt und durchlaufen. Nach dessen Abschluss wird die Variable *bool_Ent_Ver* wieder auf 3 gesetzt, also in den Wartezustand überführt. Als letzte Handlung folgt ein `JMP` zum `HauptbildschirmDarstellen` (Label).

2.2.5 EntschluesselungBeginn (Label)

Hier wird über ein `INCLUDE` der Code für das Entschlüsseln aus der Ent_Code.asm eingefügt und durchlaufen. Nach dessen Abschluss wird die Variable *bool_Ent_Ver* wieder auf 3 gesetzt, also in den Wartezustand überführt. Als letzte Handlung folgt hier ebenfalls ein `JMP` zum `HauptbildschirmDarstellen` (Label).

2.2.6 ESCEnde (Label)

In diesem Programmteil wird eine Überprüfung der Variable *bool_Ent_Ver* auf den Wert 3 vorgenommen. Sollte eine Übereinstimmung vorliegen, so wird der Sprung (`JMP`) zum `Ende` (Label) vorgenommen, andernfalls springt das Programm zum `HauptbildschirmDarstellen` (Label). Somit wird ein ungewolltes Verlassen aus dem Ent- bzw Verschlüsselungsbildschirm verhindert.

2.2.7 Ende (Label)

Als Erstes wird der Bildschirm geleert. Dies geschieht über den erneuten Aufruf des Videomodus 3, ohne jedoch etwas Weiteres darzustellen. Anschließend erfolgen die zwei Standardzeilen in `TASM` für das Zurückkehren in die DOS-Box:

```
MOV AH, 4Ch
INT 21h
```

2.3 MAUS.asm

2.3.1 MausPosiProc (PROC)

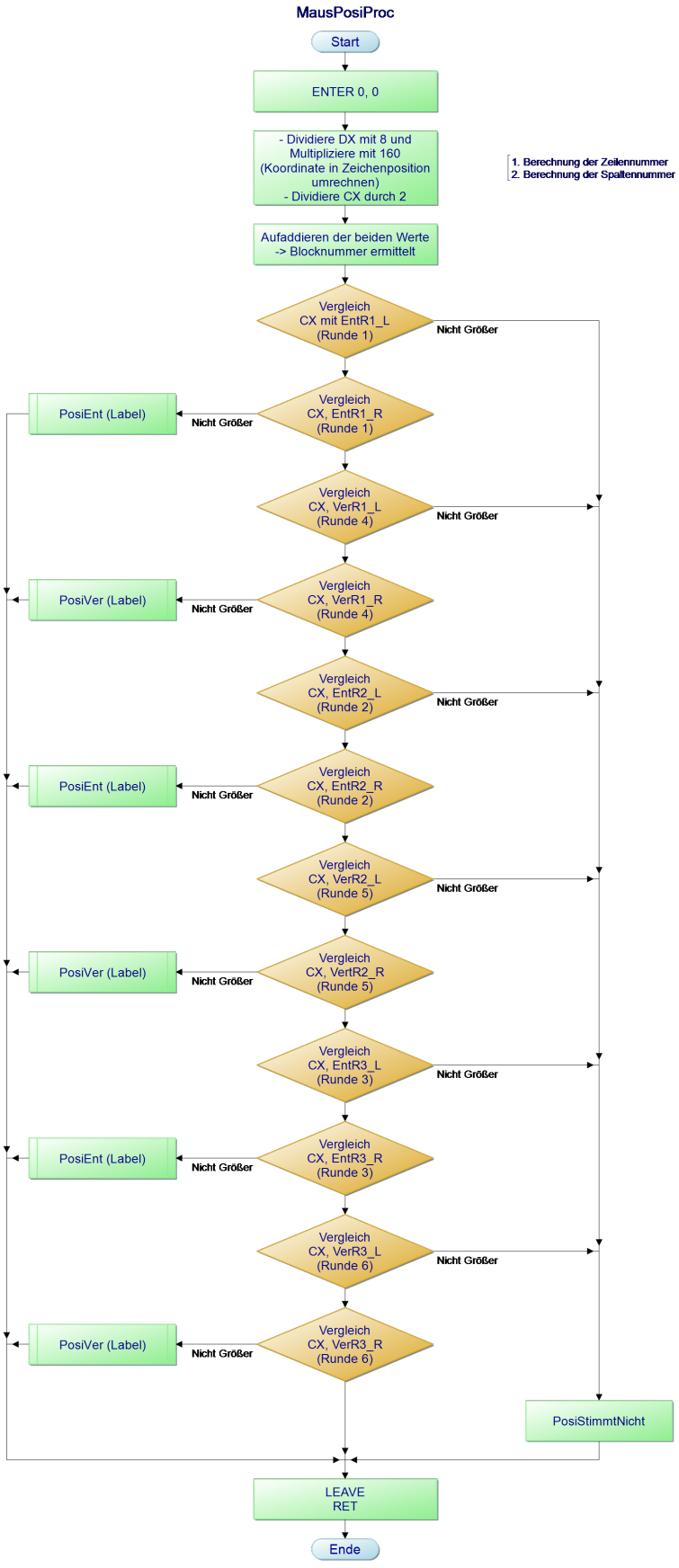
In dieser `PROC` wird überprüft, wo der sich Mauszeiger beim Klicken befand. Je nach Ergebnis wird dann ein anderer Bildschirm angezeigt. Dabei gibt es drei Fälle:

1 Keine Schaltfläche getroffen

2 Entschlüsseln getroffen

3 Verschlüsseln getroffen

Der simpelste Fall ist die Nummer 1. Hier wird einfach nichts getan, außer den Zustandspeicher *bool_Ent_Ver* auf 3 zu setzten, was als "nichts zu tun" interpretiert wird.
Fall 2 setzt diese Variable jedoch auf 1 (entspricht Entschlüsselung) und zeichnet anschließend den *Hauptbildschirm2*. Dieser beinhaltet die Flächen für die Text- und Schlüsseleingabe. Sollte jedoch kein oder ein nur unvollständiger Schlüssel eingegeben werden, so wird automatisch der vordefinierte Schlüssel aus der `config.asm` für den weiteren Ablauf verwendet.
Auf der folgenden Seite ist der Ablauf der Feststellung, ob und welche Schaltfläche getroffen wurde, graphisch dargestellt.


MausPosiProc
Start
ENTER 0, 0
- Dividiere DX mit 8 und Multipliziere mit 160 (Koordinate in Zeichenposition umrechnen)
- Dividiere CX durch 2
1. Berechnung der Zeilennummer
2. Berechnung der Spaltennummer
Aufaddieren der beiden Werte -> Blocknummer ermittelt
Vergleich CX mit EntR1_L (Runde 1)
Nicht Größer
Vergleich CX, EntR1_R (Runde 1)
PosiEnt (Label)
Nicht Größer
Vergleich CX, VerR1_L (Runde 4)
Nicht Größer
Vergleich CX, VerR1_R (Runde 4)
PosiVer (Label)
Nicht Größer
Vergleich CX, EntR2_L (Runde 2)
Nicht Größer
Vergleich CX, EntR2_R (Runde 2)
PosiEnt (Label)
Nicht Größer
Vergleich CX, VerR2_L (Runde 5)
Nicht Größer
Vergleich CX, VertR2_R (Runde 5)
PosiVer (Label)
Nicht Größer
Vergleich CX, EntR3_L (Runde 3)
Nicht Größer
Vergleich CX, EntR3_R (Runde 3)
PosiEnt (Label)
Nicht Größer
Vergleich CX, VerR3_L (Runde 6)
Nicht Größer
Vergleich CX, VerR3_R (Runde 6)
PosiVer (Label)
Nicht Größer
PosiStimmtNicht
LEAVE
RET
Ende

2.4 Ver_Code.asm

Nachdem der Bildschirm zum Verschlüsseln aufgerufen wurde, ist hier der Ablauf von Klartext-Eingabe, Schlüssel in Hex-Werten und chiffrierter Textausgabe (ebenfalls in Hex-Werten) definiert. Die Klartextlänge ist auf 64 Zeichen und die Schlüssellänge auf 128 Zeichen beschränkt. 2 Eingaben beim Schlüssel erzeugen einen internen Hex-Wert.
Durch einen `INCLUDE` von der `Ablauf_V.asm`, in der die Struktur der Verschlüsselung beschrieben ist , wird in diesem Programmteil die interne Verschlüsselung an sich aufgerufen.

2.5 Ent_Code.asm

Nachdem der Bildschirm zum Entschlüsseln über die dafür existierende `PROC` (siehe Absatz `2.13.2 BildschirmProcX`) gezeichnet wurde, kommt es hier ebenfalls zu einer Eingabe. Dieses Mal erfolgt die Texteingabe wegen des zu entschlüsselnden Zeichensatzes über hexadezimale Werte. Erlaubt sind dabei 0 bis 9 und A bis F. Nur Großbuchstaben werden akzeptiert.
Nach dem selben Muster erfolgt auch die Eingabe für den Schlüssel. Die Schlüsseleingabe wird den Standardschlüssel (welcher fest integriert ist) soweit überschreiben, wie diese Eingabe erfolgt. Es können wieder maximal 128 Zeichen eingegeben werden.
2 Eingaben erzeugen einen internen Hex-Wert.
Vor der Ausgabe des Klartextes wird in diesem Programmabschnitt ähnlich der `Ver_Code.asm` ein `INCLUDE` durchgeführt, welcher den Ablauf der Entschlüsselung aus der `Ablauf_E.asm` aufruft.

2.6 Ablauf_V.asm

In dieser ASM-Datei wird der allgemeine Ablauf geregelt, der bei einer Verschlüsselung stattfindet. Die Umsetzung in den `Labels` erfolgte nach diesem Muster (siehe nächste Seite), welches nochmal textuell ergänzend erklärt ist.

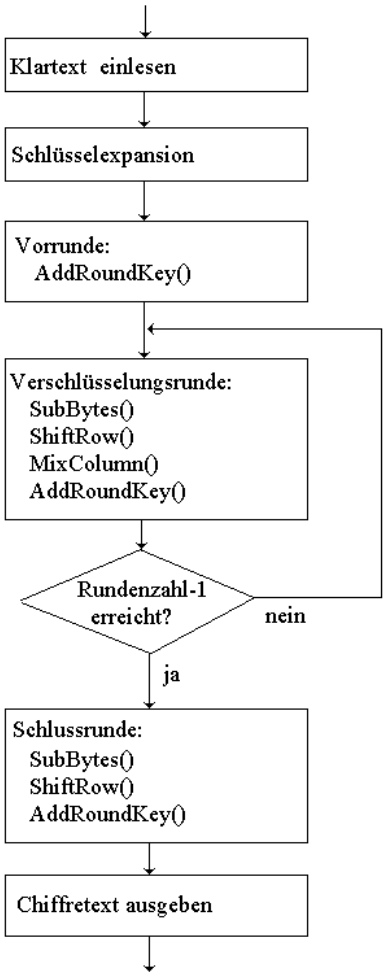


Abbildung 2.1: Schema Verschlüsselung mit AES - `Quelle im Anhang`

2.6.1 AblaufVerVorrunde (Label)

Hier geschieht lediglich die Verrechnung mit dem Rundenschlüssel nach der ersten Schlüsselexpansion.

2.6.2 AblaufVer (Label)

Dies sind die Hauptrunden der Verschlüsselung. Die Anzahl der Runden werden in der *Rundenanzahl* definiert. Durch den Beginn des Vergleichswertes in CL (Register) mit 1 wird auch tatsächlich die Anzahl der gewünschten Runden durchlaufen. Die Schlussrunde läuft anders ab, wird aber als (1) Runde mitgezählt.
In der Reihenfolge werden hier die einzelnen Werte erst substituiert (Prinzip siehe `SBOX.asm`), dann die Reihen rotiert (`ShiftRow`) mit anschließendem `ShiftColumn`. Zuletzt findet eine Schlüsselexpansion statt, damit der Rundenschlüssel erzeugt und nachfolgend auf den *TastaturBuffer* gerechnet wird.

2.6.3 AblaufVerEndrunde (Label)

Diese letzte Runde findet ähnlich den Hauptrunden statt, jedoch wird hier die Verrechnung der Spalten per `ShiftColumn` weggelassen.
Die nun erwähnte Ausgabe wird wiederrum in der `Ver_Code.asm` geregelt.

2.7 Ablauf_E.asm

In dieser ASM-Datei wird der allgemeine Ablauf geregelt, der bei einer Entschlüsselung stattfindet. Die Umsetzung in den `Labels` erfolgte nach dem Muster auf der nächsten Seite, welches nochmal textuell ergänzend erläutert ist.
AES ist ein symmetrisches Verfahren. Somit ist erkennbar, dass die Entschlüsselung wie die inverse Verschlüsslung stattfindet.

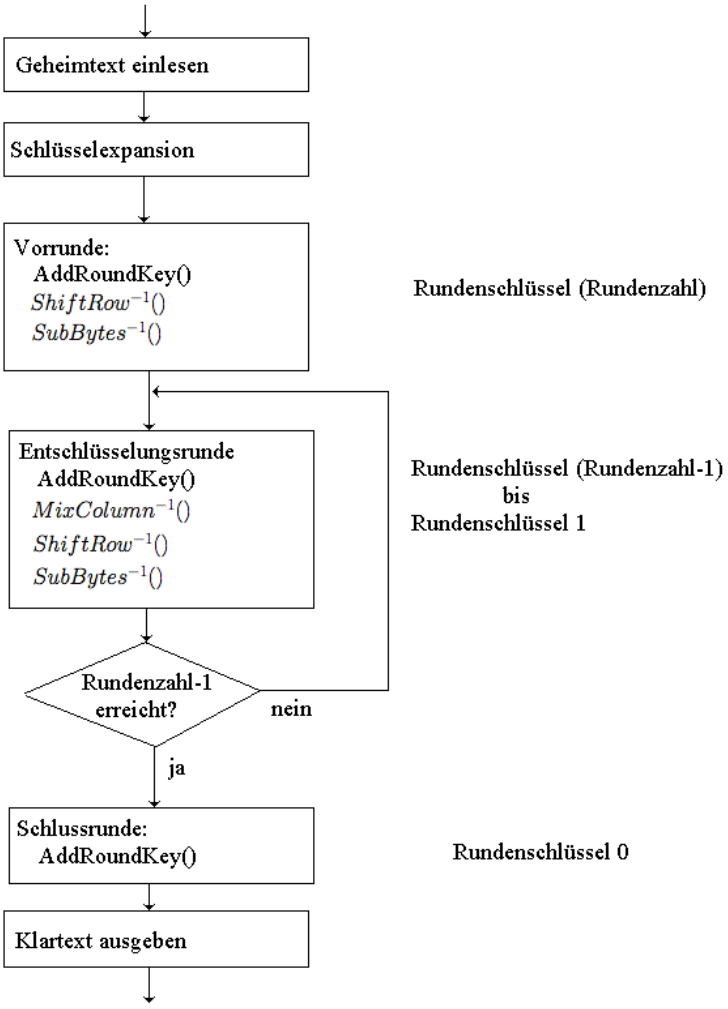


Abbildung 2.2: Schema Verschlüsselung mit AES - `Quelle im Anhang`

2.7.1 AblaufEntVorrunde (Label)

In dieser Vorrunde findet nach Schlüsselexpansion die Verrechnung mit diesem Rundenschlüssel, dann ein inverser ShiftRow (siehe `ShiftRow_1`) und schließlich die inverse Substitution mit der invertierten `SBOX` statt.

2.7.2 AblaufEnt (Label)

Hier findet solange die Hauptrunde der Entschlüsselung statt, wie der Rundzähler aus CL nicht gleich der *Rundenanzahl* ist. Damit ist gemeint, dass zunächst der Rundenschlüssel expandiert und aufgerechnet wird, dann die inverse Spaltenrechnung stattfindet, anschließend die inverse Zeilenmischung sowie die abschließende Substitution mit der inversen `SBOX`.

2.7.3 AblaufEntEndrunde (Label)

In dieser Schlussrunde wird, dank dem symmetrischen Verfahren, lediglich der Rundeschlüssel auf den *TastaturBuffer* gerechnet. Dies entspricht der Vorrunde des Verschlüsselungsablaufes.

2.8 VERS.asm

2.8.1 SchluesselInitialisieren (PROC)

Die Funtkion dieser `PROC` ist einfach, da der gesamte, fest in der `config.asm` implementierte, Standardschlüssel in den *Rundenschluessel_Expand* kopiert wird. Somit bleibt der *Rundenschluessel_Expand* bei einer nicht erfolgten Eingabe durch den Benutzer nicht leer.
Ebenfalls wird die *Rundenkonstante* initialisiert mit dem Wert $5F_{hex}$.
HINWEIS: Bei der manuellen Eingabe eines Schlüssels oder Teilschlüssels, wird der jeweilige Bereich, wie weit die Eingabe reicht, überschrieben und somit der Benutzerwunschschlüssel verwendet.

2.8.2 SchluesselAnwenden (PROC)

Mit dieser `PROC` wird das Anwenden des Rundenschlüssels auf das Array im *TastaturBuffer* beschrieben.

2.8.3 Schluesselexpansion (PROC)

Die Rundenschlüsselexpansion findet nach folgendem Muster statt:

$$k_{Runde+1,\,0} = k_{Runde,\,0} \oplus substituiert(k_{Runde,\,63}) \oplus Rundenkonstante$$

$$k_{Runde+1,\,j} = k_{Runde,\,j} \oplus k_{Runde,\,j-1}$$

Runde => Vorrunde (bei der 1. also Initialschlüssel)

j => Index des Schlüssels (zw. 0 und 63)

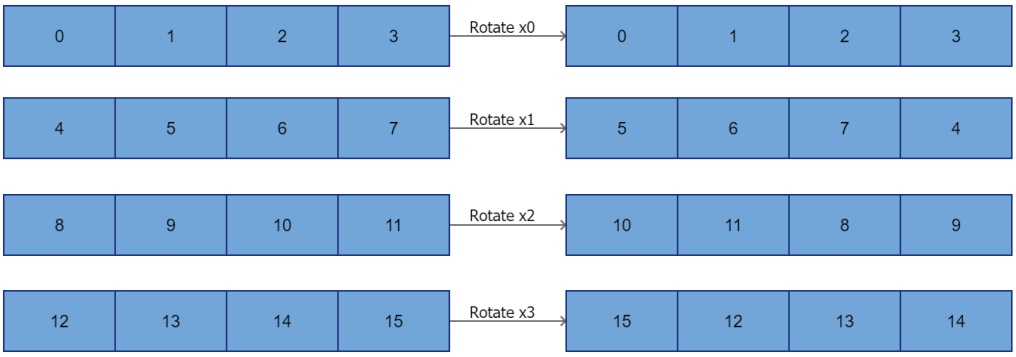
Jedoch bevor das Programm intern die 63 Schlüssel nach dem ersten Schlüssel durchläuft, wird noch die *Rundenkonstante* über ein `ROL` (Rotate Left) ebenfalls expandiert.

2.8.4 SBOXSubstitution (PROC)

In dieser `PROC` wird der gesamte *TastaturBuffer* mithilfe der `SBOX` substituiert. Wie die Substitution an sich abläuft, ist im Abschnitt 2.10 `SBOX.asm` beschrieben.

2.8.5 ShiftRow (PROC)

Beim Aufruf dieser `PROC` wird mit dem zeilenweisen Rotieren der Inhalte des *TastaturBuffers* begonnen. Dabei hat der Name hier Programm, da nach folgendem Muster das Durchrotieren stattfindet:

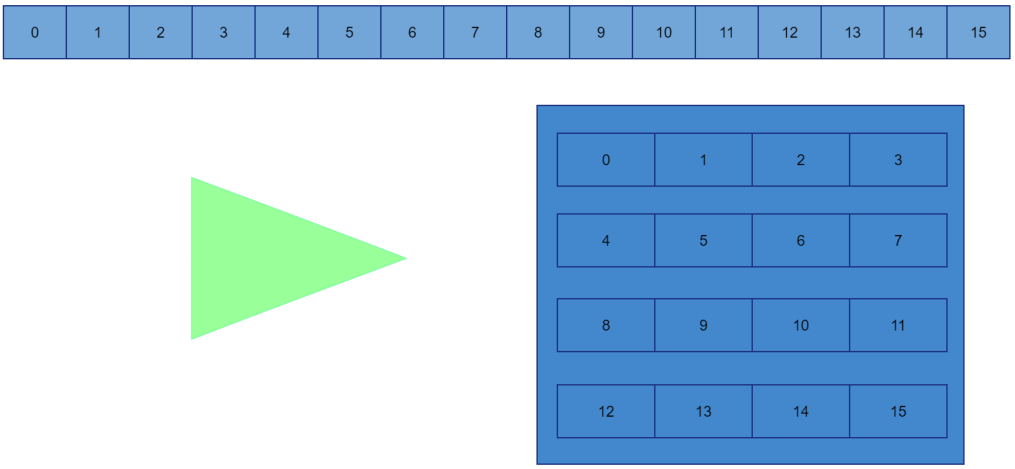


Wie zu erkennen, wird die erste Zeile des Datensatzes nicht veränadert. Der Block aus 16 Einheiten wird in vier gedachte Zeilen zerlegt. Die zweite Zeile dagegen bekommt

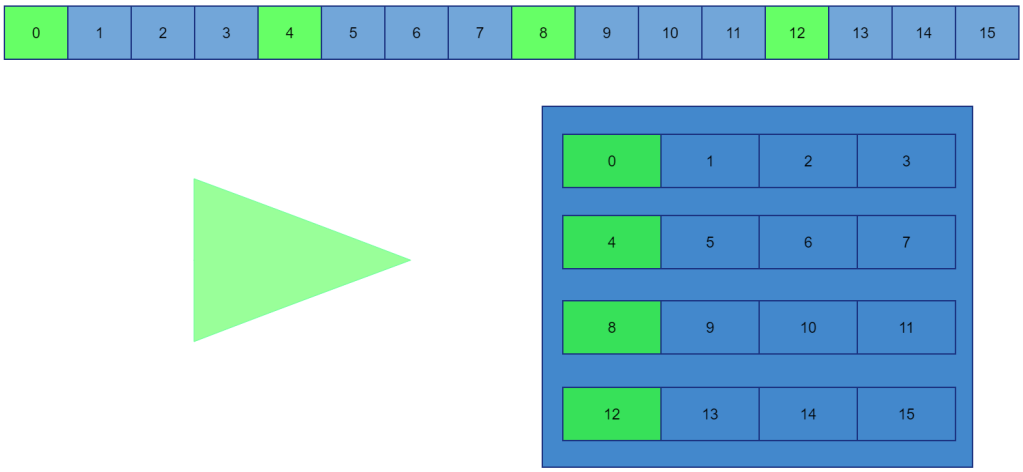
eine Rotation einmal nach Links. Somit steht die Stelle **7** nun an der ehemaligen Stelle **4**. Dadurch rückt diese (4) an die Stelle 5 und so weiter.
Zeile 3 wird um 2 Stellen und die 4. Zeile um 3 Stellen rotiert. Dieses Muster wird 4x durchlaufen, da unser 64 Zeichen langer Text aus dem *TastaturBuffer* in 4 Blöcke a 16 Einträge perfekt geteilt werden kann.

2.8.6 ShiftColumn (PROC)

Diese `PROC` setzt das Spaltenrechnen von AES um. Dabei wird mithilfe der C-Matrix eine Verrechnung der einzelnen Spalten vorgenommen, da man sich hier wieder die Blöcke aus 16 Einträgen vorstellen muss. Nachfolgende Grafik soll dies zeigen:



Bei einem Durchlauf einer Spalte werden nun die Werte ”0”, ”4”, ”8”, ”12” mit der C-Matrix nach den mathematischen Gesetzen verrechnet.



Somit werden mit einer Rechnung pro C-Matrix 4 Einträge manipuliert. Für den gesamten *TastaturBuffer* muss also die Mutliplikation mit der C-Matrix insgesamt 16x angewendet werden.

2.8.7 MalZweiProc (PROC)

Hier findet eine Fallunterscheidung statt, welche den Wert als größer 127 oder kleiner gleich 127 erkennt. Sollte er kleiner sein, so wird er durch ein SHL (shift left = schiebe nach links) mit 2 Multipliziert. Andernfalls wird eine `XOR`-Operation mit 283_{dez} durchgeführt und anschließend mal 2 gerechnet. Ein Aufruf dieser `PROC` erfolgt über die `ShiftColumn` an mehreren Stellen.

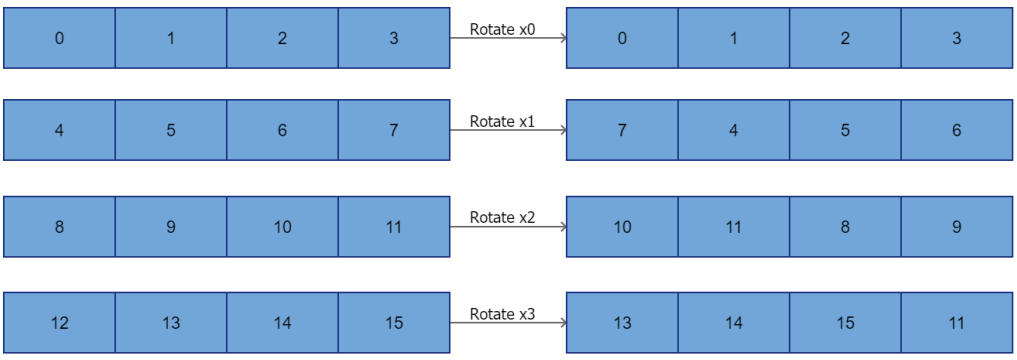
2.9 ENTS.asm

2.9.1 SBOXSubstitution_1 (PROC)

In dieser `PROC` wird der gesamte *Tastaturbuffer* mithilfe der invertierten `SBOX` substituiert. Wie die Substitution an sich abläuft, ist im Abschnitt 2.10 SBOX.asm beschrieben.

2.9.2 ShiftRow_1 (PROC)

Beim Aufruf dieser `PROC` wird mit dem zeilenweisen Rotieren der Inhalte begonnen. Dieses Rotieren findet nach folgendem Muster statt:



Aus der Grafik ist erkennbar, dass die erste Zeile keine Rotation zu verzeichnen hat. Die Zweite dagegen hat einen ShiftRight (schiebe nach rechts) mit Sicherung des Inhaltes 7 und Einschub von links an die Stelle 4.
Bei Zeile 3 wird eine 2fache Rotation durchgeführt. Somit stehen alle Einträge zwei Plätze weiter rechts. Auch hier werden die raus fallenden Inhalte von links wieder eingefügt, wie es bei einer Rotation definiert ist.
Bei der letzten Zeile findet eine Verschiebung um 3 statt, damit jedes Verschiebemuster in den 4 Zeilen einmal stattgefunden hat.

2.9.3 ShiftColumn_1 (PROC)

Leider nicht funktionsfähig...
Hier soll die Verrechnung mit der inversen C-Matrix stattfinden:

$$\begin{pmatrix} E & B & D & 9 \\ 9 & E & B & D \\ D & 9 & E & B \\ B & D & 9 & E \end{pmatrix}$$

Wie schon bei der nicht inversen Variante wird eine einzelne (gedachte) Spalte mit einer Zeile der Matrix verrechnet. Die einzelnen Einträge werden multipliziert und diese dann zum Schluss durch ein `XOR` logisch verknüpft.

2.10 SBOX.asm

In dieser asm-Datei werden zwei elementare `Array`s` angelegt. Das Erste beinhaltet die `SBOX`, welche zum Verschlüsseln benötigt wird, während das Zweite schlicht die inverse SBOX enthält (zum Entschlüsseln).
Das Grundprinzip in der Anwendung einer SBOX besteht daraus, dass mithilfe des zu wechselnden Wertes die Stelle im Array ermittelt wird, wo der neue (substituierte) Wert dann herausgenommen wird.

Als Beispiel wäre die Eingabe 36_{hex} gegeben. Dies entspricht dem dezimal Wert 54_{dez} und verweist somit auf die 54. Stelle im Array, also dem Index 53_{dez} des Array's. An dieser Stelle befindet sich der vordefinierte Wert 102_{dez} - was 66_{hex} entspricht. Bei der Substitution wird nun dieser Wert (66_{hex}) mit dem vorherigen Wert (36_{hex}) ersetzt.
Bei der inversen SBOX erfolgt dieses Ersetzten der Werte genau nach dem selben Schema.

2.11 config.asm

In dieser Datei finden sich alle Variablen. Aufgelistet wären das:

Name	*Datentyp*	*Beschreibung*
Rundenanzahl	DB	Beinhaltet die Anzahl der Runden. Damit kann man leichter dann aus z.B. 10 Runden 5 machen.
bool_Ent_Ver	DB	Beinhaltet die Variable, die genutzt wird, um festzuhalten, ob Entschluesseln, Verschluesseln oder keine der Schaltflächen gedrückt wurde.
ESC_Code	DB	1Bh => Tastaturcode von der ESC-Taste. Für bessere Lesbarkeit im Code.
Entertaste_Code	DB	0Dh => Tastaturcode von der Enter-Taste. Für bessere Lesbarkeit im Code.
Video_Seg	DW	Adresse vom Videosegment. Damit der Videomodus3 auch alles korrekt printed.
oldIOFF	DW	Offset der ehemaligen Interrupt-Service-Routine.
oldISeg	DW	Segmentadresse der ehemaligen Interrupt-Service-Routine.
Hauptbildschirm	DB	Beinhaltet den String, der das Hauptmenü beinhaltet. (siehe `config.asm`)

Name	*Datentyp*	*Beschreibung*
Hauptbildschirm2	DB	Beinhaltet den String, der den Bildschirm für die Eingabe zum Entschlüsseln beinhaltet (siehe `config.asm`)
Hauptbildschirm3	DB	Beinhaltet den String, der den Bildschirm für die Eingabe zum Verschlüsseln beinhaltet beinhaltet. (siehe `config.asm`)
Hauptbildschirm4_ Ergebnis	DB	Beinhaltet den String, der den Bildschirm für die Ausgabe des Entschlüsseln-Ergebnis beinhaltet beinhaltet. (siehe `config.asm`)
Hauptbildschirm5_ Ergebnis	DB	Beinhaltet den String, der den Bildschirm für die Ausgabe des Verschlüsseln-Ergebnis beinhaltet beinhaltet. (siehe `config.asm`)
BildschirmVar	DB	Hier wird eingetragen, welcher der 4 (nicht Haupt-) Bildschirme zu zeichnen sind. Diese Variable wird von der `BildschirmProcX` verwendet
EntR1_L	DW	=499 Beinhaltet die Blocknummer (*2) der oberen Zeile vom ENTSCHLUESSELN (Links)
EntR1_R	DW	=535 Beinhaltet die Blocknummer (*2) der oberen Zeile vom ENTSCHLUESSELN (Rechts)
EntR2_L	DW	=659 Beinhaltet die Blocknummer (*2) der mittleren Zeile vom ENTSCHLUESSELN (Links)
EntR2_R	DW	=695 Beinhaltet die Blocknummer (*2) der mittleren Zeile vom ENTSCHLUESSELN (Rechts)
EntR3_L	DW	=819 Beinhaltet die Blocknummer (*2) der unteren Zeile vom ENTSCHLUESSELN (Links)
EntR3_R	DW	=855 Beinhaltet die Blocknummer (*2) der unteren Zeile vom ENTSCHLUESSELN (Rechts)

Name	*Datentyp*	*Beschreibung*
VerR1_L	DW	=581 Beinhaltet die Blocknummer (*2) der oberen Zeile vom VERSCHLUESSELN (Links)
VerR1_R	DW	=617 Beinhaltet die Blocknummer (*2) der oberen Zeile vom VERSCHLUESSELN (Rechts)
VerR2_L	DW	=741 Beinhaltet die Blocknummer (*2) der mittleren Zeile vom VERSCHLUESSELN (Links)
VerR2_R	DW	=777 Beinhaltet die Blocknummer (*2) der mittleren Zeile vom VERSCHLUESSELN (Rechts)
VerR3_L	DW	=901 Beinhaltet die Blocknummer (*2) der unteren Zeile vom VERSCHLUESSELN (Links)
VerR3_R	DW	=937 Beinhaltet die Blocknummer (*2) der unteren Zeile vom VERSCHLUESSELN (Rechts)
TastaturBuffer	DB	Hier werden die Hex-Werte der Zeichen gespeichert, welche per Tastatureingabe erkannt wurden
TastaturBufferOrg	DB	Dient als Sicherung der Eingabe, um die Ausgabe des Originaltextes zu gewährleisten.
Konstante	DB	$=5F_{hex}$ | Ist die Vordefinierte Initialkonstante.
Rundenkonstante	DB	Ermittelte sich aus der durch `ROL` veränderten vorherigen Rundenkonstante.
Rundenschluessel_0	DB	Ist der fest implementierte Standardschlüssel, welcher während der Laufzeit auch nicht überschrieben werden kann.
Rundenschluessel_ Expand	DB	Hier wird der Rundenschluessel_0 reinkopiert. Bei einer Benutzereingabe eines eigenen Schlüssels, werden nur hier die entsprechenden Einträge verändert. Ebenfalls die erweiterten Schlüssel (siehe `Schluesselexpansion`) werden hier abgelegt.

Name	*Datentyp*	*Beschreibung*
ShiftZwSpeicher	DB	Wird benötigt für die Arbeit der **`ShiftRow`** und seinem Pendant **`ShiftRow_1`**.
ShiftBXZwSpeicher	DW	Da ein Index-Register nicht mit einem Halb-Register (H oder L) verglicher werden kann, wird das BX Register hier genutzt, was durch den internen Ablauf jedoch an einigen Stellen zwischengespeichert werden muss.
ShiftColumnZW	DW	Sichert die Zwischenergebnisse aus der ShiftColumn & ShiftColumn_1.
ShiftColumnZW2	DW	Sichert die Zwischenergebnisse aus der **`Shift-Column`** fuer die Spaltenrechnung, sonst gäbe es Rechenfehler da Werte überschrieben werden.


Hexwerte eingeben (nicht durch Leerzeichen trennen):
Schluessel (hexadezimal)
Enter betaetigen, sobald der Text vollstaendig ist
Nach der Schluesseleingabe (optional) erneut Enter betaetigen


Klartext eingeben:
Schluessel (hexadezimal)
Enter betaetigen, sobald der Text vollstaendig ist
Nach der Schluesseleingabe (optional) erneut Enter betaetigen
Eingabe Klartext nur ueber eine Zeile!!!


Klartext:
ESC betaetigen Bitte, damit eine Rueckkehr zum
Startbildschrim erfolgt.


Klartext eingegeben:
Ergebnis der Verschluesselung (hexadezimal)
ESC betaetigen Bitte, damit eine Rueckkehr zum
Startbildschrim erfolgt.

2.12 ISR.asm

Diese gesamte Datei beinhaltet einen rein zur Demonstration erstellten Codesatz, um zu beweisen, dass uns bekannt ist, wie die Erstellung und Einschreibung einer eigenen Interrupt-Service-Routine (kurz: `ISR`) ablaufen muss.

2.12.1 ISR1Ch (Label)

Hier wird beschrieben, was in der eigens erstellten ISR ablaufen soll. Unspektakulärer Weise ist dies bei diesem Programm ein `NOP`-Befehl.

2.12.2 EigeneISR (PROC)

Mit dieser `PROC` wird das Eintragen der selbst erstellten ISR durchgeführt. Näherers dazu ist im Quelltext-Kommentar zu lesen.

2.12.3 ISR_Zuruecksetzten (PROC)

Beim Aufruf dieser `PROC` erfolgt das Zurücksetzten der ISR-1Ch, damit das Programm beendet werden kann.

2.13 BildPROC.asm

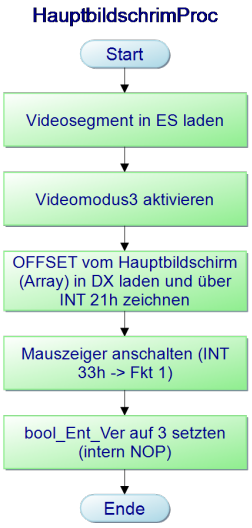
Diese zwei `PROC`‘s haben die Funktionalität, des Zeichnens der Bildschirme, welche in der `config.asm` definiert sind, auszuführen.

2.13.1 HauptbildschirmProc (PROC)

Sobald der `CALL` dieser `PROC` erfolgt, wird das Videosegment in ES geladen, damit für den Videomodus3 festgelegt wird, wo er zu zeichnen hat.
Somit wir der *Hauptbildschirm* dargestellt (näheres in den Kommentaren der Befehlszeilen).
Abschließend wird die, für die Interne Verarbeitung benötigte, Variable *bool_Ent_Ver*

auf 3 gesetzt (also nichts tun).

Daraufhin geschieht der Rücksprung zum Programm über ein `RET`.



2.13.2 BildschirmProcX (PROC)

Nach dem Aufruf dieser `PROC`, wird zunächst das Videosegment in ES geladen. Anschließend wird der Bildschirm, durch die erneute Einstellung des Videomodus3 geleert und zum abermaligen Zeichnen von Inhalten vorbereitet. Näheres dazu in den Kommentaren bei den Befehlszeilen.

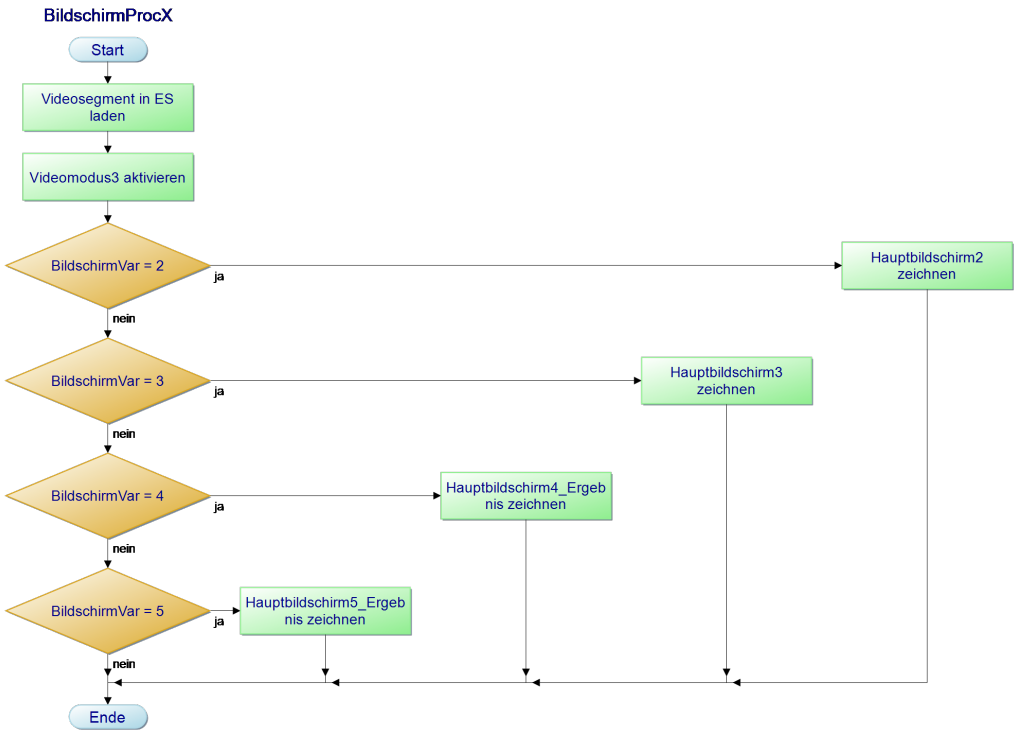
Als erstes erfolgt ein Vergleich der Variable *BildschirmVar* auf den Wert 2. Sollte dieser erfolgreich sein, so wird der *Hauptbildschirm2* gezeichnet (Entschlüsseln-Eingabe). Nach der Ausführung (graphische Darstellung des Bildschirms) springt das Programm zum Ende der `PROC` und kehrt zum vorherigen Programmabschnitt über ein `RET` zurück. Sollte der Vergleich der *BildschirmVar* mit 2 negativ ausgefallen sein, so erfolgt nun ein `CMP` auf den Inhalt mit 3. Hier wird nach dem selben Muster der Aufruf des *Hauptbildschirm3* (Verschlüsseln-Eingabe) erfolgen, mit anschließendem Sprung zum BildschirmEnde (Label).

Bei erneut fehlgeschlagenem Test, erfolgt ein Vergleich auf den Wert 4. Dieser codiert

den *Hauptbildschirm4_Ergebnis* (Verschlüsseln-Ausgabe).

Als letzter Vergleich wird die Variable *BildschirmVar* mit dem Wert 5 abgeglichen. Dieser bringt den *Hauptbildschirm5_Ergebnis* (Entschlüsseln-Ausgabe) zum Zeichnen.

Sollten durch einen Fehler im Programm oder durch externe Einflüsse die *BildschirmVar* keinen der hier 4 codierten Zustände haben, so wird in der `PROC` das Zeichnen an sich vernachlässigt. Stattdessen findet ein `JMP` zum BildschirmEnde (Label) statt. Das Unterprogramm `BildschirmProcX` führt ein `RET` durch und wird beendet.



Kapitel 3

Batch-Dateien

3.1 asmaes.bat

Diese ausführbare Datei beinhaltet folgenden Befehlssatz:

```
cd ASM
tasm aes.asm
tlink aes.obj
cd ..
```

Damit wird für den ungeübten Benutzer, aber auch für den Experten in der DOS-Box, das Kompilieren der ASM-Dateien leichter. Mit dem ersten Befehl wird eine obj- und eine map- Datei erstellt dank des TASM-Compilers. Danach wird mit dem tlink Befehl eine exe aus der obj Datei erzeugt, welche als ausführbare Datei in der DOS-Box dann gestartet werden kann.
Der Name steht für **as**se**m**blier**aes**.

3.2 tdaes.bat

Diese ausführbare Datei beinhaltet folgenden Befehlssatz:

```
cd ASM
tasm aes.asm
tlink aes.obj
```

```
td aes.asm
cd ..
```

Hier erfolgt nach dem Erstellen der exe-Datei noch der Aufruf des Turbodebuggers. Hiermit soll dem Benutzer der Aufruf des Debuggers nach Änderungen im Quellcode zum Testen vereinfacht werden.
Der Name steht hier für starte**t**urbo**d**ebuggervon**aes**.

3.3 start.bat

Diese ausführbare Datei beinhaltet folgenden Befehlssatz:

```
cd ASM
tasm aes.asm
tlink aes.obj
aes.exe
cd ..
```

Mit dieser Datei, die ebenfalls für einen Entwickler vorgesehen ist, wird nach dem Assemblieren der Start des richtigen Programms, nicht im Debugger wie in der tdaes.bat, ausgeführt. Dies soll das Testen in der Oberfläche zeitlich effektiver Gestalten, falls Änderungen oder Erweiterungen eingefügt & getestet werden. Der Name steht hier für **start**eaes.

3.4 AES.bat

Diese ausführbare Datei beinhaltet folgenden Befehlsatz:

```
cd ASM
aes.exe
cd ..
```

Mit dieser ausführbaren Datei soll für den Endbenutzer der Start in der DOS-Box erleichtert werden. Da ein ungeübter Benutzer sich im gesamten Verzeichnis und den dazugehörigen Dateien möglicherweise nur schwer zurechtfindet, ist diese Datei mit dem simplen Befehl versehen, die assemblierte exe zu starten.

Literaturverzeichnis

Bücher:

Links:

Assembler (hilfreiche Webseite)

https://stanislavs.org/helppc/idx_assembler.html

TASM (Turbo-Assembler)

https://dev-supp.de/programmierung/turbo-assembler

Joan Daemen

https://de-academic.com/dic.nsf/dewiki/695345

Vincent Rijmen

https://de-academic.com/dic.nsf/dewiki/1467056

ASCII Tabelle

https://tools.piex.at/ascii-tabelle/

Verschlüsselung

https://www.ceilers-news.de/serendipity/
775-Verfahren-der-Kryptographie,-Teil-\
6-Der-Advanced-Encryption-Standard-AES.html

Entschlüsselung

https://www.ceilers-news.de/serendipity/
777-Verfahren-der-Kryptographie,-Teil-\
7-AES-Entschluesselung-und-Sicherheit.html

Label

http://docwiki.appmethod.com/appmethod/1.16/topics/de/
Assembler-Syntax

INT 16 (Interrupt)

https://stanislavs.org/helppc/int_16.html

JMP (Jump)

https://www.i8086.de/asm/8086-88-asm-jmp.html

SBOX

https://de.wikipedia.org/wiki/S-Box

PROC - pdf Seite 272

https://www.borncity.com/web/Library/EinfASM.pdf

RET (Return)

https://www.i8086.de/asm/8086-88-asm-ret.html

ROL (Rotate Left)

https://www.i8086.de/asm/8086-88-asm-rol.html

INCLUDE

Seitenauschnitt: ”Die INCLUDE-Anweisung dient dazu, Textdateien an der Stelle der Include-Anweisung einzubinden. [...] INCLUDE wird zumeist eingesetzt, um wiederverwendbare Code-Bibliotheken zu erstellen und in die Programme einzubinden.

Syntax: INCLUDE *Dateiname* ”(abgerufen am 28.07.2021)

Quelle:

https://www.ipd.kit.edu/mitarbeiter/buchmann/microcontroller/assembler7.htm

ISR - Interrupt Service Routine

https://www.lowlevel.eu/wiki/Interrupt_Service_Routine

Liste von ISR‘s

https://stanislavs.org/helppc/int_table.html

NOP - No Operation

https://www.i8086.de/asm/8086-88-asm-nop.html

CMP - Compare (Vergleich)

https://www.i8086.de/asm/8086-88-asm-cmp.html

Array (Datentyp)

https://glossar.hs-augsburg.de/Feld_(Datentyp)

CALL

https://www.i8086.de/asm/8086-88-asm-call.html

XOR - Exklusives Oder

https://www.elektronik-kompendium.de/sites/dig/0205186.htm