UNIVERSITÄT
MANNHEIM

University of Mannheim
Faculty for Business Informatics & Business Mathematics
Theoretical Computer Science And IT Security Group

Bacherlor Thesis

# Visualization of AES as a CrypTool 2 Plugin

as part of the degree program Bachelor of Science Wirtschaftsinformatik

submitted by

Matthias Becher
Matriculation number 1375677

at June 14th, 2016

**Supervisior:** Prof. Dr. Frederik Armknecht

# Abstract

This thesis deals with the AESVisualization plugin for CrypTool 2.

The first chapter discusses the reason for this plugin and the requirements gathered.

Chapter 2 then gives an overview over related work and explains which elements have been decided to be used for this work and what had to be done differently.

Chapter 3 presents the basic functionalities and the interface of the plugin.

Chapter 4 discusses how the plugin is implemented and how it works both in regard to the interface and the logic behind it.

Chapter 5 explains how a user can obtain the plugin and how to use it. It shows how to get the whole CrypTool 2 package, how to install it and how to open the plugin. Then it explains the controls of the new plugin and how to use them.

The final chapter of the thesis gives an outlook on how the plugin might change in the future and what features will be added.

# Contents

**Abstract** . . . . . . . . . . **2**

**Acknowledgment** . . . . . . . . . . **4**

**1 Introduction** . . . . . . . . . . **5**

**2 Overview over Related Work** . . . . . . . . . . **6**
- 2.1 Rijndael Animation and Rijndael Inspektor in CrypTool 1 . . . . . 6
- 2.2 AESvisual . . . . . 7
- 2.3 Hardware Visualization of the Advanced Encryption Standard (AES) Algorithm . 8

**3 Interface** . . . . . . . . . . **9**
- 3.1 Key Expansion . . . . . 9
- 3.2 Encryption . . . . . 14

**4 Implementation of the Plugin** . . . . . . . . . . **20**
- 4.1 Interface . . . . . 20
- 4.2 Logic . . . . . 23

**5 How to** . . . . . . . . . . **27**

**6 Future Development** . . . . . . . . . . **31**

**References** . . . . . . . . . . **32**

**Eidesstattliche Erklärung / Affirmation** . . . . . . . . . . **33**

## Acknowledgment

This work was created in cooperation with the team of CrypTool 2 (CT2).

Special thanks go to Prof. Bernhard Esslinger of the University of Siegen for his challenging support. Prof. Esslinger is the overall coordinator of the CrypTool project.

Special thanks also go to Armin Krauß who gave me a head start to understand the CT2 development framework and always was an excellent expert when technical problems came up.

Both Prof. Esslinger and Mr. Krauß supported me a lot throughout the development by giving me feedback and helping with problems I had.

Finally, I would also like to thank my supervisor Prof. Dr. Frederik Armknecht for helping me with this work and giving input to improve it.

# 1 Introduction

The goal of this thesis was the development of a new plugin for CrypTool 2 (CT2). CT2 is a program described by its creators as "an open-source program offering an innovative visual programming GUI to experiment with cryptographic procedures." [1]. It is essentially an interactive learning program to help people understand cryptography and cryptanalysis. One of its key features is the visualization of various cryptographic algorithms.

The plugin discussed in this thesis is a visualization of the AES algorithm. The goal was to have a plugin that shows how the algorithm works in detail and help people understand it. The user should be able to use his own input and navigate through the plugin himself instead of just watching a presentation.

So the following feature list was created according to discussions with some lecturers and students on how a good support for learning and teaching modern cryptography could be achieved with an appropriate AES visualization:

1. The plugin should show how the algorithm works
   (a) Size of input and output
   (b) Generation of round keys
   (c) Encryption of the text using the round keys
2. Preset the input with standard values
3. Two parallel views
   (a) Overview and detailed view
   (b) Navigation via the overview
4. New random input generated by pressing a button
5. Option for a user to use his own input
6. Option to change single values and visualize the effect this has
7. Extended options for navigation
   (a) Go backwards
   (b) Proceed in single steps or whole sections
8. A help function
   (a) How does the algorithm work
   (b) How can the user deal with the plugin
9. Second language

# 2 Overview over Related Work

In this chapter an overview is given over related work. I will also explain and argue which elements I liked and wanted to have in my plugin as well and what I wanted to do differently. I only found three relatively mature visualizations for AES: the triplet in CT1, AESvisual, and the HADES implementation.

## 2.1 Rijndael Animation and Rijndael Inspektor in CrypTool 1

There was already a visualization of AES in CrypTool 1 (CT1) [2], written by E. Zabala. Figures 1a to 1c show the "Rijndael Animation". Figure 1d shows the "Rijndael Inspektor". Both are Flash programs. I liked some of the elements and decided to use them in my plugin as well. As seen in Figure 1a, the animation gives an overview over the encryption process. The row of numbers at the bottom represent the slides of the animation, and a user can navigate by clicking on these numbers. I wanted to have both these features. However, the navigation should be combined with the overview that shows which operation in which round of encryption is currently processed.

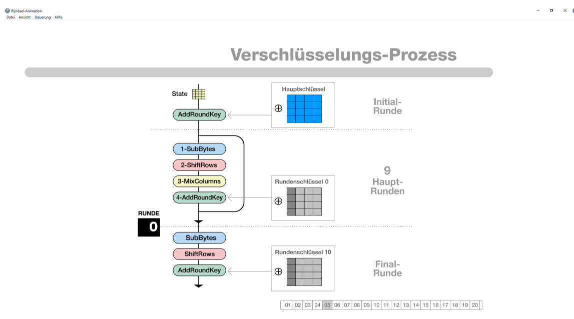
"SubByte" 1b made it easy to understand this part of the algorithm: picking one byte and then highlighting the corresponding row, column and the result. I decided to do it in a similar fashion. As seen in 1c the animation had some short explanatory texts which I also wanted in my plugin.

However, some aspects needed to be done differently. First of all I wanted to make it possible for the user to use his own key and text as input. Secondly, my plugin should visualize all the steps.

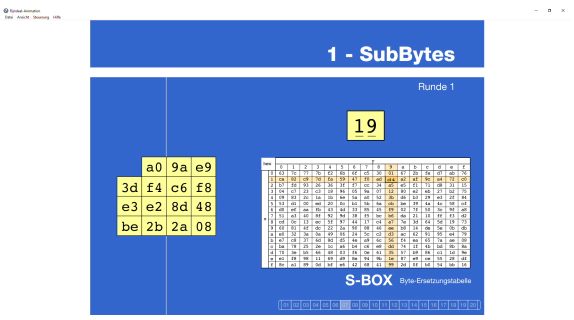
The "Rijndael Inspektor" 1d uses user input and shows all states. But it does not visualize the operations.

Another decision I made was to put the key expansion as the first operation of the plugin.

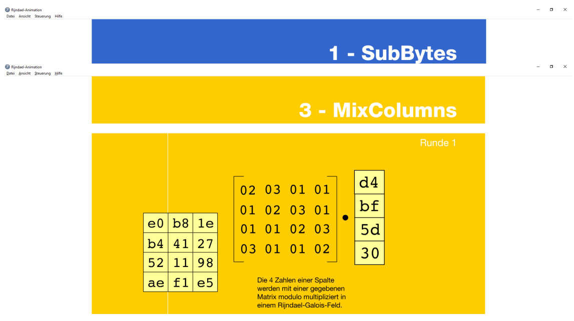
Technically, Flash has no future, so these two CT1 applications are at the end of their life.



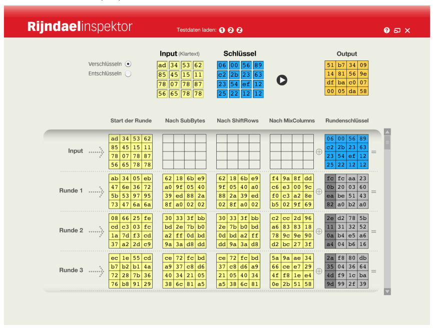
(a) An overview over AES



(b) The SubBytes step



(c) The MixColumns step



(d) The Rijndael Inspektor in CrypTool 1

Figure 1: The AES visualization in CrypTool 1 [2]

## 2.2 AESvisual

The next project I looked at was AESvisual [3]. After starting AESvisual (see Figure 2a) I noticed it had a demo mode and a practice mode. A practice mode was not planned for my plugin because I believed visualizing the algorithm would do enough to help to understand AES. This was confirmed in the paper published about AESvisual: After testing the program ”some students believed that the Demo mode would be sufficient and they did not use the Practice mode.” [4]

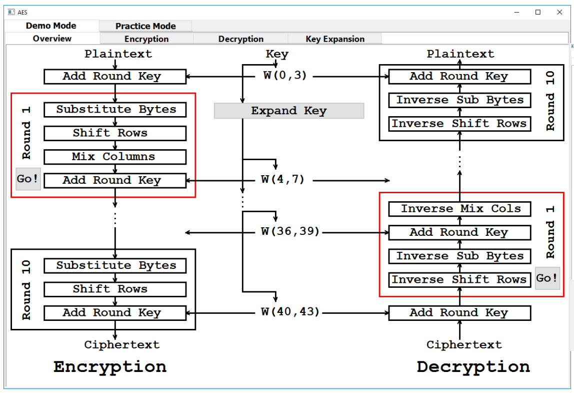
Another problem AESvisual has is that it does not scale when set to full screen (Figure 2b).

As seen in Figure 2c the program has a button to generate new random input, a feature my finished plugin will also have. AESvisual does however not support user input. This and the lack of scaling window size were also mentioned in the paper as missing features. ”Based on the student comments, the most needed extensions are (1) resizable windows, (2) allowing the user to enter his input, ...” [4].

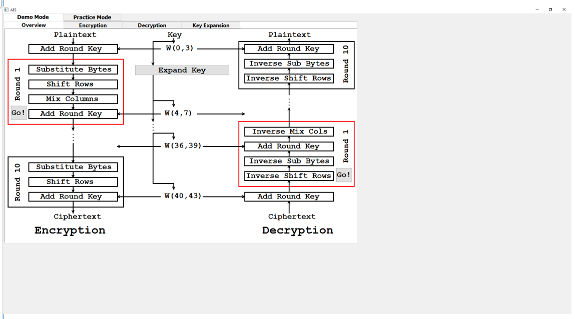
Both these features are part of the new plugin. Additionally I wanted to show all rounds of encryption and not just the first round like AESvisual does.

Another aspect I liked and wanted for my plugin are the four buttons at the top used to navigate between the operations of a round not only forward but also backward.

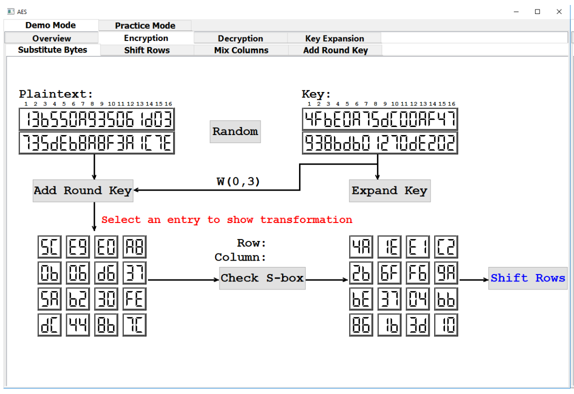
One more thing in AESvisual that needed to be done differently was the fact the S-Box was not on screen but instead popped up in a separate window (Figure 2d) when pressing the button.The goal was to show all elements of one operation in one window and have everything in there.



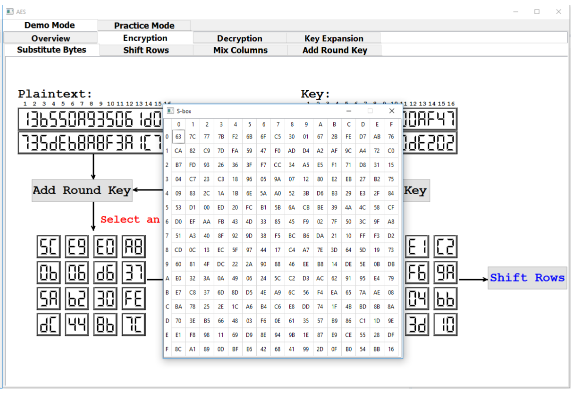
(a) The AESvisual program after startup



(b) AES visual when set to full screen



(c) The SubBytes step in AESvisual

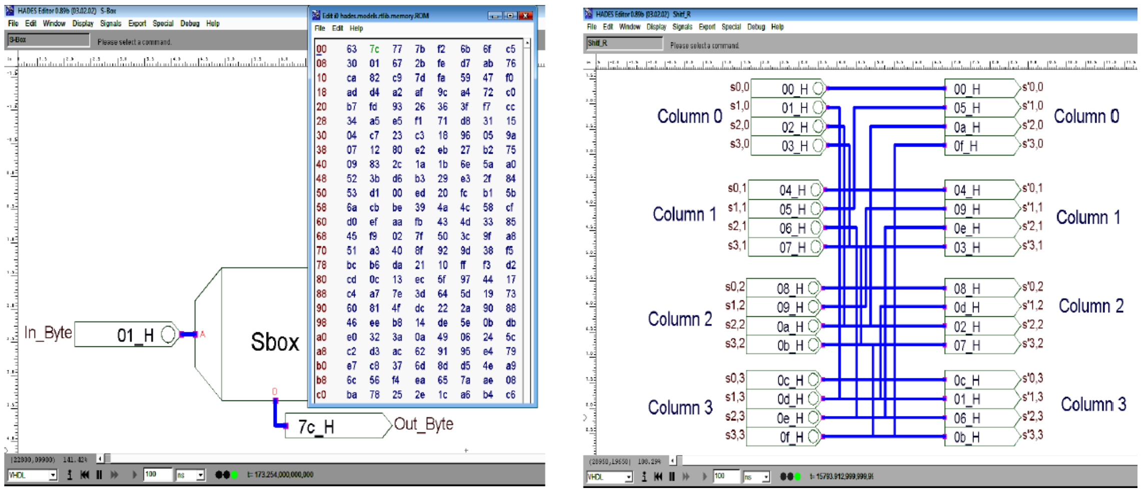


(d) AESvisual with the opened S-Box

Figure 2: The AESvisual program [3]

## 2.3 Hardware Visualization of the Advanced Encryption Standard (AES) Algorithm

The third and last project I looked at more closely was a paper [5] about hardware visualization of AES using HADES, the Hamburg Design System.
As indicated in the paper's title and seen in Figure 3 the visualization was hardware focused and very different from what I wanted in my plugin. Therefore I did not use any ideas I got from the paper for my plugin. I thought about using pipes to show the way of the data. However, that would make the screen too crowded and would not help much in terms of understanding AES.



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 00 | 63 | 7c | 77 | 7b | f2 | 6b | 6f | c5 |
| 08 | 30 | 01 | 67 | 2b | fe | d7 | ab | 76 |
| 10 | ca | 82 | c9 | 7d | fa | 59 | 47 | f0 |
| 18 | ad | d4 | a2 | af | 9c | a4 | 72 | c0 |
| 20 | b7 | fd | 93 | 26 | 36 | 3f | f7 | cc |
| 28 | 34 | a5 | e5 | f1 | 71 | d8 | 31 | 15 |
| 30 | 04 | c7 | 23 | c3 | 18 | 96 | 05 | 9a |
| 38 | 07 | 12 | 80 | e2 | eb | 27 | b2 | 75 |
| 40 | 09 | 83 | 2c | 1a | 1b | 6e | 5a | a0 |
| 48 | 52 | 3b | d6 | b3 | 29 | e3 | 2f | 84 |
| 50 | 53 | d1 | 00 | ed | 20 | fc | b1 | 5b |
| 58 | 6a | cb | be | 39 | 4a | 4c | 58 | cf |
| 60 | d0 | ef | aa | fb | 43 | 4d | 33 | 85 |
| 68 | 45 | f9 | 02 | 7f | 50 | 3c | 9f | a8 |
| 70 | 51 | a3 | 40 | 8f | 92 | 9d | 38 | f5 |
| 78 | bc | b6 | da | 21 | 10 | ff | f3 | d2 |
| 80 | cd | 0c | 13 | ec | 5f | 97 | 44 | 17 |
| 88 | c4 | a7 | 7e | 3d | 64 | 5d | 19 | 73 |
| 90 | 60 | 81 | 4f | dc | 22 | 2a | 90 | 88 |
| 98 | 46 | ee | b8 | 14 | de | 5e | 0b | db |
| a0 | e0 | 32 | 3a | 0a | 49 | 06 | 24 | 5c |
| a8 | c2 | d3 | ac | 62 | 91 | 95 | e4 | 79 |
| b0 | e7 | c8 | 37 | 6d | 8d | d5 | 4e | a9 |
| b8 | 6c | 56 | f4 | ea | 65 | 7a | ae | 08 |
| c0 | ba | 78 | 25 | 2e | 1c | a6 | b4 | c6 |

Figure 3: The S-Box and ShiftRows implementation in HADES [5]

HADES, similar to Flash, is also a "dead" technology. AESvisual doesn't seem to be further developed: After getting funding once and publishing a paper no further changes have been applied. Also requests haven't been answered.

However, for my plugin I wanted to use only technologies which are wide-spread, which are state-of-the-art, and which should be supported for several more years. In addition, I wanted to make use of an open-source project which seems to have a broad, active, and long-existing team behind it. This is fulfilled with CT2, which uses C# (modern programming language) and WPF (to deal with vector graphics in order to be able to zoom all components), and has always moved to the newest Microsoft development environment (like VS 2015).

# 3 Interface

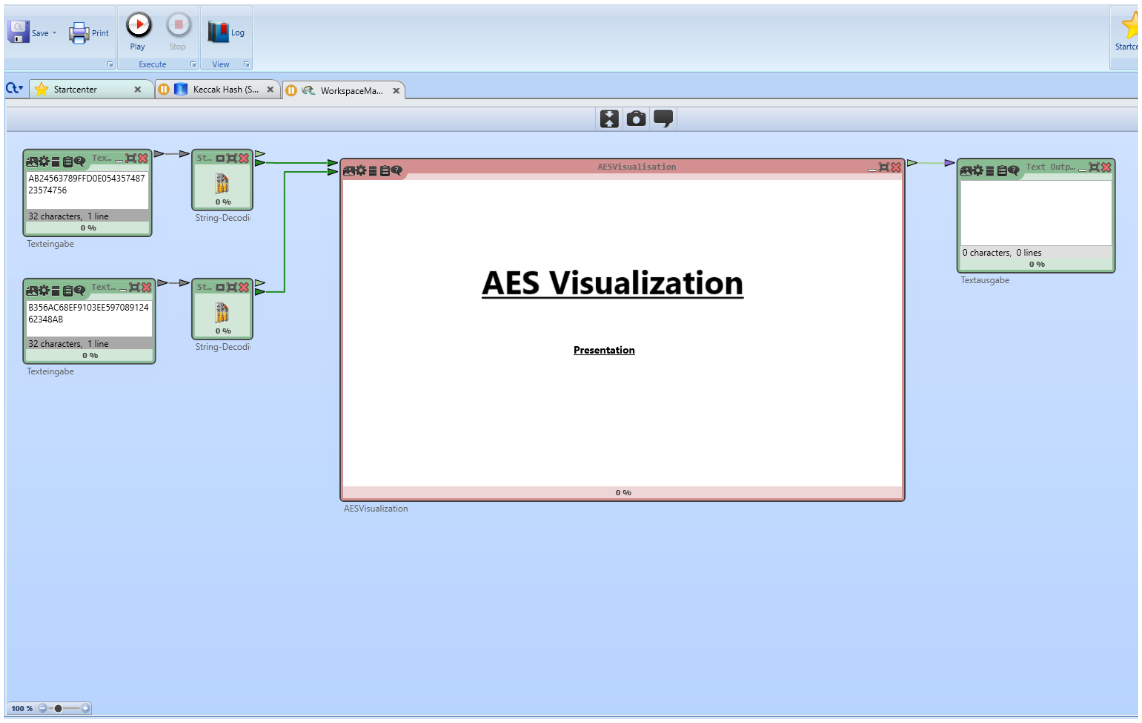
This chapter describes the interface of the plugin. I explain where the different elements are placed at and why they are placed there. In addition, I give a brief description of the functionality. A more in depth explanation of how to use the plugin is given in the chapter "How to Use".

This chapter is separated into two sections. The first one deals with the interface for key expansion and the second one with the interface for encryption.

CT2 can combine its components to create interactive workflows. Such a workflow is called "template".

Figure 4 shows a template which uses my new plugin (component). This template consists of six elements: There are two text boxes for input on the left. Connected to these are two text decoders for transforming different data types. On the right there is a component for text output.

The plugin I developed is placed in the center. Here you see it in its initial state.



AB24563789FFD0E054357487
23574756
32 characters, 1 line
0 %

B356AC68EF9103EE597089124
62348AB
32 characters, 1 line
0 %

0 characters, 0 lines
0 %

Figure 4: The template with the plugin in the maximized view

## 3.1 Key Expansion

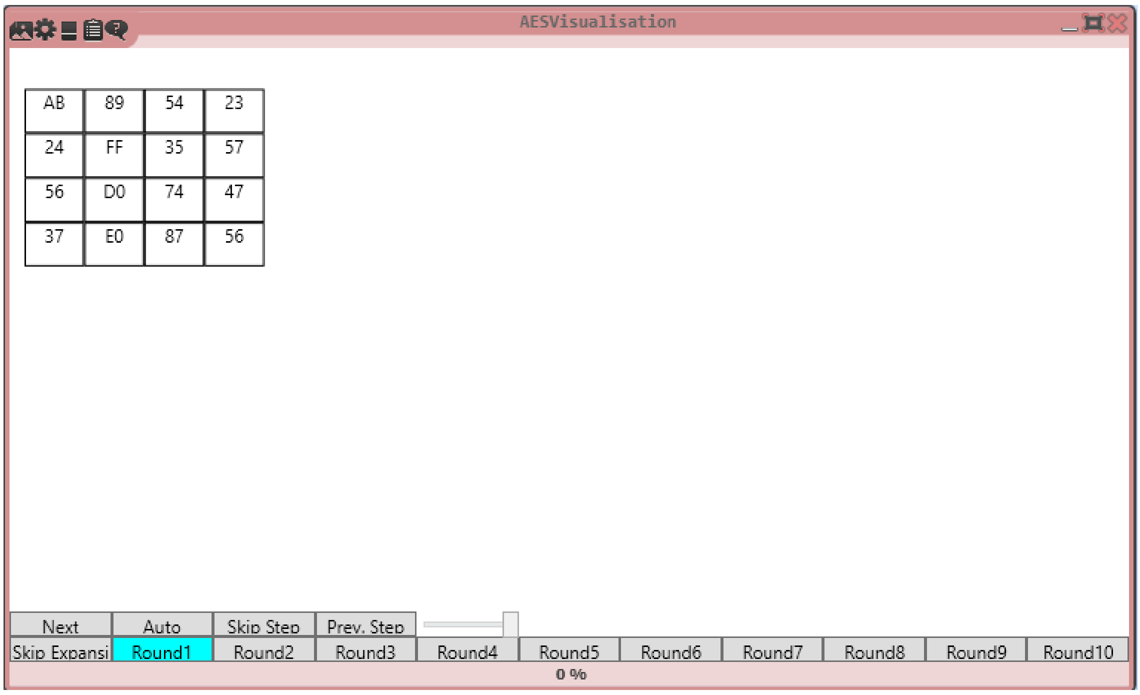
Once the execution of the plugin begins, the interface for key expansion (Figure 5) is loaded.

At the bottom are eleven buttons. The leftmost is the "Skip Expansion" button. Pressing it will bring the user directly to the encryption.

To its right are ten "Round" buttons which can be used to navigate, but also show which operation is currently active (the corresponding button is highlighted). In Figure 5, for instance, it is the second round of key expansion.

The row above the one at the very bottom consists of four buttons and one slider. The "Next" button will bring the user to the next step of the current operation. The "Auto" button enables auto step. Once auto step is activated, the plugin proceeds through all steps of the current operation without stopping. The slider controls the pause time between the individual steps. The further to the right the slider is, the shorter the pause. The ”Skip Step” button skips the next operation while the ”Prev. Step” button returns to the previous one.

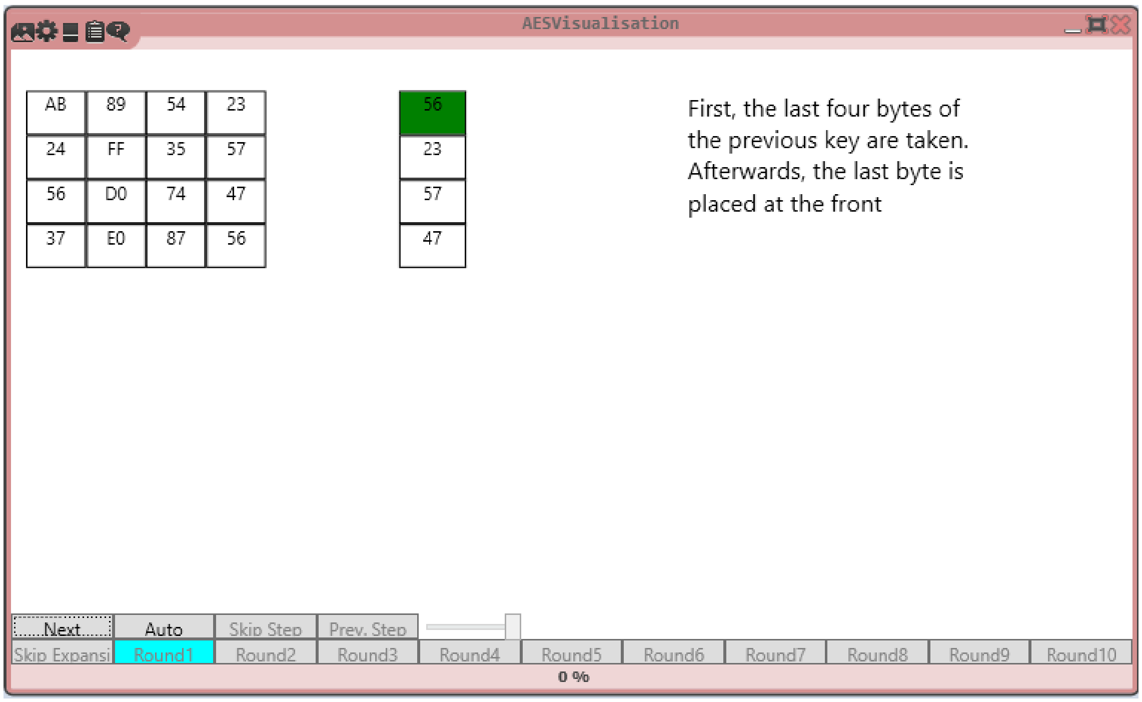
On the top left of the screenshot in Figure 5 you can see the previous key from which the next key will be derived.



| | | | |
|---|---|---|---|
| AB | 89 | 54 | 23 |
| 24 | FF | 35 | 57 |
| 56 | D0 | 74 | 47 |
| 37 | E0 | 87 | 56 |

Figure 5: The plugin at the beginning of a key expansion round

For the first step of the key expansion (Figure 6) a new column is created into which the last four bytes are copied. To show which data is used and/or changed the plugin uses color coding. There are also short texts for each step to explain what is being done.



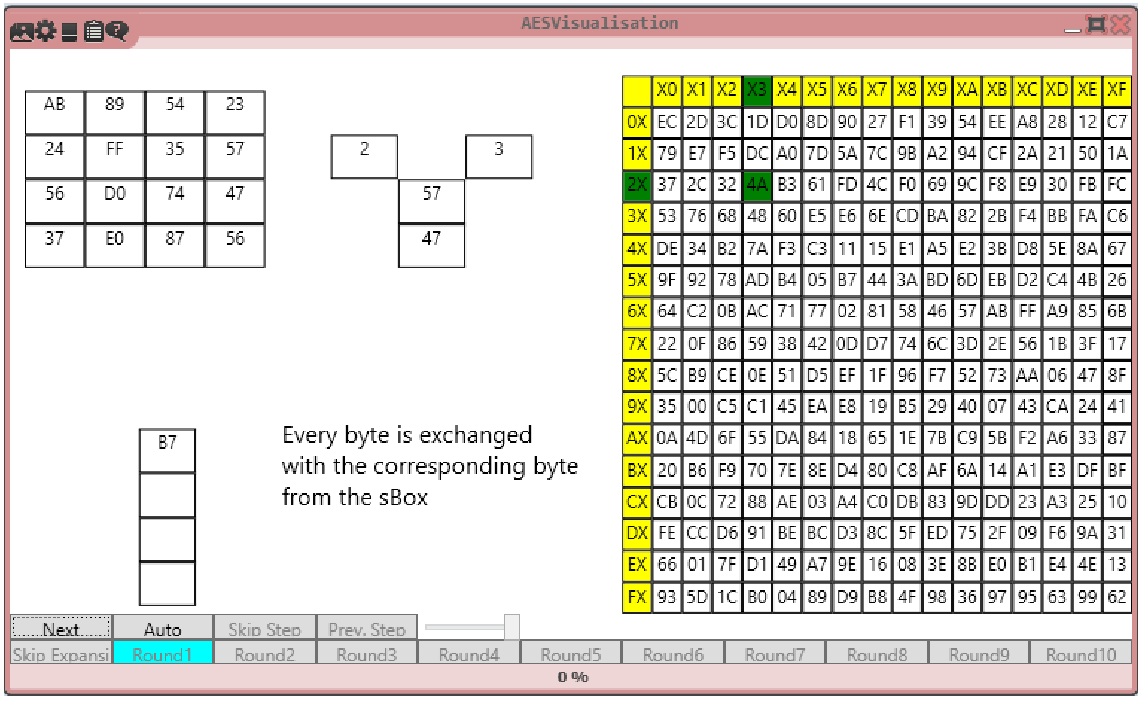
| | | | |
|---|---|---|---|
| AB | 89 | 54 | 23 |
| 24 | FF | 35 | 57 |
| 56 | D0 | 74 | 47 |
| 37 | E0 | 87 | 56 |

| |
|---|
| 56 |
| 23 |
| 57 |
| 47 |

First, the last four bytes of the previous key are taken. Afterwards, the last byte is placed at the front

Figure 6: The plugin after copying the last column and rearranging the bytes

In Figure 7 you can see the byte substitution of the key expansion. The S-Box appears on the right, therefore the explanation is moved to the center bottom. The top row and the left column of the S-Box are colored yellow to distinguish the indices from the actual values. In order to visualize this step, each byte in the transition column is split in two. Afterwards the corresponding row, column and byte of the S-Box are highlighted to show where the result is derived from. Then the result is added to the result column in the bottom left.



| | | | |
|---|---|---|---|
| AB | 89 | 54 | 23 |
| 24 | FF | 35 | 57 |
| 56 | D0 | 74 | 47 |
| 37 | E0 | 87 | 56 |

| |
|---|
| 57 |
| 47 |

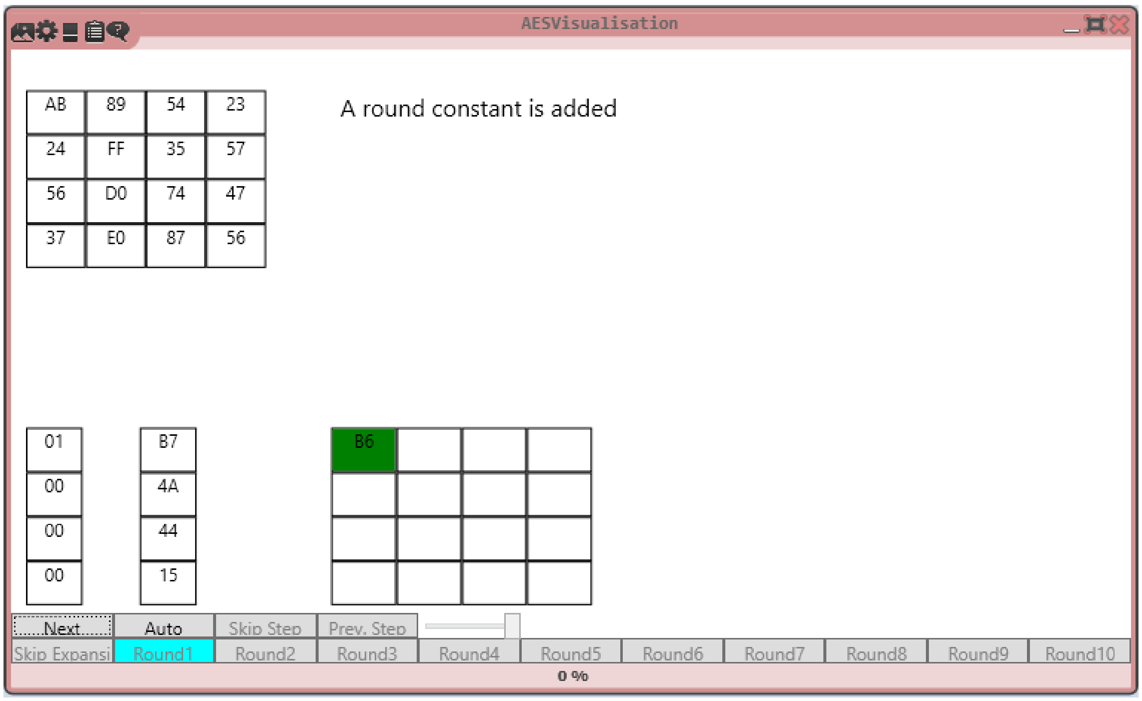
| |
|---|
| B7 |
| |
| |
| |

Every byte is exchanged with the corresponding byte from the sBox

| | X0 | X1 | X2 | X3 | X4 | X5 | X6 | X7 | X8 | X9 | XA | XB | XC | XD | XE | XF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0X | EC | 2D | 3C | 1D | D0 | 8D | 90 | 27 | F1 | 39 | 54 | EE | A8 | 28 | 12 | C7 |
| 1X | 79 | E7 | F5 | DC | A0 | 7D | 5A | 7C | 9B | A2 | 94 | CF | 2A | 21 | 50 | 1A |
| 2X | 37 | 2C | 32 | 4A | B3 | 61 | FD | 4C | F0 | 69 | 9C | F8 | E9 | 30 | FB | FC |
| 3X | 53 | 76 | 68 | 48 | 60 | E5 | E6 | 6E | CD | BA | 82 | 2B | F4 | BB | FA | C6 |
| 4X | DE | 34 | B2 | 7A | F3 | C3 | 11 | 15 | E1 | A5 | E2 | 3B | D8 | 5E | 8A | 67 |
| 5X | 9F | 92 | 78 | AD | B4 | 05 | B7 | 44 | 3A | BD | 6D | EB | D2 | C4 | 4B | 26 |
| 6X | 64 | C2 | 0B | AC | 71 | 77 | 02 | 81 | 58 | 46 | 57 | AB | FF | A9 | 85 | 6B |
| 7X | 22 | 0F | 86 | 59 | 38 | 42 | 0D | D7 | 74 | 6C | 3D | 2E | 56 | 1B | 3F | 17 |
| 8X | 5C | B9 | CE | 0E | 51 | D5 | EF | 1F | 96 | F7 | 52 | 73 | AA | 06 | 47 | 8F |
| 9X | 35 | 00 | C5 | C1 | 45 | EA | E8 | 19 | B5 | 29 | 40 | 07 | 43 | CA | 24 | 41 |
| AX | 0A | 4D | 6F | 55 | DA | 84 | 18 | 65 | 1E | 7B | C9 | 5B | F2 | A6 | 33 | 87 |
| BX | 20 | B6 | F9 | 70 | 7E | 8E | D4 | 80 | C8 | AF | 6A | 14 | A1 | E3 | DF | BF |
| CX | CB | 0C | 72 | 88 | AE | 03 | A4 | C0 | DB | 83 | 9D | DD | 23 | A3 | 25 | 10 |
| DX | FE | CC | D6 | 91 | BE | BC | D3 | 8C | 5F | ED | 75 | 2F | 09 | F6 | 9A | 31 |
| EX | 66 | 01 | 7F | D1 | 49 | A7 | 9E | 16 | 08 | 3E | 8B | E0 | B1 | E4 | 4E | 13 |
| FX | 93 | 5D | 1C | B0 | 04 | 89 | D9 | B8 | 4F | 98 | 36 | 97 | 95 | 63 | 99 | 62 |

Figure 7: The byte substitution of key expansion

For the next step (Figure 8) a new grid and a new column are added. After the round constant is added, each column from the previous key is transferred to the left column while the corresponding column from the result is transferred to the right column. The values derived from XORing both columns are entered into the result grid one after another.



A round constant is added

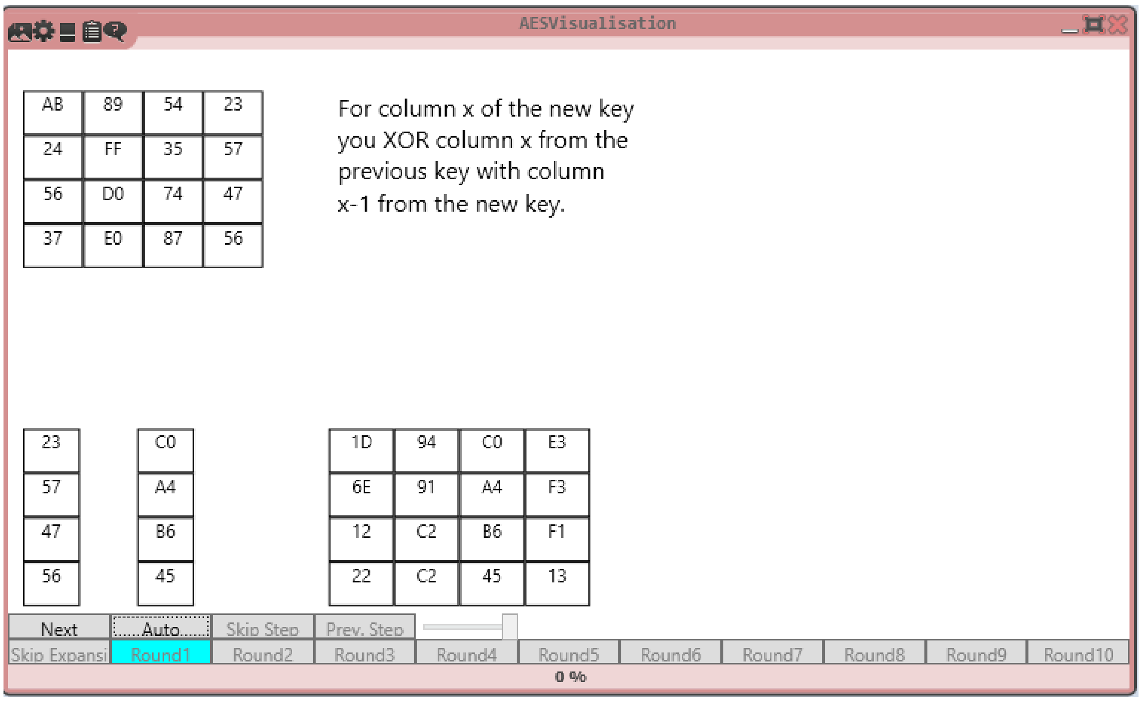
| | | | |
|---|---|---|---|
| AB | 89 | 54 | 23 |
| 24 | FF | 35 | 57 |
| 56 | D0 | 74 | 47 |
| 37 | E0 | 87 | 56 |

| | |
|---|---|
| 01 | B7 |
| 00 | 4A |
| 00 | 44 |
| 00 | 15 |

| | | | |
|---|---|---|---|
| B6 | | | |
| | | | |
| | | | |
| | | | |

Figure 8: The round constant is added after byte substitution

After all steps are completed the operation stops and shows the final result (Figure 9) in the center bottom grid.



For column x of the new key you XOR column x from the previous key with column x-1 from the new key.

| | | | |
|---|---|---|---|
| AB | 89 | 54 | 23 |
| 24 | FF | 35 | 57 |
| 56 | D0 | 74 | 47 |
| 37 | E0 | 87 | 56 |

| | |
|---|---|
| 23 | C0 |
| 57 | A4 |
| 47 | B6 |
| 56 | 45 |

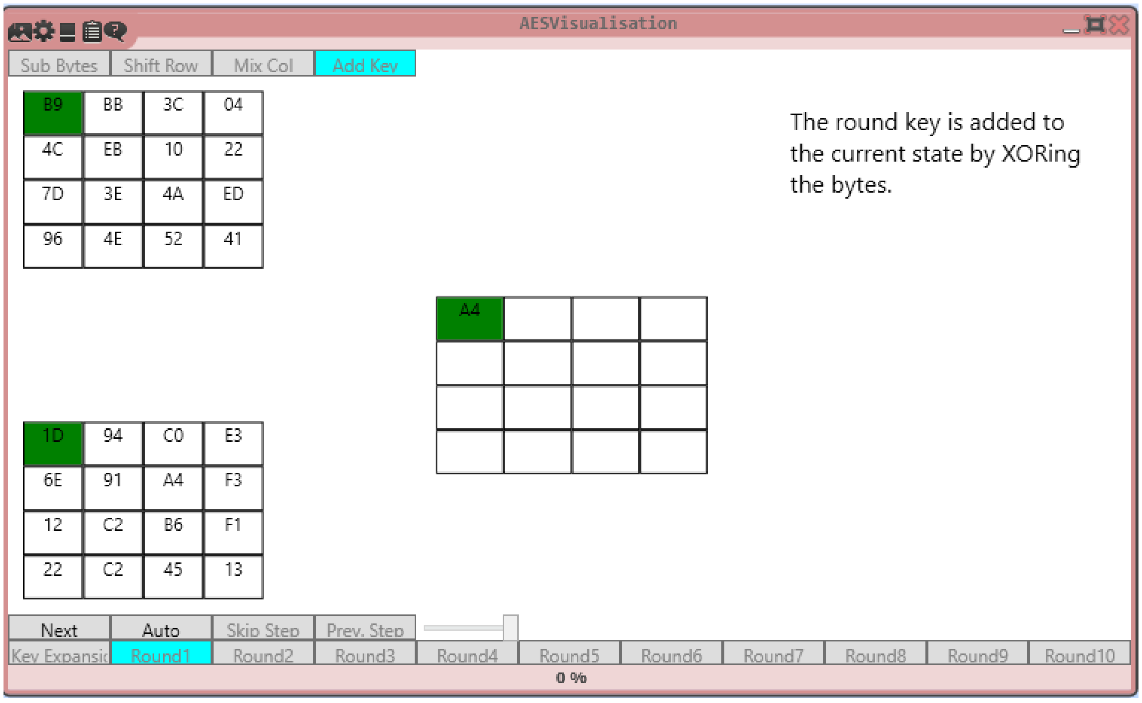
| | | | |
|---|---|---|---|
| 1D | 94 | C0 | E3 |
| 6E | 91 | A4 | F3 |
| 12 | C2 | B6 | F1 |
| 22 | C2 | 45 | 13 |

Figure 9: The plugin after completing a round of key expansion

## 3.2 Encryption

When the encryption starts, four new buttons appear in the top row. These let the user navigate through the four operations of each round and also serve as an overview. In addition, the text of the button in the bottom left corner now reads ”Key Expansion” since it is now used to go back to the key expansion.

As the encryption consists of four steps I decided to go with four interfaces. The first one depicts the ”AddKey” operation (Figure 10). There are three matrices. On the top left is the current state, bottom left the current round key (or at first the main key) and on the right is the result matrix. To visualize the operation, first one byte in the state matrix is highlighted green. Then the corresponding byte of the key matrix is highlighted green. Lastly the resulting byte appears highlighted in the result matrix. The procedure is repeated for all values and until the result matrix is filled (Figure 11). This way it is easy to see which bytes are XORed with each other and how the result is derived.



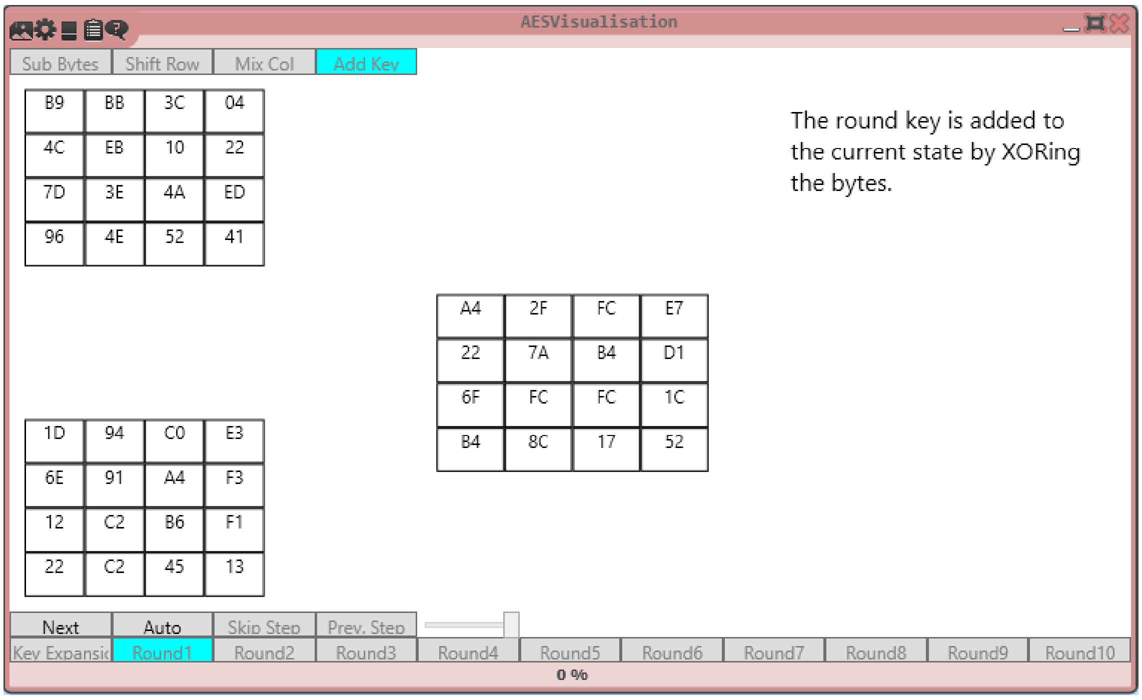
| | | | |
|---|---|---|---|
| B9 | BB | 3C | 04 |
| 4C | EB | 10 | 22 |
| 7D | 3E | 4A | ED |
| 96 | 4E | 52 | 41 |

| | | | |
|---|---|---|---|
| 1D | 94 | C0 | E3 |
| 6E | 91 | A4 | F3 |
| 12 | C2 | B6 | F1 |
| 22 | C2 | 45 | 13 |

| | | | |
|---|---|---|---|
| A4 | | | |
| | | | |
| | | | |
| | | | |

The round key is added to the current state by XORing the bytes.

Figure 10: The addKey operation after completing the first addition



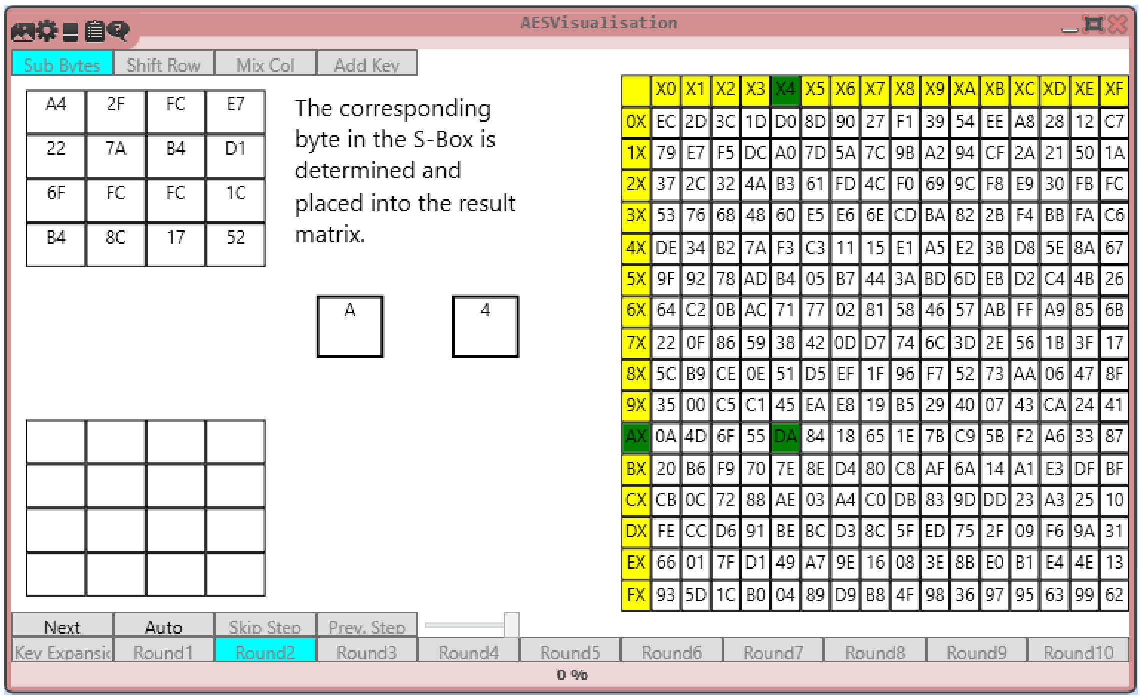
|  |  |  |  |
|---|---|---|---|
| B9 | BB | 3C | 04 |
| 4C | EB | 10 | 22 |
| 7D | 3E | 4A | ED |
| 96 | 4E | 52 | 41 |

The round key is added to the current state by XORing the bytes.

|  |  |  |  |
|---|---|---|---|
| A4 | 2F | FC | E7 |
| 22 | 7A | B4 | D1 |
| 6F | FC | FC | 1C |
| B4 | 8C | 17 | 52 |

|  |  |  |  |
|---|---|---|---|
| 1D | 94 | C0 | E3 |
| 6E | 91 | A4 | F3 |
| 12 | C2 | B6 | F1 |
| 22 | C2 | 45 | 13 |

Figure 11: The plugin after completing the addkey operation



|  |  |  |  |
|---|---|---|---|
| A4 | 2F | FC | E7 |
| 22 | 7A | B4 | D1 |
| 6F | FC | FC | 1C |
| B4 | 8C | 17 | 52 |

The corresponding byte in the S-Box is determined and placed into the result matrix.

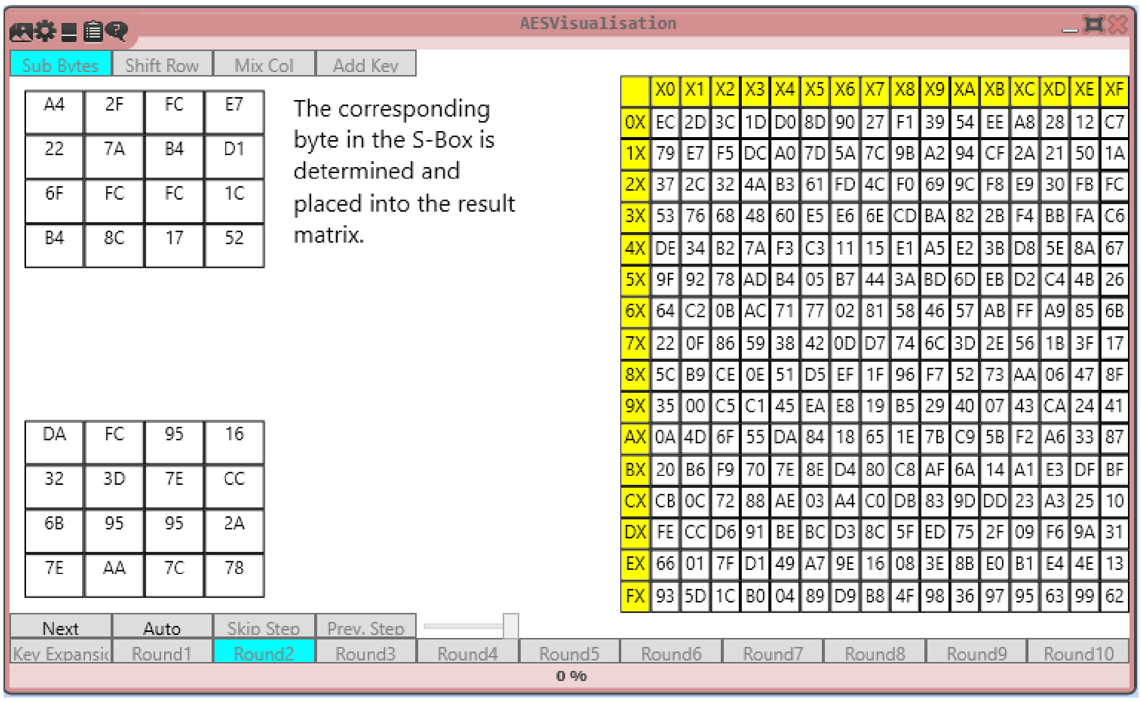
|  | X0 | X1 | X2 | X3 | X4 | X5 | X6 | X7 | X8 | X9 | XA | XB | XC | XD | XE | XF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0X | EC | 2D | 3C | 1D | D0 | 8D | 90 | 27 | F1 | 39 | 54 | EE | A8 | 28 | 12 | C7 |
| 1X | 79 | E7 | F5 | DC | A0 | 7D | 5A | 7C | 9B | A2 | 94 | CF | 2A | 21 | 50 | 1A |
| 2X | 37 | 2C | 32 | 4A | B3 | 61 | FD | 4C | F0 | 69 | 9C | F8 | E9 | 30 | FB | FC |
| 3X | 53 | 76 | 68 | 48 | 60 | E5 | E6 | 6E | CD | BA | 82 | 2B | F4 | BB | FA | C6 |
| 4X | DE | 34 | B2 | 7A | F3 | C3 | 11 | 15 | E1 | A5 | E2 | 3B | D8 | 5E | 8A | 67 |
| 5X | 9F | 92 | 78 | AD | B4 | 05 | B7 | 44 | 3A | BD | 6D | EB | D2 | C4 | 4B | 26 |
| 6X | 64 | C2 | 0B | AC | 71 | 77 | 02 | 81 | 58 | 46 | 57 | AB | FF | A9 | 85 | 6B |
| 7X | 22 | 0F | 86 | 59 | 38 | 42 | 0D | D7 | 74 | 6C | 3D | 2E | 56 | 1B | 3F | 17 |
| 8X | 5C | B9 | CE | 0E | 51 | D5 | EF | 1F | 96 | F7 | 52 | 73 | AA | 06 | 47 | 8F |
| 9X | 35 | 00 | C5 | C1 | 45 | EA | E8 | 19 | B5 | 29 | 40 | 07 | 43 | CA | 24 | 41 |
| AX | 0A | 4D | 6F | 55 | DA | 84 | 18 | 65 | 1E | 7B | C9 | 5B | F2 | A6 | 33 | 87 |
| BX | 20 | B6 | F9 | 70 | 7E | 8E | D4 | 80 | C8 | AF | 6A | 14 | A1 | E3 | DF | BF |
| CX | CB | 0C | 72 | 88 | AE | 03 | A4 | C0 | DB | 83 | 9D | DD | 23 | A3 | 25 | 10 |
| DX | FE | CC | D6 | 91 | BE | BC | D3 | 8C | 5F | ED | 75 | 2F | 09 | F6 | 9A | 31 |
| EX | 66 | 01 | 7F | D1 | 49 | A7 | 9E | 16 | 08 | 3E | 8B | E0 | B1 | E4 | 4E | 13 |
| FX | 93 | 5D | 1C | B0 | 04 | 89 | D9 | B8 | 4F | 98 | 36 | 97 | 95 | 63 | 99 | 62 |

Figure 12: The plugin after splitting the first byte and highlighting the result during subByte

The next operation I visualized is the "SubByte" operation (Figure 12). On the top left is the state matrix and on the bottom left the result matrix just like it was in the previous operation. On the right we have the S-Box as it was during the key expansion. However, now every byte needs to be substituted and not just the last four. I decided to transfer one byte after another to the center and split it up. Again I used color coding to show how the result is generated. First the left part of the byte is highlighted and then the corresponding row in the S-Box. Next the right part of the byte and then the column. And finally the actual value in the S-Box (Figure 12).

In the last step the result is transferred to the result matrix.

All these steps are repeated until all bytes are substituted (Figure 13).



| | | | |
|---|---|---|---|
| A4 | 2F | FC | E7 |
| 22 | 7A | B4 | D1 |
| 6F | FC | FC | 1C |
| B4 | 8C | 17 | 52 |

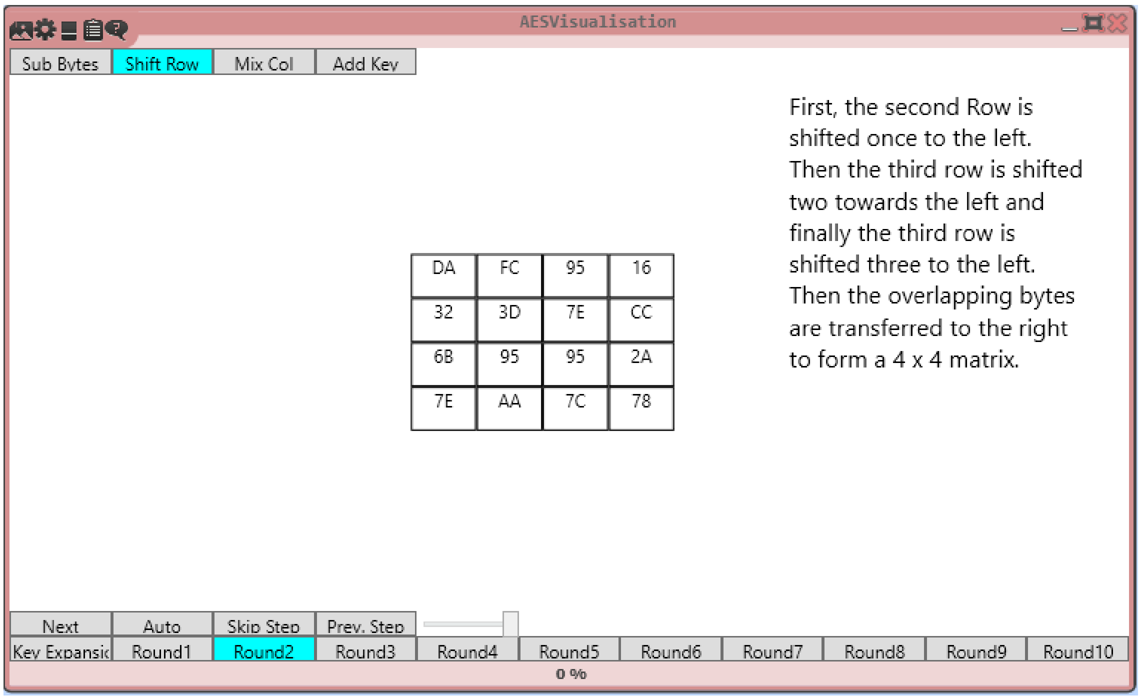
The corresponding byte in the S-Box is determined and placed into the result matrix.

| | | | |
|---|---|---|---|
| DA | FC | 95 | 16 |
| 32 | 3D | 7E | CC |
| 6B | 95 | 95 | 2A |
| 7E | AA | 7C | 78 |

| | X0 | X1 | X2 | X3 | X4 | X5 | X6 | X7 | X8 | X9 | XA | XB | XC | XD | XE | XF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0X | EC | 2D | 3C | 1D | D0 | 8D | 90 | 27 | F1 | 39 | 54 | EE | A8 | 28 | 12 | C7 |
| 1X | 79 | E7 | F5 | DC | A0 | 7D | 5A | 7C | 9B | A2 | 94 | CF | 2A | 21 | 50 | 1A |
| 2X | 37 | 2C | 32 | 4A | B3 | 61 | FD | 4C | F0 | 69 | 9C | F8 | E9 | 30 | FB | FC |
| 3X | 53 | 76 | 68 | 48 | 60 | E5 | E6 | 6E | CD | BA | 82 | 2B | F4 | BB | FA | C6 |
| 4X | DE | 34 | B2 | 7A | F3 | C3 | 11 | 15 | E1 | A5 | E2 | 3B | D8 | 5E | 8A | 67 |
| 5X | 9F | 92 | 78 | AD | B4 | 05 | B7 | 44 | 3A | BD | 6D | EB | D2 | C4 | 4B | 26 |
| 6X | 64 | C2 | 0B | AC | 71 | 77 | 02 | 81 | 58 | 46 | 57 | AB | FF | A9 | 85 | 6B |
| 7X | 22 | 0F | 86 | 59 | 38 | 42 | 0D | D7 | 74 | 6C | 3D | 2E | 56 | 1B | 3F | 17 |
| 8X | 5C | B9 | CE | 0E | 51 | D5 | EF | 1F | 96 | F7 | 52 | 73 | AA | 06 | 47 | 8F |
| 9X | 35 | 00 | C5 | C1 | 45 | EA | E8 | 19 | B5 | 29 | 40 | 07 | 43 | CA | 24 | 41 |
| AX | 0A | 4D | 6F | 55 | DA | 84 | 18 | 65 | 1E | 7B | C9 | 5B | F2 | A6 | 33 | 87 |
| BX | 20 | B6 | F9 | 70 | 7E | 8E | D4 | 80 | C8 | AF | 6A | 14 | A1 | E3 | DF | BF |
| CX | CB | 0C | 72 | 88 | AE | 03 | A4 | C0 | DB | 83 | 9D | DD | 23 | A3 | 25 | 10 |
| DX | FE | CC | D6 | 91 | BE | BC | D3 | 8C | 5F | ED | 75 | 2F | 09 | F6 | 9A | 31 |
| EX | 66 | 01 | 7F | D1 | 49 | A7 | 9E | 16 | 08 | 3E | 8B | E0 | B1 | E4 | 4E | 13 |
| FX | 93 | 5D | 1C | B0 | 04 | 89 | D9 | B8 | 4F | 98 | 36 | 97 | 95 | 63 | 99 | 62 |

Figure 13: The plugin after completing the subByte operation

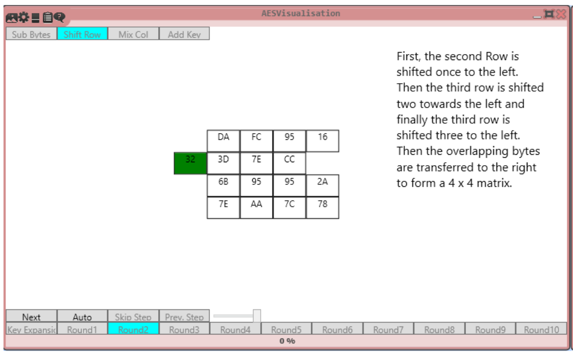
For the "ShiftRow" operation I decided to use only one matrix in the center (Figure 14). It takes four steps to get the result. In the first three steps the rows are shifted to the left and the overlapping bytes are highlighted (Figure 15a- 15c). In the last step the overlapping bytes are repositioned to form the result matrix (Figure 15d).



| | | | |
|---|---|---|---|
| DA | FC | 95 | 16 |
| 32 | 3D | 7E | CC |
| 6B | 95 | 95 | 2A |
| 7E | AA | 7C | 78 |

First, the second Row is shifted once to the left. Then the third row is shifted two towards the left and finally the third row is shifted three to the left. Then the overlapping bytes are transferred to the right to form a 4 x 4 matrix.

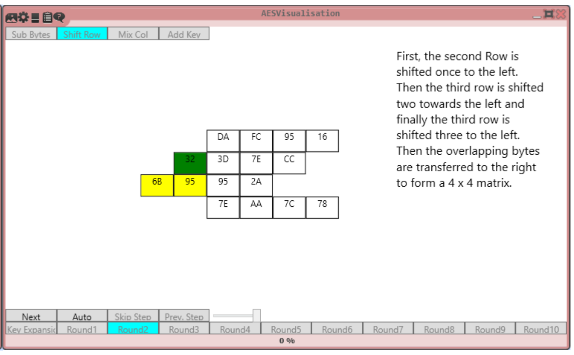
Figure 14: The plugin before commencing the shiftRow operation



| | | | | |
|---|---|---|---|---|
| | DA | FC | 95 | 16 |
| 32 | 3D | 7E | CC | |
| | 6B | 95 | 95 | 2A |
| | 7E | AA | 7C | 78 |

First, the second Row is shifted once to the left. Then the third row is shifted two towards the left and finally the third row is shifted three to the left. Then the overlapping bytes are transferred to the right to form a 4 x 4 matrix.

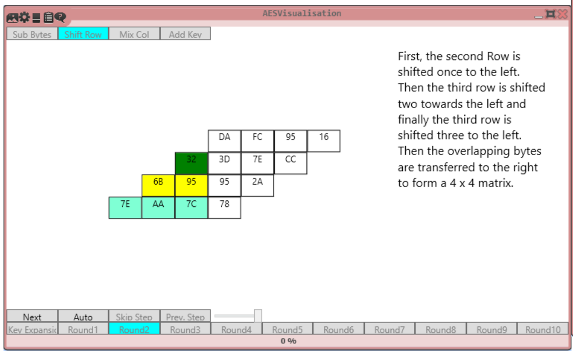
(a) The second row is shifted one field to the left



| | | | | | |
|---|---|---|---|---|---|
| | | DA | FC | 95 | 16 |
| | 32 | 3D | 7E | CC | |
| 6B | 95 | 95 | 2A | | |
| | | 7E | AA | 7C | 78 |

First, the second Row is shifted once to the left. Then the third row is shifted two towards the left and finally the third row is shifted three to the left. Then the overlapping bytes are transferred to the right to form a 4 x 4 matrix.

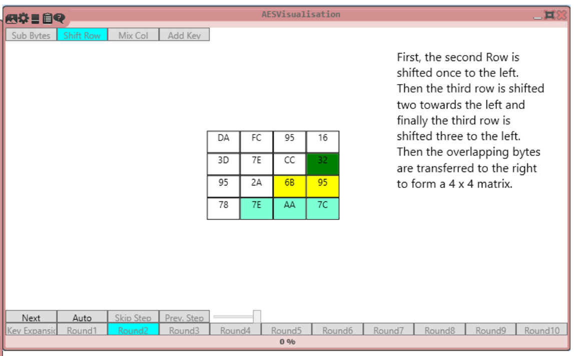
(b) The third row is shifted two fields two the left



| | | | | | | |
|---|---|---|---|---|---|---|
| | | | DA | FC | 95 | 16 |
| | | 32 | 3D | 7E | CC | |
| | 6B | 95 | 95 | 2A | | |
| 7E | AA | 7C | 78 | | | |

First, the second Row is shifted once to the left. Then the third row is shifted two towards the left and finally the third row is shifted three to the left. Then the overlapping bytes are transferred to the right to form a 4 x 4 matrix.

(c) The fourth row is shifted three fields to the left



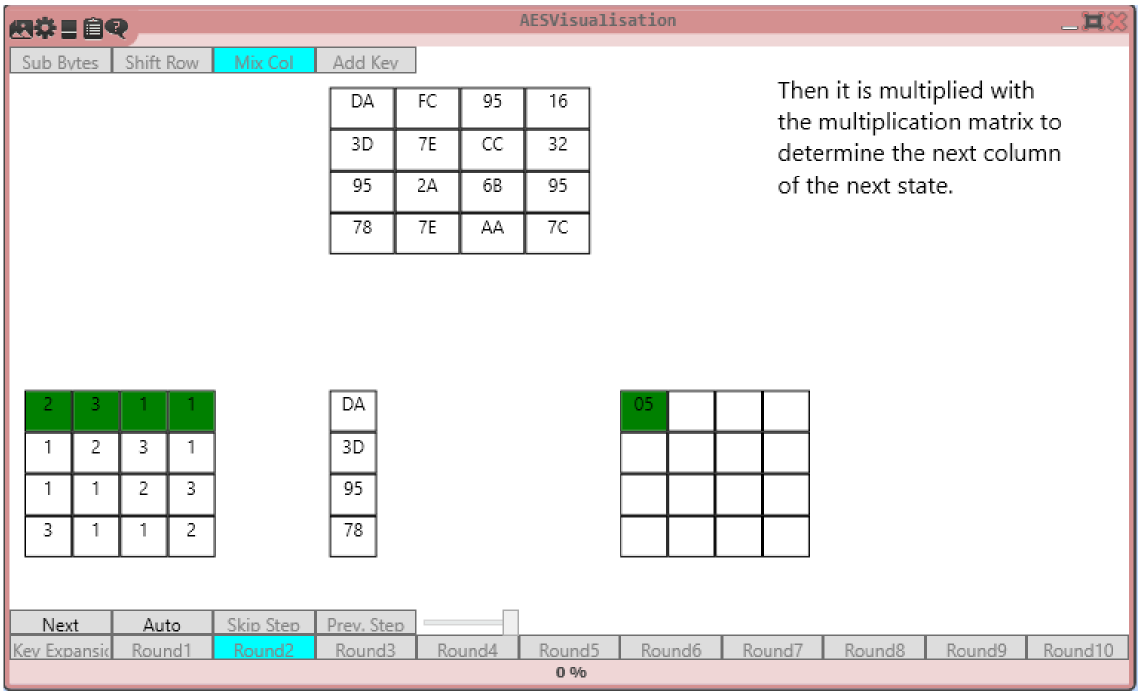
| | | | |
|---|---|---|---|
| DA | FC | 95 | 16 |
| 3D | 7E | CC | 32 |
| 95 | 2A | 6B | 95 |
| 78 | 7E | AA | 7C |

First, the second Row is shifted once to the left. Then the third row is shifted two towards the left and finally the third row is shifted three to the left. Then the overlapping bytes are transferred to the right to form a 4 x 4 matrix.

(d) The plugin after completing the shiftRow operation

Figure 15: The four steps of ShiftRow

For the next operation I chose to use three matrices and one column (Figure 16). The matrix on the top is the current state. Bottom left is the multiplication matrix and on the right the result matrix. The column in the middle shows the bytes that are being worked with at the moment. In the beginning the first column of the state matrix is loaded into the column in the middle. Then the first row of the multiplication matrix is highlighted. Subsequently the result of the multiplication is displayed and highlighted in the result matrix. This is repeated for each row of the multiplication matrix and each column of the state matrix. When all steps are completed, the result matrix is filled and the operation is done (Figure 17).



| | | | |
|---|---|---|---|
| DA | FC | 95 | 16 |
| 3D | 7E | CC | 32 |
| 95 | 2A | 6B | 95 |
| 78 | 7E | AA | 7C |

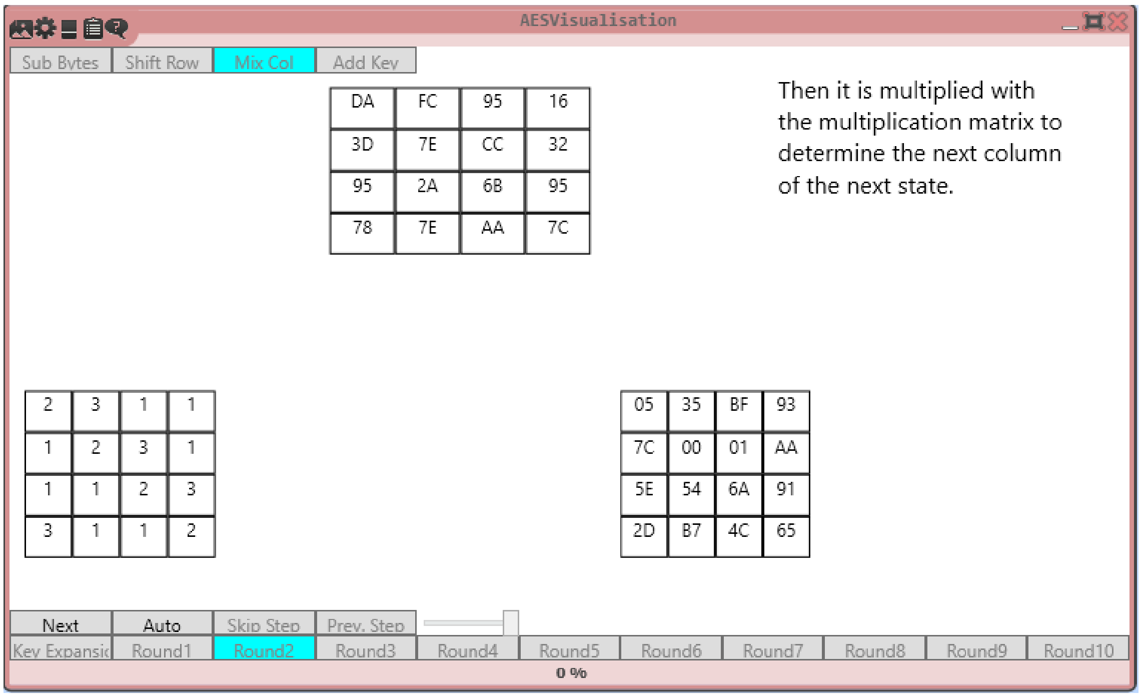
Then it is multiplied with the multiplication matrix to determine the next column of the next state.

| | | | |
|---|---|---|---|
| 2 | 3 | 1 | 1 |
| 1 | 2 | 3 | 1 |
| 1 | 1 | 2 | 3 |
| 3 | 1 | 1 | 2 |

| |
|---|
| DA |
| 3D |
| 95 |
| 78 |

| | | | |
|---|---|---|---|
| 05 | | | |
| | | | |
| | | | |
| | | | |

Figure 16: The plugin after completing the first calculation of the mixColumn step



|  |  |  |  |
|---|---|---|---|
| DA | FC | 95 | 16 |
| 3D | 7E | CC | 32 |
| 95 | 2A | 6B | 95 |
| 78 | 7E | AA | 7C |

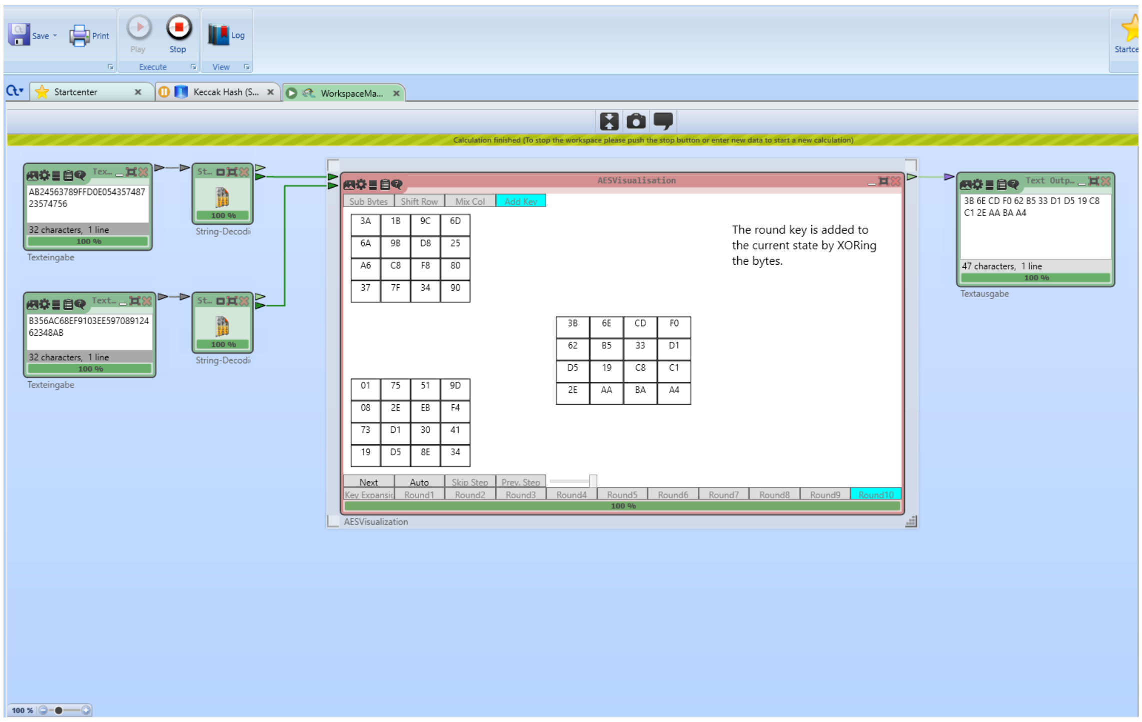
Then it is multiplied with the multiplication matrix to determine the next column of the next state.

|  |  |  |  |
|---|---|---|---|
| 2 | 3 | 1 | 1 |
| 1 | 2 | 3 | 1 |
| 1 | 1 | 2 | 3 |
| 3 | 1 | 1 | 2 |

|  |  |  |  |
|---|---|---|---|
| 05 | 35 | BF | 93 |
| 7C | 00 | 01 | AA |
| 5E | 54 | 6A | 91 |
| 2D | B7 | 4C | 65 |

Figure 17: The plugin after completing the mixColumn operation

When all operations are completed, the cipher text is displayed in the text output window of the template (Figure 18).



Calculation finished (To stop the workspace please push the stop button or enter new data to start a new calculation)

AB24563789FFD0E054357487
23574756

32 characters, 1 line

B356AC68EF9103EE597089124
62348AB

32 characters, 1 line

|  |  |  |  |
|---|---|---|---|
| 3A | 1B | 9C | 6D |
| 6A | 9B | D8 | 25 |
| A6 | C8 | F8 | 80 |
| 37 | 7F | 34 | 90 |

The round key is added to the current state by XORing the bytes.

|  |  |  |  |
|---|---|---|---|
| 3B | 6E | CD | F0 |
| 62 | B5 | 33 | D1 |
| D5 | 19 | C8 | C1 |
| 2E | AA | BA | A4 |

|  |  |  |  |
|---|---|---|---|
| 01 | 75 | 51 | 9D |
| 08 | 2E | EB | F4 |
| 73 | D1 | 30 | 41 |
| 19 | D5 | 8E | 34 |

3B 6E CD F0 62 B5 33 D1 D5 19 C8
C1 2E AA BA A4

47 characters, 1 line

Figure 18: The plugin after completion

# 4 Implementation of the Plugin

This chapter will show how I implemented the plugin.

The first part of the chapter focuses on the interface explaining how and why it is built this way.

The second part deals with the logic behind it and presents some of the most important methods.

## 4.1 Interface

For the main layout I decided to use a grid in which all other elements were placed (Figure 19). I thought about using a canvas first but preferred the grid because it was better in terms of scaling. For example when the window will be spread the columns will spread too. The buttons are coded to have the same width as the column they are in. Therefore, if the column widens the button widens to the same extend. The same goes for the rows and their height.

The grid has four rows and eleven columns. The top row contains the four buttons during encryption. The second row is where the steps are visualized.

I considered using just one row at the bottom. But the buttons would be not wide enough and the row itself too crowded, so I went with two. The third row has four buttons and the slider all of which are used solely for navigation.

The fourth contains all the "Round" buttons and one for either encryption or key expansion. Those buttons are used for navigation as well as visualization. They show in which round the process currently is.

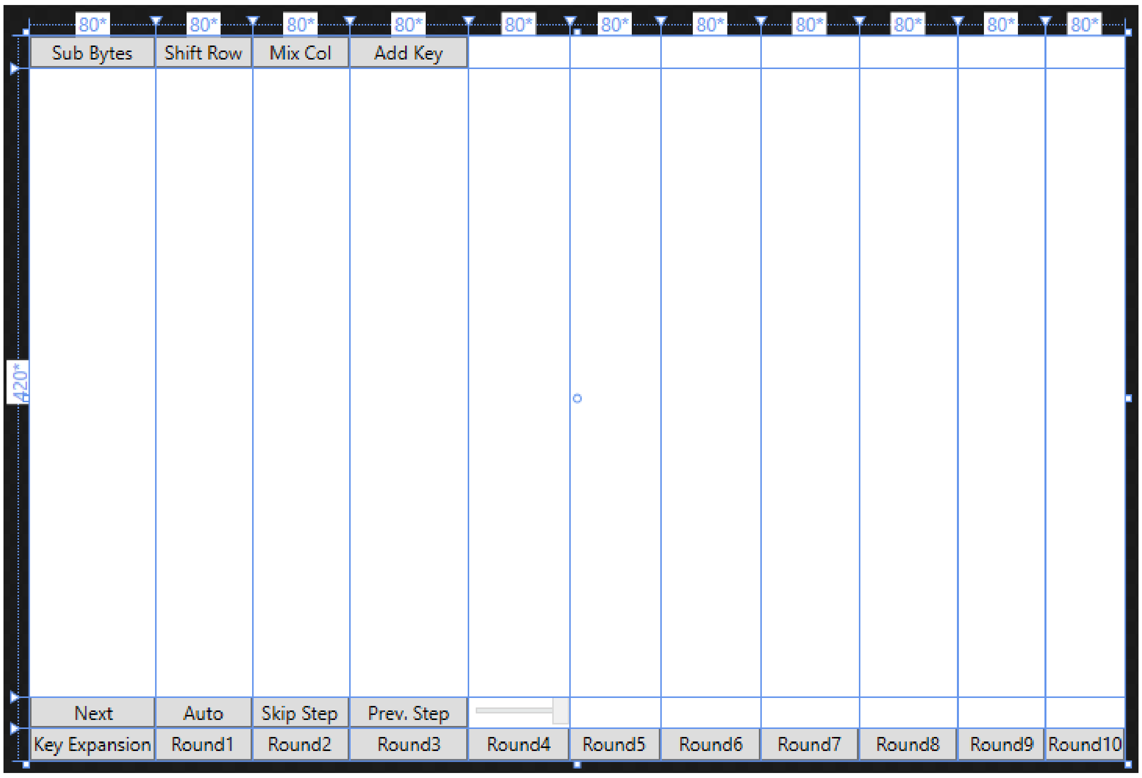
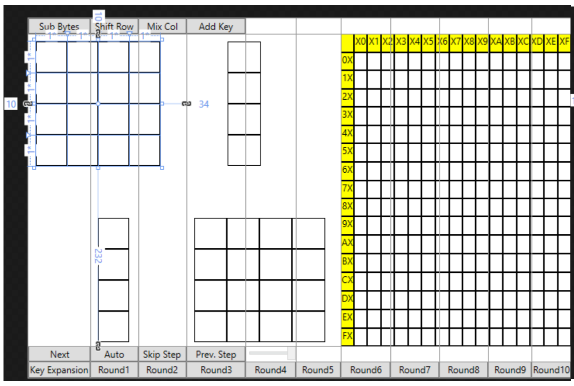
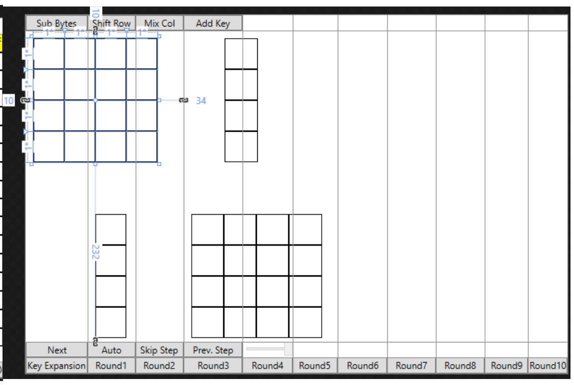


Figure 19: The main grid of the interface

To visualize the various operations I first thought about using a canvas. For the reasons previously stated I chose a grid here too. To be more precise there are several small grids instead of just one grid for each operation. One separate grid is used for each element (matrix, columns, etc.) of the operation. This allows me to set the visibility of the S-Box grid to visible (Figure 20a) or hidden (Figure 20b) if I want to show or hide it respectively later during the key expansion.



(a) The S-Box is set to visible



(b) The S-Box is set to hidden

Figure 20: The various grids used to visualize the key expansion

Each cell of a grid has a border containing a text block which displays the text. It was done this way because it enables scaling. If the window size is increased (when for example it is set to full screen) the main grid stretches. Because of this the small grids positioned inside the main grid also stretch as do the borders in each cell of said grids.

To switch between the different operations I just had to hide the grids of the previous operation and make the required ones visible. In some cases single elements need to be hidden instead of an entire grid.

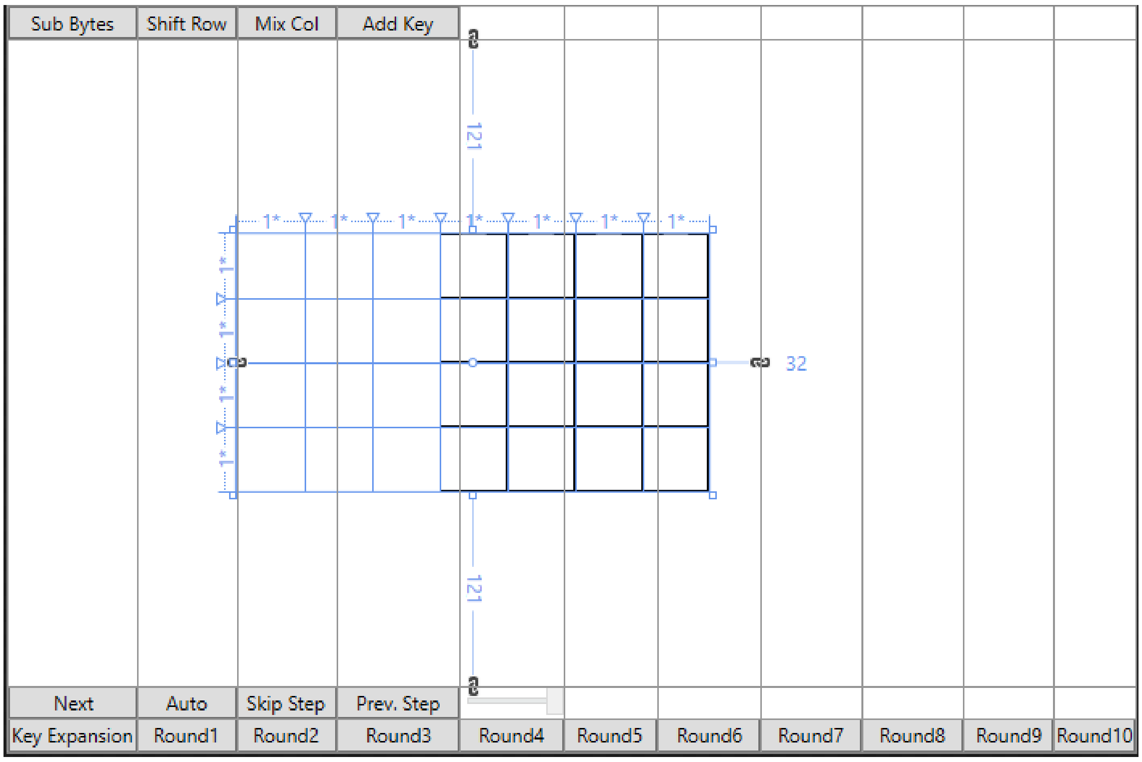
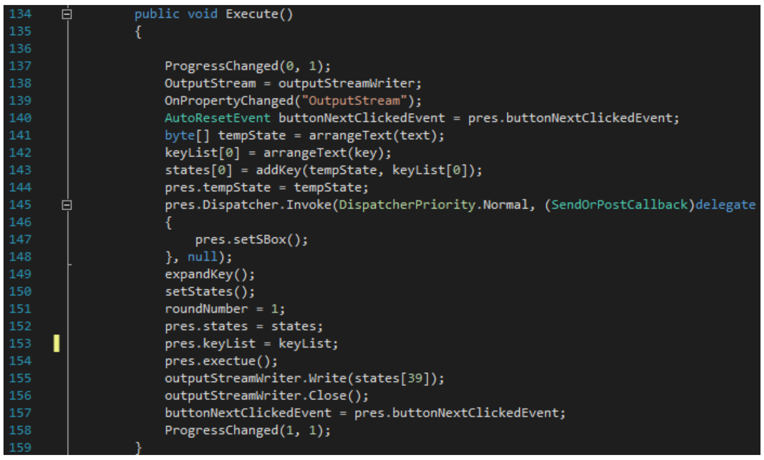


Figure 21: The grid of the shiftRow operation

For the "ShiftRow" operation for example the initial idea was to move the elements which is why I first considered using a canvas. However, I decided to work with a grid in which at first only the borders from column four onward were visible (Figure 21). The actual shifts occur by hiding some elements and making others visible. To shift the second row for example the rightmost border is hidden and the border in row two column three is made visible. In addition, the displayed values are edited so that they are still in order (Figure 15a).



```
        public void Execute()
        {

            ProgressChanged(0, 1);
            OutputStream = outputStreamWriter;
            OnPropertyChanged("OutputStream");
            AutoResetEvent buttonNextClickedEvent = pres.buttonNextClickedEvent;
            byte[] tempState = arrangeText(text);
            keyList[0] = arrangeText(key);
            states[0] = addKey(tempState, keyList[0]);
            pres.tempState = tempState;
            pres.Dispatcher.Invoke(DispatcherPriority.Normal, (SendOrPostCallback)delegate
            {
                pres.setSBox();
            }, null);
            expandKey();
            setStates();
            roundNumber = 1;
            pres.states = states;
            pres.keyList = keyList;
            pres.exectue();
            outputStreamWriter.Write(states[39]);
            outputStreamWriter.Close();
            buttonNextClickedEvent = pres.buttonNextClickedEvent;
            ProgressChanged(1, 1);
        }
```

Figure 22: The code of the execute method in the main class

## 4.2 Logic

The plugin mainly consists of two classes. The first is AESVisualization.cs which takes the input in the beginning, calculates all states, keys, etc. and returns the output.

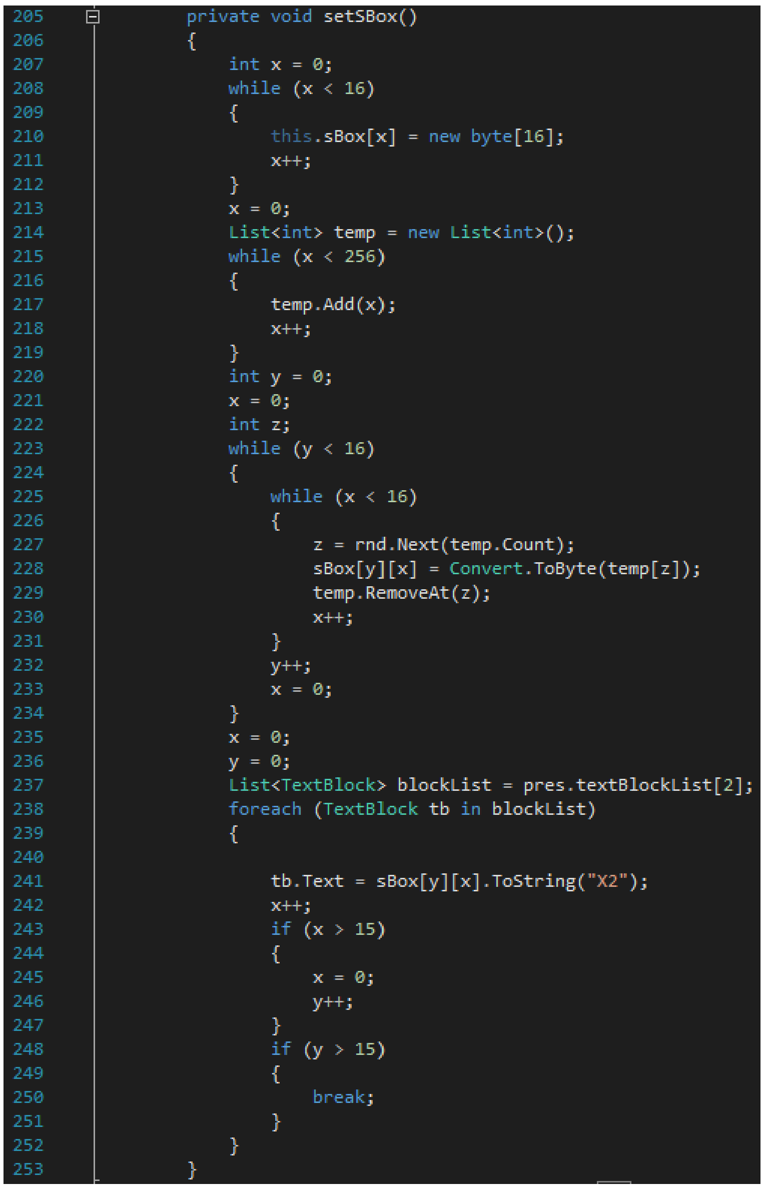
The second is AESPresentation.xaml/AESPresentation.xaml.cs which is responsible for the interface and user controls. AESVisualization.cs creates its own instance of AESPresentation called "pres".

The first big decision that had to be made was whether the states after each encryption operation would be calculated during the visualization or precalculated and stored at the start of the execution. One feature the plugin should have was to not only jump ahead to later operations but also to go back to previous ones. That means if the values were calculated during the visualization every time you went back they would have to be recalculated from the start. Therefore, I decided to precompute and store results of each operation in an array of byte arrays. The same goes for the key expansion which obviously needs to be done before the encryption.

The first step of "Execute()" (Figure 22) is setting the S-Box since a new one is generated each time the plugin is started (Figure 23). First, an array of byte arrays is created. As it consists of sixteen arrays with a length of sixteen each, it contains 256 bytes. Then a list of integers is created, ranging from 0 to 255. Now for each byte in the S-Box array a random number between zero and the length of the list is generated. Then the integer at this index is converted to a byte and inserted into the S-Box array. Lastly the integer is deleted from the list. This is repeated until all bytes in the S-Box have a value and the integer list is empty.

Once this is done the method gets a list of text blocks from pres containing the text blocks of the S-Box. Notice that these text blocks do not include the first row and the first column since those are used as indices and are preset. Finally each text block receives the according value as its text.

After the S-Box is set, the round keys are calculated and stored as are the states after each operation. The computed states and keys are then given to the instance of AESPresentation created earlier and its execute method is started.



```
        private void setSBox()
        {
            int x = 0;
            while (x < 16)
            {
                this.sBox[x] = new byte[16];
                x++;
            }
            x = 0;
            List<int> temp = new List<int>();
            while (x < 256)
            {
                temp.Add(x);
                x++;
            }
            int y = 0;
            x = 0;
            int z;
            while (y < 16)
            {
                while (x < 16)
                {
                    z = rnd.Next(temp.Count);
                    sBox[y][x] = Convert.ToByte(temp[z]);
                    temp.RemoveAt(z);
                    x++;
                }
                y++;
                x = 0;
            }
            x = 0;
            y = 0;
            List<TextBlock> blockList = pres.textBlockList[2];
            foreach (TextBlock tb in blockList)
            {

                tb.Text = sBox[y][x].ToString("X2");
                x++;
                if (x > 15)
                {
                    x = 0;
                    y++;
                }
                if (y > 15)
                {
                    break;
                }
            }
        }
```

Figure 23: The code of the setS-Box method

The execute method basically consists of one while loop (Figure 24).

In the first dispatcher everything is set up for the key expansion. First it makes sure the button in the bottom left of the interface says "Skip Expansion". This is set initially and therefore redundant for the first round. However it is necessary if the user wants to go back from encryption to key expansion. "invisible()" hides all grids to make sure no unwanted grids of previous operations are still visible. "disableButtons" does as the name says so the buttons cannot be pressed until the next stop. The "changeRoundButton" method is used to highlight the correct round button. This method must be called since the round button must be round 1 regardless of the round the user was during encryption when he wants to go back to expansion. "buttonVisible()" toggles the visibility of the four operation buttons on the top left of the interface depending on whether the "expansion" boolean is true or false. If it is true the buttons are hidden and if it is false they are visible.

When all of the above is done, everything is set for the key expansion. The "keyExpansion()" method mainly consists of one big loop repeating and visualizing all the steps of the expansion until the integer "roundNumber" reaches ten or the user presses the "Skip Expansion" key.

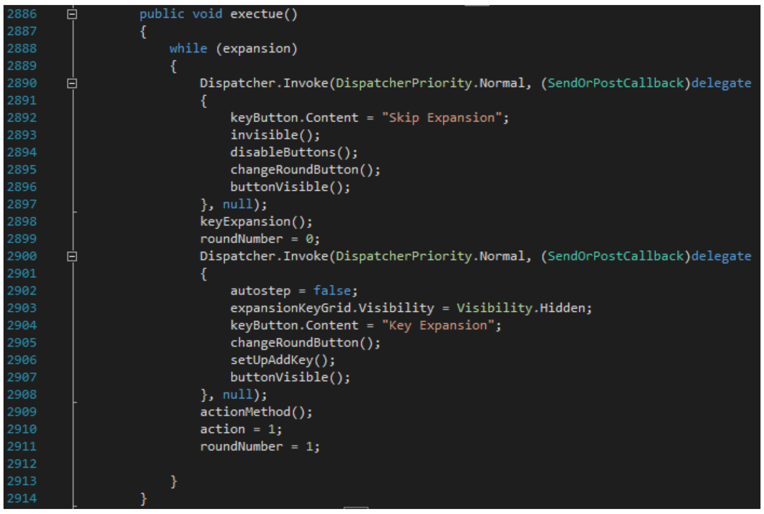
After the key expansion is done, "roundNumber" is set back to zero because the encryption needs to start there. The second dispatcher works similar to the first one, but this time it hides the remaining grids of the key expansion and sets up "addKey" which is the first step of encryption. Once the setup is done, the "actionMethod()" is called which is responsible for the visualization of the encryption (Figure 25).

The method has two while loops. The first one is used to go through all rounds while the second one goes through all operations. The "action" integer is used to determine the next action (1 = subBytes, 2 = shiftRow, 3 = mixColumn, 4 = addKey). Inside the while loop that depends on "action" is a switch statement to choose the right action. The different switch cases are all built similarly. First the operation is executed. During this time all buttons used for navigation are disabled. Once the operation is completed auto step is disabled and the method pauses to give the user a chance to see the result.

Once the user decides to continue, the next step is set up and the buttons are enabled again. In round ten there are two special situations. The first one (Figure 25a) is after "ShiftRow". The method checks which round it currently is in. In every round but the tenth the next operation is mix columns. In round ten, however, it goes directly to add key. The second situation is after "addKey" in round 10 when the encryption is done and no operation needs to be set up.

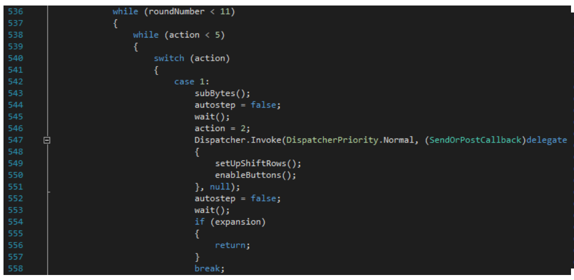
After one operation is completed and the next one is set up, the buttons are enabled so the user can navigate by pressing any of the round or operation buttons or the "Key Expansion" button. The "Round" buttons will set "roundNumber" to the corresponding value and "action" to 1. The "Operation" buttons will just change the "action" integer. In addition, all of those buttons will set up the next operation. Pressing the "Key Expansion" button will set "expansion" to true. All those buttons also tell the method to continue. The last step is always to check whether "expansion" is true or not. If so, the "actionMethod()" returns and the while loop in "execute()" (Figure 24) will start from the beginning by setting up and executing "keyExpansion()".

Once "execute()" in pres is completed the last thing to be done in the main "Execute()" (Figure 22) is to put out the last state.



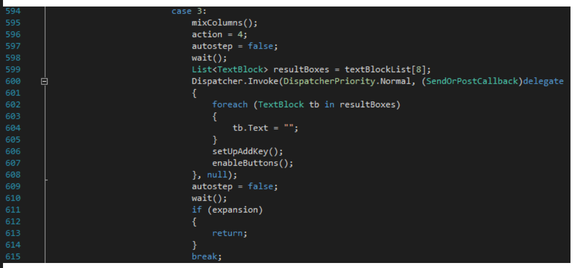
```
public void exectue()
{
    while (expansion)
    {
        Dispatcher.Invoke(DispatcherPriority.Normal, (SendOrPostCallback)delegate
        {
            keyButton.Content = "Skip Expansion";
            invisible();
            disableButtons();
            changeRoundButton();
            buttonVisible();
        }, null);
        keyExpansion();
        roundNumber = 0;
        Dispatcher.Invoke(DispatcherPriority.Normal, (SendOrPostCallback)delegate
        {
            autostep = false;
            expansionKeyGrid.Visibility = Visibility.Hidden;
            keyButton.Content = "Key Expansion";
            changeRoundButton();
            setUpAddKey();
            buttonVisible();
        }, null);
        actionMethod();
        action = 1;
        roundNumber = 1;

    }
}
```

Figure 24: The code of the execute method in the presentation class



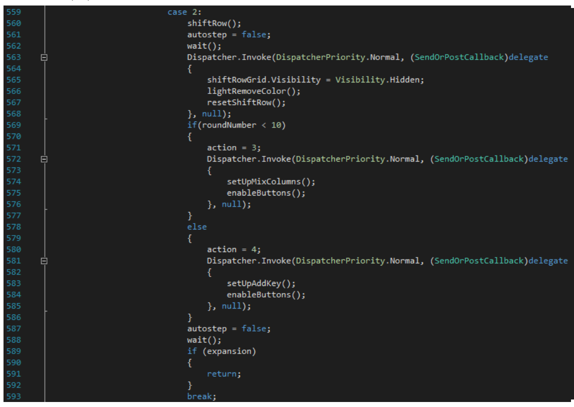
```
while (roundNumber < 11)
{
    while (action < 5)
    {
        switch (action)
        {
            case 1:
                subBytes();
                autostep = false;
                wait();
                action = 2;
                Dispatcher.Invoke(DispatcherPriority.Normal, (SendOrPostCallback)delegate
                {
                    setUpShiftRows();
                    enableButtons();
                }, null);
                autostep = false;
                wait();
                if (expansion)
                {
                    return;
                }
                break;
```

(a) The first case of the switch statement



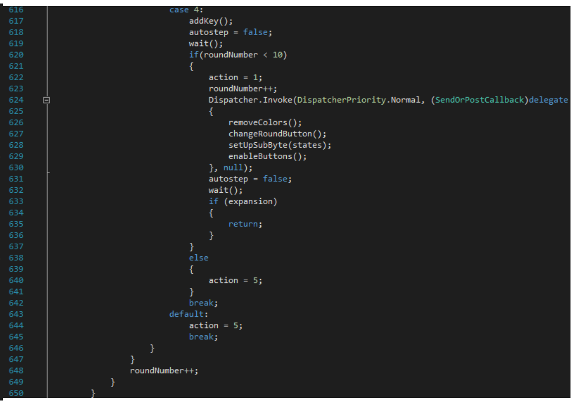
```
case 3:
    mixColumns();
    action = 4;
    autostep = false;
    wait();
    List<TextBlock> resultBoxes = textBlockList[8];
    Dispatcher.Invoke(DispatcherPriority.Normal, (SendOrPostCallback)delegate
    {
        foreach (TextBlock tb in resultBoxes)
        {
            tb.Text = "";
        }
        setUpAddKey();
        enableButtons();
    }, null);
    autostep = false;
    wait();
    if (expansion)
    {
        return;
    }
    break;
```

(b) The second case of the switch statement



```
case 2:
    shiftRow();
    autostep = false;
    wait();
    Dispatcher.Invoke(DispatcherPriority.Normal, (SendOrPostCallback)delegate
    {
        shiftRowGrid.Visibility = Visibility.Hidden;
        lightRemoveColor();
        resetShiftRow();
    }, null);
    if(roundNumber < 10)
    {
        action = 3;
        Dispatcher.Invoke(DispatcherPriority.Normal, (SendOrPostCallback)delegate
        {
            setUpMixColumns();
            enableButtons();
        }, null);
    }
    else
    {
        action = 4;
        Dispatcher.Invoke(DispatcherPriority.Normal, (SendOrPostCallback)delegate
        {
            setUpAddKey();
            enableButtons();
        }, null);
    }
    autostep = false;
    wait();
    if (expansion)
    {
        return;
    }
    break;
```

(c) The third case of the switch statement



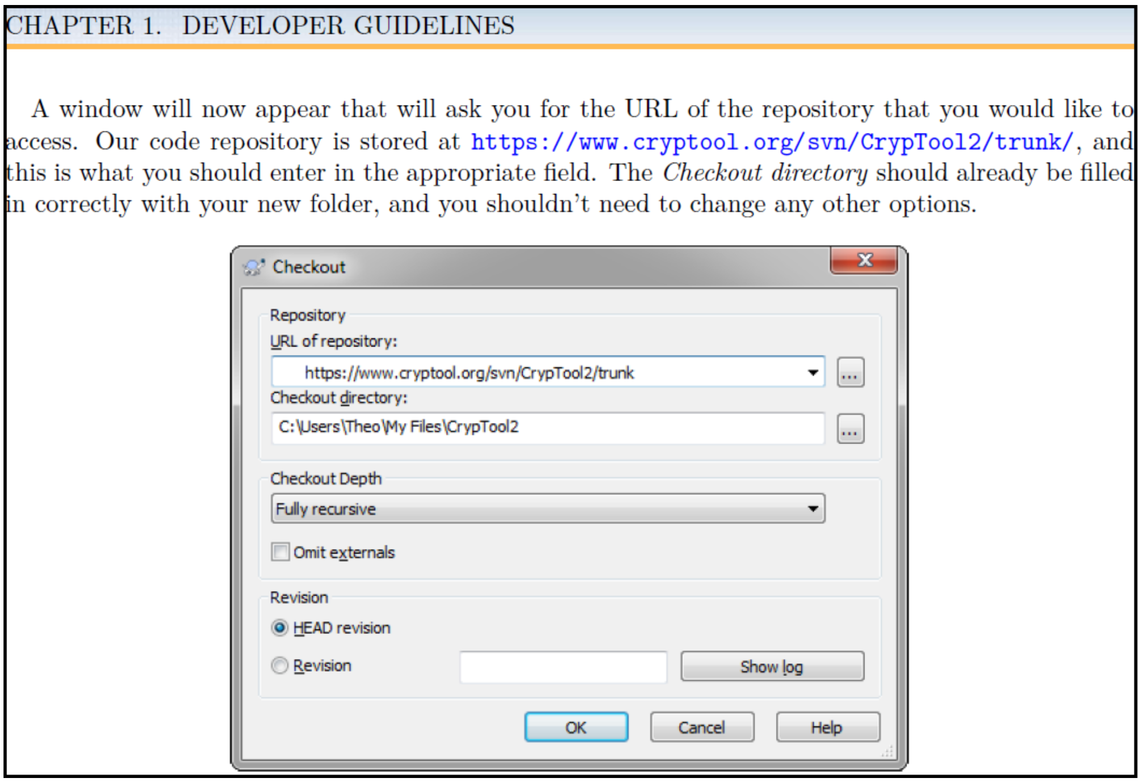
```
                case 4:
                    addKey();
                    autostep = false;
                    wait();
                    if(roundNumber < 10)
                    {
                        action = 1;
                        roundNumber++;
                        Dispatcher.Invoke(DispatcherPriority.Normal, (SendOrPostCallback)delegate
                        {
                            removeColors();
                            changeRoundButton();
                            setUpSubByte(states);
                            enableButtons();
                        }, null);
                        autostep = false;
                        wait();
                        if (expansion)
                        {
                            return;
                        }
                    }
                    else
                    {
                        action = 5;
                    }
                    break;
                default:
                    action = 5;
                    break;
            }
        }
        roundNumber++;
    }
}
```

(d) The fourth case of the switch statement

Figure 25: The code of the action method in the presentation class

# 5 How to

This chapter explains how to obtain CrypTool 2 and how to compile it. Afterwards it shows how to use the plugin.



CHAPTER 1. DEVELOPER GUIDELINES

A window will now appear that will ask you for the URL of the repository that you would like to access. Our code repository is stored at https://www.cryptool.org/svn/CrypTool2/trunk/, and this is what you should enter in the appropriate field. The *Checkout directory* should already be filled in correctly with your new folder, and you shouldn't need to change any other options.

Checkout

Repository

URL of repository: https://www.cryptool.org/svn/CrypTool2/trunk

Checkout directory: C:\Users\Theo\My Files\CrypTool2

Checkout Depth: Fully recursive

[ ] Omit externals

Revision

(●) HEAD revision

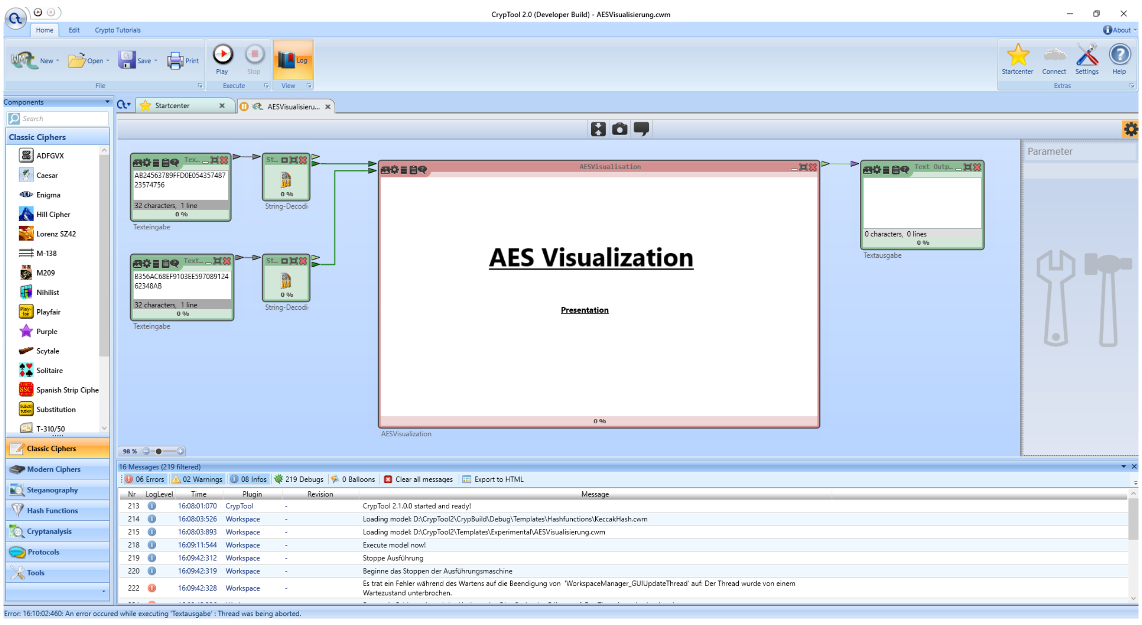
( ) Revision

Figure 26: An explanation of how to get the CrypTool 2 files from the repository

In order to use this plugin you first have to download the repository. To do this you need to create a folder to store the files in and install a SVN client, i.e. TortoiseSVN. Then, right click the folder and choose "SVN checkout...". Now follow the instruction on Figure 26 [6]. You will be asked to enter login information. Use 'anonymous' as the username and an empty password. This will give you read-only access.

Once the download is completed, open CrypTool2.0.sln with Visual Studio. Then, you have to compile it (make sure to choose "CrypWin" as the Startup Project).

After CrypTool 2 is loaded you have to open the template. To do so click 'Open' on the top left. Go to the folder which contains CrypTool 2. Go to Templates/Experimation/AESVisualization.cwm. Now the template is open (Figure 27).



AB24563789FFD0E05435748723574756
32 characters, 1 line

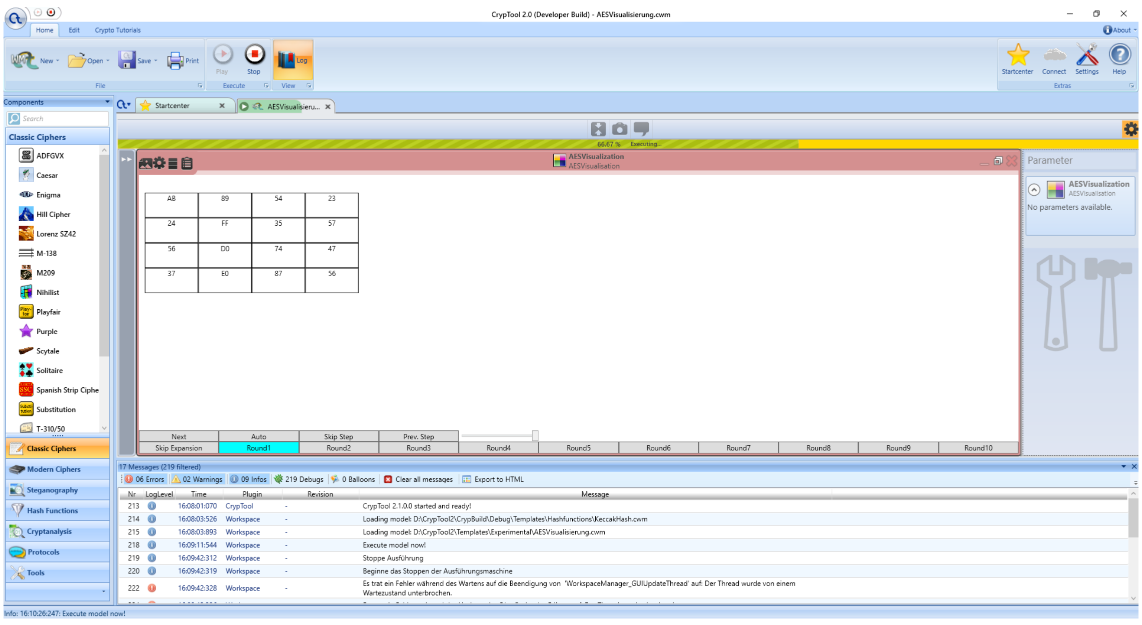
B356AC68EF9103EE59708912462348AB
32 characters, 1 line

0 characters, 0 lines

```
16 Messages (219 filtered)
06 Errors  02 Warnings  08 Infos  219 Debugs  0 Balloons  Clear all messages  Export to HTML
Nr   LogLevel  Time          Plugin     Revision  Message
213            16:08:01:070  CrypTool   -         CrypTool 2.1.0.0 started and ready!
214            16:08:03:526  Workspace  -         Loading model: D:\CrypTool2\CrypBuild\Debug\Templates\Hashfunctions\KeccakHash.cwm
215            16:08:03:893  Workspace  -         Loading model: D:\CrypTool2\Templates\Experimental\AESVisualisierung.cwm
218            16:09:11:544  Workspace  -         Execute model now!
219            16:09:42:312  Workspace  -         Stoppe Ausführung
220            16:09:42:319  Workspace  -         Beginne das Stoppen der Ausführungsmaschine
222            16:09:42:328  Workspace  -         Es trat ein Fehler während des Wartens auf die Beendigung von  'WorkspaceManager_GUIUpdateThread' auf: Der Thread wurde von einem
                                                  Wartezustand unterbrochen.
Error: 16:10:02:460: An error occured while executing 'Textausgabe' : Thread was being aborted.
```

Figure 27: The view after compiling CrypTool 2 and opening the AESVisualization template

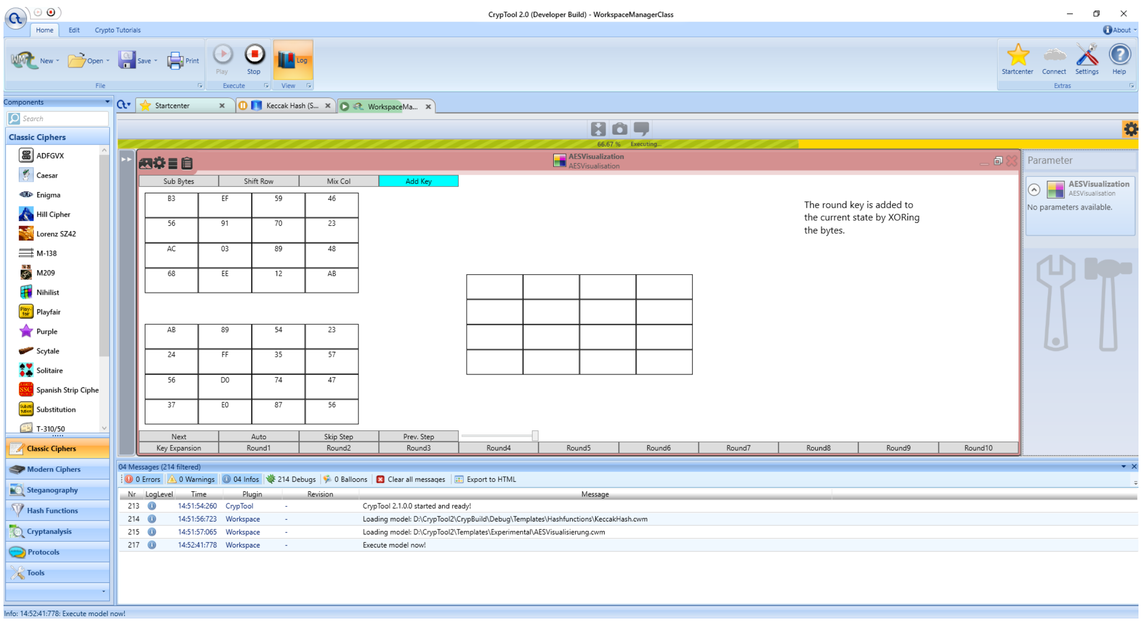
On the left you have two text inputs. The upper one is for the key and the lower one for the bytes of the message you want to encrypt. There are already values inserted when you start the template, but you can put in your own values too. In both cases, the values have to be strings of hex characters (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F), both with a length of 32 hex characters. Two characters make up one byte. So you have a key and a message both with a length of 16 byte = 128 bit. Once the values are inserted press the 'Play' button on the top to start the plugin. If you want to enlarge the AESVisualization window press the 'Fullscreen' icon on the top right of the component's title and close the row on the bottom (Figure 28).



| | | | |
|---|---|---|---|
| AB | 89 | 54 | 23 |
| 24 | FF | 35 | 57 |
| 56 | D0 | 74 | 47 |
| 37 | E0 | 87 | 56 |

```
17 Messages (219 filtered)
06 Errors  02 Warnings  09 Infos  219 Debugs  0 Balloons  Clear all messages  Export to HTML
Nr   LogLevel  Time          Plugin     Revision  Message
213            16:08:01:070  CrypTool   -         CrypTool 2.1.0.0 started and ready!
214            16:08:03:526  Workspace  -         Loading model: D:\CrypTool2\CrypBuild\Debug\Templates\Hashfunctions\KeccakHash.cwm
215            16:08:03:893  Workspace  -         Loading model: D:\CrypTool2\Templates\Experimental\AESVisualisierung.cwm
218            16:09:11:544  Workspace  -         Execute model now!
219            16:09:42:312  Workspace  -         Stoppe Ausführung
220            16:09:42:319  Workspace  -         Beginne das Stoppen der Ausführungsmaschine
222            16:09:42:328  Workspace  -         Es trat ein Fehler während des Wartens auf die Beendigung von  'WorkspaceManager_GUIUpdateThread' auf: Der Thread wurde von einem
                                                  Wartezustand unterbrochen.
Info: 16:10:26:247: Execute model now!
```

Figure 28: The plugin set to full screen during key expansion



|  |  |  |  |
|---|---|---|---|
| B3 | EF | 59 | 46 |
| 56 | 91 | 70 | 23 |
| AC | 03 | 89 | 48 |
| 68 | EE | 12 | AB |

|  |  |  |  |
|---|---|---|---|
| AB | 89 | 54 | 23 |
| 24 | FF | 35 | 57 |
| 56 | D0 | 74 | 47 |
| 37 | E0 | 87 | 56 |

The round key is added to the current state by XORing the bytes.

No parameters available.

04 Messages (214 filtered)

0 Errors · 0 Warnings · 04 Infos · 214 Debugs · 0 Balloons · Clear all messages · Export to HTML

| Nr | LogLevel | Time | Plugin | Revision | Message |
|---|---|---|---|---|---|
| 213 | ⓘ | 14:51:54:260 | CrypTool | - | CrypTool 2.1.0.0 started and ready! |
| 214 | ⓘ | 14:51:56:723 | Workspace | - | Loading model: D:\CrypTool2\CrypBuild\Debug\Templates\Hashfunctions\KeccakHash.cwm |
| 215 | ⓘ | 14:51:57:065 | Workspace | - | Loading model: D:\CrypTool2\Templates\Experimental\AESVisualisierung.cwm |
| 217 | ⓘ | 14:52:41:778 | Workspace | - | Execute model now! |

Info: 14:52:41:778: Execute model now!

Figure 29: The plugin set to full screen during the encryption

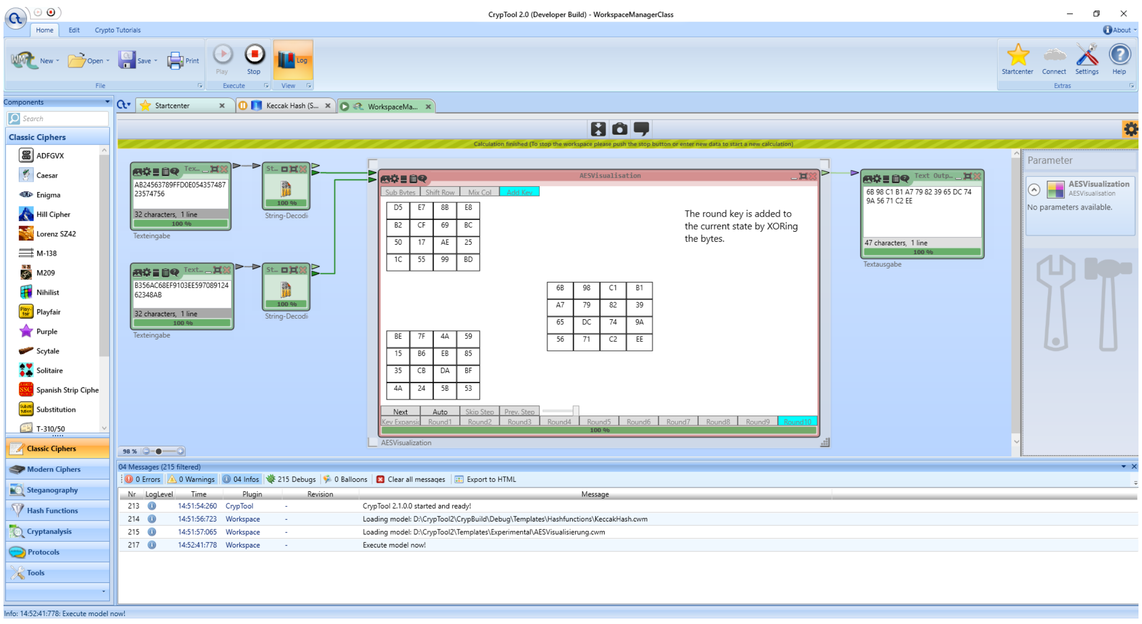
The plugin now is at the start of the first round of key expansion as indicated by the highlighted round1 button. You can change the round either by pressing any of the "Round" buttons to jump to that specific round or by pressing "Skip Step" or "Prev. Step". The latter two will bring you to the next or the previous round. In the first round the "Prev. Rund" button will not do anything. You can also skip the key expansion altogether by pressing the button on the bottom left.

If you want to start the round, press either "Next" or "Auto". Next will start the round and bring you to the next step of the operation and then wait for you to press either "Next" or "Auto". "Auto" on the other hand will start the operation and keep going through the steps pausing briefly after each one. The pause time is set by the slider to the right. It goes from long pauses on the left to short ones on the right. If you press "Auto" a second time, the plugin will stop and wait for new user input.

During every round the navigational buttons are disabled so that you have to complete the round before you can go to another one. When a round is done auto step is disabled to show you the result. Pressing "Next" or "Auto" will take you to the start of the next round and enable the navigational buttons again.

Once the key expansion is completed, a few changes happen to the interface (Figure 29). Most importantly there are four extra buttons on the top allowing you to choose between the various operations in each round. Changing the round will always bring you to the first operation in the round. The key on the bottom left now lets you return to the key expansion if you want to revisit it. Just like during key expansion the navigational buttons get disabled during every operation. They are enabled between operations to let you go to another operation or round.

Once the last operation of the last round is done, the plugin will write the result in the text output on the right and is complete (Figure 30).



Calculation finished (To stop the workspace please push the stop button or enter new data to start a new calculation)

AB24563789FFD0E054357487 23574756

32 characters, 1 line

100 %

B356AC68EF9103EE597089124 62348AB

32 characters, 1 line

100 %

| | | | |
|---|---|---|---|
| D5 | E7 | 8B | E8 |
| B2 | CF | 69 | BC |
| 50 | 17 | AE | 25 |
| 1C | 55 | 99 | BD |

| | | | |
|---|---|---|---|
| 6B | 98 | C1 | B1 |
| A7 | 79 | 82 | 39 |
| 65 | DC | 74 | 9A |
| 56 | 71 | C2 | EE |

| | | | |
|---|---|---|---|
| BE | 7F | 4A | 59 |
| 15 | B6 | EB | 85 |
| 35 | CB | DA | BF |
| 4A | 24 | 5B | 53 |

The round key is added to the current state by XORing the bytes.

6B 98 C1 B1 A7 79 82 39 65 DC 74 9A 56 71 C2 EE

47 characters, 1 line

100 %

No parameters available.

04 Messages (215 filtered)

0 Errors 0 Warnings 04 Infos 215 Debugs 0 Balloons

```
Nr   LogLevel  Time          Plugin     Revision  Message
213            14:51:54:260  CrypTool   -         CrypTool 2.1.0.0 started and ready!
214            14:51:56:723  Workspace  -         Loading model: D:\CrypTool2\CrypBuild\Debug\Templates\Hashfunctions\KeccakHash.cwm
215            14:51:57:065  Workspace  -         Loading model: D:\CrypTool2\Templates\Experimental\AESVisualisierung.cwm
217            14:52:41:778  Workspace  -         Execute model now!
```

Info: 14:52:41:778: Execute model now!

Figure 30: The plugin after completion

# 6 Future Development

As the plugin described in this thesis is not the final version, I will explain in this chapter how the plugin could be further improved in the near and the more distant future.

In the near future (the next couple of weeks) I will add the following missing features:

- Support for 192 bit and 256 bit keys. This is by far the most important feature but also the one requiring the most work. It will take changes in the interface (e.g. adding other round buttons) but also changes in the logic. Both, the method and the interface of key expansion will be completely redesigned.
- Scaling the text within objects like matrices when their size changes.
- Adoptions to support color-blind people (barrier-free accessibility). There should be a way to highlight buttons, values etc. using other mechanisms than color. One option is to use a pattern of strokes. The idea is to create two more buttons the user can use to toggle color and pattern. This will enable the user to use either one, both or neither.
- Another advanced feature is to add a third optional input representing a second text. In that text the same input string can be put in however with some minor changes (for instance in one or two characters/bytes). This second text will also be encrypted but not visualized. During the visualization of the first text, the plugin will compare the states of the two texts and highlight the differences to show the effect of changing one byte (bit).
- Enhance the in-program-help (online help) by explaining how to use the plugin, and by supporting another language (German as a second language in addition to English).

Additional features which could be developed in the long run are an enhanced navigation that allows the user to change the round or operation while an operation is currently being processed.

## References

[1] `https://www.cryptool.org/en/cryptool2`, Accessed at 2016-06-07

[2] Enrique Zabala: Rijndael Animation and Rijndael Inspektor, May 2007

[3] AESvisual.
`http://www.cs.mtu.edu/~shene/NSF-4/AES-Downloads/index.html`,
Accessed at 2016-06-08

[4] Jun Ma, Jun Tao, Melissa Keranen, Jean Mayo, Ching-Kuang Shene, and Chaoli Wang: AESvisual: A Visualization Tool for AES Cipher, 2014.
`http://www.cs.mtu.edu/~shene/NSF-4/AESvisual.pdf`

[5] Mostafa I. Soliman and Ghada Y. Abozaid: Hardware Visualization of the Advanced Encryption Standard (AES) Algorithm, October 2008

[6] S. Przybylski, A. Wacker, M. Wander, F. Enkler, and P. Vacek: Plugin Developer Manual – How to build your own plugins for CrypTool 2, March 2016.
`https://www.cryptool.org/trac/CrypTool2/wiki`, Accessed at 2016-06-07

## Eidesstattliche Erklärung / Affirmation

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer meiner Studien- oder Prüfungsleistungen war.

Mannheim, June 14th, 2016

Matthias Becher (Verfasser)