UNIVERSITY OF MANNHEIM

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

Bachelor's Thesis

Visualization of the Avalanche Effect in CT2

as part of the degree program Bachelor of Science Wirtschaftsinformatik

submitted by

Camilo Echeverri
cechever@mail.uni-mannheim.de

on October 31, 2016 (2nd revised public version, Apr 18, 2017)

Supervisors: Prof. Dr. Frederik Armknecht
Prof. Bernhard Esslinger

Abstract

Cryptographic algorithms must fulfill certain properties concerning their security. This thesis aims at providing insights into the importance of the *avalanche effect* property by introducing a new plugin for the cryptography and cryptanalysis platform *CrypTool 2*.

The thesis addresses some of the desired properties, discusses the implementation of the plugin for modern and classic ciphers, guides the reader on how to use it, applies the proposed tool in order to test the *avalanche effect* of different cryptographic ciphers and hash functions, and interprets the results obtained.

Contents

List of Abbreviations

List of Figures

List of Tables

1 Introduction

Encryption of messages, born from the human need to protect information has been practiced since ancient times. Since the introduction of computing devices and its rapid development new possibilities emerged in terms of coding techniques, allowing the creation of more elaborated and efficient encryption algorithms. Progress in technology however also facilitates the breaking of encrypted data. This is why a high quality of encryption algorithms is continuously sought after.

In order to obtain the desired quality, ciphers (especially block ciphers) and hash functions should exhibit certain properties, including the *avalanche effect (AE)*, first introduced by Feistel [4]. This property is measured as the reaction of the encrypted output caused by a small change in the input message (plaintext), in the key, or in the initialization vector (IV). Ciphers possess a strong *AE* when a single bit change in the plaintext results in a significant change of bits in the ciphertext. Ciphers without a high degree of the *AE* property are subject to cryptographic attacks which can make predictions about the input, being given only the output (in particular, the algorithm is vulnerable to chosen-ciphertext attacks). This may be sufficient to partially or completely break the algorithm (and if so, it's much more effective than *brute-force* attacks, a generic attack, in which the attacker checks for all possible keys, decrypts the ciphertext and compares the results with the plaintext).

Another property secure block ciphers should have is the *completeness* property [6], which makes sure that a change in any plaintext bit affects all output bits. This means that every plaintext bit must contribute to the final value of each output bit [15].

The combination of both *avalanche effect* and *completeness* gives rise to a new concept known as the *strict avalanche criterion (SAC)*, which is satisfied when in average half of the ciphertext bits are changed whenever an input bit is switched [16].

Both properties can be achieved through a sequence consisting of several rounds of simple operations like substitutions and permutations, also known as *substitution-permutation network (SPN)* [4], whose basic concepts are applied in several modern cryptographic block ciphers like *AES* or *PRESENT*. A depiction of an SPN can be seen in **Figure 1**.
Another construction several ciphers make use of (e. g., *DES*, *Camellia*) in order to attain the desired criteria, is known as the *Feistel Network*, which uses a series of several rounds and splits the input into two halves. It alternates performing the operations on only one half while the other one remains unaltered [10].

The thesis at hand provides a visualization of the above mentioned *avalanche effect* as a plugin implemented for the software *CrypTool 2 (CT2)*, with the intention of facilitating the understanding of this property and its importance for the design of secure cryptographic algorithms.

1.1 CrypTool 2

CrypTool 2 (CT2) is an open-source e-learning platform that provides users with interactive tools either to gain insights into the fields of cryptography and cryptanalysis for the

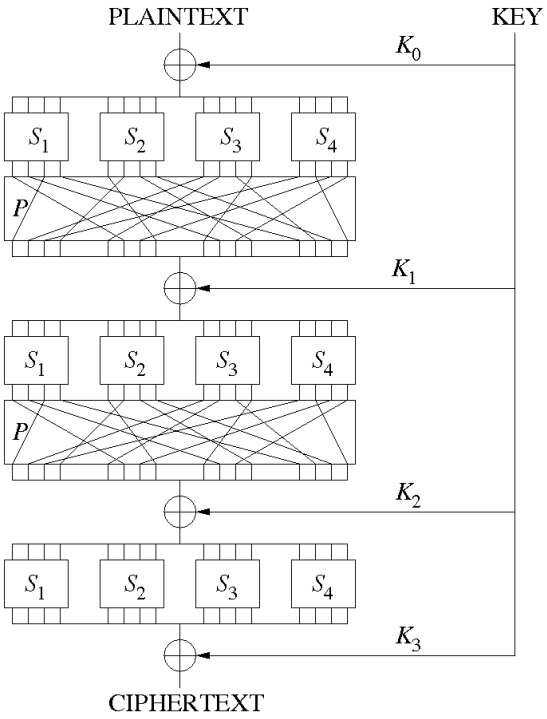


Figure 1: Substitution-permutation network [5]

amateur or to deepen their understanding on the subject for the more experienced.

CT2 uses the programming language C#, and is based on the Microsoft framework .NET with its Windows Presentation Foundation WPF for creating user interfaces and the development environment Visual Studio. How to write plugins for CT2 is described in [12].

With *CT2*, individuals can drag and drop single plugins that implement different cryptographic algorithms and cryptanalysis tools onto the program's workspace and connect them with other components. Each plugin is equipped with input and output *docking points* that enable the interaction and data exchange among them.

1.2 Outline of the Thesis

After the introduction the thesis briefly addresses the different properties related to the security of ciphers including the *avalanche effect* (AE). Subsequently it provides information on how the implemented tool is structured and how it works. It also points out related tools, as well as obstacles encountered during the implementation. Finally, it tests the introduced properties (especially the AE) on several cryptographic algorithms, as well as hash functions, and states some conclusions based on the outcome.

2 Properties of Secure Block Ciphers

As already mentioned secure cryptographic algorithms must exhibit a strong *avalanche effect* and a strong *completeness* property.

In addition, a good encryption process should also create complexity between its key, its plaintext, and its ciphertext, this is known as *confusion*. It should also scatter the changes made on the plaintext over the whole ciphertext, known as *diffusion*, both properties were introduced by Shannon [14].

Confusion is achieved by using substitution operations (*S-boxes*) and *diffusion* is created by using permutations with the purpose of creating a degree of randomness in the cipher in such a way that no patterns can be recognized, thus making it difficult for cryptanalysts to break the algorithm. The *avalanche effect* contributes to a good *diffusion*. Consequently a good *diffusion* can be accomplished by means of a high *avalanche effect*.

The goal is to make every single bit of the input affect every single bit of the output (*completeness*). Carefully selected and well thought-through layers combining *S-boxes* and scrambling bits over a certain amount of rounds help building strong ciphers that are less susceptible to statistical attacks. The final goal is not to disclose any connections or patterns in the cipher which could reveal the actual key of the encryption. The cipher should be unrecognizable, and it should appear as random as possible [15].

In this case flipping a single plaintext bit should result in a change of every single ciphertext bit with a probability of $\frac{1}{2}$ (*SAC*) [16]. Applying this property to modern block ciphers normally each block is considered separately.

2.1 Avalanche Effect

The *avalanche effect* (AE) can be measured dividing the number of switched bits by the number of total bits in the ciphertext [9] [11].[1,2]

$$\frac{Number\ of\ flipped\ bits\ in\ ciphertext}{Number\ of\ total\ bits\ in\ ciphertext}$$

2.2 Completeness

The *completeness* property is satisfied if each single output bit depends on every single input bit. If this is the case one can conclude that if a cipher is *complete* it also fulfills the *SAC*.

[1] In classical ciphers the objects are bytes instead of bits. So there we measure changed bytes instead of flipped bits.

[2] Remark concerning the comparison of the AE of different algorithms: From a methodically aspect and strictly speaking, we can only compare the effect onto 1 block for different block ciphers. If considering more than 1 block we depend on the additional chaining mode. In stream ciphers or with classical ciphers we have no defined block length. Hash algorithms by definition have a fixed length of the result and can be considered closest to block ciphers regarding the comparability of this statistical value.

3 Related Work

There are several projects dealing with the topic of the *avalanche effect*.

- The *Hash Demonstration* tool included in the open-source program *CrypTool 1 (CT1)* shows the effect a document modification has on a hash value. It provides statistical data based on the comparison between the initial and the modified input. The user can select between different hash functions [3].
- With Knape's web application *AEStetic* users can visualize – by clicking on a specific byte – which other bytes it depends on during the entire *AES* encryption process (so it looks back while the *AE* considers forward looking). Though it does not directly compare changes, it does offer insights on how changes propagate throughout the encryption process [8].

The here implemented *Avalanche Visualization (AV)* plugin is more generic: It provides statistical data as well, illustrating the effect of changes not only on hash values, but also on all different modern (symmetric) and classic cryptographic algorithms available in *CrypTool 2* (CT2). It also allows the user to walk through the encryption rounds, testing the *avalanche effect* on each stage of the encryption process (in the case of *AES* and *DES*). The functioning of the plugin will be described in more detail in the next section.

4 Plugin Design and Implementation

For the implementation of the plugin the IDE Visual Studio Community 2015 by Microsoft was used. The plugin was developed in the programming language C#.

The visualization is built upon the plugin architecture provided by *CrypTool 2* (CT2). Everybody can download the sources of the software CT2, and any developer can contribute to the expansion of the software.

The main purpose of the *AV* plugin is to facilitate the understanding of the requirement of the *avalanche* property for (modern) ciphers in an illustrative way.

4.1 General Description of the Plugin

The *AV* plugin was designed first to show the effect in detail for ciphers (methods) prepared for this purpose, and secondly to show the *avalanche effect* regarding the final result of any cipher and hash method (where no extra preparation has to be undertaken in advance). Therefore, the *AV* plugin deals with two categories of methods: *prepared methods* and *unprepared methods*.

The plugin has two input docking points of type *ICrypttoolStream*, one for the key and one for the message. Whenever the *prepared methods* option is selected, both input docking points must be used. For the *unprepared* option only the second input docking point (message) is used.

- For *prepared methods* the *Avalanche Visualization* (AV) plugin knows the method in detail. So it does not only show the statistics of the two results to be compared but also the statistics of all intermediate steps (like single rounds).

  This is currently implemented for *AES* and *DES*, but could also be done for hash functions. In this sense the *AV* plugin is expandable. For ciphers this category needs two inputs: not only the ciphertext, but also the key. This category also requires the user to choose the "Selection" parameter (*DES* or *AES*).

- For *unprepared methods* only two consecutive input messages are compared (no key can be handed over to the *AV* plugin. The messages come from the input chain – a method component (either a cipher plugin or a hash plugin) and its respective input components (plaintext and key).

  The 1st message is what the *AV* plugin gets from the input chain after pressing the "Play" button (located at the top of the CT2 window); the 2nd message is what the *AV* plugin gets after the user changes the input (e. g. the plaintext) of the connected method plugin. Afterwards, the *AV* plugin just shows in one screen statistics derived from the differences between the two messages.

The current default is: If both input entries (message and key) are delivered, the *Avalanche Visualization* plugin assumes the prepared method *AES-128*. If only the message entry gets input, then the plugin selects "Hash functions" from the *unprepared methods* category.

Category	**Use of Docking Point**	**Selection**	**Compared Object**
Prepared Methods	Message and Key	AES	Bits
		DES	
Unprepared Methods	Message only	Classic Ciphers	Bytes
		Modern Ciphers	Bits
		Hash Functions	

Table 1: Categories that can be selected with their respective parameters

Table 1 exemplifies both categories and their respective parameters according to the user's choice.

4.2 Prepared Methods

The *prepared methods* option comprises both the *Advanced Encryption Standard* (*AES*) and the *Data Encryption Standard* (*DES*).

4.2.1 AES and DES

AES uses a default key length of 128 bits, but it can also be changed to 192 or 256. *DES* uses a 64-bit key. The whole component can be divided in five main views that emerge while running through the plugin.

1. **Home View**

   It displays the main title with a brief description of what the *avalanche effect* is. It also indicates the possibility of adjusting the settings depending on what will be done next. The *home* view with two inputs (key and message for *AES*) is depicted in **Figure 2** .

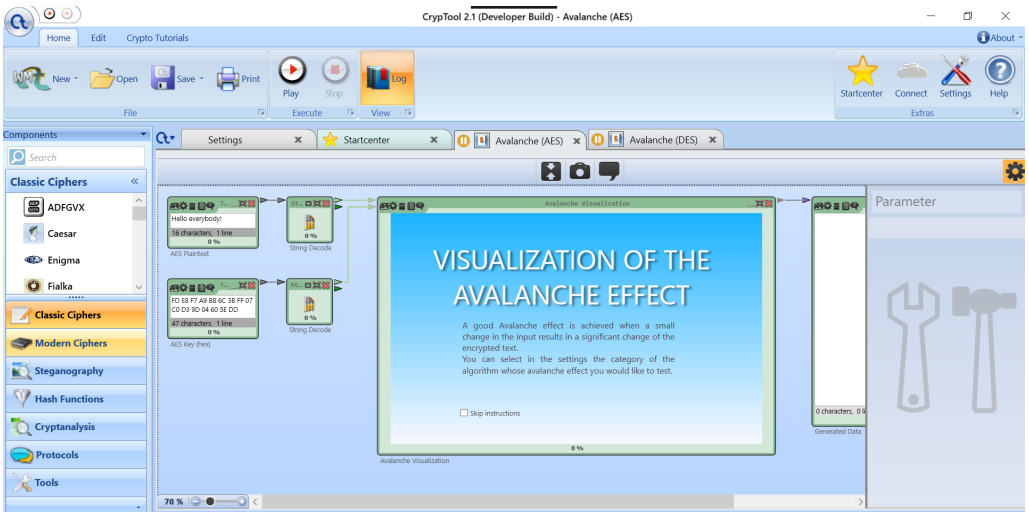
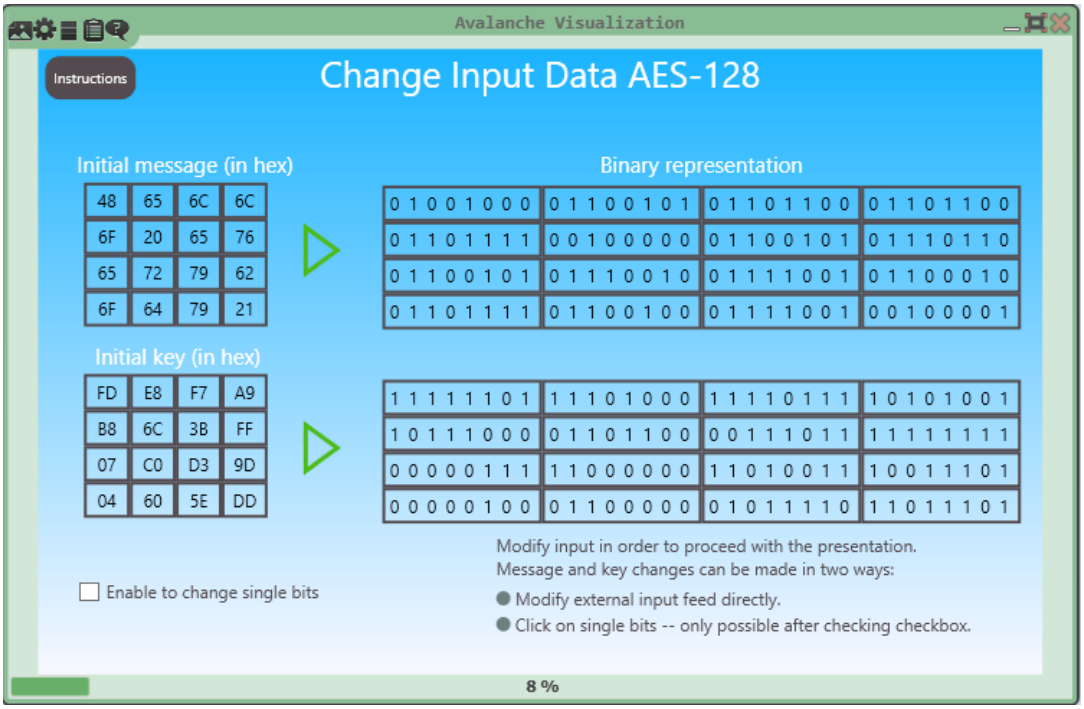


Figure 2: Home view of the *Avalanche Visualization* plugin within the CT2 environment

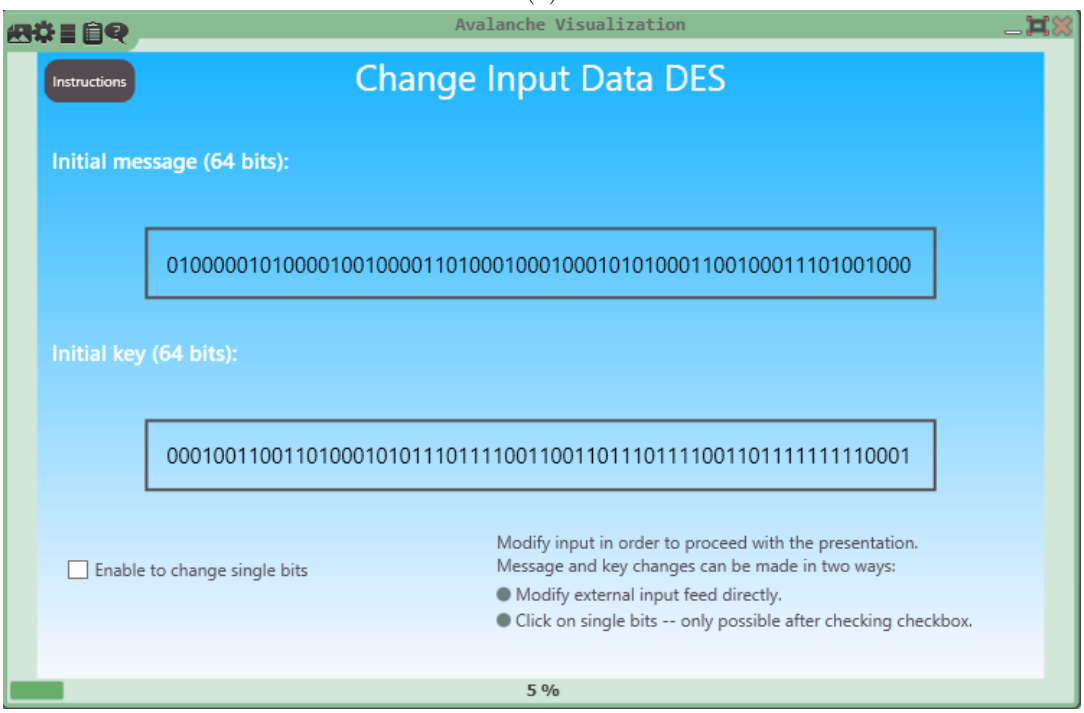
The three views (data input, comparison and general overview) are only available for prepared methods (so currently only when testing the *avalanche effect* of *AES* or *DES*). The views (home and avalanche effect) are also available for unprepared methods.

2. **Data Input View**

   Upon clicking the “Play” button located at the top of the CT2 window the *home* view disappears and the input data (key and message) are shown. The depiction of the *AES* input slightly differs from the *DES* representation. This can be seen in **Figure 3**.



(a)



(b)

Figure 3: (a) Input view *AES*, (b) Input view *DES*

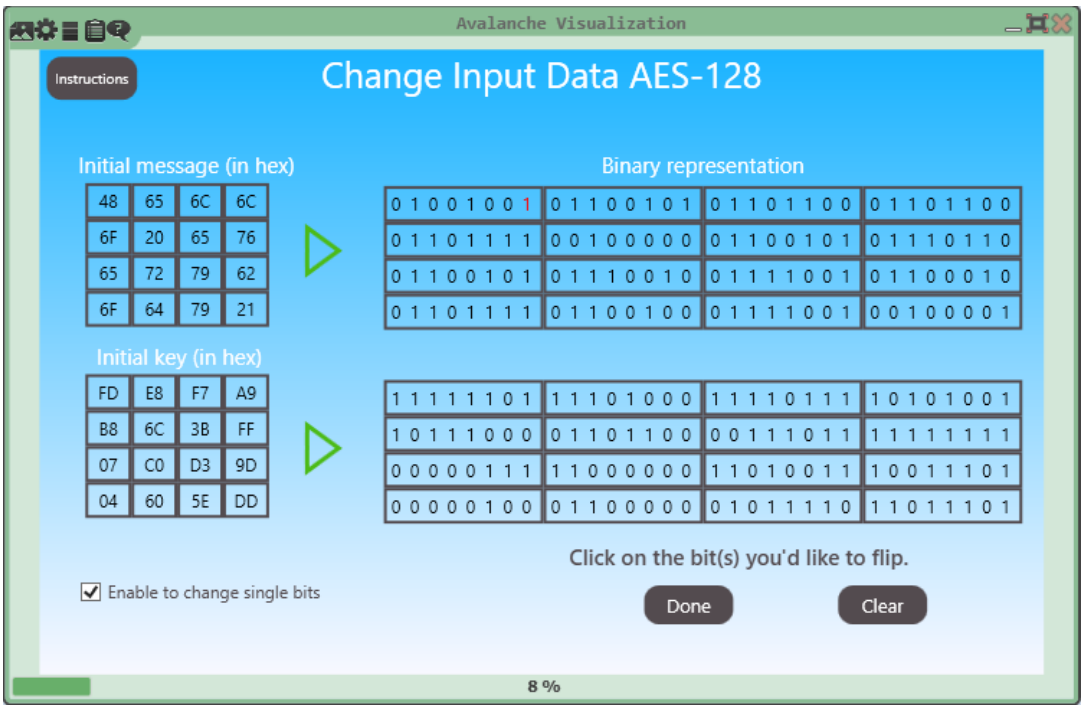


Figure 4: Flipping of one message bit for AES-128

A short text explaining the next step is present in both cases. In order to proceed there are two possibilities:

- Changes can be made by directly modifying the two external input plugins containing the key and the message to be encrypted.
- Single bit changes can be made by clicking on a specific bit after enabling this feature.

The single-bit-change feature is enabled by checking the check box *Enable to change single bits*, located on the lower left hand side corner. Subsequently the user can click on single bits representing the key or the message in order to flip them. They turn red. **Figure 4** depicts this action for *AES*.

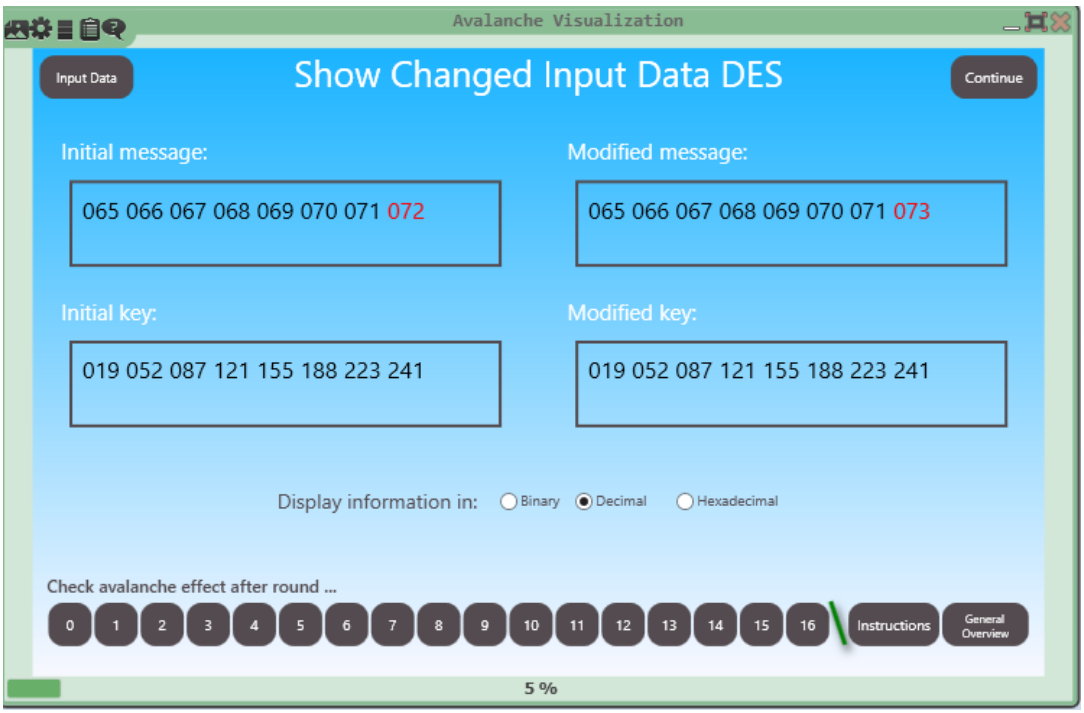


Figure 5: *DES* input data in decimal values with highlighted changes

3. **Comparison View**

   Upon changing the initial message and/or key a new view emerges, depicting the original and modified message and key with the respective changes highlighted in color.

   Up to this point the information can also be displayed in binary, decimal, or hexadecimal format in the case of *DES*, and in decimal and hexadecimal format in the case of *AES*. This can be seen in **Figure 14**. The binary representation for *AES* in this view is not available, since it would take a big part of the screen, becoming rapidly unclear and difficult to see at a glance.

   The user can now check the strength of the *avalanche effect* after each single round by clicking on any of the numerated buttons located on the lower part of the window. See **Figure 5**.

   The number of rounds displayed on the buttons panel varies according to the selected encryption standard and key length.

   The *Input Data* button leads right back to the *Data Input* view where single bits can be flipped.

   Upon clicking the *General Overview* button, a general overview of the encryption process is displayed on the window. It is described in more detail later on. The buttons panel with all rounds as well as the *Input Data* and the *General Overview* buttons are illustrated in **Figure 5**.

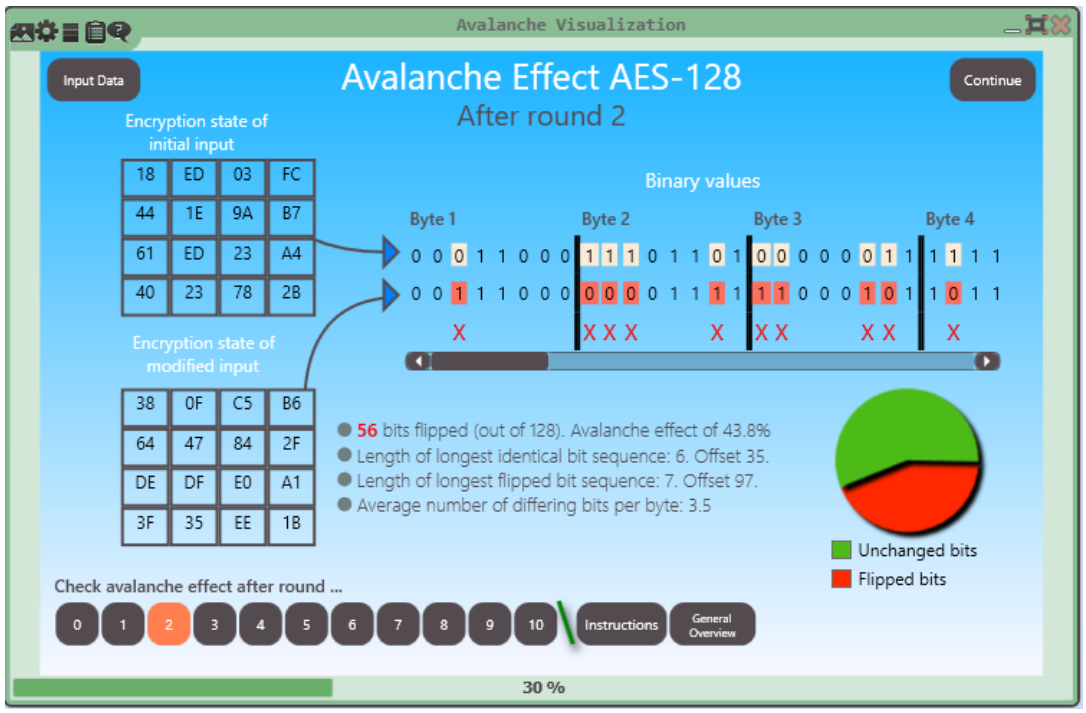


Figure 6: Components shown in the avalanche effect view of *AES-128*

4. **Avalanche Effect View**

   The view after each round comprises the following 4 components:

   (a) The current state of the cipher for the initial and the modified input, shown in two different rows respectively (binary format).
   Remark: In the case of *AES* the current states are also shown in a 4x4 matrix in hex format.

   (b) A third row contains a letter x whenever there is a bit difference between the original and the modified message, as to allow the user to visualize which bits have been flipped.

   (c) Statistical data: number of switched bits after completion of current round, value of the avalanche effect for the current round, length of the longest identical bit sequence (unchanged bit sequence) with its offset, length of the longest flipped bit sequence (changed bit sequence) with its offset, and the average number of differing bits per byte.

   (d) Pie chart depicting the ratio between changed and unchanged bits. After placing the mouse on each colored area a tooltip emerges showing the corresponding percentages.

   Figure 6 shows all components described above.

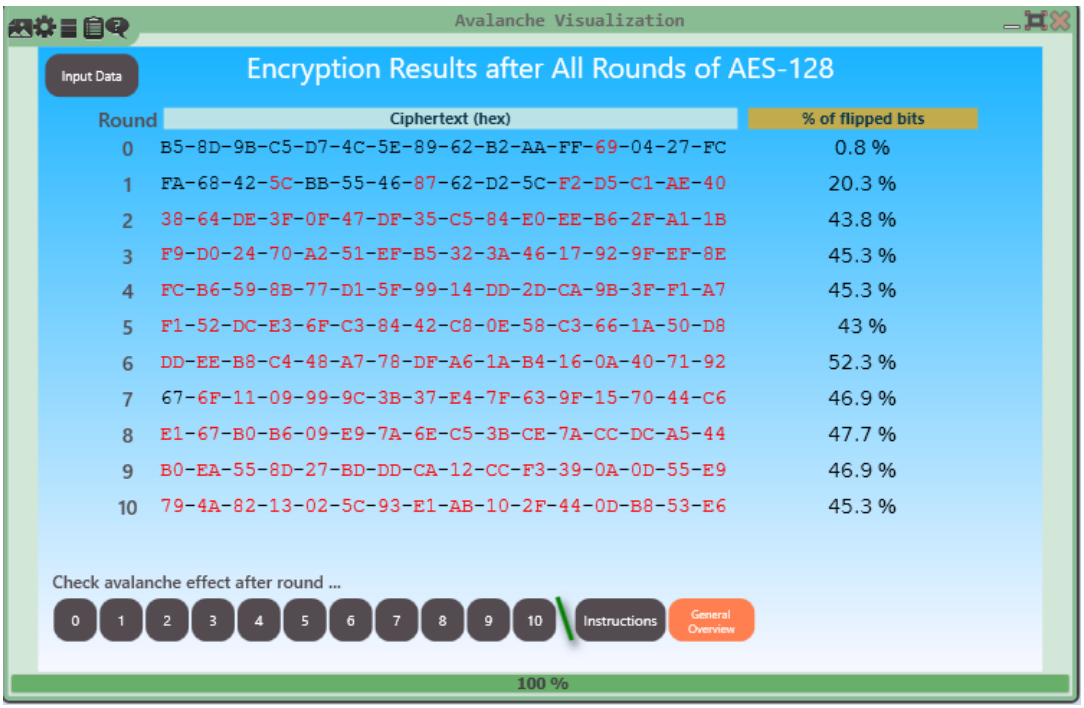


Figure 7: General overview of *AES-128*

5. **General Overview**

   In the case of *DES*, it depicts all binary values after each round. In the case of *AES*, it shows all bytes (hex values) after each round. The bits/bytes (that are changed after the input has been modified) are highlighted in red. This is depicted in **Figure 7**.

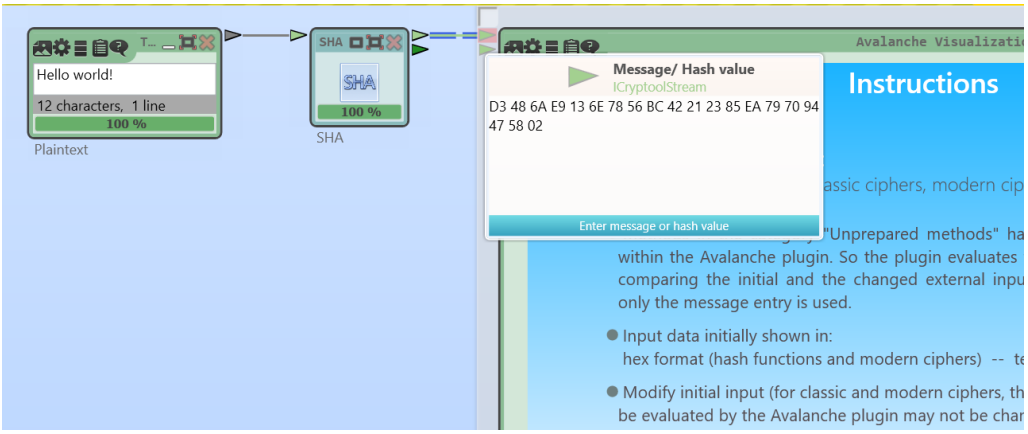


Figure 8: Input data provided by *SHA*

4.3 Unprepared Methods

As seen in **Table 1** the *unprepared methods* category comprises classic ciphers, modern ciphers, and hash functions.

4.3.1 Classic Ciphers, Modern Ciphers, and Hash Functions

By selecting any of the three options *Classic Ciphers*, *Modern Ciphers*, or *Hash Functions* the *Avalanche Visualization* plugin processes data produced by any external component belonging to any of those options. The input is provided using only one input docking point (the one for message/ hash value, as there is no key input used by the *Avalanche Visualization* plugin under this category option – see **Table 1** on page 13).

As seen in **Figure 8** the *SHA* plugin receives the plaintext and subsequently delivers the hash value to the *Avalanche Visualization* plugin by using only the upper input docking point (message/ Hash value). In **Figure 9** the *Avalanche Visualization* plugin gets the encrypted message from the *RC4* plugin after *RC4* has received all input necessary to carry out the encryption.

After the external input feed is modified there is only one single view (except for the *home* view) on the presentation containing the following elements:

- Initial and modified value of hash function or encrypted message.
- Binary representation of these values. Here, a bit-by-bit comparison takes place in order to signalize which bits have been flipped.
- Statistical data and pie chart (as seen in the previous section).

The user can make changes on the input by directly modifying the external plugin containing the initial message to be encrypted or hashed. Subsequently the modified message or modified hash function is displayed on the presentation and the flipped bits are signalized by the red letters x.

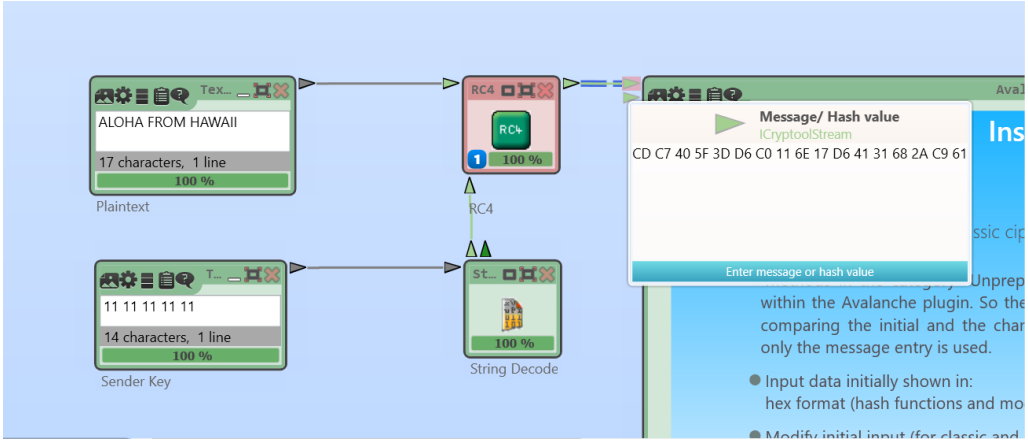


Figure 9: Input data provided by *RC4*

For modern ciphers and hash functions the input is initially shown in hex format whereas for classic ciphers the input is displayed as text. Afterwards the format can be changed via radio button. Information concerning the strength of *avalanche effect* is also depicted. **Figure 10** illustrates the view with all its elements and the input provided by the *SHA* plugin.

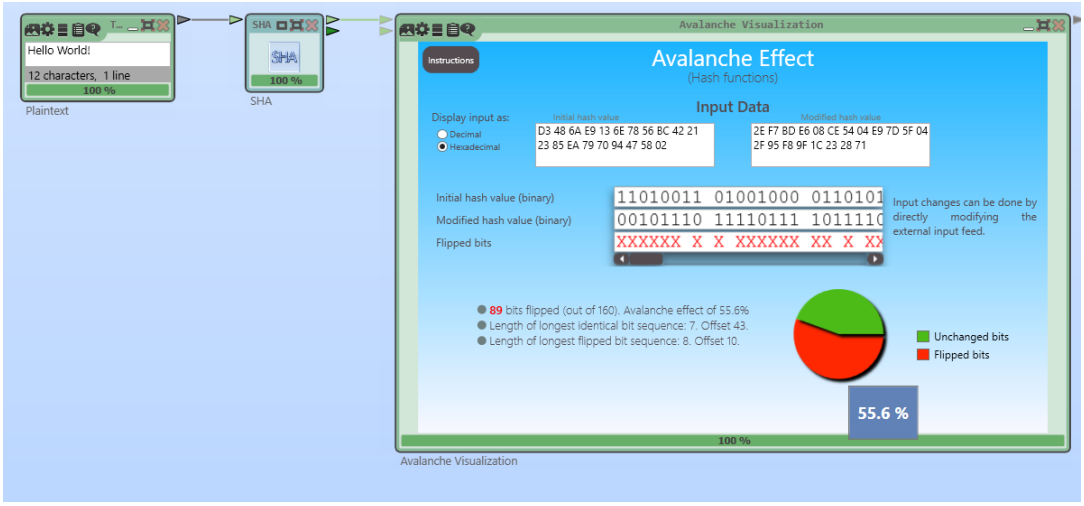


Figure 10: Testing the avalanche effect of SHA

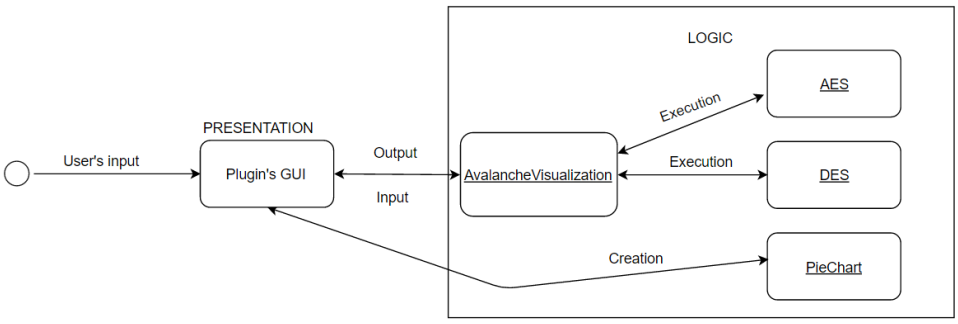


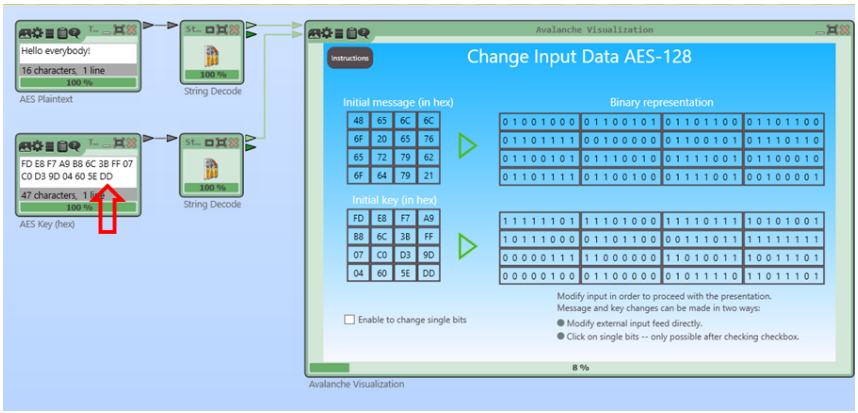
Figure 11: Simplified architecture diagram of the *Avalanche Visualization* plugin [7]

4.4 Architecture of the Code

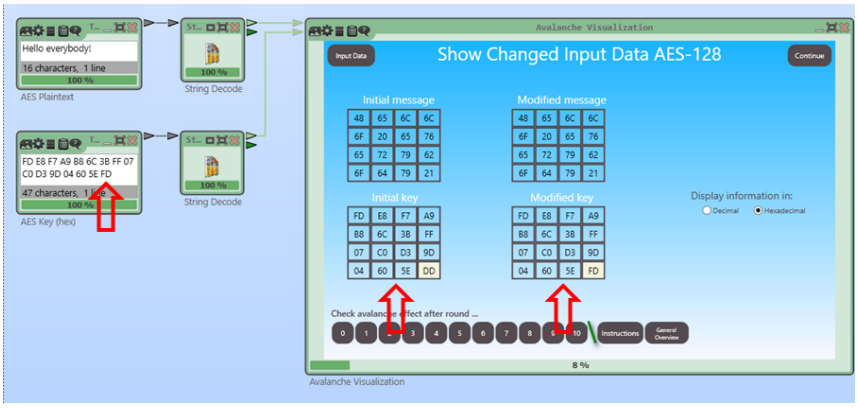
A big part of the program's logic takes place in the *AvalancheVisualization* class, which acts as a mediator between the presentation and the *AES* and *DES* classes in order to display the desired data on the UI, according to the selected category. The *PieChart* class is used for the creation of the pie chart mentioned before, which is used directly by the presentation as a UI control. A simplified depiction of the interaction between classes can be seen in **Figure 11**.

The code of the *prepared methods AES* and *DES* was already available in CT2 – it is the same that is used in the two plugins *AES Visualization* and *DES Visualization* in regards to their core functionality. This initially posed some trouble with respect to the necessary functions calls for *AES*, since a big part of the implementation was only specific to the *AES Visualization* plugin. This was quickly overcome and all code blocks dealing with operations different from the calculation of the cipher were not taken into account. So the code exists only once there, and here the respective function calls are used.

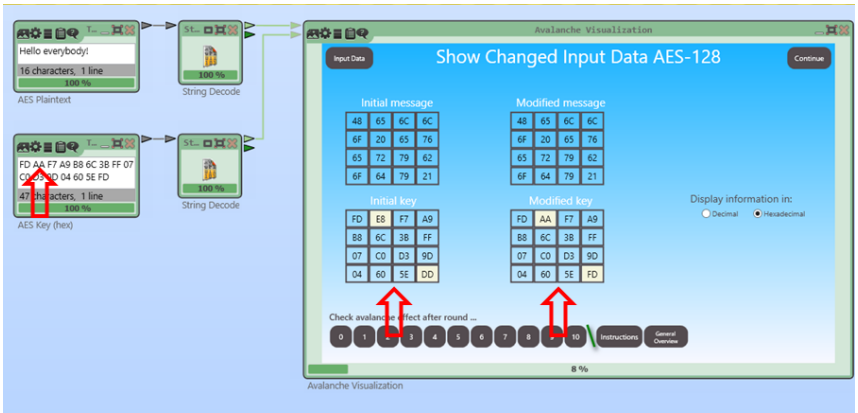
The *Avalanche Visualization* plugin uses from each of its input docking points the very first (right after the template has been started) and the very last input (after the user modifies the input feed). Obtaining a new input after a user makes changes somewhere in the input chain is a feature of the CT2 workspace the plugin developer gets for free. So you easily can handle consecutive inputs via the same docking point. The workflow of this occurrence is depicted in **Figure 12** with the initial key (very first one) signalized with a red arrow in 12a together with the *Data Input* view. In 12b the last hex number of the key is modified and the changes are shown in the *Comparison* view. Subsequently the second hex number of the key is modified and the current changes can be seen as well (12c). These modifications could also be made with the input message.



(a)



(b)



(c)

Figure 12: Screenshots after (a) initial input, (b) first modified input, (c) last modified input

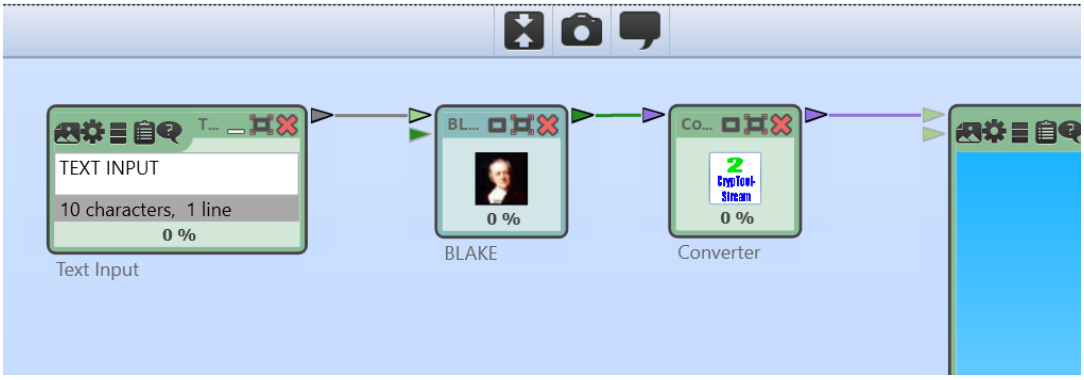


Figure 13: Using the *Converter* plugin to make both data types (*byte[]* and *ICryptoolStream*) compatible

4.5 Limitations and Future Work

Interaction with *Keccak* plugin
The only plugin which caused a problem when used as input was the *Keccak* plugin. Here the interaction between the *Avalanche Visualization* and the *Keccak* plugin didn't work: As soon as the "Play" button is pressed the progress bar on top of the CT2 workspace partially fills and after a while the whole CT2 environment freezes. The CT2 team was notified about that: As it is not caused by the *Avalanche Visualization* plugin, the CT2 team accepted to fix this.

External plugins with output of type *byte[]*
The two *ICryptoolStream* input docking points allow the *Avalanche Visualization* plugin to receive data directly from any plugin besides from those plugins whose output has the type *byte[]*. This incompatibility was easily circumvented by using the *Converter* plugin in between to convert the output to *ICryptoolStream*. For example: The *Converter* is used to provide the output from the *BLAKE* hash function to the *Avalanche Visualization* plugin (see **Figure 13**).
As a future work this could also be handled within the *Avalanche Visualization* plugin by changing its input docking points type from *ICryptoolStream* to *Object*. A quick try wasn't successful.

Enhancements requested in the submitted thesis, which have been resolved in the meantime:

- Provide an output stream of the generated statistical data.
- The second limitation mentioned above could be fixed by providing an *ICryptoolStream* output to each of those plugins in question, by changing the input docking points from the *Avalanche Visualization* plugin from type *ICryptoolStream* to *Object* as already mentioned, or by simply letting it as it is and using the *Converter*.

5 Analysis Based on the Implemented Tool

In this section different tests are carried out regarding the *avalanche effect* of different cryptographic algorithms as well as hash functions.

5.1 Avalanche Tests for AES

5.1.1 AES-128 (Modified Message, Constant Key)

For this test, a single message bit is flipped within the *Avalanche Visualization* plugin.

128-bit key (in hex): FD E8 F7 A9 B8 6C 3B FF 07 C0 D3 9D 04 60 5E DD.
Initial input message (in text): "Hello everybody!"
Input message (in hex): 48 65 6C 6C 6F 20 65 76 65 72 79 62 6F 64 79 21.

After the input message is shown in its corresponding hex and binary values the last bit of 48 (01001000) is flipped withing the presentation, resulting in the slightly modified message 49 65 6C 6C 6F 20 65 76 65 72 79 62 6F 64 79 21. Upon clicking on the *Done* button the changes are visible and highlighted in form of a 4x4 matrix. This is depicted in **Figure 14**. Since 49 is the hex value of the letter I, the same effect could also have been achieved by directly changing the letter H (48) into the letter I (49) in the external message input.

The *avalanche effect* of the cipher after applying the first *AES* round can be seen in **Figure 15**, where 13 bits are changed in total. This means the *avalanche effect* accounts for 10.2%.

After the second round 67 bits are complemented. Delivering an *avalanche effect* of 52.3%. After applying all remaining rounds the *avalanche effect* on each of them reaches values of nearly 50% or more. The longest out of all longest identical sequences (LIS) measured from all rounds consists of 29 bits. It can be seen after the first round. The second longest LIS have each a maximum of 7 bits after rounds 2, 6, and 8. (see **Table 2**).

Round	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Flipped Bits	13	67	68	72	69	61	75	53	68	66
Aval. Effect	10.2%	52.3%	53.1%	56.3%	53.9%	47.7%	58.6%	41.4%	53.1%	51.6%
LIS	29	7	4	4	4	7	6	7	4	5

Table 2: Results after testing *avalanche effect* of *AES-128* (modified message, constant key)

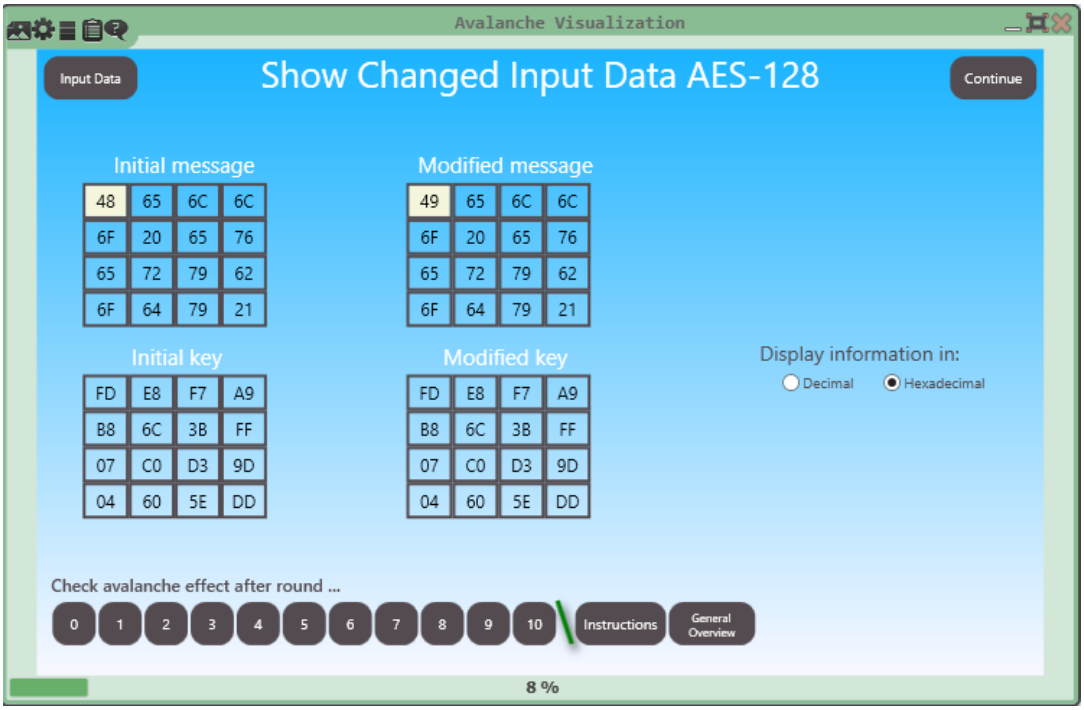


Figure 14: Comparison between initial and modified message of *AES-128*

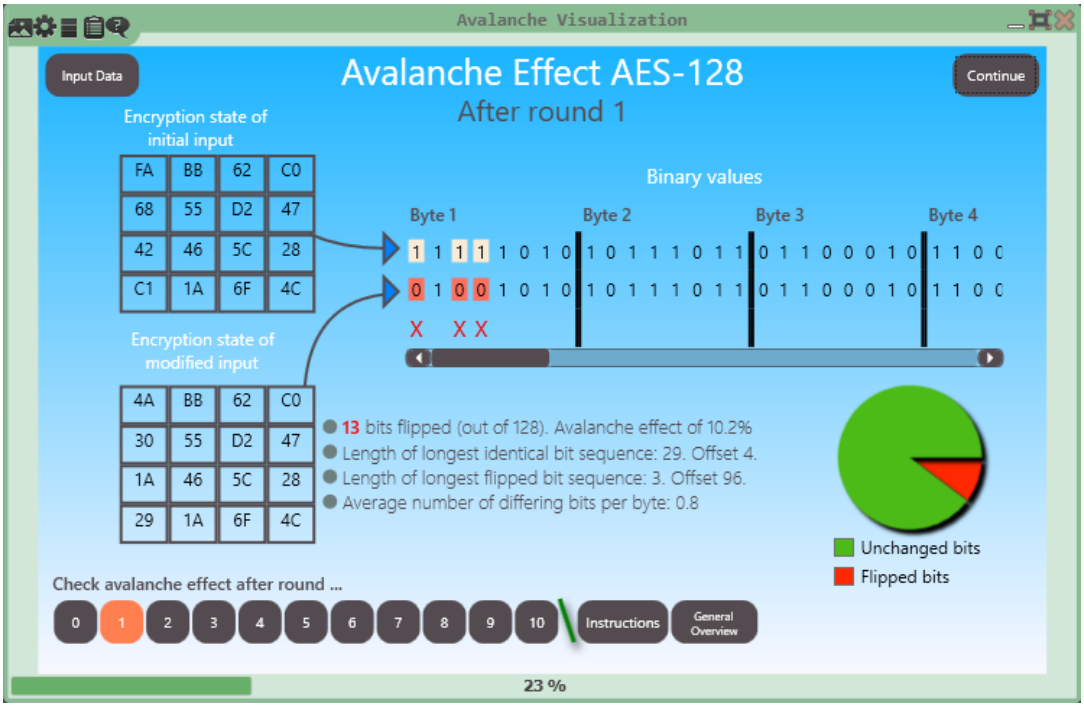


Figure 15: *Avalanche effect* after 1st round of *AES-128*

5.1.2 AES-128 (Constant Message, Modified key)

For this test, changes are made on the external input plugin delivering the key.

Initial 128-bit key (in hex): FD E8 F7 A9 B8 6C 3B FF 07 C0 D3 9D 04 60 5E DD.
Input message (in text): "Hello everybody!"
Modified key: FD E8 F7 A9 B8 6C 3B FF 07 C0 D3 9D 04 60 5E AB.

Figure 16 illustrates the changes made on the key.

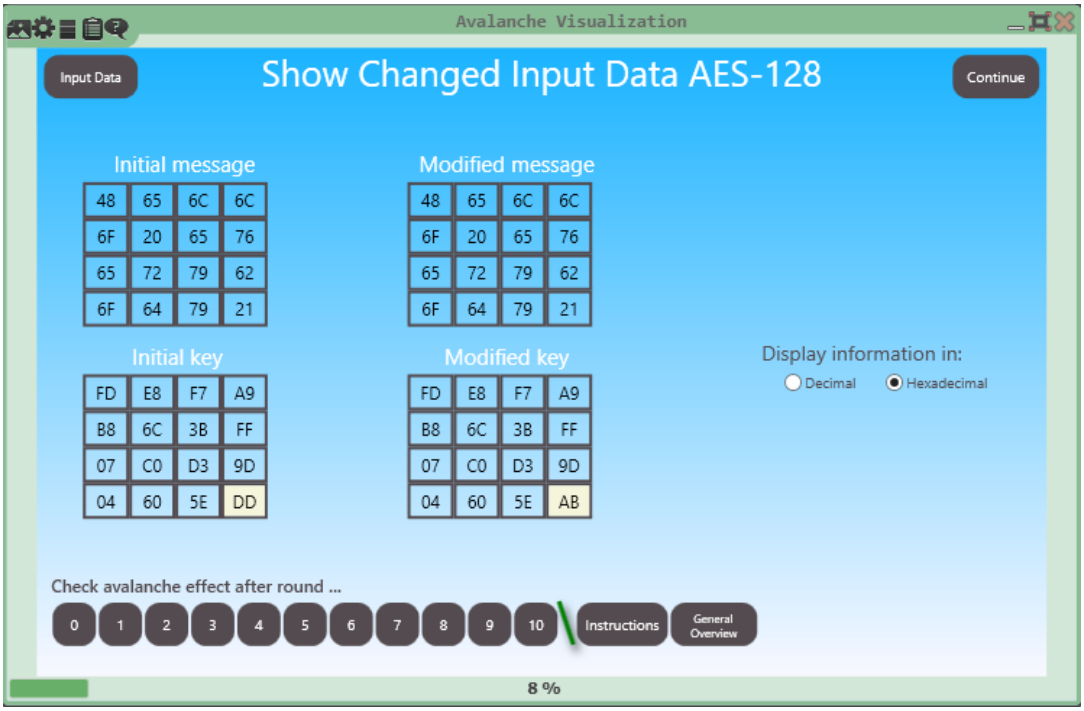


Figure 16: Comparison between initial and modified key

After running through all rounds the *avalanche effect* values can be seen in **Table 3**.

Round	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Flipped Bits	36	63	66	68	70	54	62	70	53	64
Aval. Effect	28.1%	49.2%	51.6%	53.1%	54.7%	42.2%	48.4%	54.7%	41.4%	50.0%

Table 3: Results after testing *avalanche effect* of *AES-128* (constant message, modified key)

5.1.3 AES-192 (Modified Message, Modified Key)

For this test, single bits from both (message and key) are flipped.

Initial 192-bit key (in hex):
FD E8 F7 A9 B8 6C 3B FF 07 C0 D3 9D 04 60 5E DD 13 34 57 79 9B BC DF F1.
Initial input message (in text): "Hello everybody!"
Input message (in hex): 48 65 6C 6C 6F 20 65 76 65 72 79 62 6F 64 79 21.

The very first bit of the key and the very last bit of the message are flipped within the *Avalanche Visualization* plugin as seen in **Figure 17**. This change yields:

Modified key:
7D E8 F7 A9 B8 6C 3B FF 07 C0 D3 9D 04 60 5E DD 13 34 57 79 9B BC DF F1.
Modified message (in hex):
48 65 6C 6C 6F 20 65 76 65 72 79 62 6F 64 79 20.

The test delivers the values depicted in **Table 4**.

Round	1	2	3	4	5	6	7	8	9	10	11	12
Flipped Bits	17	68	63	65	60	73	66	68	62	66	65	58
Average Effect	13.3%	53.1%	49.2%	50.8%	46.9%	57.0%	51.6%	53.1%	48.4%	51.6%	50.8%	45.3%
LIS	29	7	5	5	5	7	6	7	7	6	8	6

Table 4: Results after testing *avalanche effect* of *AES-192* (modified message, modified key)

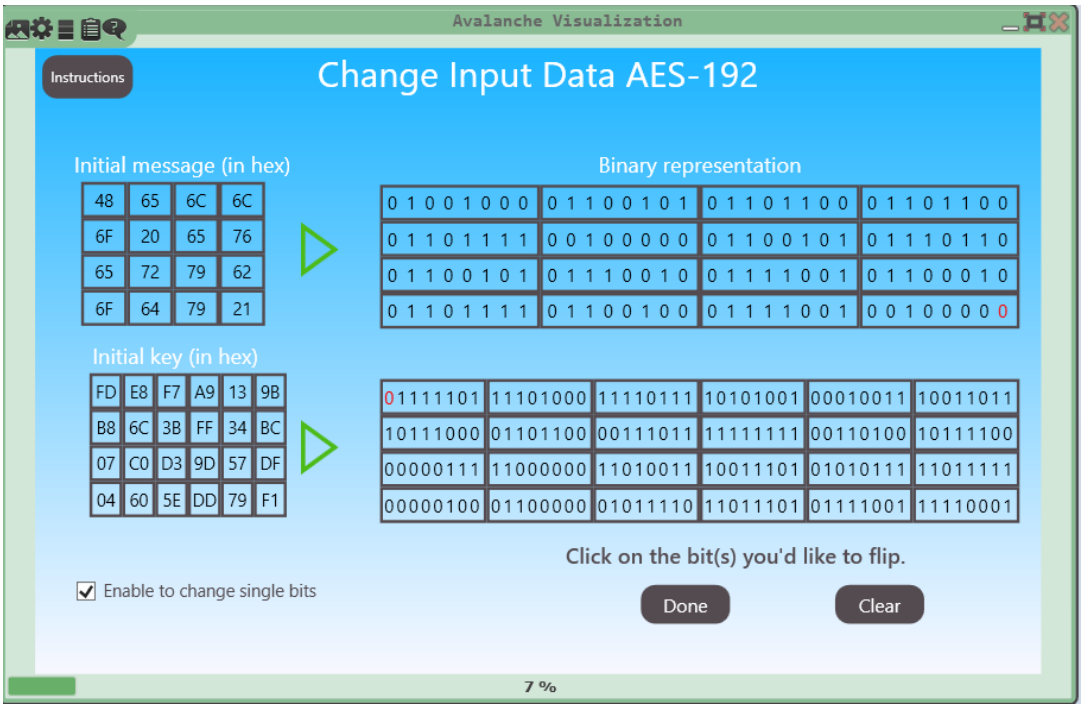


Figure 17: Flipped bits from key and message for *AES-192*

5.1.4 AES-256 (Modified Message, Constant Key)

For this test, changes are made on the external input plugin delivering the message.

256-bit key (in hex):
FD E8 F7 A9 B8 6C 3B FF 07 C0 D3 9D 04 60 5E DD 13 34 57 79 9B BC DF F1 2D 20 1E 7A 04 F2 11 C9.
Initial input message (in text): "Hello everybody!"
Input message (in hex): 48 65 6C 6C 6F 20 65 76 65 72 79 62 6F 64 79 21.

Modified message (in text): "Hello Everybody!"
Modified message (in hex): 48 65 6C 6C 6F 20 45 76 65 72 79 62 6F 64 79 21.

The results of the test can be seen in **Table 5**.

Round	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Flipped Bits	19	65	68	60	68	57	67	72	56	71	52	63	63	69
Average Effect	14.8%	50.8%	53.1%	46.9%	53.1%	44.5%	52.3%	56.3%	43.8%	55.5%	40.6%	49.2%	49.2%	53.9%
LIS	26	5	6	6	6	11	7	5	6	5	8	6	7	6

Table 5: Results after testing *avalanche effect* of *AES-256* (modified message, constant key)

5.1.5 Observations (*AES* Tests)

A single bit change of either the message or the key triggers a whole chain of changes throughout the encryption process. Right after applying round 0 (*AddKey* operation) there are very few bytes that are affected by the modification, in all of the test cases. After applying the first round of *AES* operations (*SubBytes*, *ShiftRows*, *MixColumns*, and *AddKey*) the number of bytes that are affected already increases. After the second round every single byte of the cipher is already affected by the initial bit/byte changes. This can be seen in a global overview in **Figure 18**

These small changes in key and/or message propagate rapidly throughout the encryption rounds, eventually affecting all of the *AES* bytes. Therefore, the cipher exhibits the *completeness* property introduced before. Consequently, it also reaches high values of *avalanche effect* in almost every round, starting at an early stage of the encryption process.

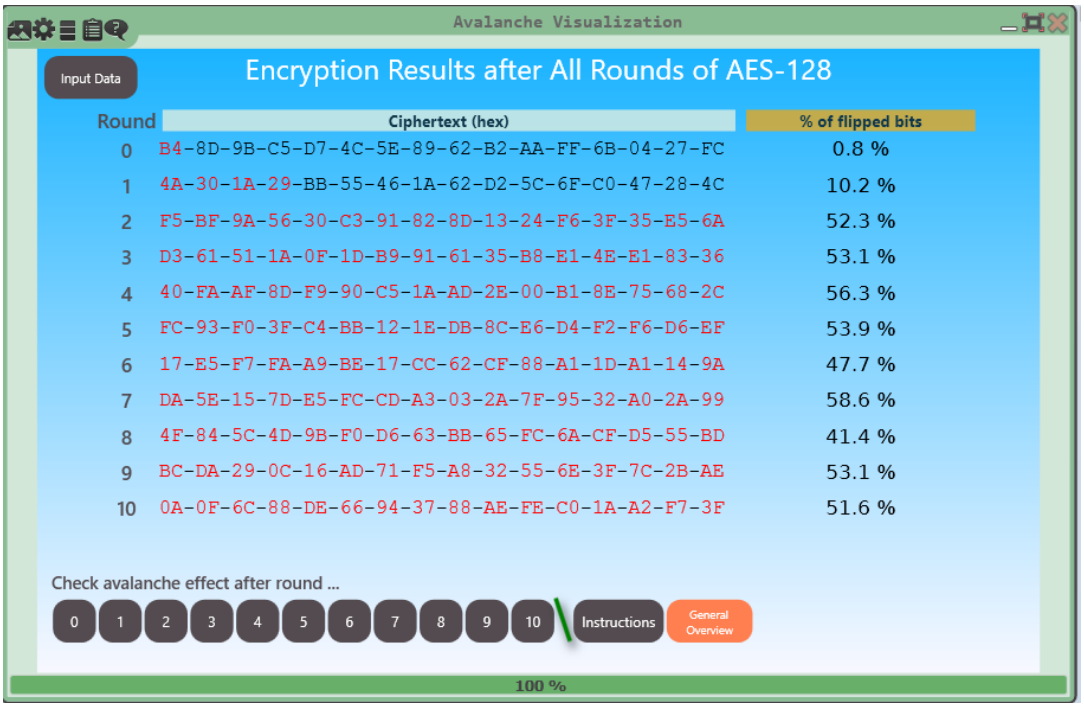


Figure 18: Affected bytes (shown in red) after a single bit has been flipped for *AES-128*

5.2 Avalanche Tests for DES

5.2.1 DES (Modified Message, Constant Key)

For this test, a single message bit is flipped within the *Avalanche Visualization* plugin.

64-bit key (in hex): 13 34 57 79 9B BC DF F1.
Initial input Message (in text): "Let's go"
Input message (in hex): 4C 65 74 27 73 20 67 6F.

The very last bit of the message is complemented, yielding:
Modified message: 4C 65 74 27 73 20 67 6E.

After the first DES round, there is only one complemented bit out of 64 bits (*avalanche effect* of 1.6%). After the second round, there are already 6 bits flipped (*avalanche* of 9.4%). After the sixteenth round, 25 bits are flipped (*avalanche* of 39.1%).

The values gained after checking all rounds of *DES* are illustrated in **Table 6**.

Round	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Flipped Bits	1	6	25	34	33	36	33	30
Average Effect	1.6%	9.4%	39.1%	53.1%	51.6%	56.3%	51.6%	46.9%
LIS	56	24	7	6	5	5	5	6

Round	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Flipped Bits	28	29	30	31	30	29	28	25
Average Effect	43.8%	45.3%	46.9%	48.4%	46.9%	45.3%	43.8%	39.1%
LIS	6	6	6	5	5	5	8	8

Table 6: Results after testing *avalanche effect* of *DES* (modified message, constant key)

Figure 19 depicts the flipped bits after the sixth round.

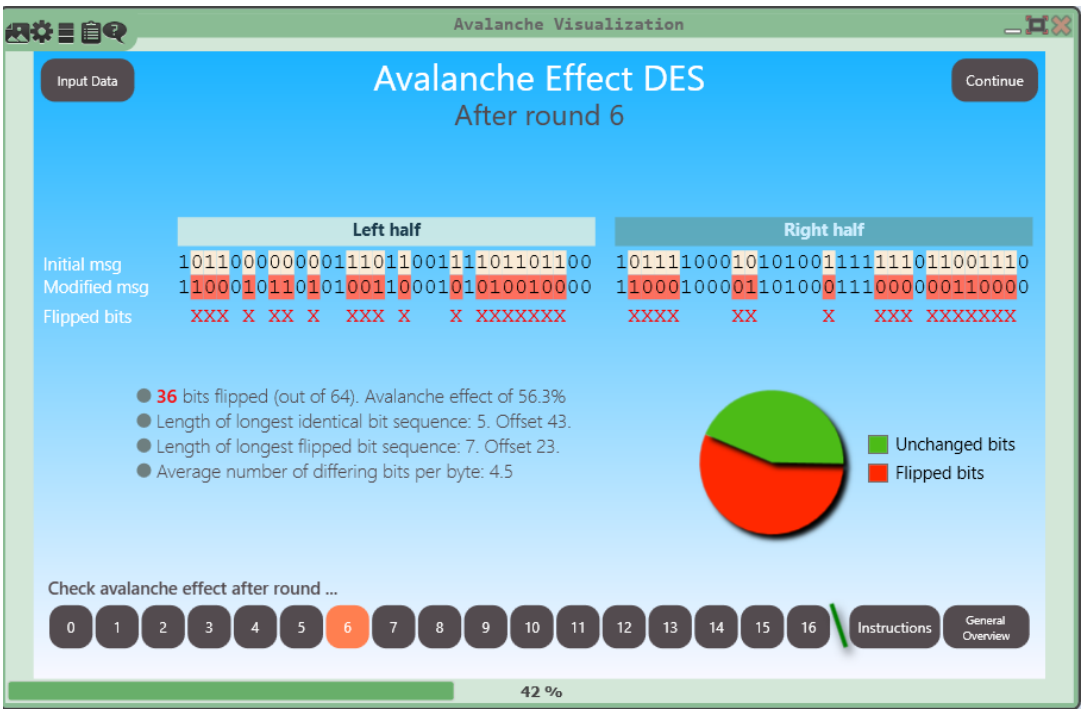


Figure 19: Avalanche effect after 6th round of *DES*

5.2.2 DES (Constant Message, Modified Key)

For this test, changes are made on the external input plugin delivering the key.

Initial 64-bit key (in hex): 13 34 57 79 9B BC DF F1.
Input Message (in text): "Let's go"

Modified key: 13 34 57 B4 9B BC DF F1.

After the first round there are 6 bits flipped. The results of all rounds are seen in **Table 7**.

Round	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Flipped Bits	6	20	27	28	30	32	39	40
Average Effect	9.4%	31.3%	42.2%	43.8%	46.9%	50.0%	60.9%	62.5%
LIS	37	9	6	4	3	4	4	4

Round	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Flipped Bits	30	28	34	37	35	30	37	42
Average Effect	46.9%	43.8%	53.1%	57.8%	54.7%	46.9%	57.8%	65.6%
LIS	8	8	8	4	6	6	4	3

Table 7: Results after testing *avalanche effect* of *DES* (constant message, modified key)

5.2.3 DES (Constant Message, Modified Key)

For this test, a single key bit is flipped within the *Avalanche Visualization* plugin.

Initial 64-bit key (in hex): 13 34 57 79 9B BC DF F1.
Input Message (in text): "Have fun"

The eighth bit of the key is complemented, yielding:
Modified key: 12 34 57 79 9B BC DF F1.

The following statistical data applies to all sixteen rounds.
No. of flipped bits: 0.

Consequently
Avalanche effect: 0.0 %
Longest identical sequence: 64 bits.

5.2.4 Observations (*DES* Tests)

During the first two *DES* tests, bit differences initially happen on only one half the 64-bit cipher, depending on the position of the bits flipped, that later on, land on a different position after round 0 (*initial permutation*) is carried out. **Figure 20** illustrates this occurrence during the first *DES* test. If more changes, scattered all over the message are made, the probability of getting bit differences on both halves, right after round 0 increases.

The number of complemented bits from each right half corresponds to the number of different bits from the left half of the previous round. This is attributed to the alternating nature of the *Feistel* structure introduced before.

In both first tests, starting from the third round on, the number of different bits gets close to 32, thus, showing an *avalanche effect* of nearly 50 %, and this in turn, satisfies the *strict avalanche criterion* (SAC). An overview of all the affected bits during the entire *DES* encryption process is portrayed in **Figure 21**.

Complementing only the 8th key bit, the third *DES* test yields a value of 0 for the *avalanche effect* and a *longest identical sequence* (LIS) of 64 bits for all encryption rounds. This is due to the fact that every 8th key bit is discarded; hence, it has no effect on the encryption process.

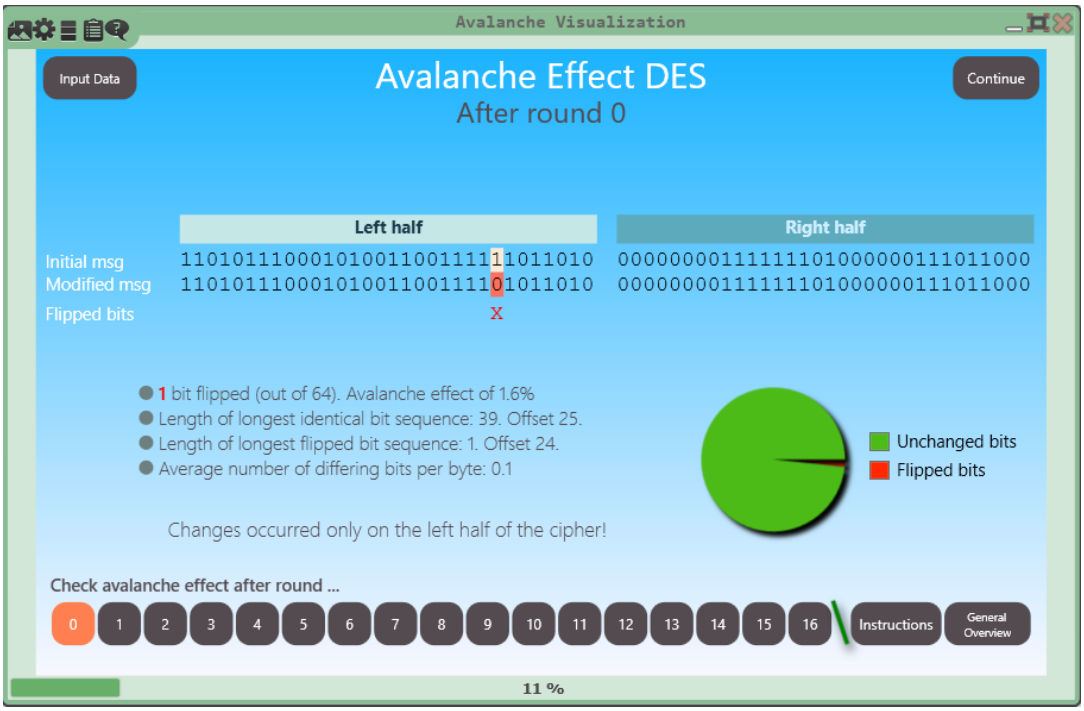


Figure 20: Bit difference only present on the left half after initial permutation (*DES*)

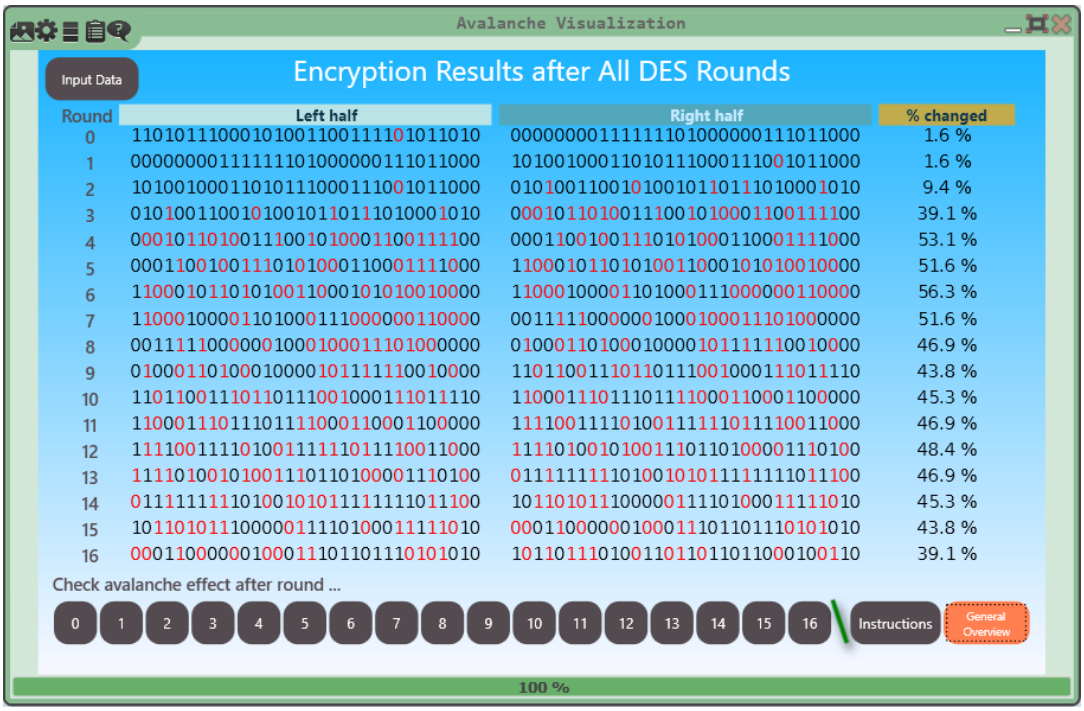


Figure 21: Affected bits (in red) after a single bit has been complemented for *DES*

5.3 Avalanche Tests for Hash Functions

5.3.1 SHA-1

Initial input message (in text): "Hello world!"

After pressing the "Play" button, the *SHA* plugin delivers the following hash value:
D3 48 6A E9 13 6E 78 56 BC 42 21 23 85 EA 79 70 94 47 58 02.

The initial input is changed by one letter, as follows:

Modified input message (in text): "Hello World!"
resulting in 2E F7 BD E6 08 CE 54 04 E9 7D 5F 04 2F 95 F8 9F 1C 23 28 71 as the new hash function. So, the test yields:

No. of flipped bits: 89 (out of 160).
Avalanche effect: 55.6 %.
Longest identical sequence: 7 bits.

5.3.2 MD5

Initial input message (in text): "Hello world!"
Modified input message (in text): "Hello world?"

Initial hash value: 86 FB 26 9D 19 0D 2C 85 F6 E0 46 8C EC A4 2A 20.
Modified hash value: 48 60 47 54 B9 FE D8 4B 3F EE B8 4C 5D C1 38 C0.

No. of flipped bits: 64 (out of 128).
Avalanche effect: 55.0 %.
Longest identical sequence: 6 bits.

5.3.3 Tiger

Initial input message (in text): "Vamos a la playa!"
Modified input message (in text): "Vamos$a la playa!"

Initial hash value:
3D 87 B9 50 65 B9 F8 5C F9 5D E1 8E 64 FA 24 2B 62 A2 CE B5 1F C2 7E EC.
Modified hash value:
F0 38 7D 93 0A F1 C9 0E 9C 43 9E C0 27 C6 2A 98 AD 45 5E 0E F3 C3 66 C2.

No. of flipped bits: 99 (out of 192).
Avalanche effect: 51.6 %.
Longest identical sequence: 9 bits.

5.3.4 Observations (Hash Functions' Tests)

The *completeness* property is satisfied, owing to the fact that despite minimal changes, all of the tested hash functions map the modified message to a complete different hash value. As a result they all exhibit a high degree of *avalanche effect* (over 50 %), thus, fulfilling the *strict avalanche criterion* (SAC).

5.4 Avalanche Tests for Classic Ciphers

5.4.1 Caesar

Initial input message (in text): "This is a test message"
Modified input message (in text): "This is a test massage"
Key used: 5.

Initial encrypted message: "Ymnx nx f yjxy rjxxflj"
Modified encrypted message: "Ymnx nx f yjxy rfxxflj"

No. of changed bytes: 1 (out of 22).
Avalanche effect: 4.54 %.
Longest identical sequence: 16 bytes.

5.4.2 Hill

Initial input message (in text): "THISISATESTMESSAGE"
Modified input message (in text): "THISISATESTMASSAGE"
Key Matrix:

$$\begin{pmatrix} 02 & 15 & 22 & 03 \\ 01 & 09 & 01 & 12 \\ 16 & 07 & 13 & 11 \\ 08 & 05 & 09 & 06 \end{pmatrix}$$

Initial encrypted message: "VXRELZQNMDLTSGYARGHW"
Modified encrypted message: "VXRELZQNMDLTWICARGHW"

No. of changed bytes: 3 (out of 20).
Avalanche effect: 15.0 %.
Longest identical sequence: 12 bytes.

5.4.3 Enigma

This test uses the default settings of the *Enigma* plugin.

Initial input message (in text): "Hello there"
Modified input message (in text): "Hallo there"

Initial encrypted message: "Lzfbd ptlnb"
Modified encrypted message: "Lqfbd ptlnb"

No. of changed bytes: 1 (out of 11).
Avalanche effect: 9.1 %.
Longest identical sequence: 9 bytes.

5.4.4 Vigenère

Initial input message (in text): "Hello there"
Modified input message (in text): "Hello thxre"
Key used (in text): "KEYWORD"

Initial encrypted message: "Rijhc kkovc"
Modified encrypted message: "Rijhc kkhvc"

No. of changed bytes: 1 (out of 11).
Avalanche effect: 9.1 %.
Longest identical sequence: 8 bytes.

5.4.5 Spanish Strip Cipher (SSC)

The first variant of the test uses a random method for the selection of the homophones.

Initial input message (in text): "Hello there"
Modified input message (in text): "Hallo there"

Initial encrypted message: "849690295164341756 39"
Modified encrypted message: "345229501533421712 39"

No. of changed bytes: 15 (out of 20).
Avalanche effect: 75%.
Longest identical sequence: 2 bytes.

The *Avalanche Visualization* plugin is also used to compare the *Spanish Strip Cipher* to *Caesar*, by performing two changes in the initial input message. This time – as a 2nd variant – a deterministic method (*Round Robin*) is used to select the homophones.

Initial input message (in text): "Classic ciphers"
First modification of input message (in text): "Classic cipherT"
Second modification of input message (in text): "Classic cipherZ"

After first modification:
No. of changed bytes Caesar: 1 (out of 15).
Avalanche effect Caesar: 6.7%.
No. of changed bytes SSC: 2 (out of 28).
Avalanche effect: 7.1%.

After second modification:
No. of changed bytes Caesar: 1 (out of 15).
Avalanche effect Caesar: 6.7%.
No. of changed bytes SSC: 2 (out of 28).
Avalanche effect: 7.1%.

The second modification is depicted in **Figure 22**.

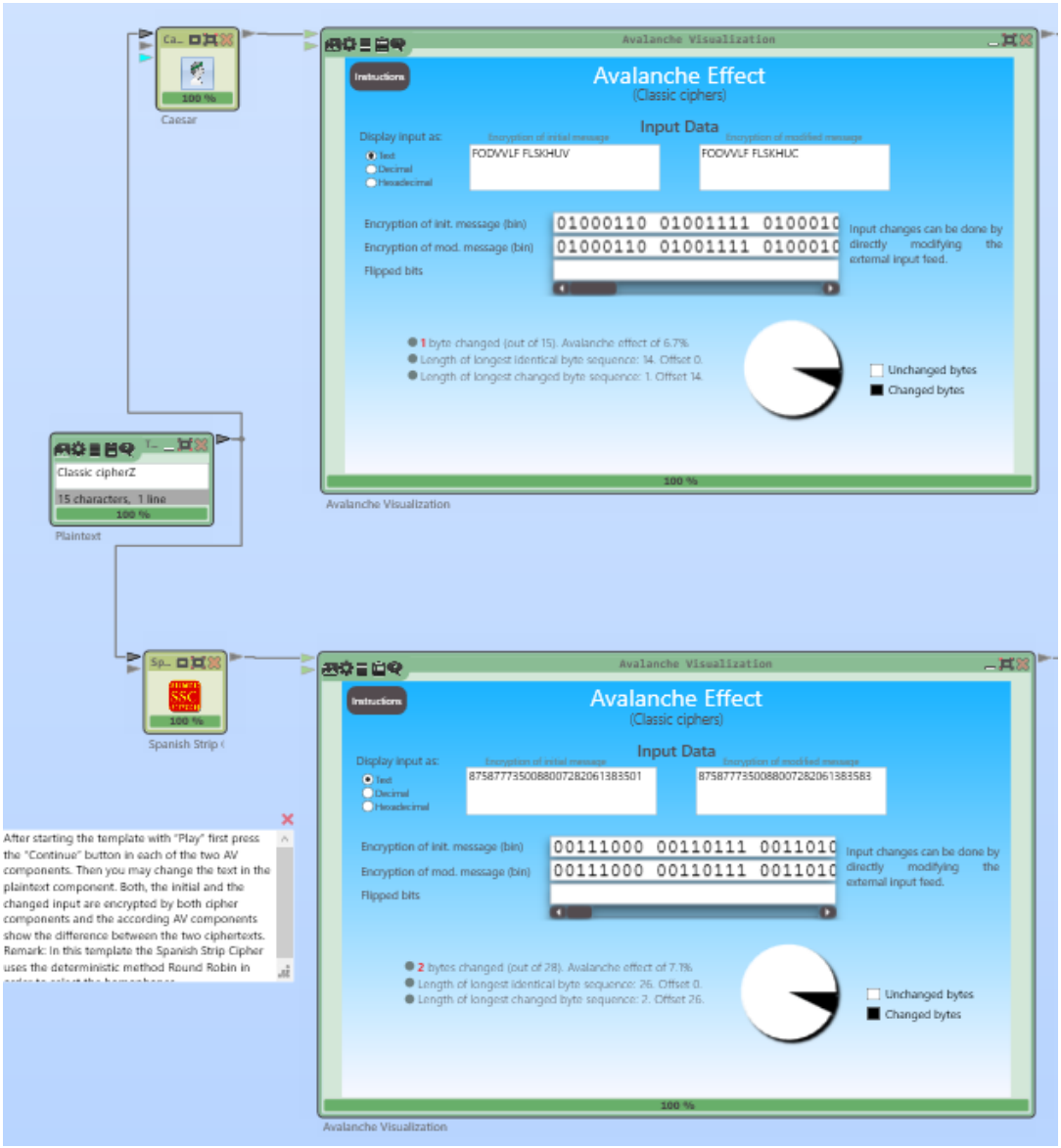


Figure 22: Comparison after 2nd modification between *Spanish Strip Cipher* and *Caesar*

5.4.6 Observations (Classic Ciphers' Tests)

Some of the tested classic substitution ciphers map the same plaintext letter to a different ciphertext letter (this can be seen e. g. if a word contains the same letter twice). This is the case for the Enigma machine and the Vigenère cipher. They are so called polyalphabetic ciphers. To say it in another way, they make use of different substitution alphabets [2].

For instance the two letters l from the word "Hello" map to "fb" and "jh" during the tests for the *Enigma* and the *Vigenère* ciphers respectively. Despite of having this characteristic, the ciphers do not exhibit a high *avalanche effect*, since the modification of one letter, only maps to one ciphertext symbol as well. For this reason they do not fulfill the *completeness* property either.

Hill cipher and *Spanish Strip Cipher* map a change of one plaintext symbol to more than one changed ciphertext symbol. In the example of the *Hill* cipher with a $4*4$ key matrix, a one-byte change mapped to three changed ciphertext bytes, thus achieving a certain level of *diffusion*. However, it does neither change around 50 % of the ciphertext bytes nor are the changed ciphertext bytes chosen randomly, hence, it does not show a high *avalanche effect*.

When using a random[3] selection of homophones with the *Spanish Strip Cipher* each symbol is mapped to a different one each time the algorithm runs, even if no modifications in the plaintext have been made. The results obtained are distorted. This is why the first test carried out with the *Spanish Strip Cipher (SSC)* is not suitable for determining the *avalanche effect*. A deterministic selection (*Round Robin*) must be used in order to determine this property in the case of comparing changes in the plaintext.

All of the tested classic ciphers exhibit a low degree of *avalanche effect*.

[3]Nonetheless, the cipher can be broken by using combinatorial and statistical methods [13].

6 Conclusion

The purpose of this work is to provide help in understanding the importance of the *avalanche effect* (AE) property by means of a visualization tool. The plugin allows the user to test this property on different ciphers and hash functions available in the CT2 environment.

The plugin deals with *prepared* and *unprepared methods*.

The test results show for the *prepared methods* (*AES* and *DES*) that both of these encryption standards possess a very strong *avalanche effect*. This occurrence was observed during all *AES* and *DES* test scenarios carried out. After a certain number of encryption rounds, the values of the *avalanche effect* remain relatively steady, reaching percentages close to 50 % until the end of the encryption (see **Figure** 23). This is also true for the the *Longest Identical Bit Sequence* (LIS) that reach values no bigger than 11 bits, after a certain number of rounds. This is an indicative, that a single change made in the input, rapidly spreads throughout the entire cipher, thus, contributing to the fulfillment of the *completeness* property.

For the *unprepared* methods there are very different outcomes, depending on the selection made. As minor changes to all hash functions produced completely different hash values, it is evident that they satisfy the *completeness* property, which in turn satisfies the *strict avalanche criterion* (SAC).

On the other hand, the classic ciphers do not exhibit the introduced properties. This is due to the fact, that most of them map a plaintext change to one ciphertext symbol, making the cipher vulnerable and easier to break.

Table 8 summarizes the test results for the ciphers *AES*, *DES*, and *Vigenère*, as well as for the hash function *SHA-1* regarding the five introduced properties: *avalanche effect*, *completeness*, *strict avalanche criterion* (SAC), *diffusion*, and *confusion*.

Summing up, the generic nature of the plugin lets the user analyze many different cryptographic ciphers and hash functions. In the future some other complex algorithms could also become part of the *prepared* methods, as to allow a deeper insight into their strength in regard to the *avalanche effect*.

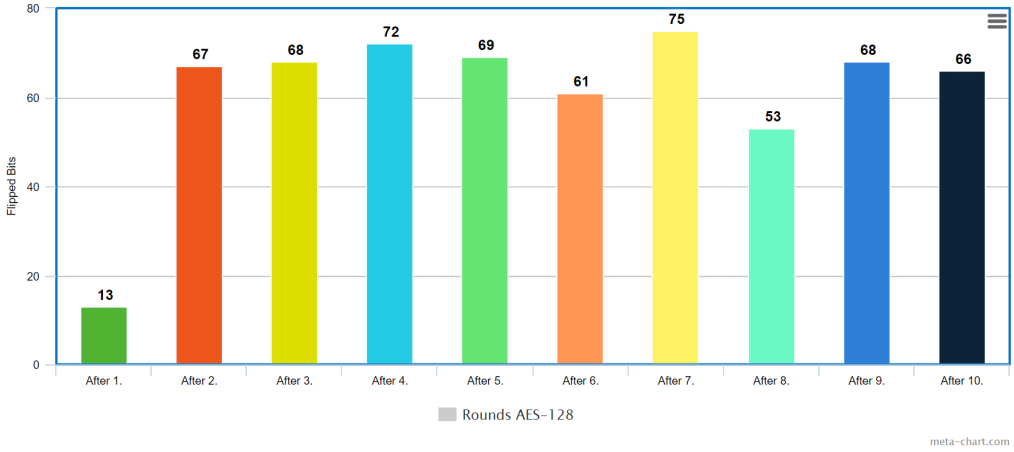


Figure 23: *Avalanche effect* after every round (*AES-128*) [1]

Method / Property	**AES**	**DES**	**SHA-1**	**Vigenère**
Avalanche Effect	strong (after round 2)	strong (after round 3)	strong	poor
Completeness	strong (after round 2)	strong (after round 4)	strong	poor
SAC	strong (after round 2)	strong (after round 4)	strong	poor
Diffusion	strong	strong	strong	poor
Confusion	strong	strong	strong	poor

Table 8: Properties and results for various methods

References

[1] Generated with Graphing/Charting tool - MetaChart. Available on: https://www.meta-chart.com/.

[2] R. F. Churchhouse. *Codes and ciphers: Julius Caesar, the Enigma, and the Internet*. Cambridge University Press, 2002.

[3] CrypTool team. Hash Demonstration - CrypTool 1. Available on: https://www.cryptool.org/en/ct1-downloads.

[4] H. Feistel. Cryptography and computer privacy. *Scientific American*, 228:15–23, 1973.

[5] GaborPete. Substitution permutation network - own work. https://commons.wikimedia.org/w/index.php?curid=6420152, 2009.

[6] H. M. Heys and S. E. Tavares. On the design of secure block ciphers. In *Proceedings of Queen's 17th Biennial Symposium on Communications, Kingston, Ontario*, 1994.

[7] JGraph Ltd. Generated with draw.io - online diagram software. Available on: https://www.draw.io/.

[8] T. Knappe. AEStetic web application. Available on: http://www.kna-st.de/aestetic/.

[9] A. K. Mandal and A. Tiwari. Analysis of avalanche effect in plaintext of des using binary codes. *International Journal of Emerging Trends and Technology in Computer Science (IJETTCS)*, 1(3):166–177, 2012.

[10] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[11] S. Ramanujam and M. Karuppiah. Designing an algorithm with high avalanche effect. *IJCSNS International Journal of Computer Science and Network Security*, 11(1):106–111, 2011.

[12] S. Przybylski, A. Wacker, M. Wander, F. Enkler, P. Vacek, and A. Krauß. Plugin Developer Manual, How to build your own plugins for CrypTool 2, 2016. https://www.cryptool.org/trac/CrypTool2/browser/trunk/Documentation/PluginHowTo/HowToDeveloper.pdf.

[13] L. A. B. Sanguino, G. Leander, C. Paar, B. Esslinger, and I. Niebel. Analyzing the spanish strip cipher by combining combinatorial and statistical methods. *Cryptologia*, 40(3):261–284, 2016.

[14] C. E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949.

[15] P. Stavroulakis and M. Stamp. *Handbook of information and communication security*. Springer Science & Business Media, 2010.

[16] A. Webster and S. E. Tavares. On the design of S-boxes. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 523–534. Springer, 1985.