Journal of Physics: Conference Series

PAPER • OPEN ACCESS

# RC4 Algorithm Visualization for Cryptography Education

To cite this article: S Sriadhi *et al* 2018 *J. Phys.: Conf. Ser.* **1028** 012057

# RC4 Algorithm Visualization for Cryptography Education

**S Sriadhi[1], Robbi Rahim[2*] and Ansari Saleh Ahmar[3]**

[1]Department of Electrical Engineering, Universitas Negeri Medan, Medan, Indonesia
[2]School of Computer and Communication Engineering, Universiti Malaysia Perlis, Kubang Gajah, Malaysia
[3]Department of Statistics, Universitas Negeri Makassar, Makassar, Indonesia

*usurobbi85@zoho.com

**Abstract.** Cryptography is a field of science that can be learned to secure data and information, cryptography is used in almost all communications both in network and non-network; and one of the algorithms could use is RC4 algorithm, publication about RC4 algorithm is quite a lot but the discussion is dominant only theory alone does not complete the RC4 algorithm process in detail and applications only show input and output none of the processes include, in this research paper illustrates the process of RC4 algorithm in detail and with visualization to demonstrates the work of RC4 algorithm to make it easiest for readers to learn cryptography

## 1. Introduction

The security of information[1]–[4] in this global age is increasingly becoming a vital necessity in various aspects of life especially if it is related to business, security, or public interest[5]–[8], the information of course made different parts who also have interests in it, in everyday life people rely heavily on information technology, from small things to complex problems and also provide many benefits for human life[5], RC4[9], [10] is a modern cryptographic algorithm that can use in various forms of security.

Several researchers have modeled a security process in the shape of simulation or visualization[11], [12], RC4 animation visually shows how the algorithm works and behaves in practice. Visualization of RC4 cryptography and operations are displayed step by step with animations consisting of multiple objects so that the behavior of the RC4 algorithm could be learned.

This paper provides a thorough understanding of the workings of RC4 cryptographic algorithms, and the visibility is show from the process of key simulation, padding, S-Box creation to the encryption and decryption process, the expected result of this paper is to provide an understanding that cryptography is not as difficult to imagine and easy to learn

## 2. Methodology

Cryptography is a science that creates a secure communication that cannot be understood or translated by anyone except the particular person[2], [3], [7], [8], [13], [14]. In this case, cryptography is not described as the science of studying the randomization of electronic data with the help of a computer program so that no one can interpret it. The benefits of cryptography such as:

a. Privacy which prevents the reading of messages by unauthorized persons.

b. The authenticity that allows the recipient of the message to know who is sending the message and the sender can also check that the receiver of the message is the person he meant.
c. Integrity assuring that messages sent are not falsified or altered by other unauthorized persons during the transmission of such messages.
d. Non-Repudiation prevents the recipient or sender of the message denying that they have received or sent the message

The RC4 algorithm has a plaintext combination encryption process using bit-wise XOR[15], [16]. RC4 uses a key length from 1 to 256 bytes used to initialize a 256-byte long table. This key use for pseudo-random processes that use XOR with the plaintext to generate ciphertext, each element in the table is changed at least once. The decryption process is done in the same way because XOR is a symmetric function[15], [16].

Visualization of RC4 algorithm for education must know the process of encryption and decryption RC4 algorithm first, here are some steps of RC4 algorithm:

A. Initialize the array S (S-Box or Substitution Box), so S0 = 0, S1 = 1, S2 = 2, ... ... Sn = n ... ..., S243 = 243, S244 = 244, S255 = 255.
B. If the key length U <256, do the padding addition of a pseudo byte, so the key length becomes 256 bytes
C. Perform permutations of the values in the S (S-Box) array, see the pseudo code below:

```
Dim tempSwap As Integer = 0
Dim j As Integer = 0
         For i As Integer = 0 To 255
             j = (j + S (i) + U (i)) Mod 256
              tempSwap = S (i)
             S (i) = S (j) '> swap value of S [i] and S [j]
             S (j) = tempSwap
         Next
```

D. Generate the key stream K and encrypt the plaintext P, for this process can see the pseudo code below:

```
Dim Input As String = Plaintext/Ciphertext
Dim tempSwap As Integer = 0 : Dim K As Integer = 0
Dim Output As Integer = 0 : Dim t As Integer = 0
Dim OutputSTSB As New System.Text.StringBuilder
Dim i As Integer = 0 : Dim j As Integer = 0
For idx As Integer = 0 To Input.Length - 1
                i = (i + 1) Mod 256
                j = (j + S(i)) Mod 256
    tempSwap = S(i) '\
                S(i) = S(j)      ' > swap value of S[i] and S[j]
                S(j) = tempSwap '/
                t = (S(i) + S(j)) Mod 256
                K = SBox(t)
                Output = K Xor Asc(Input(idx))
    OutputSTSB.Append(Chr(Output))
            Next
Dim OutputText As String = OutputSTSB.ToString
```

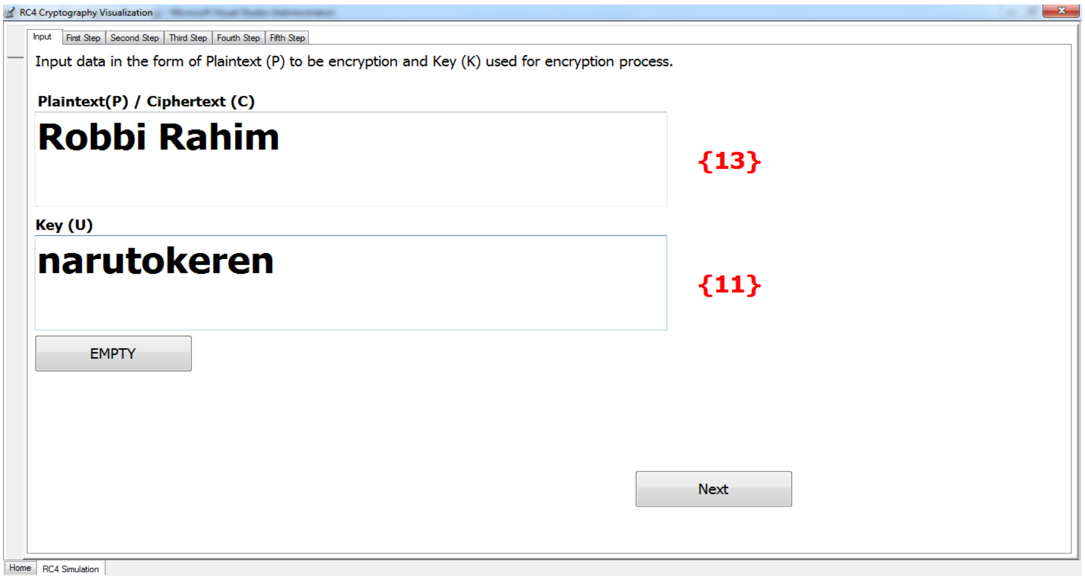
From A to D is the steps of workings RC4 algorithm that will describe in the next section, RC4 visualization model designed by using Visual Basic.Net 2008 programming language

## 3. Result and Discussion

Visualization of the RC4 algorithm for encryption and decryption process will be perform with plaintext and key, the example process as:

Plaintext = Robbi Rahim
Key = narutokeren

Plaintext and the key are insert into textfield in program and looks like in figure 1 below:



Input data in the form of Plaintext (P) to be encryption and Key (K) used for encryption process.

Plaintext(P) / Ciphertext (C)

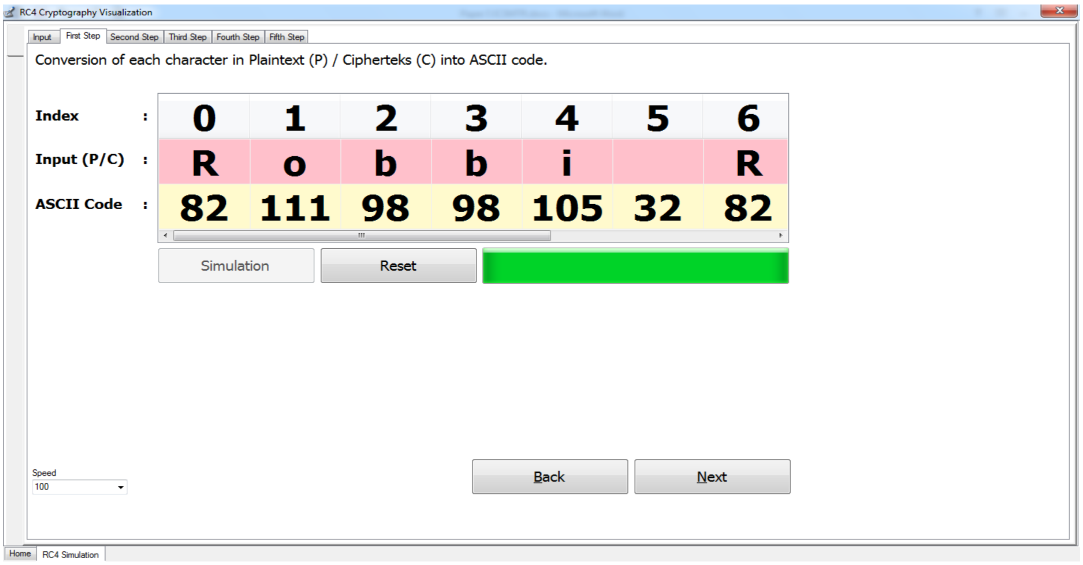
Robbi Rahim {13}

Key (U)

narutokeren {11}

**Figure 1**. Input Plaintext and Key

The next process is to convert plaintext into ASCII form, for plaintext converting model into ASCII form can see as picture 2 below:



Conversion of each character in Plaintext (P) / Cipherteks (C) into ASCII code.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Index : | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Input (P/C) : | R | o | b | b | i | | R |
| ASCII Code : | 82 | 111 | 98 | 98 | 105 | 32 | 82 |

Speed
100

**Figure 2**. Plaintext ASCII Conversion

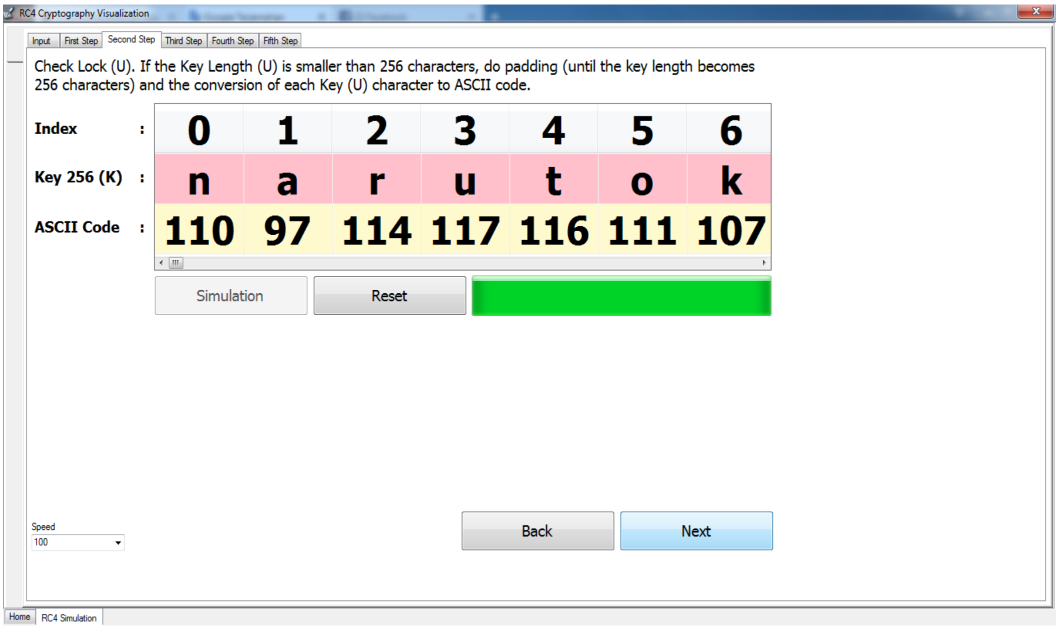
Figure 2 shows the ASCII code results for each character contained in the plaintext, for detail see table 1 below:

**Table 1**. ASCII Conversion

| Index | Plaintext | ASCII Code |
|---|---|---|
| 0 | R | 82 |
| 1 | o | 111 |
| 2 | b | 98 |

| | | |
|---|---|---|
| 3 | b | 98 |
| 4 | i | 105 |
| 5 | | 32 |
| 6 | R | 82 |
| 7 | a | 97 |
| 8 | h | 104 |
| 9 | i | 105 |
| 10 | m | 109 |

The next step is to change the key that is used up to 256 characters if less than 256 characters padding process is done as follows:



Check Lock (U). If the Key Length (U) is smaller than 256 characters, do padding (until the key length becomes 256 characters) and the conversion of each Key (U) character to ASCII code.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Index : | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Key 256 (K) : | n | a | r | u | t | o | k |
| ASCII Code : | 110 | 97 | 114 | 117 | 116 | 111 | 107 |

Speed
100

**Figure 3**. Key ASCII Conversion and Padding Function

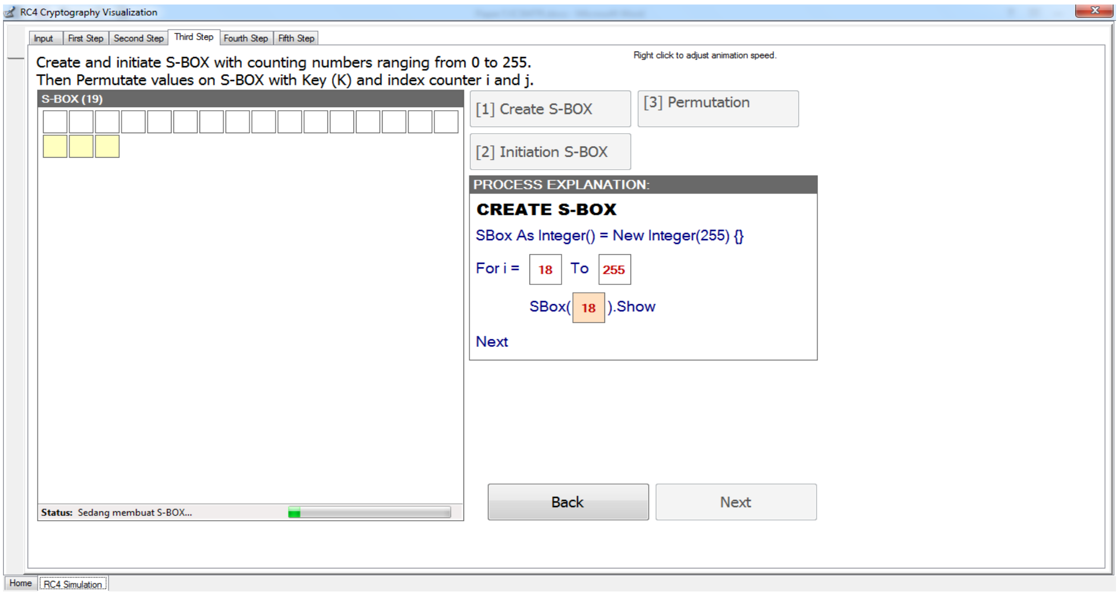
For details of ASCII generated from the used key and padding used note the following table 2:

**Table 2**. Key ASCII Conversion and Padding Function

| Index | Key | ASCII Code |
|---|---|---|
| 0 | n | 110 |
| 1 | a | 97 |
| 2 | r | 114 |
| 3 | u | 117 |
| 4 | t | 116 |
| 5 | o | 111 |
| 6 | k | 107 |
| 7 | e | 101 |
| 8 | r | 114 |
| 9 | e | 101 |
| 10 | n | 110 |
| 11 | n | 110 |

| | | |
|---|---|---|
| 12 | a | 97 |
| 13 | r | 114 |
| 14 | u | 117 |
| 15 | t | 116 |
| 16 | o | 111 |
| 17 | k | 107 |
| 18 | e | 101 |
| 19 | r | 114 |
| 20 | e | 101 |
| 21 | n | 110 |
| … | … | … |
| 245 | n | 110 |
| 246 | a | 97 |
| 247 | r | 114 |
| 248 | u | 117 |
| 249 | t | 116 |
| 250 | o | 111 |
| 251 | k | 107 |
| 252 | e | 101 |
| 253 | r | 114 |
| 254 | e | 101 |
| 255 | n | 110 |

The padding process in table 2 show in index 11 – 255 where there is a loop for the key used, the loop process is performed up to 256 characters. The next step is to create an S-Box table and initialize the S-Box table used, to create an S-Box table like Figure 4 below:



Create and initiate S-BOX with counting numbers ranging from 0 to 255.
Then Permutate values on S-BOX with Key (K) and index counter i and j.

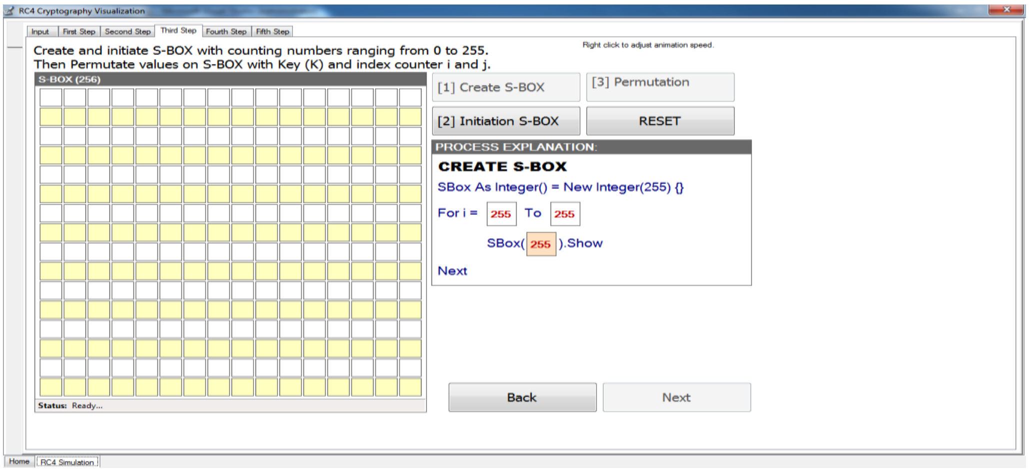
**CREATE S-BOX**

```
SBox As Integer() = New Integer(255) {}
For i = 18 To 255
        SBox( 18 ).Show
Next
```

**Status:** Sedang membuat S-BOX...

**Figure 4**. Create S-Box Table

The process of making table S-Box done gradually wherein the picture looks the number of tables S-Box made up of 255 boxes, see figure 5 below for finishing S-Box creation



Create and initiate S-BOX with counting numbers ranging from 0 to 255.
Then Permutate values on S-BOX with Key (K) and index counter i and j.

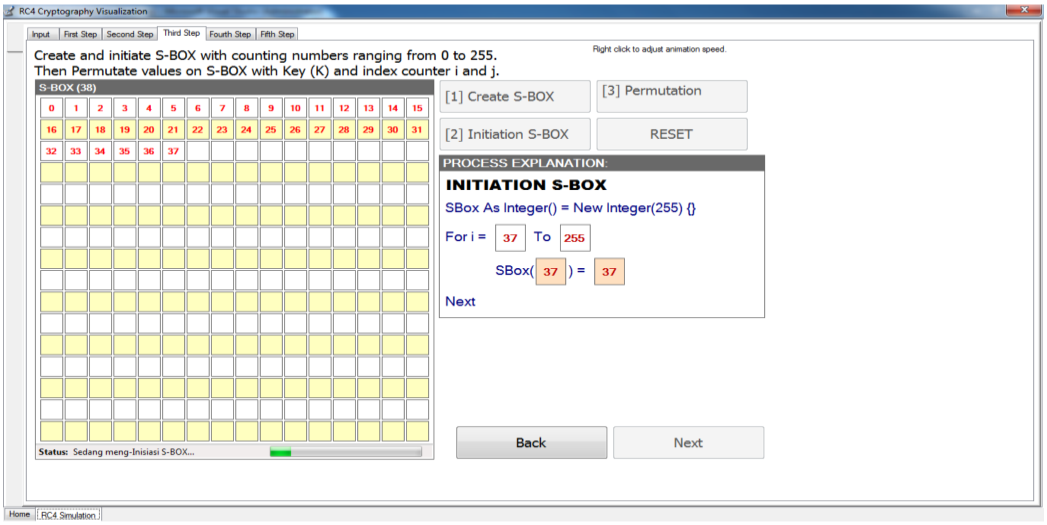
CREATE S-BOX

```
SBox As Integer() = New Integer(255) {}
For i = 255 To 255
    SBox( 255 ).Show
Next
```

Status: Ready...

**Figure 5**. Finishing S-Box Table

The next process is to initiate the value in the S-Box table with the following results:



Create and initiate S-BOX with counting numbers ranging from 0 to 255.
Then Permutate values on S-BOX with Key (K) and index counter i and j.

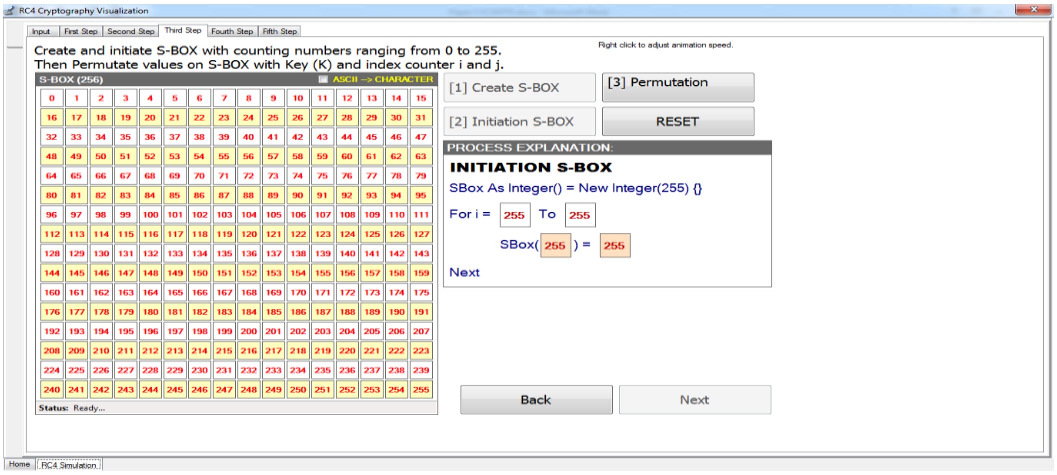
| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 32 | 33 | 34 | 35 | 36 | 37 | | | | | | | | | | |

INITIATION S-BOX

```
SBox As Integer() = New Integer(255) {}
For i = 37 To 255
    SBox( 37 ) = 37
Next
```

Status: Sedang meng-Inisiasi S-BOX...

**Figure 6**. Initiation S-Box



Create and initiate S-BOX with counting numbers ranging from 0 to 255.
Then Permutate values on S-BOX with Key (K) and index counter i and j.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
| 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 |
| 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
| 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 |
| 128 | 129 | 130 | 131 | 132 | 133 | 134 | 135 | 136 | 137 | 138 | 139 | 140 | 141 | 142 | 143 |
| 144 | 145 | 146 | 147 | 148 | 149 | 150 | 151 | 152 | 153 | 154 | 155 | 156 | 157 | 158 | 159 |
| 160 | 161 | 162 | 163 | 164 | 165 | 166 | 167 | 168 | 169 | 170 | 171 | 172 | 173 | 174 | 175 |
| 176 | 177 | 178 | 179 | 180 | 181 | 182 | 183 | 184 | 185 | 186 | 187 | 188 | 189 | 190 | 191 |
| 192 | 193 | 194 | 195 | 196 | 197 | 198 | 199 | 200 | 201 | 202 | 203 | 204 | 205 | 206 | 207 |
| 208 | 209 | 210 | 211 | 212 | 213 | 214 | 215 | 216 | 217 | 218 | 219 | 220 | 221 | 222 | 223 |
| 224 | 225 | 226 | 227 | 228 | 229 | 230 | 231 | 232 | 233 | 234 | 235 | 236 | 237 | 238 | 239 |
| 240 | 241 | 242 | 243 | 244 | 245 | 246 | 247 | 248 | 249 | 250 | 251 | 252 | 253 | 254 | 255 |

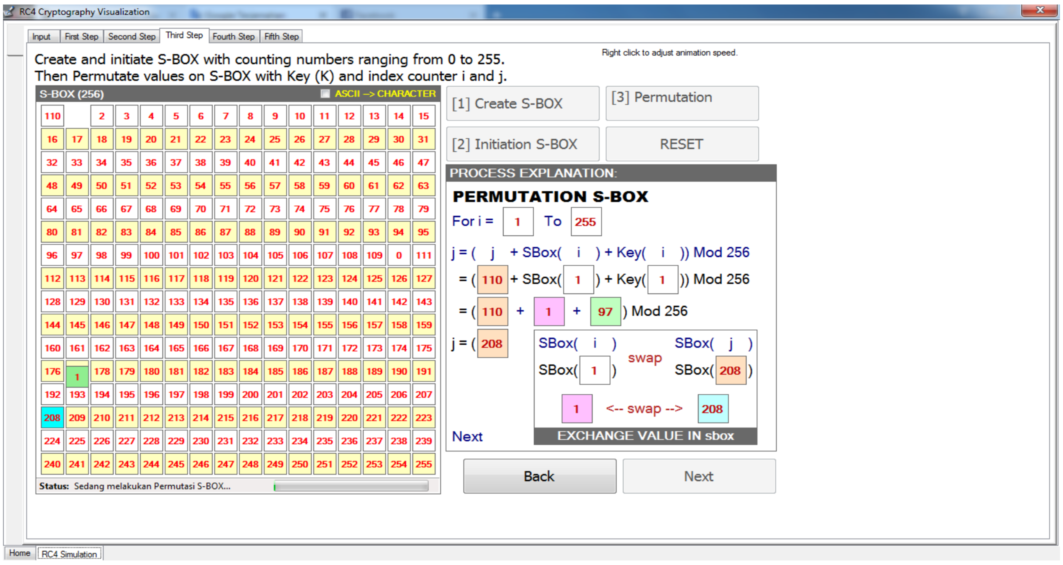
INITIATION S-BOX

```
SBox As Integer() = New Integer(255) {}
For i = 255 To 255
    SBox( 255 ) = 255
Next
```

Status: Ready...

**Figure 7**. Finish Initiation S-Box

Figure 6 and Figure 7 are the results of S-Box initiation by entering the index value, the index value entered starting from 0 to 255, the next step is to do the S-Box permutation with the generated key, and here is the result



Create and initiate S-BOX with counting numbers ranging from 0 to 255.
Then Permutate values on S-BOX with Key (K) and index counter i and j.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 110 | | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
| 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 |
| 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 0 | 111 |
| 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 |
| 128 | 129 | 130 | 131 | 132 | 133 | 134 | 135 | 136 | 137 | 138 | 139 | 140 | 141 | 142 | 143 |
| 144 | 145 | 146 | 147 | 148 | 149 | 150 | 151 | 152 | 153 | 154 | 155 | 156 | 157 | 158 | 159 |
| 160 | 161 | 162 | 163 | 164 | 165 | 166 | 167 | 168 | 169 | 170 | 171 | 172 | 173 | 174 | 175 |
| 176 | 1 | 178 | 179 | 180 | 181 | 182 | 183 | 184 | 185 | 186 | 187 | 188 | 189 | 190 | 191 |
| 192 | 193 | 194 | 195 | 196 | 197 | 198 | 199 | 200 | 201 | 202 | 203 | 204 | 205 | 206 | 207 |
| 208 | 209 | 210 | 211 | 212 | 213 | 214 | 215 | 216 | 217 | 218 | 219 | 220 | 221 | 222 | 223 |
| 224 | 225 | 226 | 227 | 228 | 229 | 230 | 231 | 232 | 233 | 234 | 235 | 236 | 237 | 238 | 239 |
| 240 | 241 | 242 | 243 | 244 | 245 | 246 | 247 | 248 | 249 | 250 | 251 | 252 | 253 | 254 | 255 |

**Status:** Sedang melakukan Permutasi S-BOX...

PROCESS EXPLANATION:

**PERMUTATION S-BOX**

For i = 1 To 255

j = ( j + SBox( i ) + Key( i )) Mod 256

= ( 110 + SBox( 1 ) + Key( 1 )) Mod 256

= ( 110 + 1 + 97 ) Mod 256

j = ( 208

SBox( i ) swap SBox( j )

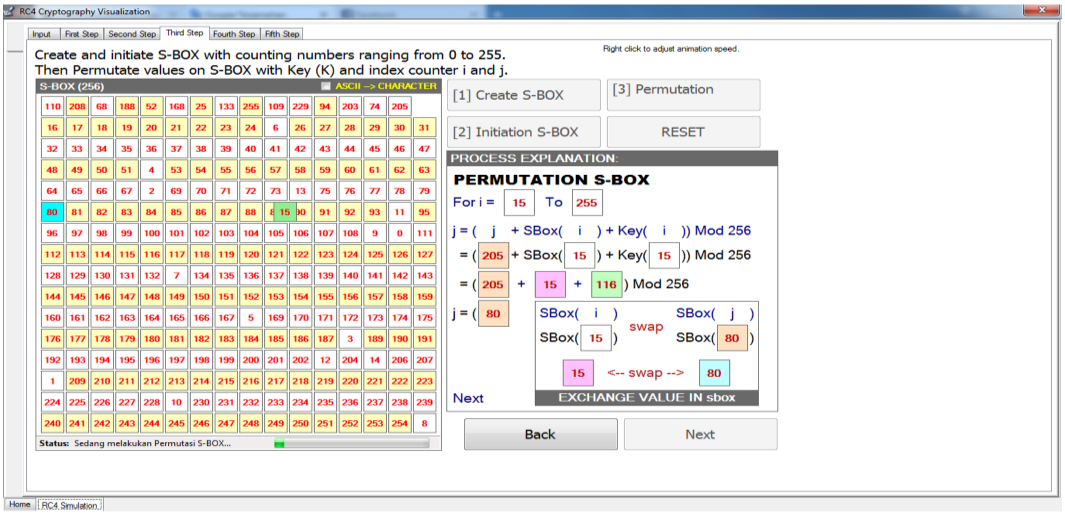
SBox( 1 ) SBox( 208 )

1 <-- swap --> 208

Next

EXCHANGE VALUE IN sbox

**Figure 8**. Permutation Process 1



Create and initiate S-BOX with counting numbers ranging from 0 to 255.
Then Permutate values on S-BOX with Key (K) and index counter i and j.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 110 | 208 | 68 | 188 | 52 | 168 | 25 | 133 | 255 | 109 | 229 | 94 | 203 | 74 | 205 | |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 6 | 26 | 27 | 28 | 29 | 30 | 31 |
| 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 48 | 49 | 50 | 51 | 4 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| 64 | 65 | 66 | 67 | 2 | 69 | 70 | 71 | 72 | 73 | 13 | 75 | 76 | 77 | 78 | 79 |
| 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 8 15 | 90 | 91 | 92 | 93 | 11 | 95 |
| 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 9 | 0 | 111 |
| 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 |
| 128 | 129 | 130 | 131 | 132 | 7 | 134 | 135 | 136 | 137 | 138 | 139 | 140 | 141 | 142 | 143 |
| 144 | 145 | 146 | 147 | 148 | 149 | 150 | 151 | 152 | 153 | 154 | 155 | 156 | 157 | 158 | 159 |
| 160 | 161 | 162 | 163 | 164 | 165 | 166 | 167 | 5 | 169 | 170 | 171 | 172 | 173 | 174 | 175 |
| 176 | 177 | 178 | 179 | 180 | 181 | 182 | 183 | 184 | 185 | 186 | 187 | 3 | 189 | 190 | 191 |
| 192 | 193 | 194 | 195 | 196 | 197 | 198 | 199 | 200 | 201 | 202 | 12 | 204 | 14 | 206 | 207 |
| 1 | 209 | 210 | 211 | 212 | 213 | 214 | 215 | 216 | 217 | 218 | 219 | 220 | 221 | 222 | 223 |
| 224 | 225 | 226 | 227 | 228 | 10 | 230 | 231 | 232 | 233 | 234 | 235 | 236 | 237 | 238 | 239 |
| 240 | 241 | 242 | 243 | 244 | 245 | 246 | 247 | 248 | 249 | 250 | 251 | 252 | 253 | 254 | 8 |

**Status:** Sedang melakukan Permutasi S-BOX...

PROCESS EXPLANATION:

**PERMUTATION S-BOX**

For i = 15 To 255

j = ( j + SBox( i ) + Key( i )) Mod 256

= ( 205 + SBox( 15 ) + Key( 15 )) Mod 256

= ( 205 + 15 + 116 ) Mod 256

j = ( 80

SBox( i ) swap SBox( j )

SBox( 15 ) SBox( 80 )

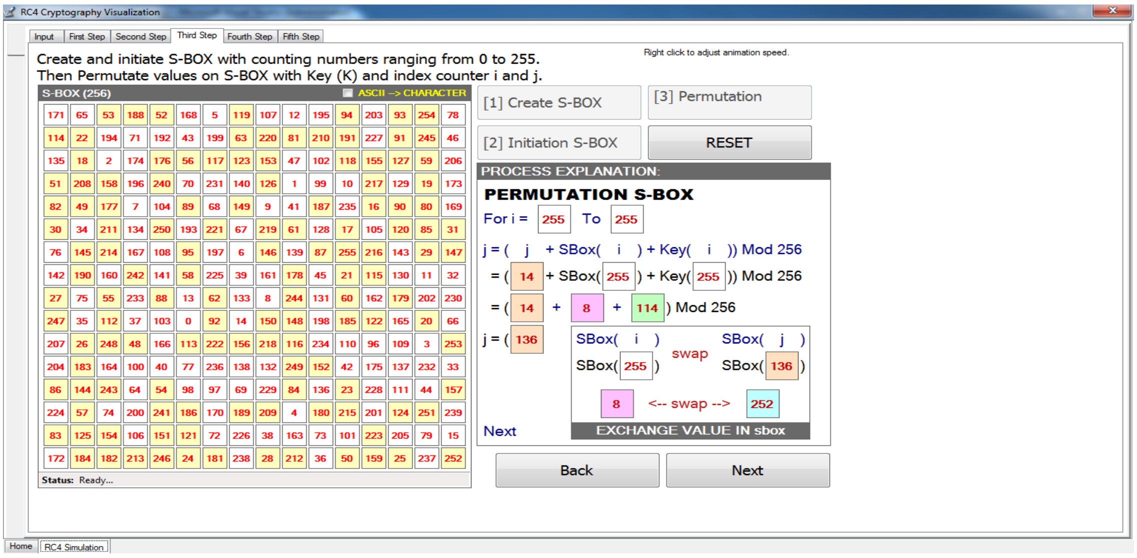
15 <-- swap --> 80

Next

EXCHANGE VALUE IN sbox

**Figure 9**. Permutation Process 2

The permutation process is performed until all values in the S-Box table are mutated by using a degenerate key index; Figure 8 and Figure 9 are the results of permutations for the S-Box 2 table and the S-Box 10 table



RC4 Cryptography Visualization

Input | First Step | Second Step | Third Step | Fourth Step | Fifth Step

Create and initiate S-BOX with counting numbers ranging from 0 to 255.
Then Permutate values on S-BOX with Key (K) and index counter i and j.

Right click to adjust animation speed.

S-BOX (256) ☐ ASCII –> CHARACTER

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 171 | 65 | 53 | 188 | 52 | 168 | 5 | 119 | 107 | 12 | 195 | 94 | 203 | 93 | 254 | 78 |
| 114 | 22 | 194 | 71 | 192 | 43 | 199 | 63 | 220 | 81 | 210 | 191 | 227 | 91 | 245 | 46 |
| 135 | 18 | 2 | 174 | 176 | 56 | 117 | 123 | 153 | 47 | 102 | 118 | 155 | 127 | 59 | 206 |
| 51 | 208 | 158 | 196 | 240 | 70 | 231 | 140 | 126 | 1 | 99 | 10 | 217 | 129 | 19 | 173 |
| 82 | 49 | 177 | 7 | 104 | 89 | 68 | 149 | 9 | 41 | 187 | 235 | 16 | 90 | 80 | 169 |
| 30 | 34 | 211 | 134 | 250 | 193 | 221 | 67 | 219 | 61 | 128 | 17 | 105 | 120 | 85 | 31 |
| 76 | 145 | 214 | 167 | 108 | 95 | 197 | 6 | 146 | 139 | 87 | 255 | 216 | 143 | 29 | 147 |
| 142 | 190 | 160 | 242 | 141 | 58 | 225 | 39 | 161 | 178 | 45 | 21 | 115 | 130 | 11 | 32 |
| 27 | 75 | 55 | 233 | 88 | 13 | 62 | 133 | 8 | 244 | 131 | 60 | 162 | 179 | 202 | 230 |
| 247 | 35 | 112 | 37 | 103 | 0 | 92 | 14 | 150 | 148 | 198 | 185 | 122 | 165 | 20 | 66 |
| 207 | 26 | 248 | 48 | 166 | 113 | 222 | 156 | 218 | 116 | 234 | 110 | 96 | 109 | 3 | 253 |
| 204 | 183 | 164 | 100 | 40 | 77 | 236 | 138 | 132 | 249 | 152 | 42 | 175 | 137 | 232 | 33 |
| 86 | 144 | 243 | 64 | 54 | 98 | 97 | 69 | 229 | 84 | 136 | 23 | 228 | 111 | 44 | 157 |
| 224 | 57 | 74 | 200 | 241 | 186 | 170 | 189 | 209 | 4 | 180 | 215 | 201 | 124 | 251 | 239 |
| 83 | 125 | 154 | 106 | 151 | 121 | 72 | 226 | 38 | 163 | 73 | 101 | 223 | 205 | 79 | 15 |
| 172 | 184 | 182 | 213 | 246 | 24 | 181 | 238 | 28 | 212 | 36 | 50 | 159 | 25 | 237 | 252 |

**Status:** Ready...

[1] Create S-BOX | [3] Permutation
[2] Initiation S-BOX | RESET

PROCESS EXPLANATION:

**PERMUTATION S-BOX**

For i = 255 To 255

j = ( j + SBox( i ) + Key( i )) Mod 256
= ( 14 + SBox( 255 ) + Key( 255 )) Mod 256
= ( 14 + 8 + 114 ) Mod 256
j = ( 136

SBox( i ) swap SBox( j )
SBox( 255 ) SBox( 136 )
8 <-- swap --> 252
EXCHANGE VALUE IN sbox

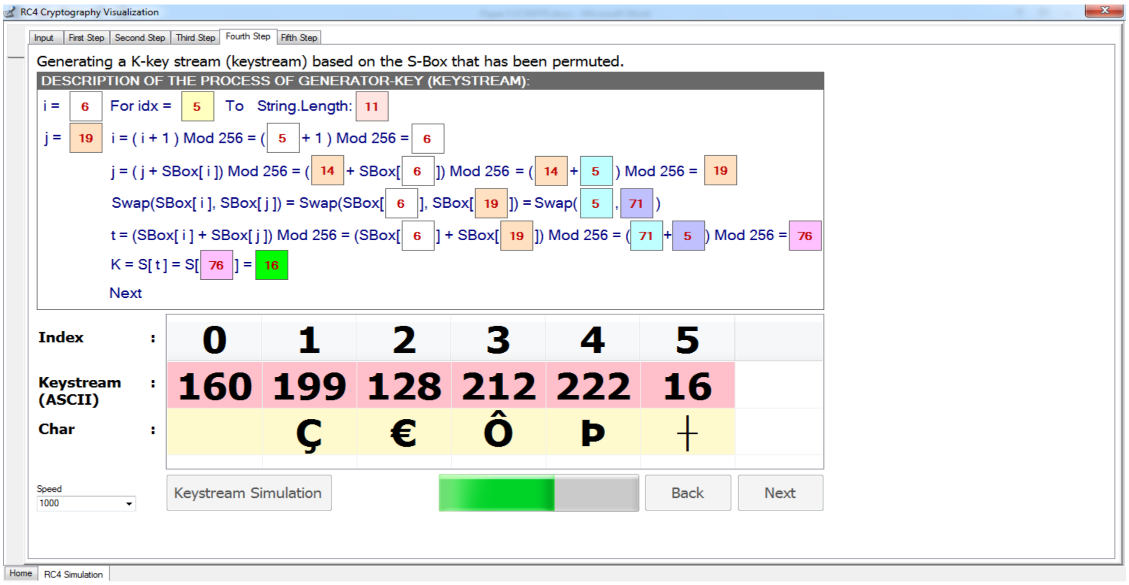
Next

Back | Next

Home | RC4 Simulation

**Figure 10**. Permutation S-Box Result

Figure 10 is the result of the permutation process, then to facilitate the learning of RC4 algorithm especially in the permutation process (figure 6 – 10), see the frame of explanation in the simulation, the next step is to form K Key Stream based on the S-Box table that has been generated, Key Stream is the key generated by the process of encryption and decryption for more details note the picture 11 below:



RC4 Cryptography Visualization

Input | First Step | Second Step | Third Step | Fourth Step | Fifth Step

Generating a K-key stream (keystream) based on the S-Box that has been permuted.

DESCRIPTION OF THE PROCESS OF GENERATOR-KEY (KEYSTREAM):

i = 6 For idx = 5 To String.Length: 11
j = 19 i = ( i + 1 ) Mod 256 = ( 5 + 1 ) Mod 256 = 6
j = ( j + SBox[ i ]) Mod 256 = ( 14 + SBox[ 6 ]) Mod 256 = ( 14 + 5 ) Mod 256 = 19
Swap(SBox[ i ], SBox[ j ]) = Swap(SBox[ 6 ], SBox[ 19 ]) = Swap( 5 , 71 )
t = (SBox[ i ] + SBox[ j ]) Mod 256 = (SBox[ 6 ] + SBox[ 19 ]) Mod 256 = ( 71 + 5 ) Mod 256 = 76
K = S[ t ] = S[ 76 ] = 16
Next

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Index** | : | 0 | 1 | 2 | 3 | 4 | 5 |
| **Keystream (ASCII)** | : | 160 | 199 | 128 | 212 | 222 | 16 |
| **Char** | : | | Ç | € | Ô | Þ | ┼ |

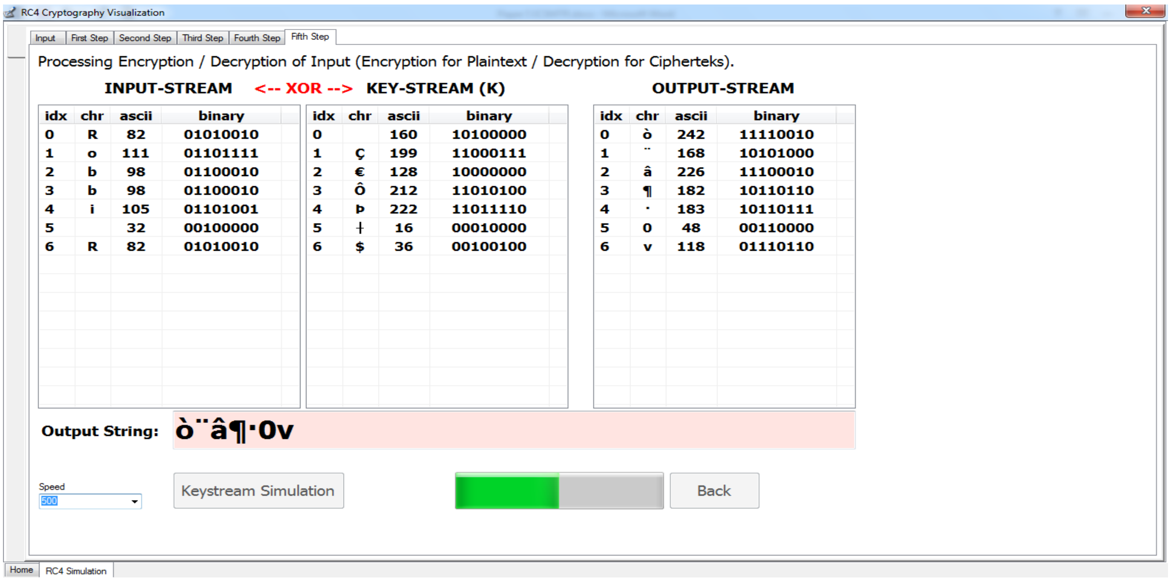
Speed 1000

Keystream Simulation | Back | Next

Home | RC4 Simulation

**Figure 11**. Key Stream Generation

Figure 11 shows the Key Stream process where the process value is displayed based on the S-Box table value and the ASCII value of the key used, and the last process is encrypting between the plaintext with the key (K) Key Stream generated by Figure 11



Processing Encryption / Decryption of Input (Encryption for Plaintext / Decryption for Cipherteks).

INPUT-STREAM <-- XOR --> KEY-STREAM (K) OUTPUT-STREAM

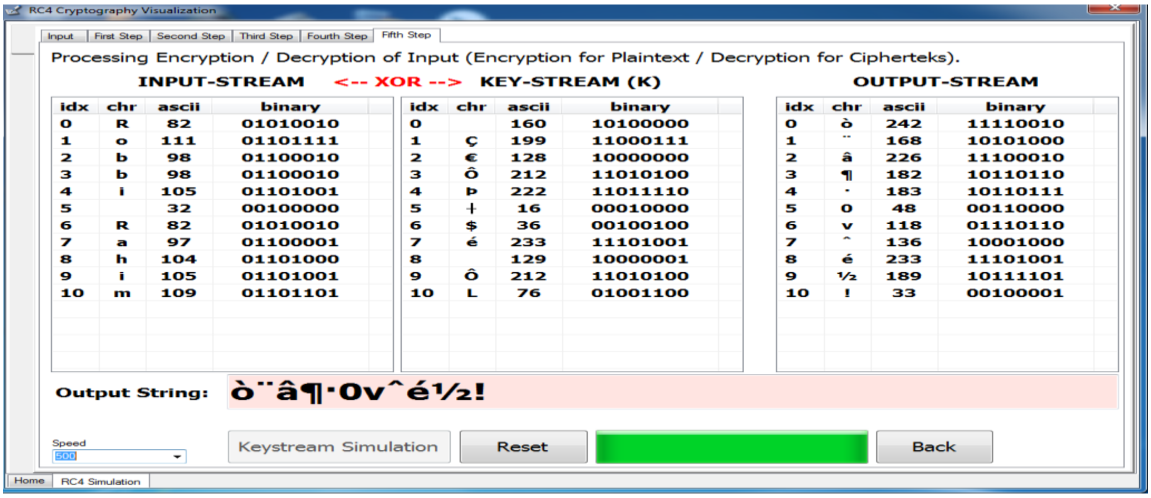
| idx | chr | ascii | binary |
|---|---|---|---|
| 0 | R | 82 | 01010010 |
| 1 | o | 111 | 01101111 |
| 2 | b | 98 | 01100010 |
| 3 | b | 98 | 01100010 |
| 4 | i | 105 | 01101001 |
| 5 |  | 32 | 00100000 |
| 6 | R | 82 | 01010010 |

| idx | chr | ascii | binary |
|---|---|---|---|
| 0 |  | 160 | 10100000 |
| 1 | Ç | 199 | 11000111 |
| 2 | € | 128 | 10000000 |
| 3 | Ô | 212 | 11010100 |
| 4 | Þ | 222 | 11011110 |
| 5 | ┼ | 16 | 00010000 |
| 6 | $ | 36 | 00100100 |

| idx | chr | ascii | binary |
|---|---|---|---|
| 0 | ò | 242 | 11110010 |
| 1 | ¨ | 168 | 10101000 |
| 2 | â | 226 | 11100010 |
| 3 | ¶ | 182 | 10110110 |
| 4 | · | 183 | 10110111 |
| 5 | 0 | 48 | 00110000 |
| 6 | v | 118 | 01110110 |

Output String: ò¨â¶·0v

**Figure 12**. Process Encryption



Processing Encryption / Decryption of Input (Encryption for Plaintext / Decryption for Cipherteks).

INPUT-STREAM <-- XOR --> KEY-STREAM (K) OUTPUT-STREAM

| idx | chr | ascii | binary |
|---|---|---|---|
| 0 | R | 82 | 01010010 |
| 1 | o | 111 | 01101111 |
| 2 | b | 98 | 01100010 |
| 3 | b | 98 | 01100010 |
| 4 | i | 105 | 01101001 |
| 5 |  | 32 | 00100000 |
| 6 | R | 82 | 01010010 |
| 7 | a | 97 | 01100001 |
| 8 | h | 104 | 01101000 |
| 9 | i | 105 | 01101001 |
| 10 | m | 109 | 01101101 |

| idx | chr | ascii | binary |
|---|---|---|---|
| 0 |  | 160 | 10100000 |
| 1 | Ç | 199 | 11000111 |
| 2 | € | 128 | 10000000 |
| 3 | Ô | 212 | 11010100 |
| 4 | Þ | 222 | 11011110 |
| 5 | ┼ | 16 | 00010000 |
| 6 | $ | 36 | 00100100 |
| 7 | é | 233 | 11101001 |
| 8 |  | 129 | 10000001 |
| 9 | Ô | 212 | 11010100 |
| 10 | L | 76 | 01001100 |

| idx | chr | ascii | binary |
|---|---|---|---|
| 0 | ò | 242 | 11110010 |
| 1 | ¨ | 168 | 10101000 |
| 2 | â | 226 | 11100010 |
| 3 | ¶ | 182 | 10110110 |
| 4 | · | 183 | 10110111 |
| 5 | 0 | 48 | 00110000 |
| 6 | v | 118 | 01110110 |
| 7 | ˆ | 136 | 10001000 |
| 8 | é | 233 | 11101001 |
| 9 | ½ | 189 | 10111101 |
| 10 | ! | 33 | 00100001 |

Output String: ò¨â¶·0vˆé½!

**Figure 13**. Finishing Process Encryption

Figure 12 and Figure 13 show the encryption process between the plaintext and the key, the encryption method of the RC4 algorithm uses the XOR function between plaintext and key-stream, so the encryption process between plaintext and key is as follows:

| | |
|---|---|
| Plaintext | = Robbi Rahim |
| Key | = narutokeren |
| Ciphertext | = ò¨â¶·0vˆé½! |

Decryption process is no different than the encryption process and also each step decryption process also shown like an encryption process.

## Conclusion

The visualization of RC4 can show how RC4 algorithm works step by step, RC4 algorithm visualization makes it easy for cryptographic learning for students, lecturers or those who want to know RC4 cryptography, the future development this research RC4 algorithm can be in the form of better simulation.

## References

[1] R. Rahim and A. Ikhwan, “Study of Three Pass Protocol on Data Security,” *Int. J. Sci. Res.*, vol. 5, no. 11, pp. 102–104, Nov. 2016.

[2] E. Hariyanto and R. Rahim, “Arnold’s Cat Map Algorithm in Digital Image Encryption,” *Int. J. Sci. Res.*, vol. 5, no. 10, pp. 1363–1365, Oct. 2016.

[3] R. Rahim and A. Ikhwan, “Cryptography Technique with Modular Multiplication Block Cipher and Playfair Cipher,” *Int. J. Sci. Res. Sci. Technol.*, vol. 2, no. 6, pp. 71–78, 2016.

[4] R. Rahim, “128 Bit Hash of Variable Length in Short Message Service Security,” *Int. J. Secur. Its Appl.*, vol. 11, no. 1, pp. 45–58, Jan. 2017.

[5] W. H. Haji and S. Mulyono, “Implementasi Rc4 Stream Cipher Untuk Keamanan Basis Data,” *Implementasi Rc4 Stream Cipher Untuk Keamanan Basis Data*, vol. 2012, no. Snati, pp. 15–16, 2012.

[6] R. Rahim, “Man-in-the-middle-attack prevention using interlock protocol method,” *ARPN J. Eng. Appl. Sci.*, vol. 12, no. 22, pp. 6483–6487, 2017.

[7] R. Rahim, M. Dahria, M. Syahril, and B. Anwar, “Combination of the Blowfish and Lempel-Ziv-Welch algorithms for text compression,” *World Trans. Eng. Technol. Educ.*, vol. 15, no. 3, pp. 292–297, 2017.

[8] H. Nurdiyanto, R. Rahim, and N. Wulan, “Symmetric Stream Cipher using Triple Transposition Key Method and Base64 Algorithm for Security Improvement,” *J. Phys. Conf. Ser.*, vol. 930, no. 1, p. 12005, Dec. 2017.

[9] S. Maitra and G. Paul, “Analysis of RC4 and proposal of additional layers for better security margin,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2008, vol. 5365 LNCS, pp. 27–39.

[10] D. Hendarsyah and R. Wardoyo, “Implementasi Protokol Diffie-Hellman Dan Algoritma RC4 Untuk Keamanan Pesan SMS,” *IJCCS*, vol. 5, no. 1, pp. 14–25, 2011.

[11] G. Cattaneo, A. De Santis, and U. Ferraro Petrillo, “Visualization of cryptographic protocols with GRACE,” *J. Vis. Lang. Comput.*, vol. 19, no. 2, pp. 258–290, 2008.

[12] M. A. M. Maeref, F. Alghali, and K. Abied, “An Advance Visual Model for Animating Behavior of Cryptographic Protocols,” *J. Comput.*, vol. 10, no. 5, pp. 336–346, 2015.

[13] H. Nurdiyanto and R. Rahim, “Enhanced pixel value differencing steganography with government standard algorithm,” in *2017 3rd International Conference on Science in Information Technology (ICSITech)*, 2017, pp. 366–371.

[14] R. Rahim, H. Winata, I. Zulkarnain, and H. Jaya, “Prime Number: an Experiment Rabin-Miller and Fast Exponentiation,” *J. Phys. Conf. Ser.*, vol. 930, no. 1, p. 12032, Dec. 2017.

[15] L. Stošić and M. Bogdanović, “RC4 stream cipher and possible attacks on WEP,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 3, no. 3, pp. 110–114, 2012.

[16] P. Jindal and B. Singh, “RC4 Encryption-A Literature Survey,” *Procedia Comput. Sci.*, vol. 46, pp. 697–705, 2015.