Use of Rc4 Method In Android-Based Sms Security Application

Oktovantua Butarbutar  
Teknik Informatika, STMIK Jakarta STI&K  
Email: oktovantua45@gmail.com

Abstract. In maintaining the confidentiality of SMS, it takes a way to secure information that is important or confidential, namely by encrypting the SMS text, the level of information security of the message can be increased. One way to use for data and or information security is to use cryptographic system, this application uses RC4 algorithm which has the advantage of having a high level of security and speed process. This makes RC4 the best choice for the encryption process required by the information world towards the next century. The purpose of this research is to prevent the occurrence of information/data taker without being known by the owner.

Keywords
Information System
Cryptography
RC4

1. INTRODUCTION

Data security and confidentiality issues are one of the most important aspects of an information system. This is very much related to how important the information is to be sent and received by people interested in the data. Information will no longer be useful if in the middle of the way data is hijacked or intercepted by unauthorized persons [1]. The information on the data will be lost so that the recipient will get different information. One of the ways used for data or information security is to use cryptographic systems, this application uses RC4 algorithms that have the advantage of having a high level of security and speed processes [2]. This makes RC4 the best choice for the encryption process required by the information world towards the next step. In maintaining the confidentiality of SMS, it takes a way to secure information that is important or confidential, namely by encrypting the SMS text, the level of information security of the message can be increased. The author in addressing the problem of sending messages, trying to create a messaging application with RC4 algorithm to encrypt data running on android operating system so that owners of android-based mobile phones (phones) can exchange SMS data more safely and conveniently [1], [3].

A text messaging service component of most telephones, Internet, and mobile-device systems is known as short message service (SMS) [4]. Standardized communication protocols are used to permit smart phones to transfer short text messages. Short message service is also commonly referred to as a “text message.” The user can conduct a message of up to 160 characters to another device with a SMS. In SMS, longer messages will automatically be fragmented into several parts. This type of text messaging is supported by most cell phones. The formal name for text messaging is SMS. Short message service is a way to conduct short, text-only messages from one phone to another. These messages are usually conducted over a cellular data network. The procedure for conducting SMS is launching the Messages application on the phone. Tap on the Compose Message button. Enter the phone number or name of the contact you want to text. Type your message and finally hit Send. These days, there exist a number of security issues and vulnerabilities related to SMS [3].

2. METHOD

2.1 Rc4 Method

Cryptography is derived from the Greek "Cryptos" meaning "Secret" and "graphein" means "writing". So, cryptography means "Secret Writing". Definition stated in [SCH96] : Cryptography is the science and art of keeping messages safe. The Rivest Code 4 (RC4) cryptographic algorithm is one of the symmetric key algorithms created by RSA Data Security Inc (RSADSI) in the form of a cipher stream [4]. The algorithm was invented in 1987 by Ronald Rivest and became a symbol of...
RSA security (short for three inventor names: Rivest Shamir Adleman) [10]. RC4 uses a key length from 1 to 256 bytes that is used to initialize a 256-byte table. This table is used for the following generations of pseudo-randoms that use XOR with plaintext to generate ciphertext [5], [6]. Each element in the table is exchanged at least once. RC4 is one type of cipher stream, which is processing units or data inputs at a time. In this way encryption or decryption can be implemented at variable lengths. This algorithm does not have to wait for a certain amount of data input before it is processed, or add additional bytes to encrypt. RC4 encryption methods are very fast approximately 10 times faster than DES.

2.2 RC4 Encryption and Decryption

The encryption and decryption processes have the same process so that only one function is performed to run both processes. The following will be given a section describing the series of processes executed to describe or decrypt [8]:

3. RESULTS AND DISCUSSION

3.1 Encryption-Decryption Process Analysis

RC4 Stream Cipher algorithm to perform encryption, the process begins with the initialization of the first Sbox, S [0], S [1]......, S [255], with the numbers 0 to 255. First fill sequentially [0] = 0, S [1] = 1,......, S [255] = 255. Then initialize another array (another S-Box), e.g. array K with a length of 256. Fill array K with the keys to repeat until the entire array K [0], K [1]......, K [255] are fully populated.

1. S-Box initialization process (Array S)
   For i = 0 to 255
   S [i] = i

2. S-Box initialization process (Array K)
   // Array key array with key length "leght"
   For i = 0 to 255
   S [i] = i

3. Then perform the S-Box randomization step
   I = 0; j = 0
   For i = 0 to 255
   { 
     J = (j + S [i] + [k] mod 256)
     Swap S [i] And S [j]
   }

4. Create a byte pseudorandom
   i = (i + 1 ) mod 256
   j = (j + S [i]) mod 256
   swap S [i] And S [j]
t = (S[i] + S[j]) mod 256
K = S[t]

5. Byte K is XOR-kan with plaintext to produce ciphertext or in-XOR- kan with ciphertext to produce plaintext.

Here is the implementation of RC4 algorithm with 256 bytes mode
1. Initialization of S-Box with a length of 256 bytes, with S[0] = 0, S[1] = 1, S[2] = 2 and S[3] = 3,....., S[255] so that array S becomes:

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

2. Initialize the 10-byte key array, Ki. Suppose the key is taken from 10 bytes i.e. "OPEN[space]key" then the sentence will be converted into Decimal form "66 85 75 65 32 107 117 110 99 105". Repeat the key until it meets the entire K array so that array K becomes:

<table>
<thead>
<tr>
<th>Iter</th>
<th>iKey</th>
<th>charKey[i]</th>
<th>Sbox[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B</td>
<td>66</td>
<td>23</td>
</tr>
<tr>
<td>2</td>
<td>U</td>
<td>85</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>K</td>
<td>75</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>65</td>
<td>26</td>
</tr>
<tr>
<td>5</td>
<td>[space]</td>
<td>32</td>
<td>27</td>
</tr>
<tr>
<td>6</td>
<td>K</td>
<td>107</td>
<td>28</td>
</tr>
<tr>
<td>7</td>
<td>U</td>
<td>117</td>
<td>29</td>
</tr>
<tr>
<td>8</td>
<td>N</td>
<td>110</td>
<td>30</td>
</tr>
<tr>
<td>9</td>
<td>C</td>
<td>99</td>
<td>31</td>
</tr>
<tr>
<td>10</td>
<td>I</td>
<td>105</td>
<td>32</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>66</td>
<td>33</td>
</tr>
<tr>
<td>12</td>
<td>U</td>
<td>85</td>
<td>34</td>
</tr>
<tr>
<td>13</td>
<td>K</td>
<td>75</td>
<td>35</td>
</tr>
<tr>
<td>14</td>
<td>A</td>
<td>65</td>
<td>36</td>
</tr>
<tr>
<td>15</td>
<td>[space]</td>
<td>32</td>
<td>37</td>
</tr>
<tr>
<td>16</td>
<td>K</td>
<td>107</td>
<td>38</td>
</tr>
<tr>
<td>17</td>
<td>U</td>
<td>117</td>
<td>39</td>
</tr>
<tr>
<td>18</td>
<td>N</td>
<td>110</td>
<td>40</td>
</tr>
<tr>
<td>19</td>
<td>C</td>
<td>99</td>
<td>41</td>
</tr>
<tr>
<td>20</td>
<td>I</td>
<td>105</td>
<td>42</td>
</tr>
<tr>
<td>21</td>
<td>B</td>
<td>66</td>
<td>43</td>
</tr>
<tr>
<td>22</td>
<td>U</td>
<td>85</td>
<td>44</td>
</tr>
</tbody>
</table>

Jurnal Info Sains: Informatikan dan Sains is licensed under a Creative Commons Attribution-Non Commercial 4.0 International License (CC BY-NC 4.0)
3. Next mix the operation where it will use the variable i And j to index array S[i] And K[i]. First we give the initial value for i And j with 0. The mixing operation is a repeat of the formula (i + S[i] + K[i]) mod 256 followed by the exchange of S[i] with S[j]. For this example, because we are using an array with a length of 256 bytes so the algorithm becomes:

For i = 0 to 256
j = (j + S[i] + K[i]) mod 256
swap S[i] And S[j]

Iteration to 4:

\[
i = 3, \text{ So} \]
\[
j = (j + S[3] + K[3]) \mod 256
\]
\[
= 41
\]

Swap S [3] And S [41]

Iteration to 5:

\[
i = 4, \text{ So} \]
\[
j = (j + S[4] + K[4]) \mod 256
\]
\[
= 77
\]

Swap S [4] And S [77]

Iteration to 6:

\[
i = 5, \text{ So} \]
\[
j = (j + S[5] + K[5]) \mod 256
\]
\[
= 189
\]


Iteration to 7:

\[
i = 6, \text{ So} \]
\[
j = (j + S[6] + K[6]) \mod 256
\]
\[
= 56
\]

Swap S [6] And S [56]

Iteration to 8:

\[
i = 7, \text{ So} \]
\[
j = (j + S[7] + K[7]) \mod 256
\]
\[
= 173
\]


Iteration to 9:

\[
i = 8, \text{ So} \]
\[
j = (j + S[8] + K[8]) \mod 256
\]
\[
= 24
\]


Iteration to 10:

\[
i = 9, \text{ So} \]
\[
j = (j + S[9] + K[9]) \mod 256
\]
\[
= 138
\]


Iteration to 254:

\[
i = 253, \text{ So} \]
\[
j = (j + S[253] + K[253]) \mod 256
\]
\[
= 229
\]

Swap S [253] And S [229]
\[
\begin{align*}
&= (j + S[253] + K[253]) \mod 256 \quad = 34 \\
&= (27+67+65) \mod 256 \quad \text{Swap S[5] And S[34]} \\
&= 159 \quad \text{Iteration to 256:}
\end{align*}
\]

5. Next is the encryption process which is XOR-kan pseudorandom byte with plaintext, e.g. plaintext "ITU[space]DIA". Plaintext consists of 7 characters So occurs 7 iterations. Before iterating, convert the characters to binary number forms.

<table>
<thead>
<tr>
<th>Character</th>
<th>Decimal</th>
<th>Biner</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>73</td>
<td>0100 1001</td>
</tr>
<tr>
<td>T</td>
<td>84</td>
<td>0101 0100</td>
</tr>
<tr>
<td>[space]</td>
<td>32</td>
<td>0010 0000</td>
</tr>
<tr>
<td>U</td>
<td>85</td>
<td>0101 0101</td>
</tr>
<tr>
<td>D</td>
<td>68</td>
<td>0100 0100</td>
</tr>
<tr>
<td>I</td>
<td>73</td>
<td>0100 1001</td>
</tr>
<tr>
<td>A</td>
<td>65</td>
<td>0100 0101</td>
</tr>
</tbody>
</table>

Here’s Iteration to 1: Initialize i And j with i = 0; j = 0;

\[
i = (i + 1) \mod 256 \\
= (0 + 1) \\
= 1
\]

And

\[
j = (j + S[i]) \mod 256 \\
= (j + S[1]) \mod 256 \\
= (0 + 44) \mod 256 \\
= 44
\]

\[
\text{Swap S[1] And S[44]} \\
t = (S[i] + S[j]) \mod 256 \\
= (S[1] + S[44]) \mod 256 \\
= (71 + 44) \mod 256 \\
= 115
\]

\[
\]

\[
\text{Byte K di-XOR with plaintext “I”} \\
\text{Chipertext} \\
\text{Plainteks} 0100 1001
\]

\[
XOR 0101 1100 \\
K = S[t] = S[17] = 202 = 1000 1000
\]

\[
\text{Byte K di-XOR with plaintext “U”} \\
\text{Chipertext} 159
\]

Jurnal Info Sains: Informatikan dan Sains is licensed under a Creative Commons Attribution-Non Commercial 4.0 International License (CC BY-NC 4.0)
Iteration to 4:

\[ i = (i + 1) \mod 256 \]
\[ = (3 + 1) \]
\[ = 4 \]

And

\[ j = (j + S[i] \mod 256) \]
\[ = (j + S[4]) \mod 256 \]
\[ = (177 + 106) \mod 256 \]
\[ = 27 \]

Swap \( S[4] \) And \( S[27] \)

\[ t = (S[i] + S[j]) \mod 256 \]
\[ = (S[4] + S[27]) \mod 256 \]
\[ = (51 + 106) \mod 256 \]
\[ = 157 \]

\( K = S[t] = S[157] = 75 = 0100\ 1010 \)

Byte \( K \) di-XOR-with plaintext “[Space]”

Plainteks \hspace{1em} \text{[Space]} \hspace{1em} 32

0010 0000

Key ( \( K \) ) \hspace{1em} 0100 1011

XOR \hspace{1em} 0110 1011

Chipertext \hspace{1em} 107

\[ K \]

Iteration to 5:

\[ i = (i + 1) \mod 256 \]
\[ = (4 + 1) \]
\[ = 5 \]

And

\[ j = (j + S[i] \mod 256) \]
\[ = (j + S[5]) \mod 256 \]
\[ = (27 + 185) \mod 256 \]
\[ = 212 \]

Swap \( S[5] \) And \( S[212] \)

\[ t = (S[i] + S[j]) \mod 256 \]
\[ = (S[5] + S[212]) \mod 256 \]
\[ = (116 + 185) \mod 256 \]
\[ = 45 \]

\( K = S[t] = S[45] = 223 = 1101\ 1111 \)

Byte \( K \) di-XOR-with plaintext “D”

Plainteks \hspace{1em} D \hspace{1em} 68

0100 0100

Key ( \( K \) ) \hspace{1em} 1101 1111

XOR \hspace{1em} 1010 1011

Chipertext \hspace{1em} 155

\[ K \]

Iteration to 6:

\[ i = (i + 1) \mod 256 \]
\[ = (5 + 1) \]
\[ = 6 \]

And

\[ j = (j + S[i] \mod 256) \]
\[ = (j + S[6]) \mod 256 \]
\[ = (212 + 56) \mod 256 \]
\[ = 12 \]


\[ t = (S[i] + S[j]) \mod 256 \]
\[ = (S[6] + S[12]) \mod 256 \]
\[ = (141 + 56) \mod 256 \]
\[ = 197 \]

\( K = S[t] = S[197] = 42 = 0010\ 1010 \)

Byte \( K \) di-XOR-with plaintext “I”

Plainteks \hspace{1em} I \hspace{1em} 73

0100 1001

Key ( \( K \) ) \hspace{1em} 0010 1010

XOR \hspace{1em} 0110 0011

Chipertext \hspace{1em} 99

\[ C \]

Iteration to 7:

\[ i = (i + 1) \mod 256 \]
\[ = (6 + 1) \]
\[ = 7 \]

And

\[ j = (j + S[i] \mod 256) \]
\[ = (j + S[7]) \mod 256 \]
\[ = (12 + 48) \mod 256 \]
\[ = 60 \]

Swap \( S[7] \) And \( S[60] \)

\[ t = (S[i] + S[j]) \mod 256 \]
\[ = (S[7] + S[60]) \mod 256 \]
\[ = (64 + 48) \mod 256 \]
\[ = 112 \]

\( K = S[t] = S[112] = 42 = 0010\ 0111 \)

Byte \( K \) di-XOR-with plaintext “A”

Plainteks \hspace{1em} A \hspace{1em} 65

0100 0001

Key ( \( K \) ) \hspace{1em} 0010 0111

XOR \hspace{1em} 0110 0110

Chipertext \hspace{1em} 102

\[ F \]

So the results of encryption obtained after going through several iterations are as follows:
6. Next is the decryption process which is XOR-kan pseudo random byte with Chipertext, And Chipertext is "G \ Ū k > c f". Chipertext consists of 7 characters so occurs 7 iterations. Before iterating, convert characters to binary number form.

<table>
<thead>
<tr>
<th>Character</th>
<th>Decimal</th>
<th>Biner</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>71</td>
<td>0100 0111</td>
</tr>
<tr>
<td>\</td>
<td>92</td>
<td>0101 1100</td>
</tr>
<tr>
<td>Ū</td>
<td>159</td>
<td>1001 1111</td>
</tr>
<tr>
<td>k</td>
<td>107</td>
<td>0110 1011</td>
</tr>
<tr>
<td>&gt;</td>
<td>155</td>
<td>1001 1011</td>
</tr>
<tr>
<td>c</td>
<td>99</td>
<td>0110 0011</td>
</tr>
<tr>
<td>f</td>
<td>102</td>
<td>0110 0110</td>
</tr>
</tbody>
</table>

The data is sent in the form of a chipertext so that once it reaches the recipient the message can be re-converted to plaintext by XOR-kan with the same key.

Chipertext:  G \ Ū k > c f
Kuncii: BUKA [space] key
PlaintextITU [space] DIA

Berikut iterasi 1:
Plaintext: G
          0100 0111
Key (K): 0000 1110
XOR: 0100 1001
Chipertext: 73
          0110 1011

Here's iteration 2:
Plaintext: \ 
          0101 1100
Key (K): 0000 1000
XOR: 0101 0100
Chipertext: 84
          1001 1011

Here's iteration 3:
Plaintext: Ū
          1100 1111
Key (K): 1100 1011
XOR: 0010 0000
Chipertext: 68
          1111 0000

Berikut iterasi 4:
Plaintext: I
          0110 1011
Key (K): 0100 1011
XOR: 1001 1010
Chipertext: 44
          0000 0100

Here's iteration 5:
Plaintext: >
          1111 1000
Key (K): 1101 1111
XOR: 0100 0100
Chipertext: 44
          0000 0100

Here's iteration 6:
Plaintext: G
          0100 0111
Key (K): 1001 1111
XOR: 1000 0100
Chipertext: 73
          0110 1011
Here's iteration 7:

<table>
<thead>
<tr>
<th>Plaintext</th>
<th>Key (K)</th>
<th>XOR</th>
<th>Chipertext</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0110 0110</td>
<td>0100 1001</td>
<td>73</td>
</tr>
<tr>
<td>99</td>
<td>0100 1010</td>
<td>0100 1001</td>
<td>65</td>
</tr>
</tbody>
</table>

4. CONCLUSION

With aAndyes SMS encryption and decryption application with RC4 algorithm, it can provide convenience for users to encrypt SMS messages without having to do calculations. This application uses RC4 algorithm which has the advantage of having a high level of security and speed processes. Android-based SMS encryption apps can help users describe SMS messages before they're sent.

REFERENCE


