A Description of the Camellia Encryption Algorithm

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2004). All Rights Reserved.

Abstract

This document describes the Camellia encryption algorithm. Camellia is a block cipher with 128-bit block size and 128-, 192-, and 256-bit keys. The algorithm description is presented together with key scheduling part and data randomizing part.

1. Introduction

1.1. Camellia

Camellia was jointly developed by Nippon Telegraph and Telephone Corporation and Mitsubishi Electric Corporation in 2000 [CamelliaSpec]. Camellia specifies the 128-bit block size and 128-, 192-, and 256-bit key sizes, the same interface as the Advanced Encryption Standard (AES). Camellia is characterized by its suitability for both software and hardware implementations as well as its high level of security. From a practical viewpoint, it is designed to enable flexibility in software and hardware implementations on 32-bit processors widely used over the Internet and many applications, 8-bit processors used in smart cards, cryptographic hardware, embedded systems, and so on [CamelliaTech]. Moreover, its key setup time is excellent, and its key agility is superior to that of AES.
Camellia has been scrutinized by the wide cryptographic community during several projects for evaluating crypto algorithms. In particular, Camellia was selected as a recommended cryptographic primitive by the EU NESSIE (New European Schemes for Signatures, Integrity and Encryption) project [NESSIE] and also included in the list of cryptographic techniques for Japanese e-Government systems which were selected by the Japan CRYPTREC (Cryptography Research and Evaluation Committees) [CRYPTREC].

2. Algorithm Description

Camellia can be divided into "key scheduling part" and "data randomizing part".

2.1. Terminology

The following operators are used in this document to describe the algorithm.

\&  bitwise AND operation.
|   bitwise OR operation.
^   bitwise exclusive-OR operation.
<<  logical left shift operation.
>>  logical right shift operation.
<<< left rotation operation.
~y  bitwise complement of y.
0x  hexadecimal representation.

Note that the logical left shift operation is done with the infinite data width.

The constant values of MASK8, MASK32, MASK64, and MASK128 are defined as follows.

```
MASK8   = 0xff;
MASK32  = 0xffffffff;
MASK64  = 0xffffffffffffffff;
MASK128 = 0xffffffffffffffffffffffffffffffff;
```

2.2. Key Scheduling Part

In the key schedule part of Camellia, the 128-bit variables of KL and KR are defined as follows. For 128-bit keys, the 128-bit key K is used as KL and KR is 0. For 192-bit keys, the leftmost 128-bits of key K are used as KL and the concatenation of the rightmost 64-bits of K and the complement of the rightmost 64-bits of K are used as KR. For 256-bit keys, the leftmost 128-bits of key K are used as KL and the rightmost 128-bits of K are used as KR.
128-bit key K:
   KL = K;    KR = 0;

192-bit key K:
   KL = K >> 64;
   KR = ((K & MASK64) << 64) | (~(K & MASK64));

256-bit key K:
   KL = K >> 128;
   KR = K & MASK128;

The 128-bit variables KA and KB are generated from KL and KR as follows. Note that KB is used only if the length of the secret key is 192 or 256 bits. D1 and D2 are 64-bit temporary variables. F-function is described in Section 2.4.

D1 = (KL ^ KR) >> 64;
D2 = (KL ^ KR) & MASK64;
D2 = D2 ^ F(D1, Sigma1);
D1 = D1 ^ F(D2, Sigma2);
D1 = D1 ^ (KL >> 64);
D2 = D2 ^ (KL & MASK64);
D2 = D2 ^ F(D1, Sigma3);
D1 = D1 ^ F(D2, Sigma4);
KA = (D1 << 64) | D2;
D1 = (KA ^ KR) >> 64;
D2 = (KA ^ KR) & MASK64;
D2 = D2 ^ F(D1, Sigma5);
D1 = D1 ^ F(D2, Sigma6);
KB = (D1 << 64) | D2;

The 64-bit constants Sigma1, Sigma2, ..., Sigma6 are used as "keys" in the F-function. These constant values are, in hexadecimal notation, as follows.

Sigma1 = 0xA09E667F3BCC908B;
Sigma2 = 0xB67AE8584CAA73B2;
Sigma3 = 0xC6EF372FE94F82BE;
Sigma4 = 0x54FF53A5F1D36F1C;
Sigma5 = 0x10E527FADE682D1D;
Sigma6 = 0xB05688C2B3E6C1FD;

64-bit subkeys are generated by rotating KL, KR, KA, and KB and taking the left- or right-half of them.
For 128-bit keys, 64-bit subkeys kw1, ..., kw4, k1, ..., k18, ke1, ..., ke4 are generated as follows.

kw1 = (KL <<< 0) >> 64;
kw2 = (KL <<< 0) & MASK64;
k1 = (KA <<< 0) >> 64;
k2 = (KA <<< 0) & MASK64;
k3 = (KL <<< 15) >> 64;
k4 = (KL <<< 15) & MASK64;
k5 = (KA <<< 15) >> 64;
k6 = (KA <<< 15) & MASK64;
ke1 = (KA <<< 30) >> 64;
ke2 = (KA <<< 30) & MASK64;
k7 = (KL <<< 45) >> 64;
k8 = (KL <<< 45) & MASK64;
k9 = (KA <<< 45) >> 64;
k10 = (KL <<< 60) & MASK64;
k11 = (KA <<< 60) >> 64;
k12 = (KA <<< 60) & MASK64;
ke3 = (KL <<< 77) >> 64;
ke4 = (KL <<< 77) & MASK64;
k13 = (KL <<< 94) >> 64;
k14 = (KL <<< 94) & MASK64;
k15 = (KA <<< 94) >> 64;
k16 = (KA <<< 94) & MASK64;
k17 = (KL <<< 111) >> 64;
k18 = (KL <<< 111) & MASK64;
kw3 = (KA <<< 111) >> 64;
kw4 = (KA <<< 111) & MASK64;

For 192- and 256-bit keys, 64-bit subkeys kw1, ..., kw4, k1, ..., k24, ke1, ..., ke6 are generated as follows.

kw1 = (KL <<< 0) >> 64;
kw2 = (KL <<< 0) & MASK64;
k1 = (KB <<< 0) >> 64;
k2 = (KB <<< 0) & MASK64;
k3 = (KR <<< 15) >> 64;
k4 = (KR <<< 15) & MASK64;
k5 = (KA <<< 15) >> 64;
k6 = (KA <<< 15) & MASK64;
ke1 = (KR <<< 30) >> 64;
ke2 = (KR <<< 30) & MASK64;
k7 = (KB <<< 30) >> 64;
k8 = (KB <<< 30) & MASK64;
k9 = (KL <<< 45) >> 64;
k10 = (KL <<< 45) & MASK64;
k11 = (KA <<< 45) >> 64;
RFC 3713: Camellia Encryption Algorithm

k12 = (KA <<< 45) & MASK64;
k13 = (KR <<< 60) >> 64;
k14 = (KR <<< 60) & MASK64;
k15 = (KB <<< 60) >> 64;
k16 = (KB <<< 60) & MASK64;
k17 = (KL <<< 77) >> 64;
k18 = (KL <<< 77) & MASK64;
k19 = (KR <<< 94) >> 64;
k20 = (KR <<< 94) & MASK64;
k21 = (KA <<< 94) >> 64;
k22 = (KA <<< 94) & MASK64;
k23 = (KL <<< 111) >> 64;
k24 = (KL <<< 111) & MASK64;
kw3 = (KB <<< 111) >> 64;
kw4 = (KB <<< 111) & MASK64;

2.3. Data Randomizing Part

2.3.1. Encryption for 128-bit keys

128-bit plaintext M is divided into the left 64-bit D1 and the right 64-bit D2.

D1 = M >> 64;
D2 = M & MASK64;

Encryption is performed using an 18-round Feistel structure with FL- and FLINV-functions inserted every 6 rounds. F-function, FL-function, and FLINV-function are described in Section 2.4.

D1 = D1 ^ kw1;       // Prewhitening
D2 = D2 ^ kw2;
D2 = D2 ^ F(D1, k1); // Round 1
D1 = D1 ^ F(D2, k2); // Round 2
D2 = D2 ^ F(D1, k3); // Round 3
D1 = D1 ^ F(D2, k4); // Round 4
D2 = D2 ^ F(D1, k5); // Round 5
D1 = D1 ^ F(D2, k6); // Round 6
D1 = FL   (D1, ke1); // FL
D2 = FLINV(D2, ke2); // FLINV
D2 = D2 ^ F(D1, k7); // Round 7
D1 = D1 ^ F(D2, k8); // Round 8
D2 = D2 ^ F(D1, k9); // Round 9
D1 = D1 ^ F(D2, k10); // Round 10
D2 = D2 ^ F(D1, k11);    // Round 11
D1 = D1 ^ F(D2, k12);    // Round 12
D1 = FL   (D1, ke3);     // FL
D2 = FLINV(D2, ke4);     // FLINV
D2 = D2 ^ F(D1, k13);    // Round 13
D1 = D1 ^ F(D2, k14);    // Round 14
D2 = D2 ^ F(D1, k15);    // Round 15
D1 = D1 ^ F(D2, k16);    // Round 16
D2 = D2 ^ F(D1, k17);    // Round 17
D1 = D1 ^ F(D2, k18);    // Round 18
D2 = D2 ^ kw3;           // Postwhitening
D1 = D1 ^ kw4;

128-bit ciphertext C is constructed from D1 and D2 as follows.

C = (D2 << 64) | D1;

2.3.2. Encryption for 192- and 256-bit keys

128-bit plaintext M is divided into the left 64-bit D1 and the right
64-bit D2.

D1 = M >> 64;
D2 = M & MASK64;

Encryption is performed using a 24-round Feistel structure with FL-
and FLINV-functions inserted every 6 rounds. F-function, FL-function,
and FLINV-function are described in Section 2.4.

D1 = D1 ^ kw1;           // Prewhitening
D2 = D2 ^ kw2;
D2 = D2 ^ F(D1, k1);     // Round 1
D1 = D1 ^ F(D2, k2);     // Round 2
D2 = D2 ^ F(D1, k3);     // Round 3
D1 = D1 ^ F(D2, k4);     // Round 4
D2 = D2 ^ F(D1, k5);     // Round 5
D1 = D1 ^ F(D2, k6);     // Round 6
D1 = FL   (D1, ke1);     // FL
D2 = FLINV(D2, ke2);     // FLINV
D2 = D2 ^ F(D1, k7);     // Round 7
D1 = D1 ^ F(D2, k8);     // Round 8
D2 = D2 ^ F(D1, k9);     // Round 9
D1 = D1 ^ F(D2, k10);    // Round 10
D2 = D2 ^ F(D1, k11);    // Round 11
D1 = D1 ^ F(D2, k12);    // Round 12
D1 = FL   (D1, ke3);     // FL
D2 = FLINV(D2, ke4);     // FLINV
D2 = D2 ^ F(D1, k13);    // Round 13
128-bit ciphertext $C$ is constructed from $D_1$ and $D_2$ as follows.

$$C = (D_2 << 64) | D_1;$$

### 2.3.3. Decryption

The decryption procedure of Camellia can be done in the same way as the encryption procedure by reversing the order of the subkeys.

That is to say:

**128-bit key:**
- $kw_1 \leftrightarrow kw_3$
- $kw_2 \leftrightarrow kw_4$
- $k_1 \leftrightarrow k_{18}$
- $k_2 \leftrightarrow k_{17}$
- $k_3 \leftrightarrow k_{16}$
- $k_4 \leftrightarrow k_{15}$
- $k_5 \leftrightarrow k_{14}$
- $k_6 \leftrightarrow k_{13}$
- $k_7 \leftrightarrow k_{12}$
- $k_8 \leftrightarrow k_{11}$
- $k_9 \leftrightarrow k_{10}$
- $ke_1 \leftrightarrow ke_4$
- $ke_2 \leftrightarrow ke_3$

**192- or 256-bit key:**
- $kw_1 \leftrightarrow kw_3$
- $kw_2 \leftrightarrow kw_4$
- $k_1 \leftrightarrow k_{24}$
- $k_2 \leftrightarrow k_{23}$
- $k_3 \leftrightarrow k_{22}$
k4 <-> k21
k5 <-> k20
k6 <-> k19
k7 <-> k18
k8 <-> k17
k9 <-> k16
k10 <-> k15
k11 <-> k14
k12 <-> k13
ke1 <-> ke6
ke2 <-> ke5
ke3 <-> ke4

2.4. Components of Camellia

2.4.1. F-function

F-function takes two parameters. One is 64-bit input data F_IN. The other is 64-bit subkey KE. F-function returns 64-bit data F_OUT.

F(F_IN, KE)
begin
  var x as 64-bit unsigned integer;
  var t1, t2, t3, t4, t5, t6, t7, t8 as 8-bit unsigned integer;
  var y1, y2, y3, y4, y5, y6, y7, y8 as 8-bit unsigned integer;
  x = F_IN ^ KE;
  t1 = x >> 56;
  t2 = (x >> 48) & MASK8;
  t3 = (x >> 40) & MASK8;
  t4 = (x >> 32) & MASK8;
  t5 = (x >> 24) & MASK8;
  t6 = (x >> 16) & MASK8;
  t7 = (x >>  8) & MASK8;
  t8 = x & MASK8;
  t1 = SBOX1[t1];
  t2 = SBOX2[t2];
  t3 = SBOX3[t3];
  t4 = SBOX4[t4];
  t5 = SBOX2[t5];
  t6 = SBOX3[t6];
  t7 = SBOX4[t7];
  t8 = SBOX1[t8];
  y1 = t1 ^ t3 ^ t4 ^ t6 ^ t7 ^ t8;
  y2 = t1 ^ t2 ^ t4 ^ t5 ^ t7 ^ t8;
  y3 = t1 ^ t2 ^ t3 ^ t5 ^ t6 ^ t8;
  y4 = t2 ^ t3 ^ t4 ^ t5 ^ t6 ^ t7;
  y5 = t1 ^ t2 ^ t6 ^ t7 ^ t8;
  y6 = t2 ^ t3 ^ t5 ^ t7 ^ t8;
\[ y_7 = t_3 \oplus t_4 \oplus t_5 \oplus t_6 \oplus t_8; \]
\[ y_8 = t_1 \oplus t_4 \oplus t_5 \oplus t_6 \oplus t_7; \]
\[ \text{F\_OUT} = (y_1 \ll 56) | (y_2 \ll 48) | (y_3 \ll 40) | (y_4 \ll 32) \]
\[ | (y_5 \ll 24) | (y_6 \ll 16) | (y_7 \ll 8) | y_8; \]

return F\_OUT;
end.

SBOX1, SBOX2, SBOX3, and SBOX4 are lookup tables with 8-bit input/output data. SBOX2, SBOX3, and SBOX4 are defined using SBOX1 as follows:

\[ \text{SBOX2}[x] = \text{SBOX1}[x] \ll 1; \]
\[ \text{SBOX3}[x] = \text{SBOX1}[x] \ll 7; \]
\[ \text{SBOX4}[x] = \text{SBOX1}[x \ll 1]; \]

SBOX1 is defined by the following table. For example, SBOX1[0x3d] equals 86.

SBOX1:

<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>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:</td>
<td>112</td>
<td>130</td>
<td>44</td>
<td>236</td>
<td>179</td>
<td>39</td>
<td>192</td>
<td>229</td>
<td>228</td>
<td>133</td>
<td>87</td>
<td>53</td>
<td>234</td>
<td>12</td>
<td>174</td>
</tr>
<tr>
<td>10:</td>
<td>35</td>
<td>239</td>
<td>107</td>
<td>147</td>
<td>69</td>
<td>25</td>
<td>165</td>
<td>33</td>
<td>237</td>
<td>14</td>
<td>79</td>
<td>78</td>
<td>29</td>
<td>101</td>
<td>146</td>
</tr>
<tr>
<td>20:</td>
<td>134</td>
<td>184</td>
<td>175</td>
<td>143</td>
<td>124</td>
<td>235</td>
<td>31</td>
<td>206</td>
<td>62</td>
<td>48</td>
<td>220</td>
<td>95</td>
<td>94</td>
<td>197</td>
<td>11</td>
</tr>
<tr>
<td>30:</td>
<td>166</td>
<td>225</td>
<td>57</td>
<td>202</td>
<td>213</td>
<td>71</td>
<td>93</td>
<td>61</td>
<td>217</td>
<td>1</td>
<td>90</td>
<td>214</td>
<td>81</td>
<td>86</td>
<td>108</td>
</tr>
<tr>
<td>40:</td>
<td>139</td>
<td>13</td>
<td>154</td>
<td>102</td>
<td>251</td>
<td>204</td>
<td>176</td>
<td>45</td>
<td>116</td>
<td>18</td>
<td>43</td>
<td>32</td>
<td>240</td>
<td>177</td>
<td>132</td>
</tr>
<tr>
<td>50:</td>
<td>223</td>
<td>76</td>
<td>203</td>
<td>194</td>
<td>52</td>
<td>126</td>
<td>118</td>
<td>5</td>
<td>109</td>
<td>183</td>
<td>169</td>
<td>49</td>
<td>209</td>
<td>23</td>
<td>4</td>
</tr>
<tr>
<td>60:</td>
<td>20</td>
<td>88</td>
<td>58</td>
<td>97</td>
<td>222</td>
<td>27</td>
<td>17</td>
<td>28</td>
<td>50</td>
<td>15</td>
<td>156</td>
<td>22</td>
<td>83</td>
<td>24</td>
<td>242</td>
</tr>
<tr>
<td>70:</td>
<td>254</td>
<td>68</td>
<td>207</td>
<td>178</td>
<td>195</td>
<td>181</td>
<td>122</td>
<td>145</td>
<td>36</td>
<td>8</td>
<td>232</td>
<td>168</td>
<td>96</td>
<td>252</td>
<td>105</td>
</tr>
<tr>
<td>80:</td>
<td>170</td>
<td>208</td>
<td>160</td>
<td>125</td>
<td>161</td>
<td>137</td>
<td>98</td>
<td>151</td>
<td>84</td>
<td>91</td>
<td>30</td>
<td>149</td>
<td>224</td>
<td>255</td>
<td>100</td>
</tr>
<tr>
<td>90:</td>
<td>16</td>
<td>196</td>
<td>0</td>
<td>72</td>
<td>163</td>
<td>247</td>
<td>117</td>
<td>219</td>
<td>138</td>
<td>3</td>
<td>230</td>
<td>218</td>
<td>9</td>
<td>63</td>
<td>221</td>
</tr>
<tr>
<td>a0:</td>
<td>135</td>
<td>92</td>
<td>131</td>
<td>2</td>
<td>205</td>
<td>74</td>
<td>144</td>
<td>51</td>
<td>115</td>
<td>103</td>
<td>246</td>
<td>243</td>
<td>157</td>
<td>127</td>
<td>191</td>
</tr>
<tr>
<td>b0:</td>
<td>82</td>
<td>155</td>
<td>216</td>
<td>38</td>
<td>200</td>
<td>55</td>
<td>198</td>
<td>59</td>
<td>129</td>
<td>150</td>
<td>111</td>
<td>75</td>
<td>19</td>
<td>190</td>
<td>99</td>
</tr>
<tr>
<td>c0:</td>
<td>233</td>
<td>121</td>
<td>167</td>
<td>140</td>
<td>159</td>
<td>110</td>
<td>188</td>
<td>142</td>
<td>41</td>
<td>245</td>
<td>249</td>
<td>182</td>
<td>47</td>
<td>253</td>
<td>180</td>
</tr>
<tr>
<td>d0:</td>
<td>120</td>
<td>152</td>
<td>6</td>
<td>106</td>
<td>231</td>
<td>70</td>
<td>113</td>
<td>186</td>
<td>212</td>
<td>37</td>
<td>171</td>
<td>66</td>
<td>136</td>
<td>162</td>
<td>141</td>
</tr>
<tr>
<td>e0:</td>
<td>114</td>
<td>7</td>
<td>185</td>
<td>85</td>
<td>248</td>
<td>238</td>
<td>172</td>
<td>10</td>
<td>54</td>
<td>73</td>
<td>42</td>
<td>104</td>
<td>60</td>
<td>56</td>
<td>241</td>
</tr>
<tr>
<td>f0:</td>
<td>64</td>
<td>40</td>
<td>211</td>
<td>123</td>
<td>187</td>
<td>201</td>
<td>67</td>
<td>193</td>
<td>21</td>
<td>227</td>
<td>173</td>
<td>244</td>
<td>119</td>
<td>199</td>
<td>128</td>
</tr>
</tbody>
</table>

### 2.4.2. FL- and FLINV-functions

FL-function takes two parameters. One is 64-bit input data FL\_IN. The other is 64-bit subkey KE. FL-function returns 64-bit data FL\_OUT.

**FL(FL\_IN, KE)**

begin
  var x1, x2 as 32-bit unsigned integer;
  var k1, k2 as 32-bit unsigned integer;
  x1 = FL\_IN \gg 32;

x2 = FL_IN & MASK32;
k1 = KE >> 32;
k2 = KE & MASK32;
x2 = x2 ^ ((x1 & k1) <<< 1);
x1 = x1 ^ (x2 | k2);
FL_OUT = (x1 << 32) | x2;
end.

FLINV-function is the inverse function of the FL-function.

FLINV(FLINV_IN, KE)
begin
  var y1, y2 as 32-bit unsigned integer;
  var k1, k2 as 32-bit unsigned integer;
  y1 = FLINV_IN >> 32;
  y2 = FLINV_IN & MASK32;
  k1 = KE >> 32;
  k2 = KE & MASK32;
  y1 = y1 ^ (y2 | k2);
  y2 = y2 ^ ((y1 & k1) <<< 1);
  FLINV_OUT = (y1 << 32) | y2;
end.

3. Object Identifiers

The Object Identifier for Camellia with 128-bit key in Cipher Block Chaining (CBC) mode is as follows:

   id-camellia128-cbc OBJECT IDENTIFIER ::= 
       { iso(1) member-body(2) 392 200011 61 security(1) 
         algorithm(1) symmetric-encryption-algorithm(1) 
         camellia128-cbc(2) }

The Object Identifier for Camellia with 192-bit key in Cipher Block Chaining (CBC) mode is as follows:

   id-camellia192-cbc OBJECT IDENTIFIER ::= 
       { iso(1) member-body(2) 392 200011 61 security(1) 
         algorithm(1) symmetric-encryption-algorithm(1) 
         camellia192-cbc(3) }

The Object Identifier for Camellia with 256-bit key in Cipher Block Chaining (CBC) mode is as follows:

   id-camellia256-cbc OBJECT IDENTIFIER ::= 
       { iso(1) member-body(2) 392 200011 61 security(1) 
         algorithm(1) symmetric-encryption-algorithm(1) 
         camellia256-cbc(4) }
The above algorithms need Initialization Vector (IV). To determine the value of IV, the above algorithms take parameters as follows:

CamelliaCBCParameter ::= CamelliaIV -- Initialization Vector

CamelliaIV ::= OCTET STRING (SIZE(16))

When these object identifiers are used, plaintext is padded before encryption according to RFC2315 [RFC2315].

4. Security Considerations

The recent advances in cryptanalytic techniques are remarkable. A quantitative evaluation of security against powerful cryptanalytic techniques such as differential cryptanalysis and linear cryptanalysis is considered to be essential in designing any new block cipher. We evaluated the security of Camellia by utilizing state-of-the-art cryptanalytic techniques. We confirmed that Camellia has no differential and linear characteristics that hold with probability more than $2^{-128}$, which means that it is extremely unlikely that differential and linear attacks will succeed against the full 18-round Camellia. Moreover, Camellia was designed to offer security against other advanced cryptanalytic attacks including higher order differential attacks, interpolation attacks, related-key attacks, truncated differential attacks, and so on [Camellia].

5. Informative References


Appendix A. Example Data of Camellia

Here are test data for Camellia in hexadecimal form.

128-bit key
Key       : 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
Plaintext : 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
Ciphertext: 67 67 31 38 54 96 69 73 08 57 06 56 48 ea be 43

192-bit key
Key       : 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
            : 00 11 22 33 44 55 66 77
Plaintext : 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
Ciphertext: b4 99 34 01 b3 e9 96 f8 4e e5 ce e7 d7 9b 09 b9

256-bit key
Key       : 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
            : 00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
Plaintext : 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
Ciphertext: 9a cc 23 7d ff 16 d7 6c 20 ef 7c 91 9e 3a 75 09
Acknowledgements

Shiho Moriai worked for NTT when this document was developed.

Authors’ Addresses

Mitsuru Matsui
Mitsubishi Electric Corporation
Information Technology R&D Center
5-1-1 Ofuna, Kamakura
Kanagawa 247-8501, Japan
Phone: +81-467-41-2190
Fax: +81-467-41-2185
EMail: matsui@iss.isl.melco.co.jp

Junko Nakajima
Mitsubishi Electric Corporation
Information Technology R&D Center
5-1-1 Ofuna, Kamakura
Kanagawa 247-8501, Japan
Phone: +81-467-41-2190
Fax: +81-467-41-2185
EMail: june15@iss.isl.melco.co.jp

Shiho Moriai
Sony Computer Entertainment Inc.
Phone: +81-3-6438-7523
Fax: +81-3-6438-8629
EMail: shiho@rd.scei.sony.co.jp
camellia@isl.ntt.co.jp (Camellia team)