SKIPJACK and KEA Algorithm Specifications

Version 2.0

29 May 1998
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.   Introduction</td>
<td>3</td>
</tr>
<tr>
<td>II.  Algorithms</td>
<td>3</td>
</tr>
<tr>
<td>A. SKIPJACK Modes of Operation</td>
<td>3</td>
</tr>
<tr>
<td>B. SKIPJACK Specification</td>
<td>5</td>
</tr>
<tr>
<td>1. Notation and Terminology</td>
<td>5</td>
</tr>
<tr>
<td>2. Basic Structure</td>
<td>5</td>
</tr>
<tr>
<td>3. Stepping Rule Equations</td>
<td>6</td>
</tr>
<tr>
<td>4. Stepping Sequence</td>
<td>6</td>
</tr>
<tr>
<td>5. G-Permutation</td>
<td>7</td>
</tr>
<tr>
<td>6. Cryptovariable Schedule</td>
<td>7</td>
</tr>
<tr>
<td>7. F-Table</td>
<td>8</td>
</tr>
<tr>
<td>C.   KEA Specification</td>
<td>9</td>
</tr>
<tr>
<td>D.   E-Mail Applications of KEA</td>
<td>13</td>
</tr>
<tr>
<td>1. Sending E-Mail</td>
<td>13</td>
</tr>
<tr>
<td>2. Receiving E-Mail</td>
<td>15</td>
</tr>
<tr>
<td>III. ANNEX - Test Vectors</td>
<td>18</td>
</tr>
<tr>
<td>A. SKIPJACK Codebook Mode</td>
<td>18</td>
</tr>
<tr>
<td>B. Key Exchange Algorithm</td>
<td>19</td>
</tr>
<tr>
<td>C. KEA Exchange for E-Mail</td>
<td>21</td>
</tr>
<tr>
<td>IV.  References</td>
<td>23</td>
</tr>
</tbody>
</table>
I. Introduction

This document provides details of the SKIPJACK and KEA algorithms. The algorithms are supported in single chip cryptoproducts such as CLIPPER (SKIPJACK only), CAPSTONE, KEYSTONE, REGENT, KRYPTON and the FORTEZZA and FORTEZZA Plus PC Card firmware which runs on them, and also in other FORTEZZA family products.

II. Algorithms

This document will discuss the following algorithms:

- **SKIPJACK**   Codebook Encryptor/Decryptor Algorithm
- **KEA**        Key Exchange Algorithm

### A. SKIPJACK Modes of Operation

SKIPJACK is a 64 bit codebook utilizing an 80-bit cryptovariable. The modes of operation are a subset of the FIPS-81 description of modes of operation for DES [1]. These include:

- **Output Feed-back (OFB) Modes**   64 bit
- **Cipher Feed-Back (CFB) Modes**   64 bit/32 bit/16 bit/8 bit
- **Codebook**                        64 bit
- **Cipher-Block Chaining (CBC)**     64 bit

![Diagram](image)

Figure 1: “Output Feed-Back Modes Diagram
Figure 2: “Cipher Feed-Back Mode Diagram”

Figure 3: “Codebook Mode Diagram”

Figure 4: “Cipher-Block Chaining Mode Diagram”
B. SKIPJACK Specification

1. Notation and terminology

$V^n$: the set of all $n$-bit values.

word: an element of $V^{16}$; a 16-bit value.

byte: an element of $V^8$; an 8-bit value.

permutation of $V^n$: an invertible (one-to-one and onto) function from $V^n$ to $V^n$. That is, the values are permuted within $V^n$, not the bits within the value.

$X \oplus Y$ the bitwise exclusive-or of $X$ and $Y$.

$X \parallel Y$ $X$ concatenated with $Y$. Let $X, Y$ be bytes, then $X \parallel Y = X \times 2^8 + Y$ is a word. Furthermore, $X$ is the high-order byte, and $Y$ is the low-order byte.

2. Basic Structure: SKIPJACK encrypts 4-word (i.e., 8-byte) data blocks by alternating between the two stepping rules (A and B) shown below. A step of rule A does the following:

a. $G$ permutes $w_1$,

b. the new $w_1$ is the xor of the $G$ output, the counter, and $w_4$,

c. words $w_2$ and $w_3$ shift one register to the right; i.e., become $w_3$, and $w_4$ respectively,

d. the new $w_2$ is the $G$ output,

e. the counter is incremented by one.

Rule B works similarly.

![Diagram of Rule A](image)

![Diagram of Rule B](image)

Figure 5: “SKIPJACK Stepping Rules”
3. **Stepping rule equations.** In the equations below, the superscript is the step number.

\[
\begin{align*}
\text{ENCRYPT} \\
\text{Rule A} & \quad w_{1}^{k+1} = G^{k}(w_{1}^{k}) \oplus w_{4}^{k} \oplus \text{counter}^{k} \\
& \quad w_{2}^{k+1} = G^{k}(w_{2}^{k}) \\
& \quad w_{3}^{k+1} = w_{2}^{k} \\
& \quad w_{4}^{k+1} = w_{3}^{k} \\
\text{Rule B} & \quad w_{1}^{k+1} = w_{4}^{k} \\
& \quad w_{2}^{k+1} = G^{k}(w_{1}^{k}) \\
& \quad w_{3}^{k+1} = w_{1}^{k} \oplus w_{2}^{k} \oplus \text{counter}^{k} \\
& \quad w_{4}^{k+1} = w_{3}^{k}
\end{align*}
\]

\[
\begin{align*}
\text{DECRYPT} \\
\text{Rule A}^{-1} & \quad w_{1}^{k-1} = [G^{k-1}]^{-1}(w_{2}^{k}) \\
& \quad w_{2}^{k-1} = w_{3}^{k} \\
& \quad w_{3}^{k-1} = w_{4}^{k} \\
& \quad w_{4}^{k-1} = w_{1}^{k} \oplus w_{2}^{k} \oplus \text{counter}^{k-1} \\
\text{Rule B}^{-1} & \quad w_{1}^{k-1} = [G^{k-1}]^{-1}(w_{2}^{k}) \\
& \quad w_{2}^{k-1} = [G^{k-1}]^{-1}(w_{2}^{k}) \oplus w_{3}^{k} \oplus \text{counter}^{k-1} \\
& \quad w_{3}^{k-1} = w_{4}^{k} \\
& \quad w_{4}^{k-1} = w_{1}^{k}
\end{align*}
\]

4. **Stepping sequence:** The algorithm requires a total of 32 steps.

a. To encrypt: The input is \(w_{i}^{0}, 1 \leq i \leq 4\), (i.e., \(k = 0\) for the beginning step). Start the counter at 1. Step according to Rule A for 8 steps, then switch to Rule B and step 8 more times. Return to rule A for the next 8 steps, then complete the encryption with 8 steps in Rule B. The counter increments by one after each step. The output is \(w_{i}^{32}, 1 \leq i \leq 4\).

b. To decrypt: the input is \(w_{i}^{32}, 1 \leq i \leq 4\), (i.e., \(k = 32\) for the beginning step). Start the counter at 32. Step according to Rule B\(^{-1}\) for 8 steps, then switch to Rule A\(^{-1}\) and step 8 more times. Return to Rule B\(^{-1}\) for the next 8 steps, then complete the decryption with 8 steps in rule A\(^{-1}\). the counter decrements by one after every step. The output is \(w_{i}^{0}, 1 \leq i \leq 4\).
5. **G-permutation**: The crypto-variable-dependent permutation \( G \) on \( V^{16} \) is a four-round Feistel structure. The round function is a fixed byte-substitution table (permutation on \( V^8 \)), which will be called the \( F \)-table. Each round of \( G \) also incorporates a byte of crypto-variable. We give two characterizations of the function below:

a. recursively (mathematically): \( G^k(w = g_1 \parallel g_2) = g_5 \parallel g_6 \) where \( g_i = F(g_{i-1} \oplus cv_{4k+i-3}) \oplus g_{i-2} \) and where \( k \) is the step number (the first step is 0), \( F \) is the substitution table, and \( cv_{4k+i-3} \) is the \((4k + i - 3)\)th byte in the crypto-variable schedule. Thus,

\[
\begin{align*}
g_3 & = F(g_2 \oplus cv_{4k}) \oplus g_1 \\
g_4 & = F(g_3 \oplus cv_{4k+1}) \oplus g_2 \\
g_5 & = F(g_4 \oplus cv_{4k+2}) \oplus g_3 \\
g_6 & = F(g_5 \oplus cv_{4k+3}) \oplus g_4 \\
\end{align*}
\]

Similarly, for the inverse, \( [G^k]^{-1}(w = g_5 \parallel g_6) = g_1 \parallel g_2 \) where

\[
g_{i-2} = F(g_{i-1} \oplus cv_{4k+i-3}) \oplus g_i.
\]

b. schematically:

![G-permutation diagram](image)

Figure 6: “G-permutation diagram”

6. **Crypto-variable schedule**: The crypto-variable is 10 bytes long (labelled 0 through 9) and used in its natural order. So the schedule subscripts given in the definition of the \( G \)-permutation are to be interpreted mod-10.
7. **F Table**: The SKIPJACK F-table is given below in hexadecimal notation. The high order bits of the input index the row and the low order 4 bits index the column. For example, \( F(7a) = d6 \).

<table>
<thead>
<tr>
<th></th>
<th>x0</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
<th>x6</th>
<th>x7</th>
<th>x8</th>
<th>x9</th>
<th>xA</th>
<th>xB</th>
<th>xC</th>
<th>xD</th>
<th>xE</th>
<th>xF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x</td>
<td>a3</td>
<td>d7</td>
<td>09</td>
<td>83</td>
<td>f8</td>
<td>48</td>
<td>f6</td>
<td>f4</td>
<td>b3</td>
<td>21</td>
<td>15</td>
<td>78</td>
<td>99</td>
<td>b1</td>
<td>af</td>
<td>f9</td>
</tr>
<tr>
<td>1x</td>
<td>e7</td>
<td>2d</td>
<td>4d</td>
<td>8a</td>
<td>ce</td>
<td>4c</td>
<td>ca</td>
<td>2e</td>
<td>52</td>
<td>95</td>
<td>d9</td>
<td>1e</td>
<td>4e</td>
<td>38</td>
<td>44</td>
<td>28</td>
</tr>
<tr>
<td>2x</td>
<td>0a</td>
<td>df</td>
<td>02</td>
<td>a0</td>
<td>17</td>
<td>f1</td>
<td>60</td>
<td>68</td>
<td>12</td>
<td>b7</td>
<td>7a</td>
<td>c3</td>
<td>e9</td>
<td>fa</td>
<td>3d</td>
<td>53</td>
</tr>
<tr>
<td>3x</td>
<td>96</td>
<td>84</td>
<td>6b</td>
<td>ba</td>
<td>f2</td>
<td>63</td>
<td>9a</td>
<td>19</td>
<td>7c</td>
<td>ae</td>
<td>e5</td>
<td>f5</td>
<td>f7</td>
<td>16</td>
<td>6a</td>
<td>a2</td>
</tr>
<tr>
<td>4x</td>
<td>39</td>
<td>b6</td>
<td>7b</td>
<td>0f</td>
<td>c1</td>
<td>93</td>
<td>81</td>
<td>1b</td>
<td>ee</td>
<td>b4</td>
<td>1a</td>
<td>ea</td>
<td>d0</td>
<td>91</td>
<td>2f</td>
<td>b8</td>
</tr>
<tr>
<td>5x</td>
<td>55</td>
<td>9a</td>
<td>da</td>
<td>85</td>
<td>3f</td>
<td>41</td>
<td>bf</td>
<td>e0</td>
<td>5a</td>
<td>58</td>
<td>80</td>
<td>5f</td>
<td>66</td>
<td>0b</td>
<td>d8</td>
<td>90</td>
</tr>
<tr>
<td>6x</td>
<td>35</td>
<td>d5</td>
<td>c0</td>
<td>a7</td>
<td>33</td>
<td>06</td>
<td>65</td>
<td>69</td>
<td>45</td>
<td>00</td>
<td>94</td>
<td>56</td>
<td>6d</td>
<td>98</td>
<td>9b</td>
<td>76</td>
</tr>
<tr>
<td>7x</td>
<td>97</td>
<td>fc</td>
<td>b2</td>
<td>c2</td>
<td>b0</td>
<td>fe</td>
<td>db</td>
<td>20</td>
<td>e1</td>
<td>eb</td>
<td>d6</td>
<td>e4</td>
<td>dd</td>
<td>47</td>
<td>4a</td>
<td>1d</td>
</tr>
<tr>
<td>8x</td>
<td>42</td>
<td>ed</td>
<td>9e</td>
<td>6e</td>
<td>49</td>
<td>3c</td>
<td>cd</td>
<td>43</td>
<td>27</td>
<td>d2</td>
<td>07</td>
<td>d4</td>
<td>de</td>
<td>c7</td>
<td>67</td>
<td>18</td>
</tr>
<tr>
<td>9x</td>
<td>89</td>
<td>cb</td>
<td>30</td>
<td>1f</td>
<td>8d</td>
<td>c6</td>
<td>8f</td>
<td>aa</td>
<td>c8</td>
<td>74</td>
<td>dc</td>
<td>c9</td>
<td>5d</td>
<td>5c</td>
<td>31</td>
<td>a4</td>
</tr>
<tr>
<td>Ax</td>
<td>70</td>
<td>88</td>
<td>61</td>
<td>2c</td>
<td>9f</td>
<td>0d</td>
<td>2b</td>
<td>87</td>
<td>50</td>
<td>82</td>
<td>54</td>
<td>64</td>
<td>26</td>
<td>7d</td>
<td>03</td>
<td>40</td>
</tr>
<tr>
<td>Bx</td>
<td>34</td>
<td>4b</td>
<td>1c</td>
<td>73</td>
<td>d1</td>
<td>c4</td>
<td>fd</td>
<td>3b</td>
<td>cc</td>
<td>fb</td>
<td>7f</td>
<td>ab</td>
<td>e6</td>
<td>3e</td>
<td>5b</td>
<td>a5</td>
</tr>
<tr>
<td>Cx</td>
<td>ad</td>
<td>04</td>
<td>23</td>
<td>9c</td>
<td>14</td>
<td>51</td>
<td>22</td>
<td>f0</td>
<td>29</td>
<td>79</td>
<td>71</td>
<td>7e</td>
<td>ff</td>
<td>8c</td>
<td>0e</td>
<td>e2</td>
</tr>
<tr>
<td>Dx</td>
<td>0c</td>
<td>ef</td>
<td>bc</td>
<td>72</td>
<td>75</td>
<td>6f</td>
<td>37</td>
<td>a1</td>
<td>ec</td>
<td>d3</td>
<td>8e</td>
<td>62</td>
<td>8b</td>
<td>86</td>
<td>10</td>
<td>e8</td>
</tr>
<tr>
<td>Ex</td>
<td>08</td>
<td>77</td>
<td>11</td>
<td>be</td>
<td>92</td>
<td>4f</td>
<td>24</td>
<td>c5</td>
<td>32</td>
<td>36</td>
<td>9d</td>
<td>cf</td>
<td>f3</td>
<td>a6</td>
<td>bb</td>
<td>ac</td>
</tr>
<tr>
<td>Fx</td>
<td>5e</td>
<td>6c</td>
<td>a9</td>
<td>13</td>
<td>57</td>
<td>25</td>
<td>b5</td>
<td>e3</td>
<td>bd</td>
<td>a8</td>
<td>3a</td>
<td>01</td>
<td>05</td>
<td>59</td>
<td>2a</td>
<td>46</td>
</tr>
</tbody>
</table>
C. KEA Specification

KEA is a key exchange algorithm. All calculations for KEA require a 1024-bit prime modulus. This modulus and related values are to be generated as per the DSS specification [2]. The KEA is based upon a Diffie-Hellman protocol utilizing SKIPJACK to reduce final values to an 80 bit key.

KEA operations require exponents of length 160 bits. One exponent used in KEA is a user specific secret component.

The KEA provides security commensurate with that provided by SKIPJACK. This is on the order of $2^{80}$ operations.

KEA requires that each user be able to validate the public values received from others, but does not specify how that is to be done.

The devices must be provided the following data in order to implement the Key Exchange Algorithm (KEA).

\[
\begin{align*}
p & \quad \text{1024-bit prime modulus which defines the field where} \\
 & \quad p = p_{1023}p_{1022} \cdots p_0 \\
q & \quad \text{160-bit prime divisor of } p - 1 \text{ for public component checking} \\
 & \quad q = q_{159}q_{158} \cdots q_0 \\
g & \quad \text{1024-bit base for the exponentiation. An element of} \\
 & \quad \text{order } q \text{ in the multiplicative group mod } p. \\
 & \quad g = g_{1023}g_{1022} \cdots g_0 \\
x & \quad \text{160-bit user secret number chosen so that} \quad (0 < x < q) \\
 & \quad x = x_{159}x_{158} \cdots x_0 \\
Y & \quad \text{1024-bit public value corresponding to private value } x \\
 & \quad Y = g^x \mod p = Y_{1023}Y_{1022} \cdots Y_0 \\
pad & \quad \text{80 bit padding value} \\
 & \quad pad = pad_{79}pad_{78} \cdots pad_0 \\
 & \quad = 72f1a87e92824198ab0b \text{ hex.} \\
r & \quad \text{160-bit random number} \\
 & \quad r = r_{159}r_{158} \cdots r_0 \\
\end{align*}
\]

A signaling requirement for the determination of the initiator and the recipient of an exchange is not necessary. A description of the process follows. For two users A and B, the subscripts A and B are used to denote the ‘owner’ of the respective values.

a. A and B exchange or obtain from a directory the certificate(s) of the far terminal. From the certificate(s), the public value $Y$ of the other terminal can be obtained along with associated user identification and other information.
b. Each device validates the public key $Y$ to determine that it is indeed the public key of a valid user on the network. If the validation fails, the process terminates. If the validation checks, go to step c.

c. Each device exchanges the random component. Device A generates a 160-bit private random number $r_A$ and sends the public version of this number

$$R_A = g^{r_A} \mod p$$

Device B generates a 160-bit $r_B$ and sends

$$R_B = g^{r_B} \mod p$$

Each of these public random components is 1024-bits in length.

d. After receiving the public random components and the far end public key, each device will check to verify both the received values are of order $q$. Device A will compute and verify:

$$1 < R_B, Y_B < p$$

$$(R_B)^q \equiv 1 \mod p \text{ and } (Y_B)^q \equiv 1 \mod p$$

Device B will compute and verify

$$1 < R_A, Y_A < p$$

$$(R_A)^q \equiv 1 \mod p \text{ and } (Y_A)^q \equiv 1 \mod p$$

If the verification checks, go to step e. Should the verification fail, stop.

e. Device A will take $Y_B$ and compute the value $t_{AB}$. Device B will compute the equivalent value $t_{BA}$ using the received random component

$$t_{AB} = (Y_B)^{r_A} \mod p = g^{x_B r_A} \mod p$$

$$t_{BA} = (R_A)^{r_B} \mod p = g^{x_A r_B} \mod p = g^{x_B r_A} \mod p$$

f. Each device computes $u$ in a similar manner as they computed $t$

$$u_{BA} = (Y_A)^{r_B} \mod p = g^{x_A r_B} \mod p$$

$$u_{AB} = (R_B)^{r_A} \mod p = g^{x_B r_A} \mod p = g^{x_A r_B} \mod p$$

g. Each device computes $w$ and checks to make sure that

$$w = (t + u) \mod p \neq 0$$

If this check passes, go to step h. Else stop.
h. This result is split into two sections

\[ v_1 = \left( \frac{w}{2^{1023-160}} \right) \mod 2^{80} \quad v_2 = \left( \frac{w}{2^{16}} \mod 2^{80} \right) \]

i.e. if we number the bits in \( w \) as \( w_{1023} \cdots w_0 \) from MSB to LSB, then

\[ v_1 = w_{1023} \cdots w_{944} \text{ and } v_2 = w_{943} \cdots w_{864} \]

i. The Key is

\[
Key = 2^{16} \left[ E_{E_{v_1 \oplus pad}} \left( E_{v_1 \oplus pad} \left[ \frac{v_2}{2^{26}} \mod 2^{64} \right] \right) \right] \\
\oplus \left[ \left( E_{v_1 \oplus pad} \left[ \frac{v_2}{2^{16}} \mod 2^{64} \right] \right) \oplus \left( v_2 \mod 2^{16} \right) \right]
\]

Note that this function represents the encryption of \( v_2 \) with \( v_1 \) XOR pad. Pictorially,

Figure 7: “Key Formation Diagram”
A summary of a full KEA exchange between devices A and B is as follows:

<table>
<thead>
<tr>
<th>Device A</th>
<th>Device B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p, q, g$</td>
<td>$p, q, g$</td>
</tr>
<tr>
<td>$x_A$</td>
<td>$x_B$</td>
</tr>
<tr>
<td>$Y_A = g^{x_A} \mod p$</td>
<td>$Y_B = g^{x_B} \mod p$</td>
</tr>
<tr>
<td>obtain other devices public</td>
<td>obtain other devices public</td>
</tr>
<tr>
<td>via certificate or sent in msg</td>
<td>via certificate or sent in msg</td>
</tr>
<tr>
<td>$r_A$</td>
<td>$r_B$</td>
</tr>
<tr>
<td>$R_A = g^{r_A} \mod p$</td>
<td>$R_B = g^{r_B} \mod p$</td>
</tr>
<tr>
<td>exchange public random numbers</td>
<td></td>
</tr>
</tbody>
</table>

$t_{AB} = (Y_B)^{r_A} \mod p$  
$u_{AB} = (R_B)^{x_A} \mod p$  
$w = (t_{AB} + u_{AB}) \mod p$  
$v_1, v_2$  
$Key$  

check all values received  
compute $t = g^{r_A x_B} \mod p$  
compute $u = g^{x_A r_B} \mod p$  
compute $w$ and check $w \neq 0$  
extract $v_1$ and $v_2$ from $w$  
form $Key$ from $v_1, v_2, pad$  
$Key$
D. E-Mail Applications of KEA

For electronic mail applications where the recipient does not participate in the formation of the key, the recipients contribution to the random exchange is replaced with the public key of the recipient. For the following, let A be the sender and B be the recipient of the E-mail message. We first begin with the formation of the E-mail message.

1. Sending E-Mail

a. Device A obtains from a directory or a local cache the certificate(s) of the far terminal. From the certificate(s), the public value $Y_B$ of terminal B can be obtained along with associated user identification and other information.

b. Device A validates the public key $Y_B$ to determine that it is indeed the public key of a valid user on the network. If the validation fails, the process terminates. If the validation checks, go to step c.

c. Device A will then verify:

$$1 < Y_B < p \text{ and } (Y_B)^a \equiv 1 \mod p$$

If the validation checks, go to step d. Should the verification fail, stop.

d. Device A generates the random number $r_A$ and computes $R_A$ which is placed in the message packet to be sent to the far terminal.

$$R_A = g^{r_A} \mod p$$

This random component is 1024 bits in length.

e. Device A will then take $Y_B$ and compute the value $t_{AB}$.

$$t_{AB} = (Y_B)^{r_A} \mod p = g^{x_B r_A} \mod p$$

f. Device A computes

$$u_{AB} = (Y_B)^{x_A} \mod p = g^{x_B x_A} \mod p = g^{x_A x_B} \mod p$$

g. Device A then computes $w$ and checks to make sure that

$$w = (t_{AB} + u_{AB}) \mod p \neq 0$$

If this check passes, go to step h. Else stop.
h. This result is split into two sections
\[ v_1 \equiv \left( \frac{w}{2^{1024-80}} \right) \mod 2^{80} \quad v_2 = \left( \frac{w}{w(1024-160)} \right) \mod 2^{80} \]
i.e., if we number the bits in \( w \) and \( w_{1023} \ldots w_0 \) from MSB to LSB, then
\[ v_1 = w_{1023} \ldots w_{944} \quad \text{and} \quad v_2 = w_{943} \ldots w_{864} \]
i. The Key is
\[
Key = 2^{16} \left[ E_{v_1 \oplus \text{pad}} \left( E_{v_1 \oplus \text{pad}} \left[ \frac{v_2}{2^{16}} \mod 2^{64} \right] \right) \right] \\
\oplus \left[ \left( \frac{v_2}{2^{48}} \mod 2^{64} \right) \oplus \left( v_2 \mod 2^{16} \right) \right]
\]
Note that function represents the encryption of \( v_2 \) with \( v_1 \) XOR pad. Pictorially,

Figure 8: “Key Formation Diagram”
2. Receiving E-Mail

a. Device B obtains the certificate(s) of the far terminal, A, in the received E-mail message. From the certificate(s), the public value \( Y_A \) of terminal A can be obtained along with associated user identification and other information.

b. Device B validates the public key \( Y_A \) to determine that it is indeed the public key of a valid user on the network. If the validation fails, the process terminates. If the validation checks, go to step c.

c. Device B receives the random component that A generated.

\[
R_A = g^{r_A} \mod p
\]

This random component is 1024-bits in length.

d. Device B will compute and verify:

\[
1 < R_A, Y_A < p
\]

\[
(R_A)^q \equiv 1 \mod p \quad \text{and} \quad (Y_A)^q \equiv 1 \mod p
\]

If the verification checks, go to step e. Should the verification fail, stop.

e. Device B will take \( R_A \) and compute the value \( t_{BA} \).

\[
t_{BA} = (R_A)^{x_B} \mod p = g^{x_A x_B} \mod p
\]

f. Device B computes:

\[
u_{BA} = (Y_A)^{x_B} \mod p = g^{x_A x_B} \mod p
\]

g. Device B computes \( w \) and checks to make sure that

\[
w = (t_{BA} + u_{BA}) \mod p \neq 0
\]

If this check passes, go to step h. Else stop.

h. This result is split into two sections

\[
v_1 = \left( \frac{w}{2^{(1024-80)}} \right) \mod 2^{80} \quad v_2 = \left( \frac{w}{2^{(1024-160)}} \right) \mod 2^{80}
\]

i.e., if we number the bits in \( w \) as \( w_{1023} \ldots w_0 \) from MSB to LSB, then

\[
v_1 = w_{1023} \ldots w_{944} \quad \text{and} \quad v_2 = w_{943} \ldots w_{864}
\]
i. The Key is

\[
Key = 2^{16} \left[ E_{v_1 \oplus \text{pad}} \left( E_{v_1 \oplus \text{pad}} \left[ \frac{v_2}{2^{16}} \mod 2^{64} \right] \right) \right] \\
\oplus \left[ \left( E_{v_1 \oplus \text{pad}} \left[ \frac{v_2}{2^{16}} \mod 2^{64} \right] \right) \oplus \left( v_2 \mod 2^{16} \right) \right]
\]

Note that function represents the encryption of v2 with v1 XOR pad. Pictorally,

Figure 9: “Key Formation Diagram”
A summary of an E-mail KEA exchange between devices A and B is as follows:

### Device A

- $p, q, g$: common to both devices
- $x_A$: private key of each device
- $Y_A = g^{x_A} \mod p$
- $Y_B$: obtain other devices public via certificate or sent in msg
- $r_A$: A generates a random number
- $R_A = g^r \mod p$

### Device B

- $x_B$: private key of each device
- $Y_B$
- $Y_A$

### Key Equations

- $t_{AB} = (Y_B)^{r_A} \mod p$
- $u_{AB} = (Y_B)^{x_A} \mod p$
- $w = (t_{AB} + u_{AB}) \mod p$
- $v_1, v_2$: extract $v_1$ and $v_2$ from $w$
- $Key$: form $Key$ from $v_1, v_2, pad$

- $t_{BA} = (R_A)^{x_B} \mod p$
- $u_{BA} = (Y_A)^{x_B} \mod p$
- $w = (t_{BA} + u_{BA}) \mod p$
- $v_1, v_2$: extract $v_1$ and $v_2$ from $w$
- $Key$: form $Key$ from $v_1, v_2, pad$
### III. ANNEX - Test Vectors

All values are hexadecimal. This data does not imply or specify any interface convention. All information is presented with the Most Significant Bit/Byte/Word to the left. X represents “don’t-care”.

#### A. SKIPJACK - CODEBOOK MODE

**Plaintext input:** 33221100ddccbbaa  
**Cryptovariable:** 0098877665544332211  
**Intermediate steps:**

<table>
<thead>
<tr>
<th></th>
<th>w1</th>
<th>w2</th>
<th>w3</th>
<th>w4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>33221100</td>
<td>ddcbbbbaa</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>b0040baf</td>
<td>1100ddcc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>e6883b46</td>
<td>0baf1100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3c762d75</td>
<td>3b460baf</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4c4547ee</td>
<td>2d753b46</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>b949820a</td>
<td>47ee2d75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>f0e3dd90</td>
<td>820a47ee</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>f9b9be50</td>
<td>dd90820a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>d79b5599</td>
<td>be50dd90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>dd901e0b</td>
<td>820bee50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>be504c52</td>
<td>c91820b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>820b7f51</td>
<td>f209c391</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>c391f9c2</td>
<td>fd56f209</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>f20925ff</td>
<td>3a5efdd56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>fd5665da</td>
<td>d7f83a5e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>3a5e69d9</td>
<td>9883d7f8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>d7f88990</td>
<td>53979883</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>9c000492</td>
<td>89905397</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>9fdcccc59</td>
<td>04928990</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>3731beb2</td>
<td>cc590492</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>7af7e7d</td>
<td>beb2cc59</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>7759b15</td>
<td>7e7dbbe2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>fb6445c0</td>
<td>bb157e7d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>6f7f1115</td>
<td>45c0bb15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>65a7deaa</td>
<td>111545c0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>45c0e0f9</td>
<td>bb141115</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>11153913</td>
<td>a523bb14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>bb148ee6</td>
<td>281da523</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>a523bfe2</td>
<td>35ee281d</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>281d0d84</td>
<td>1adc35ee</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>35ee6f1</td>
<td>25871adc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>1adc60ee</td>
<td>d3002587</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>2587caee2</td>
<td>7a12d300</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Ciphertext output:** 2587caee27a12d300
B. Key Exchange Algorithm (KEA)

p=9d4c66d 42ea91c8 28d67d49 94a9f01b 8e5b5b73 0d0faae7 bd569dd1 914e3ad4 759c8053 31eda145 9fb56be8 a8de4736 652a82b2 76e82acd 63f5b78d 0b75a03e b34d397d be7b3740 8f72136a cb0879fe 61c718a3 7f5f154b5 078a7649 fb3d4f4b c481e010 62c5241f 229fa580 423368dd 51090dfb 25351f0c 5800de05 b92ba6a9
q=97ad85fd 2b371ed0 69818ab3 c6ee8773 d9db029d
g=59534434 e897c82 51e5fa9d 02ab8b75 c0fc57b0 969f880d a366a100 01912a01 96bcb81c 41ac8485 031ac598 b5481eae 2726719 d8dg915a 6105973g 72386c0a 6a2c732c d6700d34 1f54bf28 d12d692d e2fa05f5 5e898c2e 20bb8a26 02db1ba0 7de672e3 b96d9ac2 ga188450 63d918c3 2ed71266 b783311a 0a8d08ac 487bea44
ra=6201dd56 237c222a 3f54bc7e 794bdf32 41c67ea6
xa=62319ac4 7de14518 0abd322c 59e2b600 2781e4g4
Ya=2d29ecd0 2e3497a6 72228d8e bc286313 d149f458 1b3e586d 0151024c 02e8b23d a09a430e 2ca5ed1a 4b2d7725 62316e4d 2804d226 788284ed
655cf546 10d36f66 fab1a0a2 e2d3c661 4401901d 9758d566 722aff1f 734b2adb d2b67f13 00ce455f 00968ca7 91a87678 67363d7d 49ee74a2 8dc349d9 fdff96b0 01f0fc1f 0690ec96
xb=63decda0 4487eb71 31dff4fs 1cfbae39 446b9b3d
rb=52bfa1d7 2f1cf0fb 0ff6d5df 15fb7483 167eb0e7
Yb=7730d4bb f3a2efdb 218e7041 3e861020 14ec06c6 205f5419 293b65c6 9a971e5e 55eb79a0 bbd90a2b 14c5240e de6cfdd5 8c7c19c5 269d57df
f60b61c1 db2ff648 64bee519 87f27003 4bc390ad 73168209 5e42608c 3d7987f9 649fbb71 6887633e b574b39c c73df899 51fc1bd6 d389d48 fe2244b8 29afdd05 06ab9221 ba562c07

Computed by A:
Ra=97c1fd8a 69fc8f34 a74c7ec3 c1ab176a b91fa0ea d0e6b097 06ae07a1 fbf8d0a6 67032ea4 798082b8 caea827b 4f604b71 e6c24469 211363ea
4bd2122f 4aa6ab9 4857ff06 9db03701 2b289057 b4855e70 f8f7ac4f
92fa1fe7 6c2a5c82 781ee611 1cfbdf7 a6eb9dc3 59af8c0a b632ef3a
2af82e52 c0a7f6a6 a2c961ea fc67f418

Computed by B:
Rb=91f61808 38f03d5b 6be538ff 6e0bf3cb 9d8afbbe ef199334 b389708b
b0c848da 860f0f27 62cc9a48 e496f8fc 94945538 c6f1719 57c864f1
e2eca2ba ddb340da f406e636 bbc6368e 4658fb01 1a41cbef 5adb4086
42d03cec 4e85920c 8e7530bd e2b78cb8 7cbeae364 31de373c d2ebaf29
d8412g32 8550dd8c f33e03c2 1a5056a0
Results for user A:

uab=1585dbba c06b963d 6ef5a30e 5c40220b 76fe0528 660be31a c496d1cb 0883baae 5a0331e9 ce3fe382 f47a353c edc6896d fdb4c0b5 67aaf72 4ba0ff6f 2c0fa428 fcb07a32 bf6fb88e 22c5ca47 7c9bb9cd 882da4f5 4cc57980 c174352f 13434623 ce3df2d4 14a9e0fb 7a905fe8 4ab282d5 e76e703a 55dbb38 27c2979f 08ea28c8
tab=8032eb2c b6753a9 c5faf66e a1eb6ef1 de0d3f48 c86be240 8f807e65 8622b9f3 87e0f50f a5868bf5 29ff008d 3ad55e9c 4366bad4 ae4190ce bc3ae56f 34bf70b6 3ca021dd 563005db bc7e62bb ccc9127a 3603bf00 be8fde9b f46bf538 86c4a761 4b43adfe 7282efe4 f9c146b7 1e9f89d6 2bd3c7ed 7d127719 ebf0e0f8 79e0d0d9
w=95b86ce7 76e0ca77 34f099cc fe2b90fd 550b4471 2e77c55b 54175031 8e67481 e1e426f9 73c6af78 1e7g35ca 289be80a 411b7b8a 15ec8e41 07dbe4de 60cf14df 3g509c10 159fbe69 df442d03 4964cc47 be3163f6 0b55481c b5e02a67 9a07ed85 1981a0d2 872cd0e0 7451a69f 69520cac 13432827 d2ed3252 13b37897 82caf9a1
v1=95b8c6e7 76e0ca77 34f00000
v2=99ccfe2b 90fd550b 44710000
v1 XOR pad = e7496e99 e4628b7f 9ffbb3xx

Key for user A = 740839de e833add4 6b41xxxx

Results for user B:
tab=8032eb2c b6753a9 c5faf66e a1eb6ef1 de0d3f48 c86be240 8f807e66 8622b9f3 87e0f50f a5868bf5 29ff008d 3ad55e9c 4366bad4 ae4190ce bc3ae56f 34bf70b6 3ca021dd 563005db bc7e62bb ccc9127a 3603bf00 be8fde9b f46bf538 86c4a761 4b43adfe 7282efe4 f9c146b7 1e9f89d6 2bd3c7ed 7d127719 ebf0e0f8 79e0d0d9
uab=1585dbba c06b963d 6ef5a30e 5c40220b 76fe0528 660be31a c496d1cb 0883baae 5a0331e9 ce3fe382 f47a353c edc6896d fdb4c0b5 67aaf72 4ba0ff6f 2c0fa428 fcb07a32 bf6fb88e 22c5ca47 7c9bb9cd 882da4f5 4cc57980 c174352f 13434623 ce3df2d4 14a9e0fb 7a905fe8 4ab282d5 e76e703a 55dbb38 27c2979f 08ea28c8
w=95b8c6e7 76e0ca77 34f099cc fe2b90fd 550b4471 2e77c55b 54175031 8e67481 e1e426f9 73c6af78 1e7935ca 289be80a 411b7b8a 15ec8e41 07dbe4de 60cf14df 3g509c10 159fbe69 df442d03 4964cc47 be3163f6 0b55481c b5e02a67 9a07ed85 1981a0d2 872cd0e0 7451a69f 69520cac 13432827 d2ed3252 13b37897 82caf9a1
v1=95b8c6e7 76e0ca77 34f00000
v2=99ccfe2b 90fd550b 44710000
v1 XOR pad = e7496e99 e4628b7f 9ffbb3xx

Key for user B = 740839de e833add4 6b41xxxx
C. KEA Exchange for E-Mail

p=9d4c6e6d 42e91c8 28d67d49 94a9f01b 8e5b5b73 0d0faae7 bd569dd1
914e3ad4 759c053 31eda145 9fb56be8 a8de4736 652a82b2 76e82acc1
63f5b78d 0b75a03e b34d397d be7b3740 8f72136a cb0879fe 61c718a3
7f5f15b5b 078a7649 fb3d4fb4 c458e101 62c5241f 229fa580 423368dd
5109dfb 25351f0c 5800de05 b92ba6a9
q=97ad85fd 2b371ed0 69818ab3 c6ee8773 d9db029d
g=595d3443 ec897c82 51e5fa9d 02ab8b75 c0cf57b0 969f880d a366a100
01912a01 96bcb81c 41ac8485 031ac598 b5481eae 2726b719 d8d9915a
61059734 72386c0a 6a2c732c d6700d34 1f54bf28 d12d692d e2f0af5f
5e898c2e 20bb8a26 02db1a01 7de672e3 b96d9ac2 9a188450 63d918c3
2ed71266 b783311a 08d08ac 487bea44
ra=6201dd56 237c228a 3f54bc7e 794bdf32 41c67ea6
xa=62319ac4 7de14518 0abd32c2 59e2b600 2781e494
Ya=2d29ec0 2e3497a6 7222d8de bc286131 d149f458 1b3e586d 0151024c
02e8b23d a09a430e 2ca5ed1a 4b2d7725 62316e4d 2804d226 788284ec1
655cf546 10d3f66 5fab1a02 e2d3c661 44d1909d 9758d566 724aff1f
734b2adb d2b67f13 00ce455f 00968ca7 91a87678 673637d7 49ee74a2
8dc349d9 fdfdb96b 01f0fc1f 0690ec96
xb=63decda7 4487eb71 31df4f5 1cfbae39 446b93d
Yb=7730d4bb f3a2efdb 218e7041 3e861020 14ec06c2 205f5419 293h65c6
9a971e54 55eb79a0 b9d9ab2 14c5240e de6cfd5 8c7c19c5 269d57df
f60b61c1 db2ff648 64bee519 87f27003 4bc390ad 73168209 5e42608c
3d7987f9 649bff71 6887633e b574b39c c73df899 51fc1bd6 d3899d4
fe2244b8 29afd405 06a92221 ba562c07

Computed by A:
Ra=97c1fd8a 69fc8f34 a74c7ec3 clab176a b91fa0ea d0e6b097 06ae07a1.
fbf8d0a6 67032eea4 7980b2b8 cacea27b 4f604b71 e6c24469 211363ea
4bd2122f 4a6a6f9b 4857ff06 9db03701 2b289057 b4855e70 f8f7ac4f
92fafa77 6c2a5c82 781ee611 1cfbdff7 a6eb9dc3 59a8fca0 632e5f3a
2af82e52 c0a7f6a6 a2c961ea fc67f418

Results for user A:
tab=8032eb2c b67534a9 c5fafa6be a1eb6ef1 de0d3f48 c86be240 8f807e65
8622bdf3 87e0f50f a5868bf5 29ff008d 3ad55e9c 4366bad4 ae4190ce
bc3ae56f 34bf70b6 3ca021dd 563005db bc7e62bb ccc9127a 3603bf00
be8fcfe9b f46bf538 68c4a761 4b43ade7 7282efe4 f9c146b7 1ef989d6
2bd3c7ed 7d1277f9 ebf0ef0f 79e0d0d9
uab=17087175 91f6dbf6 b0a0c05e 0ee49abd 49586033 93a7fd3 3d99bc61
68ad318a 7cf81fa8 74f4eb04 4333abe0 6423eb2f 1eb3cbdb 33067152
242d7cf8 987f208d cdf3797 639c8cd5 6a0bdc1b 2bdf6734 35e6dca9
06bd6d71 a4516738 b91f2a52 689a2d60 802de96d 150fe661 469a2643
Key for user A = 97fd1c6b d86bc439 115bxxxx

Results for user B:
tab=8032eb2c b67534a9 c5faf6be a1eb6ef1 de0d3f48 c86be240 8f8c7e66 8622b9f3 87e0f50f a5868bf5 29ff008d 3ad55e9c 4366bad4 ae4190ce bc3ae56f 34bf70b6 3ca021dd 563005db bc7e62bb ccc9127a 3603bf00 be8fcede9b f46bf538 86c4a761 4b43adfe 7282efe4 f9c146b7 1e9f89d6 2bd3c7ed 7d127719 ebf0e0f8 79e0dd9
uab=17087175 9f16dfbf b0a0c05e 0ee49abd 49586033 93aa7df3 3d99bc61 68ad318a 7cf81fa8 74f4eb04 4433abe0 6423eb2f 1ebb3c3b 33067152 24d7cfe8 987f208d cdf3797 6398ccd5 6a0bdc1b 2bfd6734 35edcc9 06bd6d71 a4516738 b91f2a52 689a2d60 802e96d1 150fe661 469a2643 18c8d8f5 9ec040ea c623c51a 91d861d1
w=973b5ca2 558c1467 769bb71c b0d09af 27659f7c 5c166033 cd1a3ac7 eecfeb7e 04d914b8 1a7b76f9 6e32ac6d 9ef949cb 6221f7af e1480220 e0686267 cd3e9144 0c7f5974 b9c8d2b1 268a3ed6 f8c679ae 6be29bc9 c54d3c0d 98bd5c71 3fe3d1b3 b3dabb5e f2b0d952 0ed12d18 6539b019 449ca0e3 1bd2804 b214a613 0bb932aa
v1=973b5ca2 558c1467 769bxxxx
v2=b71cb0d0 09af2765 9f7cxxxx
v1 XOR pad = e5caf4dc c70e55f1 dd90xxxx

Key for user B = 97fd1c6b d86bc439 115bxxxx
IV. References:

1. US DEPARTMENT OF COMMERCE Technology Administration/National Institute of Standards and Technology, DES MODES OF OPERATION, FIPS PUB 81, 2 December 1980.

About this document

This document is a reproduction of the SKIPJACK and KEA Algorithm Specifications declassified by the NSA, announced June 23, 1998.\footnote{Details available from NIST, at <http://csrc.nist.gov/encryption/skipjack-kea.htm>.
}

NSA released the document as two PDF files, “SKIPJACK and KEA specifications” (Pages 1–17) and “Annex containing test vectors” (Pages 18–23). These were unfortunately difficult to read, so a number of interested parties contributed to its reproduction in a clear and more easily reproducible format.

Section II.B was typeset in TeX by Whitfield Diffie on July 5, 1998, using the HTML source from John Young. The F Table source came from Perry Metzger.

Section II.C and figures 7–9 were regenerated by Lewis McCarthy, June 24, 1998.

Section II.D and figures 1–6 were regenerated by Matt Curtin, June 26, 1998. All remaining sections were converted to \LaTeX\ source from John Young’s HTML on July 3, 1998. All of these pieces were assembled and first typeset on July 3, 1998.

This is version:

$\textit{Id: skipjack-kea.tex,v 1.8 1998/07/09 11:36:43 cmcurtin Exp cmcurtin}$

Please send corrections to Matt Curtin at <cmcurtin@interhack.net>.
