SKIPJACK and KEA Algorithm Specifications

Version 2.0

29 May 1998

Table of Contents

I. Introduction

This document provides details of the SKIPJACK and KEA algorithms. The algorithms are supported in single chip cryptoprocessors 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

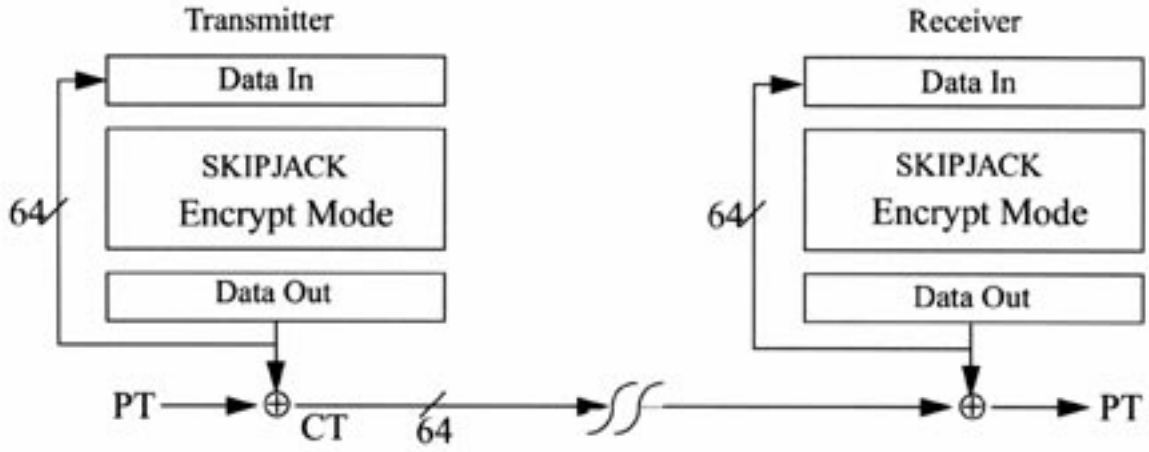


Figure 1. "Output Feed-Back Modes Diagram"

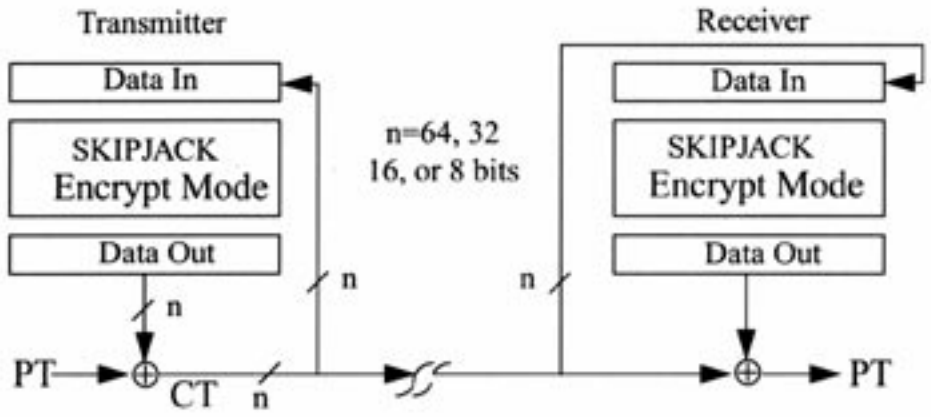


Figure 2. "Cipher Feed-Back Mode Diagram"

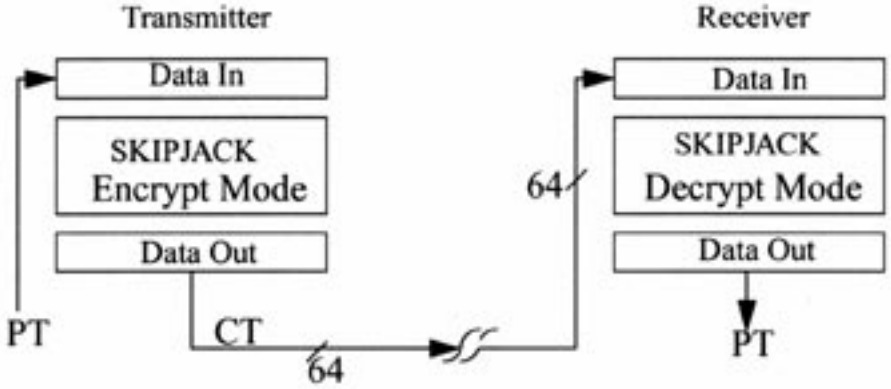


Figure 3. "Codebook Mode Diagram"

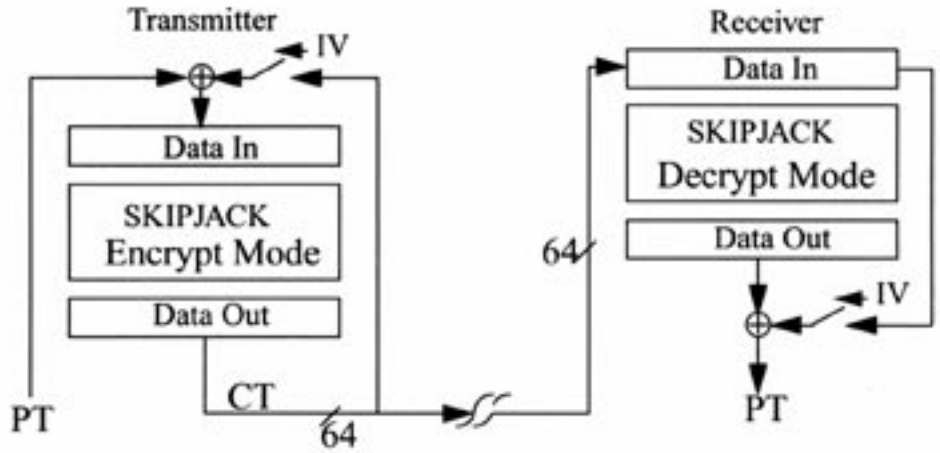


Figure 4. "Cipher-Block Chaining Mode Diagram"

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 on 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.
$X \| Y$	X concatenated with Y. Let X, Y be bytes, then $X \| Y = X \times 2^8 + Y$ 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.

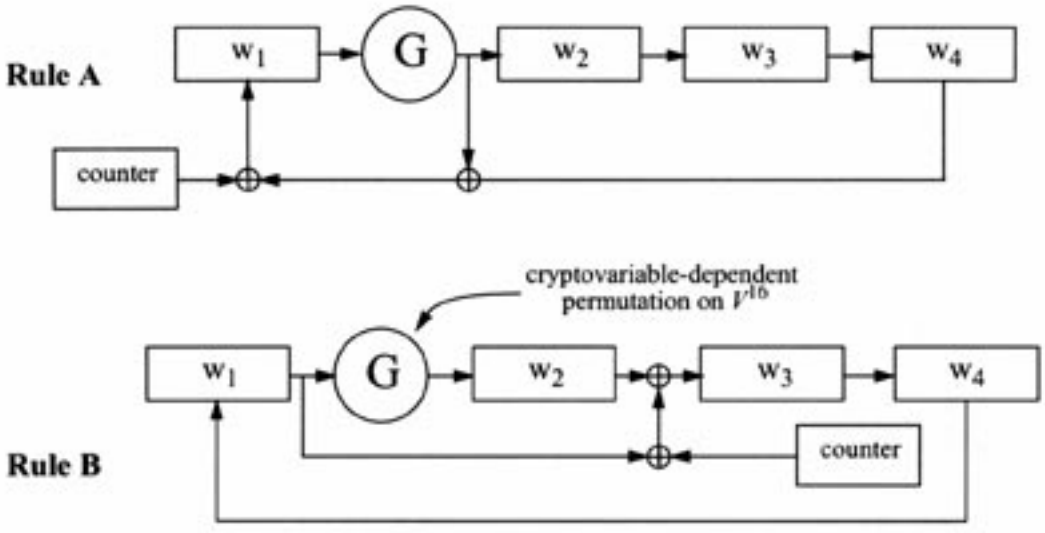


Figure 5. "SKIPJACK Stepping Rules"

3. Stepping rule equations. In the equations below, the superscript is the step number.

ENCRYPT

Rule A

$$w_1^{k+1} = G^k(w_1^k) \oplus w_4^k \oplus counter^k$$
$$w_2^{k+1} = G^k(w_1^k)$$
$$w_3^{k+1} = w_2^k$$
$$w_4^{k+1} = w_3^k$$

Rule B

$$w_1^{k+1} = w_4^k$$
$$w_2^{k+1} = G^k(w_1^k)$$
$$w_3^{k+1} = w_1^k \oplus w_2^k \oplus counter^k$$
$$w_4^{k+1} = w_3^k$$

DECRYPT

Rule A^{-1}

$$w_1^{k-1} = [G^{k-1}]^{-1}(w_2^k)$$
$$w_2^{k-1} = w_3^k$$
$$w_3^{k-1} = w_4^k$$
$$w_4^{k-1} = w_1^k \oplus w_2^k \oplus counter^{k-1}$$

Rule B^{-1}

$$w_1^{k-1} = [G^{k-1}]^{-1}(w_2^k)$$
$$w_2^{k-1} = [G^{k-1}]^{-1}(w_2^k) \oplus w_3^k \oplus counter^{k-1}$$
$$w_3^{k-1} = w_4^k$$
$$w_4^{k-1} = w_1^k$$

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

   a. To encrypt: The input is w_i^0, $1 \le i \le 4$, (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}, $1 \le i \le 4$.

   b. To decrypt: The input is w_i^{32}, $1 \le i \le 4$, (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, $1 \le i \le 4$.

5. G-permutation: The cryptovariable-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 cryptovariable. We give two characterizations of the function below:

a. recursively (mathematically): $G^k(w = g_1 \| g_2) = g_5 \| g_6$ 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 cryptovariable schedule. Thus,

$$g_3 = F(g_2 \oplus cv_{4k}) \oplus g_1$$
$$g_4 = F(g_3 \oplus cv_{4k+1}) \oplus g_2$$
$$g_5 = F(g_4 \oplus cv_{4k+2}) \oplus g_3$$
$$g_6 = F(g_5 \oplus cv_{4k+3}) \oplus g_4$$

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

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

b. schematically:

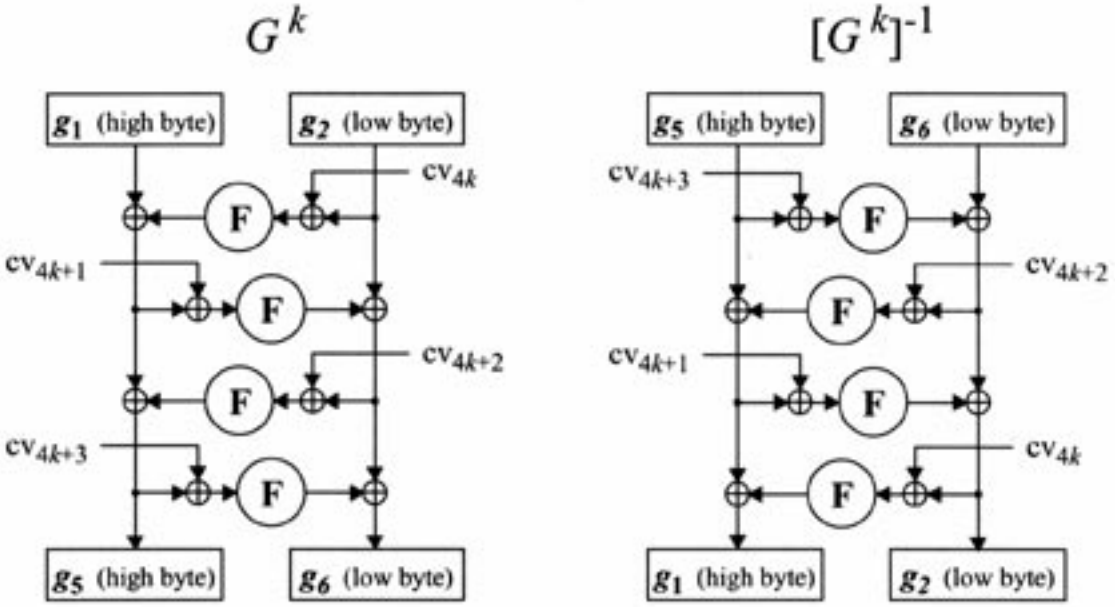


Figure 6. " G-permutation diagram"

6. Cryptovariable schedule: The cryptovariable 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 4 bits of the input index the row and the low order 4 bits index the column. For example, F(7a) = d6.

	x0	**x1**	**x2**	**x3**	**x4**	**x5**	**x6**	**x7**	**x8**	**x9**	**xA**	**xB**	**xC**	**xD**	**xE**	**xF**
0x	a3	d7	09	83	f8	48	f6	f4	b3	21	15	78	99	b1	af	f9
1x	e7	2d	4d	8a	ce	4c	ca	2e	52	95	d9	1e	4e	38	44	28
2x	0a	df	02	a0	17	f1	60	68	12	b7	7a	c3	e9	fa	3d	53
3x	96	84	6b	ba	f2	63	9a	19	7c	ae	e5	f5	f7	16	6a	a2
4x	39	b6	7b	0f	c1	93	81	1b	ee	b4	1a	ea	d0	91	2f	b8
5x	55	b9	da	85	3f	41	bf	e0	5a	58	80	5f	66	0b	d8	90
6x	35	d5	c0	a7	33	06	65	69	45	00	94	56	6d	98	9b	76
7x	97	fc	b2	c2	b0	fe	db	20	e1	eb	d6	e4	dd	47	4a	1d
8x	42	ed	9e	6e	49	3c	cd	43	27	d2	07	d4	de	c7	67	18
9x	89	cb	30	1f	8d	c6	8f	aa	c8	74	dc	c9	5d	5c	31	a4
Ax	70	88	61	2c	9f	0d	2b	87	50	82	54	64	26	7d	03	40
Bx	34	4b	1c	73	d1	c4	fd	3b	cc	fb	7f	ab	e6	3e	5b	a5
Cx	ad	04	23	9c	14	51	22	f0	29	79	71	7e	ff	8c	0e	e2
Dx	0c	ef	bc	72	75	6f	37	a1	ec	d3	8e	62	8b	86	10	e8
Ex	08	77	11	be	92	4f	24	c5	32	36	9d	cf	f3	a6	bb	ac
Fx	5e	6c	a9	13	57	25	b5	e3	bd	a8	3a	01	05	59	2a	46

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).

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

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

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

Device B generates a 160-bit r_B and sends

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

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

d. After receiving the public random component 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:

$$1 < R_B, Y_B < p$$

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

Device B will compute and verify

$$1 < R_A, Y_A < p$$

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

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

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

$$t_{BA} = (R_A)^{x_B} \bmod p = g^{r_A x_B} \bmod p = g^{x_B r_A} \bmod p$$

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

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

$$u_{AB} = (R_B)^{x_A} \bmod p = g^{r_B x_A} \bmod p = g^{x_A r_B} \bmod p$$

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

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

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

h. This result is split into two sections

$$v_1 = \left(\frac{w}{2^{(1024-80)}}\right) \bmod 2^{80} \qquad v_2 = \left(\frac{w}{2^{(1024-160)}}\right) \bmod 2^{80}$$

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

$v_1 = w_{1023} \ldots w_{944}$ and $v_2 = w_{943} \ldots w_{864}$

i. The Key is

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

$$\oplus\left[\left(\frac{E_{v_1 \oplus pad}\left[\frac{v_2}{2^{16}} \bmod 2^{64}\right]}{2^{48}}\right) \oplus (v_2 \bmod 2^{16})\right]$$

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

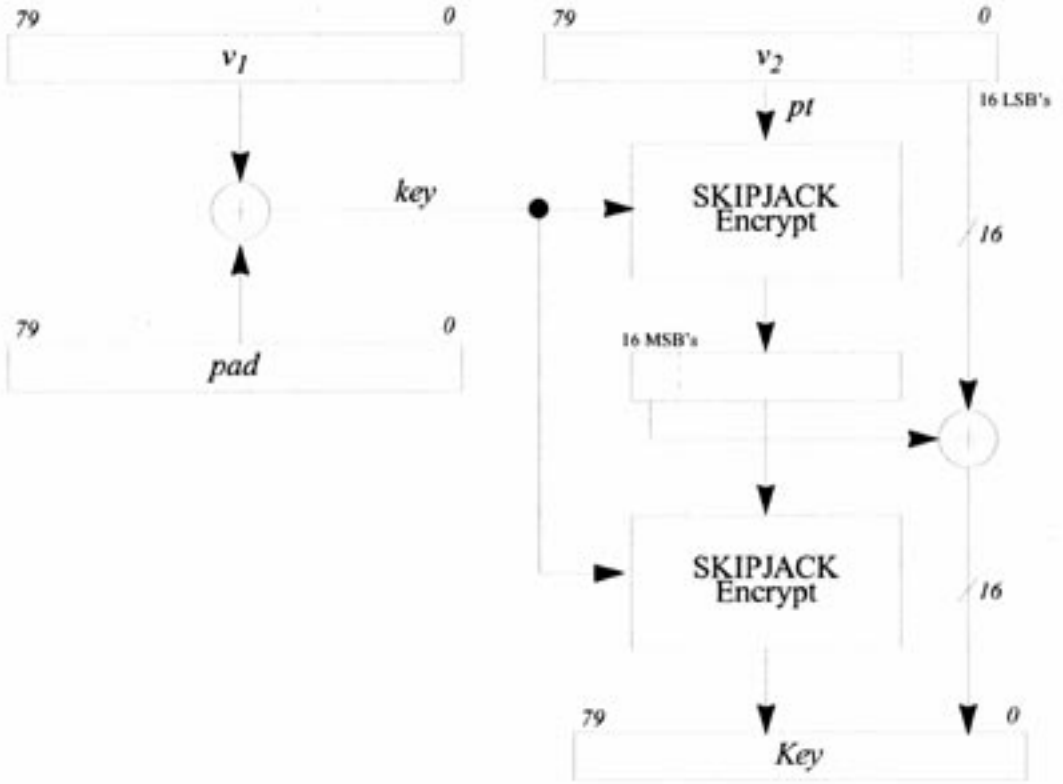
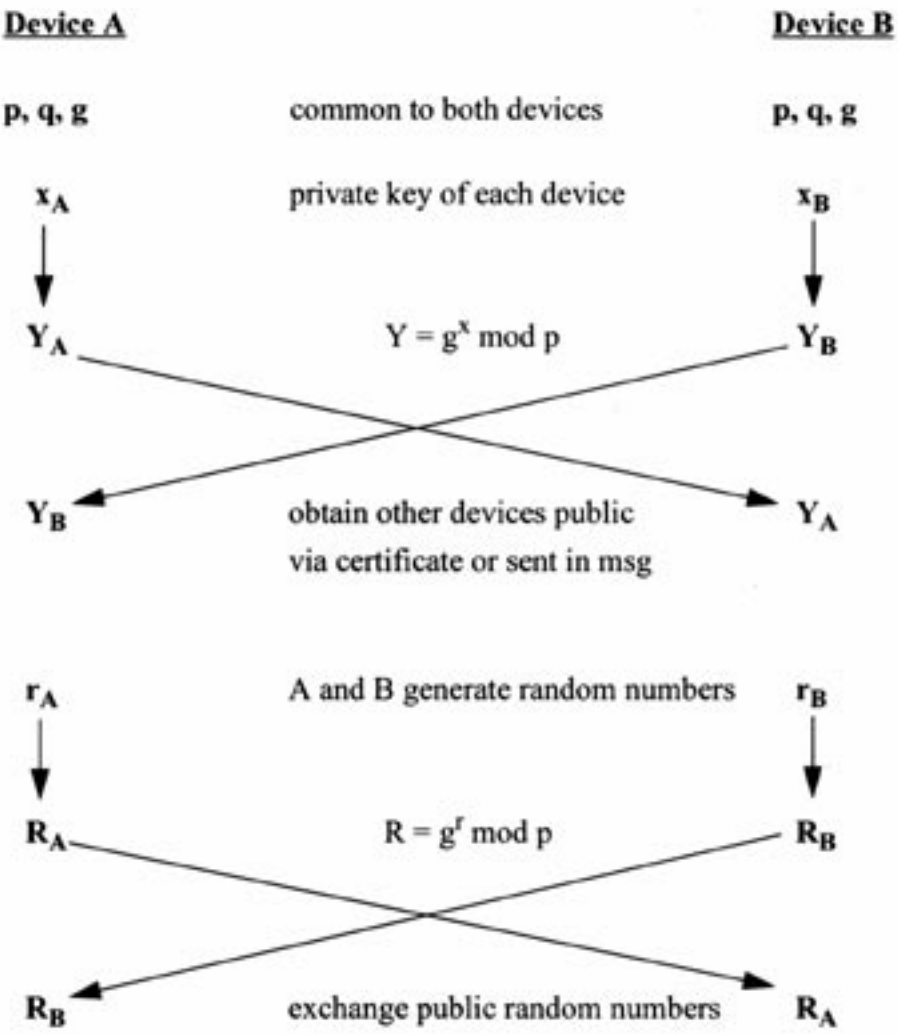


Figure 7. "Key Formation Diagram"

A summary of a full KEA exchange between devices A and B is as follows:



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:

$$1 < Y_B < p \quad \text{and} \quad (Y_B)^q \equiv 1 \bmod p$$

If the verification 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.

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

This random component is 1024 bits in length.

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

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

f. Device A computes

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

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

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

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

h. This result is split into two sections

$$v_1 = \left(\frac{w}{2^{(1024-80)}}\right) \bmod 2^{80} \qquad v_2 = \left(\frac{w}{2^{(1024-160)}}\right) \bmod 2^{80}$$

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

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

$$\oplus\left[\left(\frac{E_{v_1 \oplus pad}\left[\frac{v_2}{2^{16}} \bmod 2^{64}\right]}{2^{48}}\right) \oplus (v_2 \bmod 2^{16})\right]$$

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

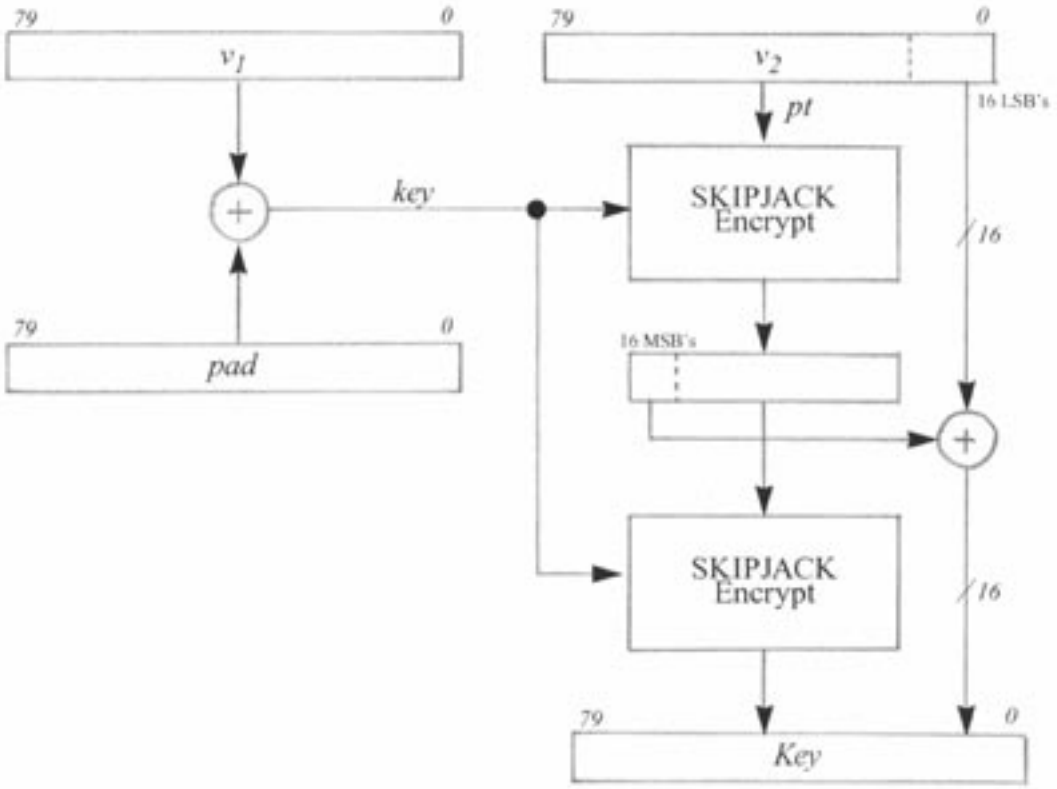


Figure 8. " Key Formation Diagram"

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.

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

This random component is 1024-bits in length.

d. Device B will compute and verify:

$$1 < R_A, Y_A < p$$

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

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}.

$$t_{BA} = (R_A)^{x_B} \bmod p = g^{r_A x_B} \bmod p$$

f. Device B computes:

$$u_{BA} = (Y_A)^{x_B} \bmod p = g^{x_A x_B} \bmod p$$

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

$$w = (t_{BA} + u_{BA}) \bmod p \neq 0$$

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

h. This result is split into two sections

$$v_1 = \left(\frac{w}{2^{(1024-80)}}\right) \bmod 2^{80} \qquad v_2 = \left(\frac{w}{2^{(1024-160)}}\right) \bmod 2^{80}$$

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

$v_1 = w_{1023} \ldots w_{944}$ and $v_2 = w_{943} \ldots w_{864}$

i. The Key is

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

$$\oplus\left[\left(\frac{E_{v_1 \oplus pad}\left[\frac{v_2}{2^{16}} \bmod 2^{64}\right]}{2^{48}}\right) \oplus (v_2 \bmod 2^{16})\right]$$

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

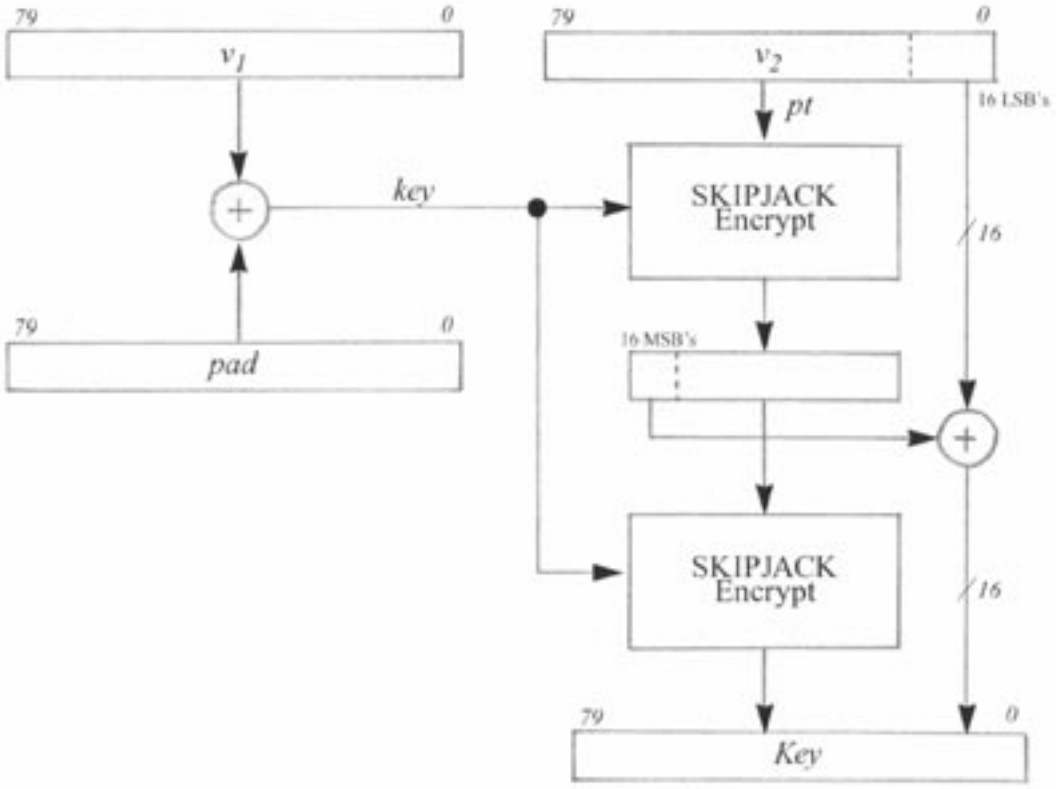
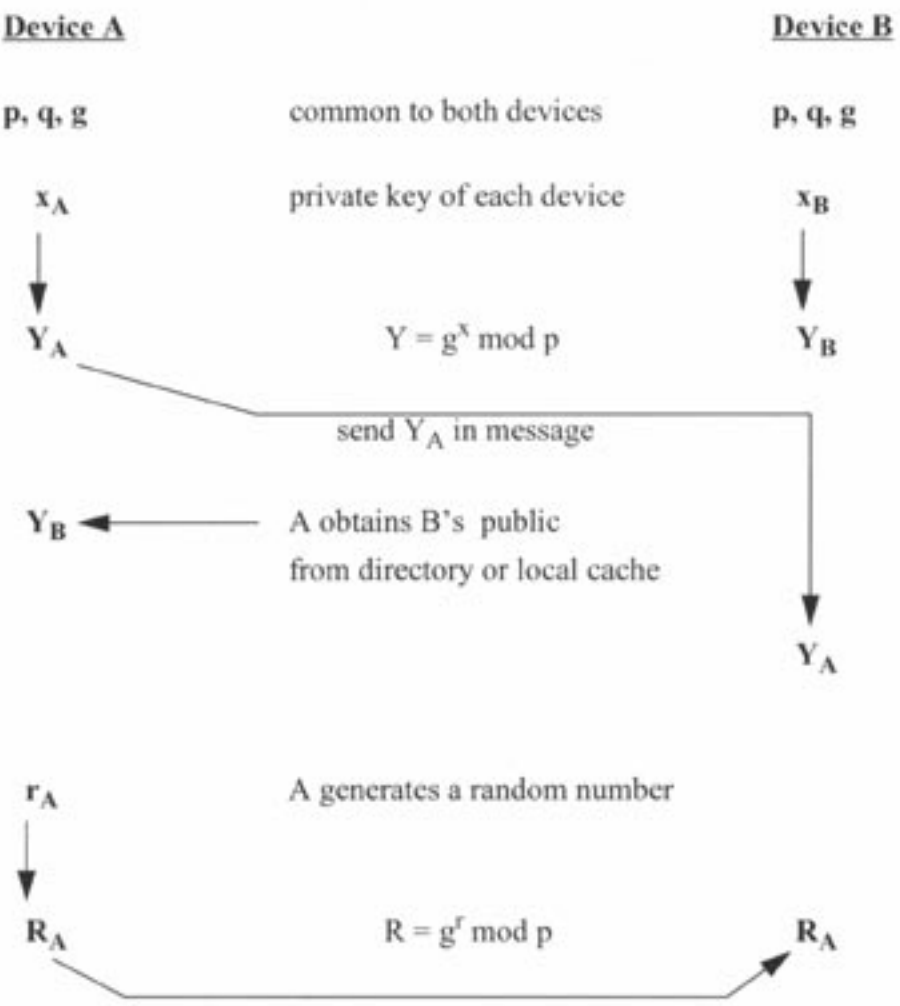


Figure 9. " Key Formation Diagram"

A summary of an E-mail KEA exchange between devices A and B is as follows:



	check all values received	
$t_{AB} = (Y_B)^{rA} \bmod p$	compute $t = g^{rAxB} \bmod p$	$t_{BA} = (R_A)^{xB} \bmod p$
$u_{AB} = (Y_B)^{xA} \bmod p$	compute $u = g^{xAxB} \bmod p$	$u_{BA} = (Y_A)^{xB} \bmod p$
$w = (t_{AB} + u_{AB}) \bmod p$	compute w and check $w \neq 0$	$w = (t_{BA}+u_{BA}) \bmod p$
v1, v2	extract v1 and v2 from w	v1, v2
Key	form Key from v1, v2, pad	Key