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

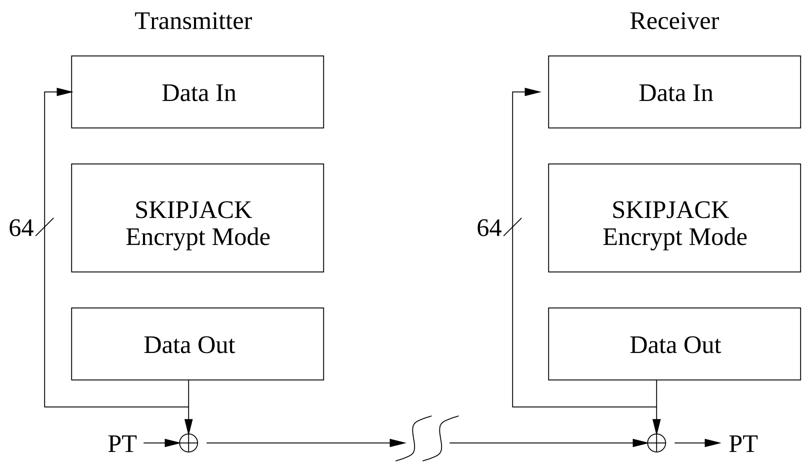


Figure 1: "Output Feed-Back Modes Diagram

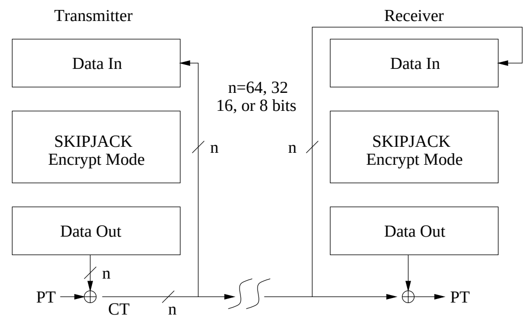


Figure 2: “Cipher Feed-Back Mode Diagram”

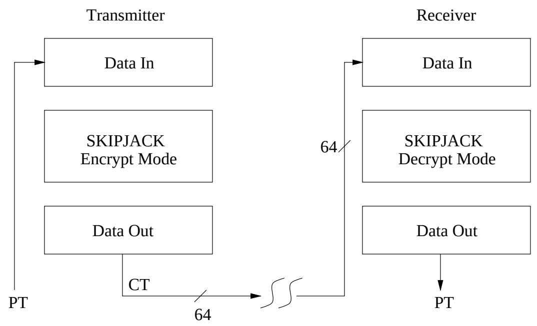


Figure 3: “Codebook Mode Diagram”

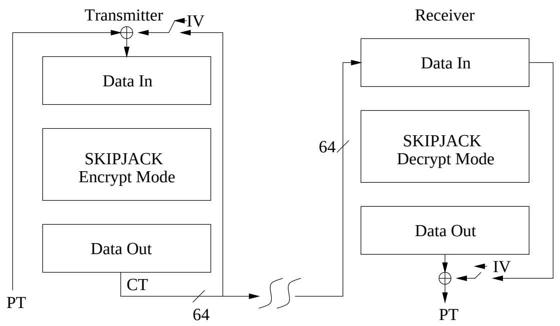


Figure 4: “Cipher-Block Chaining Mode Diagram”

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 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.

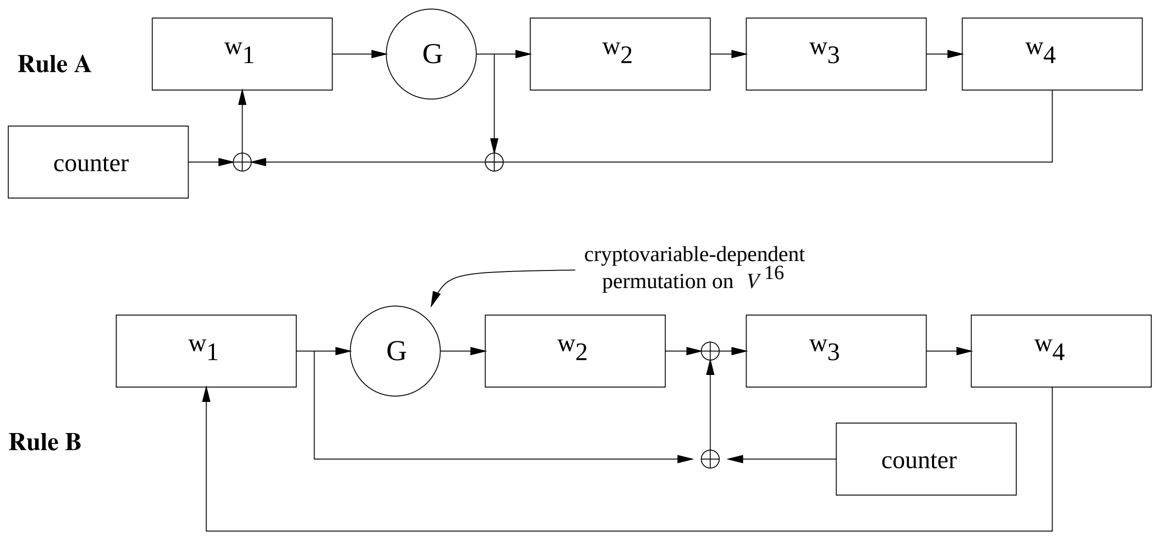


Figure 5: “SKIPJACK Stepping Rules”

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

ENCRYPT

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

DECRYPT

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

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

a. To encrypt: The input is $w_i^0, 1 \leq i \leq 4$, (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 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): $G^k(w = g_1 \parallel g_2) = g_5 \parallel g_6$ 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 $(4k+i-3)^{\text{th}}$ byte in the cryptovariable schedule. Thus,

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

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:

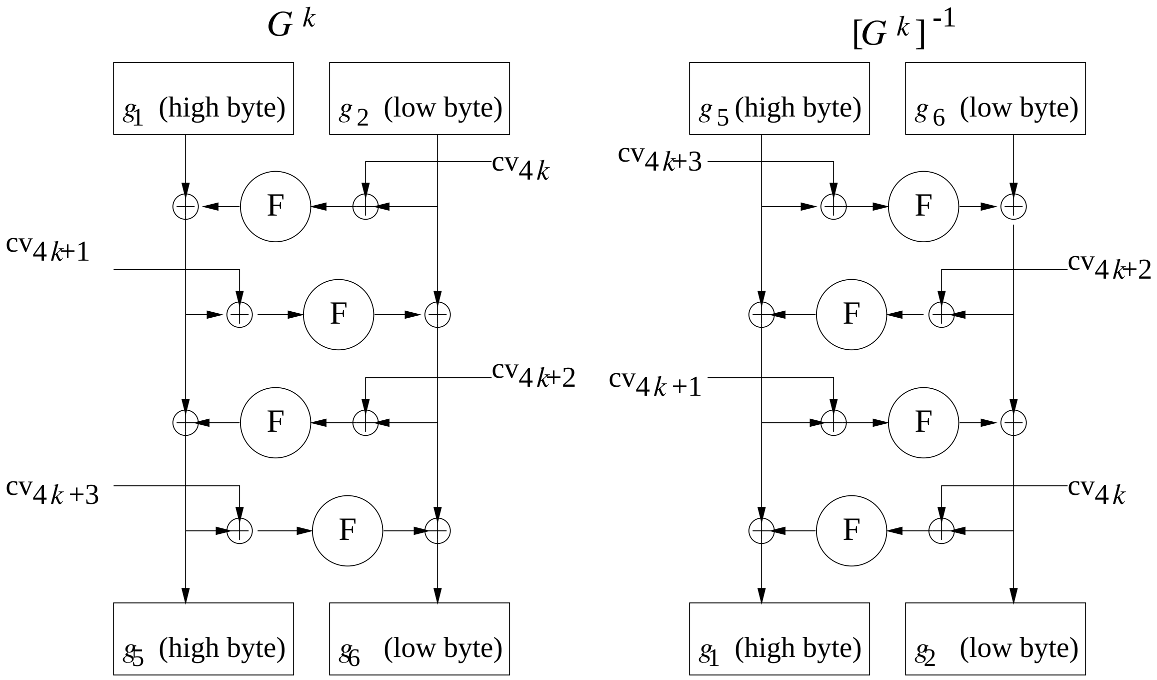


Figure 6: “G-permutation diagram”

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 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 $(0 < x < q)$
	$x = x_{159}x_{158}\cdots x_0$
Y	1024-bit public value corresponding to private value x
	$Y = g^x \bmod p = Y_{1023}Y_{1022}\cdots Y_0$
pad	80 bit padding value
	$pad = pad_{79}pad_{78}\cdots pad_0$
	= 72f1a87e92824198ab0b hex.
r	160-bit random number
	$r = r_{159}r_{158}\cdots r_0$

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 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:

$$1 < R_B, Y_B < p$$

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

Device B will compute and verify

$$1 < R_A, Y_A < p$$

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

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 $w_{1023} \cdots w_0$ 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

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

Note that this function represents the encryption of v_2 with v_1 XOR *pad*. Pictorally,

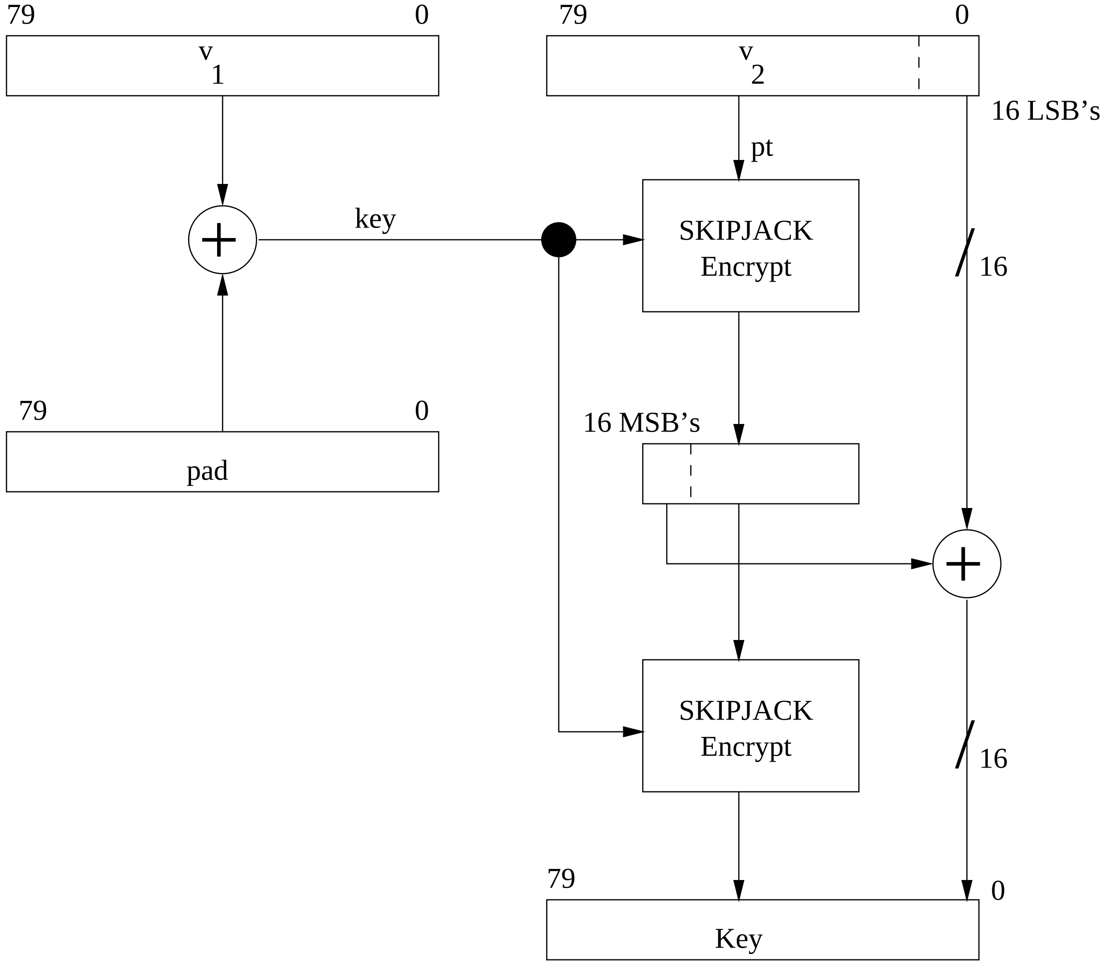
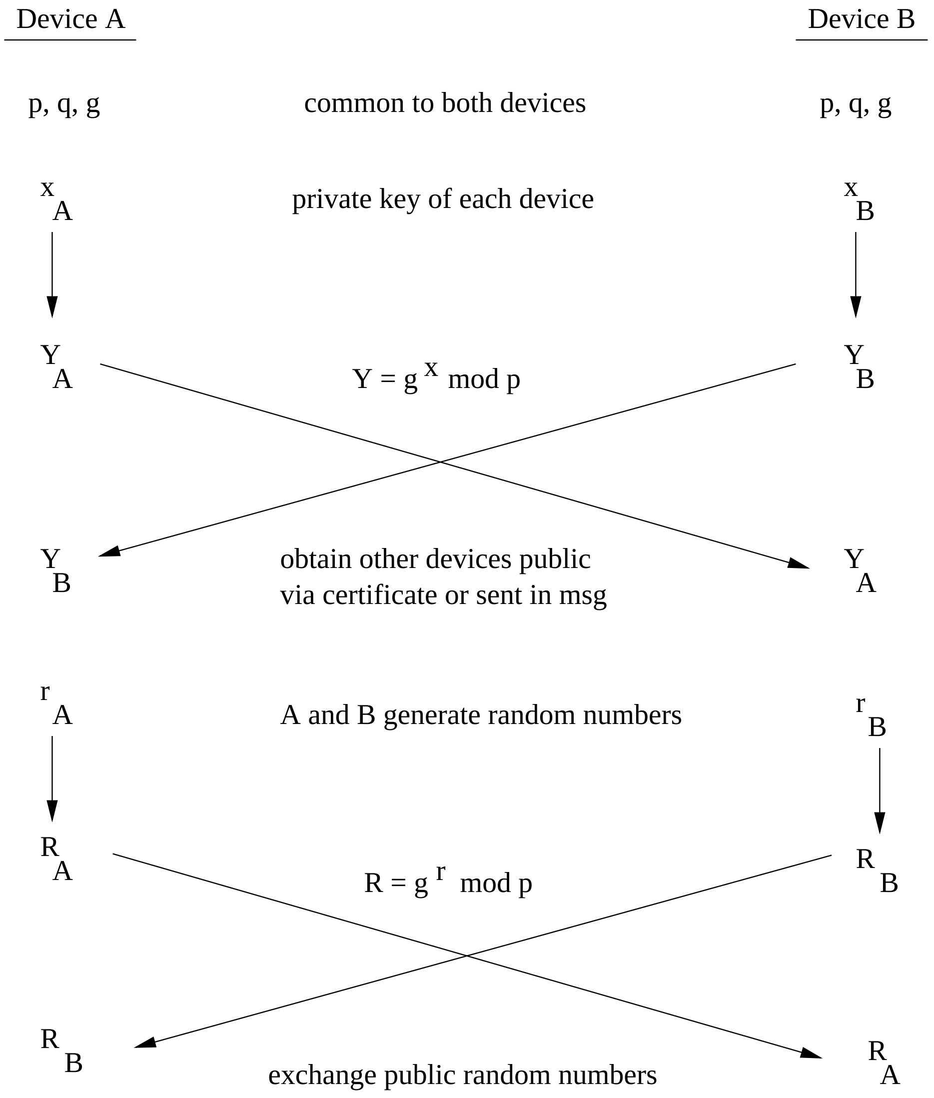


Figure 7: "Key Formation Diagram"

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



	check all values received	
$t_{AB} = (Y_B)^{r_A} \mod p$	compute $t = g^{r_A x_B} \mod p$	$t_{BA} = (R_A)^{x_B} \mod p$
$u_{AB} = (R_B)^{x_A} \mod p$	compute $u = g^{x_A r_B} \mod p$	$u_{BA} = (Y_A)^{r_B} \mod p$
$w = (t_{AB} + u_{AB}) \mod p$	compute w and check $w \neq 0$	$w = (t_{BA} + u_{BA}) \mod p$
v_1, v_2	extract v_1 and v_2 from w	v_1, v_2
Key	form Key from v_1, v_2, pad	Key

D. E-Mail Applications of KEA

For electronic mail appliations 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:

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

If the validation checks, go to step d. Should the verifiation 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

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

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

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

i. The Key is

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

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

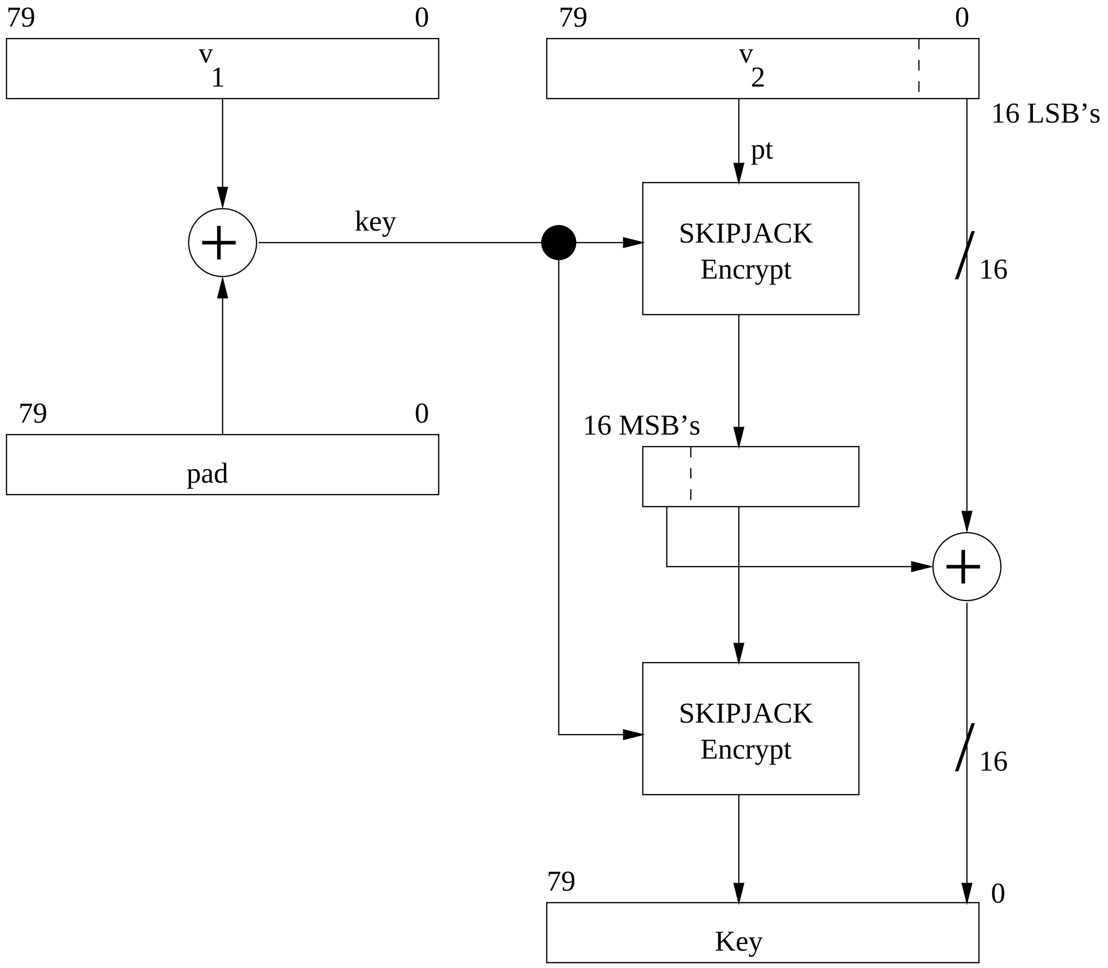


Figure 8: "Key Formation Diagram"

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:

$$1 < R_A, Y_A < p$$

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

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

e. Device 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

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

i. The Key is

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

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

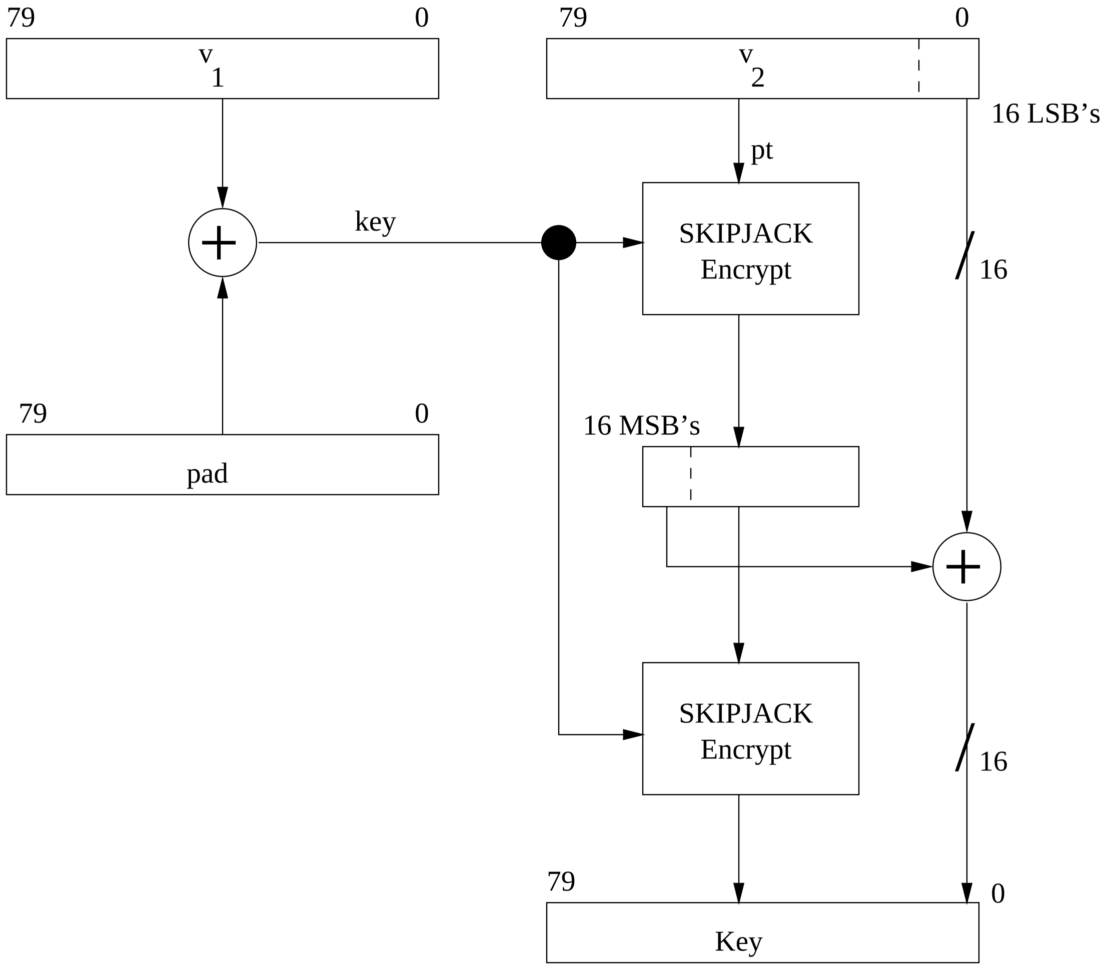
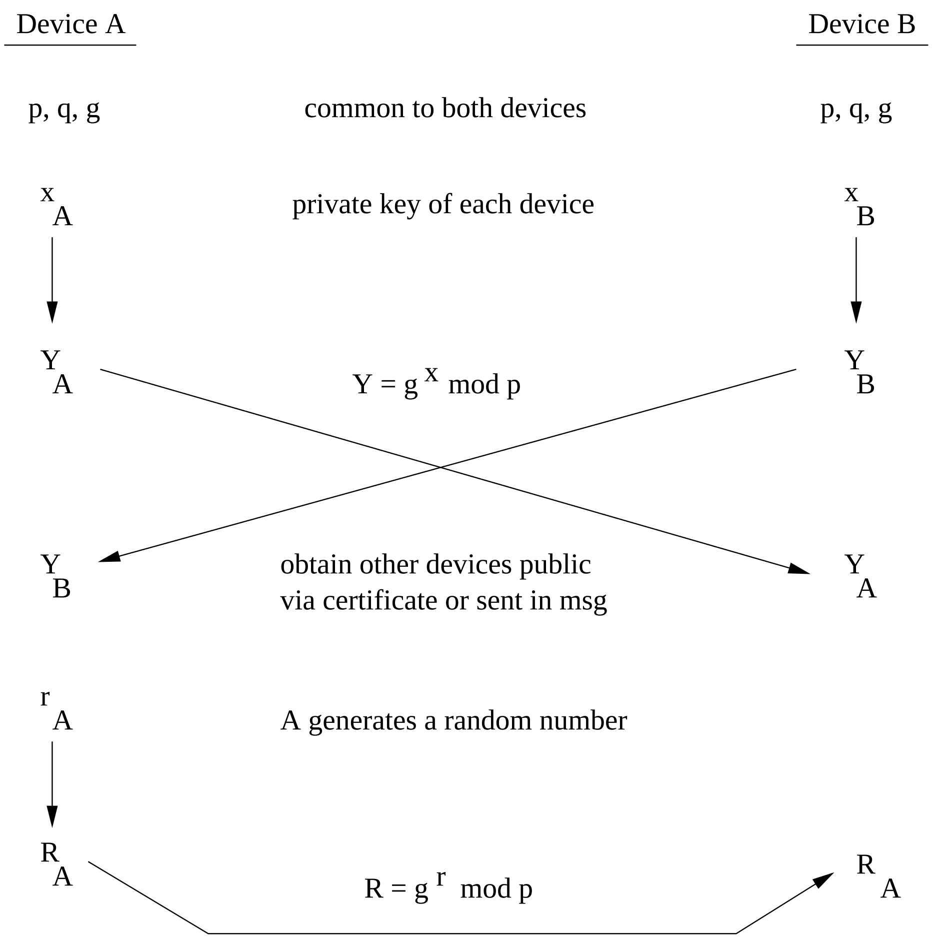


Figure 9: “Key Formation Diagram”

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



	check all values received	
$t_{AB} = (Y_B)^{r_A} \mod p$	compute $t = g^{r_A x_B} \mod p$	$t_{BA} = (R_A)^{x_B} \mod p$
$u_{AB} = (Y_B)^{x_A} \mod p$	compute $u = g^{x_A x_B} \mod p$	$u_{BA} = (Y_A)^{x_B} \mod p$
$w = (t_{AB} + u_{AB}) \mod p$	compute w and check $w \neq 0$	$w = (t_{BA} + u_{BA}) \mod p$
v_1, v_2	extract v_1 and v_2 from w	v_1, v_2
Key	form Key from v_1, v_2, pad	Key

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: `00998877665544332211`
Intermediate steps:

	w1 w2	w3 w4
0	33221100	ddccbbaa
1	b0040baf	1100ddcc
2	e6883b46	0baf1100
3	3c762d75	3b460baf
4	4c4547ee	2d753b46
5	b949820a	47ee2d75
6	f0e3dd90	820a47ee
7	f9b9be50	dd90820a
8	d79b5599	be50dd90
9	dd901e0b	820bbe50
10	be504c52	c391820b
11	820b7f51	f209c391
12	c391f9c2	fd56f209
13	f20925ff	3a5efd56
14	fd5665da	d7f83a5e
15	3a5e69d9	9883d7f8
16	d7f88990	53979883
17	9c000492	89905397
18	9fdccc59	04928990
19	3731beb2	cc590492
20	7afb7e7d	beb2cc59
21	7759bb15	7e7dbeb2
22	fb6445c0	bb157e7d
23	6f7f1115	45c0bb15
24	65a7deaa	111545c0
25	45c0e0f9	bb141115
26	11153913	a523bb14
27	bb148ee6	281da523
28	a523bfe2	35ee281d
29	281d0d84	1adc35ee
30	35eee6f1	25871adc
31	1adc60ee	d3002587
32	2587cae2	7a12d300

Ciphertext output: `2587cae27a12d300`

B. Key Exchange Algorithm (KEA)

p=9d4c6e6d 42ea91c8 28d67d49 94a9f01b 8e5b5b73 0d0faae7 bd569dd1
914e3ad4 759c8053 31eda145 9fb56be8 a8de4736 652a82b2 76e82acd
63f5b78d 0b75a03e b34d397d be7b3740 8f72136a cb0879fe 61c718a3
7f5154b5 078a7649 fb3d4fb4 c481e010 62c5241f 229fa580 423368dd
51090dbf 25351f0c 5800de05 b92ba6a9
q=97ad85fd 2b371ed0 69818ab3 c6ee8773 d9db029d
g=595d3443 ec897c82 51e5fa9d 02ab8b75 c0fc57b0 969f880d a366a100
01912a01 96bcb81c 41ac8485 031ac598 b5481eae 2726b719 d8dg915a
6105973g 72386c0a 6a2c732c d6700d34 1f54bf28 d12d692d e2fa05f5
5e898c2e 20bb8a26 02db1ba0 7de672e3 b96d9ac2 ga188450 63d918c3
2ed71266 b783311a 0a8d08ac 487bea44

ra=6201dd56 237c228a 3f54bc7e 794bdf32 41c67ea6
xa=62319ac4 7de14518 0abd322c 59e2b600 2781e4g4
Ya=2d29ecd0 2e3497a6 7222d8de bc286131 d149f458 1b3e586d 0151024c
02e8b23d a09a430e 2ca5ed1a 4b2d7725 62316e4d 2804d226 788284ed
655cf546 10d38f66 fab1a0a2 e2d3c661 4401901d 9758d566 722aff1f
734b2adb d2b67f13 00ce455f 00968ca7 91a87678 67363d7d 49ee74a2
8dc349d9 fdfdb96b 01f0fc1f 0690ec96

xb=63decdad 4487eb71 31dff4fS 1cfbae39 446b9b3d
rb=52bfa1d7 2f1cf0fb 0ff6d5df 15fb7483 167eb0e7
Yb=7730d4bb f3a2efdb 218e7041 3e861020 14cec06c 205f5419 293b65c6
9a971e54 55eb79a0 bdb90ab2 14c5240e de6cfdd5 8c7c19c5 269d57df
f60b61c1 db2ff648 64bee519 87f27003 4bc390ad 73168209 5e42608c
3d7987f9 649fbf71 6887633e b574b39c c73df899 51fc1bd6 d3889d48
fe2244b8 29afd405 06ab9221 ba562c07

Computed by A:
Ra=97c1fd8a 69fc8f34 a74c7ec3 c1ab176a b91fa0ea d0e6b097 06ae07a1
fbf8d0a6 67032ea4 798082b8 caea827b 4f604b71 e6c24469 211363ea
4bd2122f 4aa6afb9 4857ff06 9db03701 2b289057 b4855e70 f8f7ac4f
92fa1fe7 6c2a5c82 781ee611 1c1fbdf7 a6eb9dc3 59a8fca0 b632ef3a
2af82e52 c0a7f6a6 a2c961ea fc67f418

Computed by B:
Rb=91f61808 38f03d5b 6be538ff 6e0bf3cb 9d8afbbe ef199334 b389708b
b0c848da 860f0f27 62cc94a8 e496f8fc 94945538 cf6f1719 57cee4f1
e2eca2ba ddb340da f406e636 bbc6368e 4658fbf0 1a41cbef 5adb4086
42d03cec 4e85920c 8e7530bd e2b78cb8 7cbae364 31de373c d2ebaf29
d8412g32 8550dd8c f33e03c2 1a5056a0

Resu1ts for user A:
uab=1585dbba c06b963d 6ef5a30e 5c40220b 76fe0528 660be31a c496d1cb
0883ba8e 5a0331e9 ce3fe382 f47a353c edc6896d fdb4c0b5 67aafd72
4ba0ff6f 2c0fa428 fcb07a32 bf6fb88e 22c5ca47 7c9bb9cd 882da4fS
4cc57980 c174352f 13434623 ce3df2d4 14a9e0fb 7a905fe8 4ab282d5
e76e703a 55dabb38 27c2979f 08ea28c8
tab=8032eb2c b67534a9 c5faf6be a1eb6ef1 de0d3f48 c86be240 8f807e65
8622b9f3 87e0f50f a5868bf5 29ff008d 3ad55e9c 4366bad4 ae4190ce
bc3ae56f 34bf70b6 3ca021dd 563005db bc7e62bb ccc9127a 3603bf00
be8fce9b f46bf538 86c4a761 4b43adfe 7282efe4 f9c146b7 1e9f89d6
2bd3c7ed 7d127719 ebf0e0f8 79e0d0d9
w=95b8c6e7 76e0cae7 34f099cc fe2b90fd 550b4471 2e77c55b 54175031
8ea67481 e1e426f9 73c66f78 1e7g35ca 289be80a 411b7b8a 15ec8e41
07dbe4de 60cf14df 3g509c10 159fbe69 df442d03 4964cc47 be3163f6
0b55481c b5e02a67 9a07ed85 1981a0d2 872cd0e0 7451a69f 69520cac
13423827 d2ed3252 13b37897 82caf9a1
v1=95b8c6e7 76e0cae7 34f0XXXX
v2=99ccfe2b 90fd550b 4471XXXX
v1 XOR pad = e7496e99 e4628b7f 9ffbXXXX

Key for user A = 740839de e833add4 6b41XXXX

Results for user B:
tab=8032eb2c b67534a9 cSfaf6be a1eb6ef1 de0d3f48 c86be240 8f807e66
8622b9f3 87e0f50f a5868bf5 29ff008d 3ad55e9c 4366bad4 ae4190ce
bc3ae56f 34bf70b6 3ca021dd 563005db bc7e62bb ccc9127a 3603bf00
be8fce9b f46bf538 86c4a761 4b43adfe 7282efe4 f9c146b7 1e9f89d6
2bd3c7ed 7d127719 ebf0e0f8 79e0d0d9
uab=1585dbba c06b963d 6ef5a30e 5c40220b 76fe0528 660be31a c496d1cb
0883ba8e 5a0331e9 ce3fe382 f47a353c edc68g6d fdb4c0b5 67aafd72
4ba0ff6f 2c0fa428 fcb07a32 bf6fb88e 22c5ca47 7c9bb9cd 882da4f5
4cc57980 c174352f 13434623 ce3df2d4 14a9e0fb 7a905fe8 4ab282d5
e76e703a 55dabb38 27c2979f 08ea28c8
w=95b8c6e7 76e0cae7 34f099cc fe2b90fd 550b4471 2e77c55b 54175031
8ea67481 e1e426f9 73c66f78 1e7935ca 289be80a 411b7b8a 15ec8e41
07dbe4de 60cf14df 39509c10 159fbe69 df442d03 4964cc47 be3163f6
0b55481c b5e02a67 9a07ed85 1981a0d2 872cd0e0 7451a69f 69520cac
13423827 d2ed3252 13b37897 82cafga1
v1=95b8c6e7 76e0cae7 34f0XXXX
v2=99ccfe2b 90fd550b 4471XXXX
v1 XOR pad = e7496e99 e4628b7f 9ffbXXXX

Key for user B = 740839de e833add4 6b41XXXX

C. KEA Exchange for E-Mail

p=9d4c6e6d 42ea91c8 28d67d49 94a9f01b 8e5b5b73 0d0faae7 bd569dd1
914e3ad4 759c8053 31eda145 9fb56be8 a8de4736 652a82b2 76e82acc1
63fSb78d 0b75a03e b34d397d be7b3740 8f72136a cb0879fe 61c718a3
7f5154b5 078a7649 fb3d4fb4 c481e010 62c5241f 229fa580 423368dd
51090dbf 25351f0c 5800de05 b92ba6a9
q=97ad85fd 2b371ed0 69818ab3 c6ee8773 d9db029d
g=595d3443 ec897c82 51e5fa9d 02ab8b75 c0fc57b0 969f880d a366a100
01912a01 96bcb81c 41ac8485 031ac598 bS481eae 2726b719 d8d9915a
61059734 72386c0a 6a2c732c d6700d34 1f54bf28 d12d692d e2fa05f5
5e898c2e 20bb8a26 02db1ba0 7de672e3 b96d9ac2 9a188450 63d918c3
2ed71266 b783311a 0a8d08ac 487bea44

ra=6201dd56 237c228a 3f54bc7e 794bdf32 41c67ea6
xa=62319ac4 7de14518 0abd322c 59e2b600 2781e494
Ya=2d29ecd0 2e3497a6 7222d8de bc286131 d149f458 1b3e586d 0151024c
02e8b23d a09a430e 2ca5ed1a 4b2d7725 62316e4d 2804d226 788284ec1
655cf546 10d38f66 fab1a0a2 e2d3c661 4401901d 9758d566 722aff1f
734b2adb d2b67f13 00ce455f 00968ca7 91a87678 67363d7d 49ee74a2
8dc349d9 fdfdb96b 01f0fc1f 0690ec96

xb=63decdad 4487eb71 31dff4f5 1cfbae39 446b9b3d
Yb=7730d4bb f3a2efdb 218e7041 3e861020 14cec06c 205f5419 293h65c6
9a971e54 55eb79a0 bdb90ab2 14c5240e de6cfdd5 8c7c19c5 269d57df
f60b61c1 db2ff648 64bee519 87f27003 4bc390ad 73168209 5e42608c
3d7987f9 649fbf71 6887633e b574b39c c73df899 51fc1bd6 d3889d4
fe2244b8 29afd405 06ab9221 ba562c07

Computed by A:
Ra=97c1fd8a 69fc8f34 a74c7ec3 c1ab176a b91fa0ea d0e6b097 06ae07a1.
fbf8d0a6 67032ea4 798082b8 caea827b 4f604b71 e6c24469 211363ea
4bd2122f 4aa6afb9 4857ff06 9db03701 2b289057 b4855e70 f8f7ac4f
92fa1fe7 6c2a5c82 781ee611 1c1fbdf7 a6eb9dc3 59a8fca0 b632ef3a
2af82e52 c0a7f6a6 a2c961ea fc67f418

Resu1ts for user A:
tab=8032eb2c b67534a9 c5faf6be a1eb6ef1 de0d3f48 c86be240 8f807e65
8622bgf3 87e0f50f a5868bf5 29ff008d 3ad55e9c 4366bad4 ae4190ce
bc3ae56f 34bf70b6 3ca021dd 563005db bc7e62bb ccc9127a 3603bf00
be8fce9b f46bf538 86c4a761 4b43adfe 7282efe4 f9c146b7 1e9f89d6
2bd3c7ed 7d127719 ebf0e0f8 79e0d0d9
uab=17087175 9f16dfbf b0a0c05e 0ee49abd 49586033 93aa7df3 3d99bc61
68ad318a 7cf81fa8 74f4eb04 4433abe0 6423eb2f 1ebb3cdb 33067152
242d7cf8 987f208d cfdf3797 6398ccd5 6a0bdc1b 2bfd6734 35dedcc9
06bd6d71 a4516738 b91f2a52 689a2d60 802de96d 150fe661 469a2643

18c8d8f5 9ec040ea c623c51a 91d861d1
w=973b5ca2 558c1469 769bb71c b0d009af 27659f7c 5c166033 cd1a3ac7
eecfeb7e 04d914b8 1a7b76f9 6e32ac6d 9ef949cb 6221f7af e1480220
e0686267 cd3e9144 0c7f5974 b9c8d2b1 268a3ed6 f8c679ae 6be29bc9
c54d3c0d 98bd5c71 3fe3d1b3 b3dddb5e f2b0d952 0ed12d18 6539b019
449ca0e3 1bd2b804 b214a613 0bb932aa
v1=973b5ca2 558c1469 769bXXXX
v2=b71cb0d0 09af2765 9f7cXXXX
v1 XOR pad = e5caf4dc c70e55f1 dd90XXXX

Key for user A = 97fd1c6b d86bc439 115bXXXX

Resu1ts for user B:
tab=8032eb2c b67534a9 c5faf6be a1eb6ef1 de0d3f48 c86be240 8f8C7e66
8622b9f3 87e0f50f a5868bf5 29ff008d 3ad55e9c 4366bad4 ae4190ce
bc3ae56f 34bf70b6 3ca021dd 563005db bc7e62bb ccc9127a 3603bfC0
be8fce9b f46bf538 86c4a761 4b43adfe 7282efe4 f9c146b7 1e9f89d6
2bd3c7ed 7d127719 ebf0e0f8 79e0d0d9
uab=17087175 9f16dfbf b0a0c05e 0ee49abd 49586033 93aa7df3 3d99bc61
68ad318a 7cf81fa8 74f4eb04 4433abe0 6423eb2f 1ebb3cdb 33067152
242d7cf8 987f208d cfdf3797 6398ccd5 6a0bdc1b 2bfd6734 35dedcc9
06bd6d71 a4516738 b91f2a52 689a2d60 802de96d 150fe661 469a2643
18c8d8f5 9ec040ea c623c51a 91d861d1
w=973b5ca2 558C146g 769bb71c b0d009af 27659f7c 5c166033 cd1a3ac7
eecfeb7e 04d914b8 1a7b76f9 6e32ac6d 9ef949cb 6221f7af e1480220
e0686267 cd3e9144 0c7f5974 b9c8d2b1 268a3ed6 f8c679ae 6be29bc9
c54d3c0d 98bd5c71 3fe3d1b3 b3dddb5e f2b0d952 0ed12d18 6539b019
449ca0e3 1bd2b804 b214a613 0bb932aa
v1=973b5ca2 558c1469 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.
2. US DEPARTMENT OF COMMERCE Technology Administration/National Institute of Standards and Technology, DIGITAL SIGNATURE STANDARD (DSS), FIPS PUB 186, 19 May 1994.

About this document

This document is a reproduction of the SKIPJACK and KEA Algorithm Specifications declassified by the NSA, announced June 23, 1998.[1]

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:

`$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>.

[1] Details available from NIST, at <http://csrc.nist.gov/encryption/skipjack-kea.htm>.