Network Working Group R. Rivest
Request for Comments: 2268 MIT Laboratory for Computer Science
Category: Informational and RSA Data Security, Inc.
March 1998

A Description of the RC2(r) Encryption Algorithm

Status of this Memo

Copyright Notice

1. Introduction

This memo is an RSA Laboratories Technical Note. It is meant for informational use by the Internet community.

This memo describes a conventional (secret-key) block encryption algorithm, called RC2, which may be considered as a proposal for a DES replacement. The input and output block sizes are 64 bits each. The key size is variable, from one byte up to 128 bytes, although the current implementation uses eight bytes.

The algorithm is designed to be easy to implement on 16-bit microprocessors. On an IBM AT, the encryption runs about twice as fast as DES (assuming that key expansion has been done).

1.1 Algorithm description

We use the term "word" to denote a 16-bit quantity. The symbol + will denote twos-complement addition. The symbol & will denote the bitwise "and" operation. The term XOR will denote the bitwise "exclusive-or" operation. The symbol ~ will denote bitwise complement. The symbol ^ will denote the exponentiation operation. The term MOD will denote the modulo operation.

There are three separate algorithms involved:

Key expansion. This takes a (variable-length) input key and produces an expanded key consisting of 64 words K[0],...,K[63].

Encryption. This takes a 64-bit input quantity stored in words R[0], ..., R[3] and encrypts it "in place" (the result is left in R[0], ..., R[3]).

Decryption. The inverse operation to encryption.

2. Key expansion

Since we will be dealing with eight-bit byte operations as well as 16-bit word operations, we will use two alternative notations

for referring to the key buffer:

For word operations, we will refer to the positions of the buffer as K[0], ..., K[63]; each K[i] is a 16-bit word.

For byte operations, we will refer to the key buffer as L[0], ..., L[127]; each L[i] is an eight-bit byte.

These are alternative views of the same data buffer. At all times it will be true that

$$K[i] = L[2*i] + 256*L[2*i+1].$$

(Note that the low-order byte of each K word is given before the high-order byte.)

We will assume that exactly T bytes of key are supplied, for some T in the range 1 <= T <= 128. (Our current implementation uses T = 8.) However, regardless of T, the algorithm has a maximum effective key length in bits, denoted T1. That is, the search space is 2^(8*T), or 2^T1, whichever is smaller.

The purpose of the key-expansion algorithm is to modify the key buffer so that each bit of the expanded key depends in a complicated way on every bit of the supplied input key.

The key expansion algorithm begins by placing the supplied T-byte key into bytes L[0], ..., L[T-1] of the key buffer.

The key expansion algorithm then computes the effective key length in bytes T8 and a mask TM based on the effective key length in bits T1. It uses the following operations:

```
T8 = (T1+7)/8;
TM = 255 MOD 2^(8 + T1 - 8*T8);
```

Thus TM has its 8 - (8*T8 - T1) least significant bits set.

For example, with an effective key length of 64 bits, T1 = 64, T8 = 8 and TM = 0xff. With an effective key length of 63 bits, T1 = 63, T8 = 8 and TM = 0x7f.

Here PITABLE[0], ..., PITABLE[255] is an array of "random" bytes based on the digits of PI = 3.14159... . More precisely, the array PITABLE is a random permutation of the values 0, ..., 255. Here is the PITABLE in hexadecimal notation:

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

The key expansion operation consists of the following two loops and intermediate step:

```
for i = T, T+1, ..., 127 do
  L[i] = PITABLE[L[i-1] + L[i-T]];

L[128-T8] = PITABLE[L[128-T8] & TM];

for i = 127-T8, ..., 0 do
  L[i] = PITABLE[L[i+1] XOR L[i+T8]];
```

(In the first loop, the addition of L[i-1] and L[i-T] is performed modulo 256.)

The "effective key" consists of the values L[128-T8],..., L[127]. The intermediate step’s bitwise "and" operation reduces the search space for L[128-T8] so that the effective number of key bits is T1. The expanded key depends only on the effective key bits, regardless of the supplied key K. Since the expanded key is not itself modified during encryption or decryption, as a pragmatic matter one can expand the key just once when encrypting or decrypting a large block of data.

3. Encryption algorithm

The encryption operation is defined in terms of primitive "mix" and "mash" operations.

Here the expression "x rol k" denotes the 16-bit word x rotated left by k bits, with the bits shifted out the top end entering the bottom end.

3.1 Mix up R[i]

The primitive "Mix up R[i]" operation is defined as follows, where s[0] is 1, s[1] is 2, s[2] is 3, and s[3] is 5, and where the indices of the array R are always to be considered "modulo 4," so that R[i-1] refers to R[3] if i is 0 (these values are

"wrapped around" so that R always has a subscript in the range 0 to 3 inclusive):

```
R[i] = R[i] + K[j] + (R[i-1] & R[i-2]) + ((~R[i-1]) & R[i-3]);
j = j + 1;
R[i] = R[i] rol s[i];
```

In words: The next key word K[j] is added to R[i], and j is advanced. Then R[i-1] is used to create a "composite" word which is added to R[i]. The composite word is identical with R[i-2] in those positions where R[i-1] is one, and identical to R[i-3] in those positions where R[i-1] is zero. Then R[i] is rotated left by s[i] bits (bits rotated out the left end of R[i] are brought back in at the right). Here j is a "global" variable so that K[j] is always the first key word in the expanded key which has not yet been used in a "mix" operation.

3.2 Mixing round

A "mixing round" consists of the following operations:

```
Mix up R[0]
Mix up R[1]
Mix up R[2]
Mix up R[3]
```

3.3 Mash R[i]

The primitive "Mash R[i]" operation is defined as follows (using the previous conventions regarding subscripts for R):

```
R[i] = R[i] + K[R[i-1] & 63];
```

In words: R[i] is "mashed" by adding to it one of the words of the expanded key. The key word to be used is determined by looking at the low-order six bits of R[i-1], and using that as an index into the key array K.

3.4 Mashing round

A "mashing round" consists of:

```
Mash R[0]
Mash R[1]
Mash R[2]
Mash R[3]
```

3.5 Encryption operation

The entire encryption operation can now be described as follows. Here j is a global integer variable which is affected by the mixing operations.

1. Initialize words R[0], ..., R[3] to contain the 64-bit input value.

2. Expand the key, so that words K[0], ..., K[63] become defined.

3. Initialize j to zero.

4. Perform five mixing rounds.

5. Perform one mashing round.

6. Perform six mixing rounds.

7. Perform one mashing round.

8. Perform five mixing rounds.

Note that each mixing round uses four key words, and that there are 16 mixing rounds altogether, so that each key word is used exactly once in a mixing round. The mashing rounds will refer to up to eight of the key words in a data-dependent manner. (There may be repetitions, and the actual set of words referred to will vary from encryption to encryption.)

4. Decryption algorithm

The decryption operation is defined in terms of primitive operations that undo the "mix" and "mash" operations of the encryption algorithm. They are named "r-mix" and "r-mash" (r- denotes the reverse operation).

Here the expression "x ror k" denotes the 16-bit word x rotated right by k bits, with the bits shifted out the bottom end entering the top end.

4.1 R-Mix up R[i]

The primitive "R-Mix up R[i]" operation is defined as follows, where s[0] is 1, s[1] is 2, s[2] is 3, and s[3] is 5, and where the indices of the array R are always to be considered "modulo 4," so that R[i-1] refers to R[3] if i is 0 (these values are "wrapped around" so that R always has a subscript in the range 0 to 3 inclusive):

```
R[i] = R[i] ror s[i];
R[i] = R[i] - K[j] - (R[i-1] & R[i-2]) - ((~R[i-1]) & R[i-3]);
j = j - 1;
```

In words: R[i] is rotated right by s[i] bits (bits rotated out the right end of R[i] are brought back in at the left). Here j is a "global" variable so that K[j] is always the key word with greatest index in the expanded key which has not yet been used in a "r-mix" operation. The key word K[j] is subtracted from R[i], and j is decremented. R[i-1] is used to create a "composite" word which is subtracted from R[i]. The composite word is identical with R[i-2] in those positions where R[i-1] is one, and identical to R[i-3] in those positions where R[i-1] is zero.

4.2 R-Mixing round

An "r-mixing round" consists of the following operations:

```
R-Mix up R[3]
R-Mix up R[2]
R-Mix up R[1]
R-Mix up R[0]
```

4.3 R-Mash R[i]

The primitive "R-Mash R[i]" operation is defined as follows (using the previous conventions regarding subscripts for R):

```
R[i] = R[i] - K[R[i-1] & 63];
```

In words: R[i] is "r-mashed" by subtracting from it one of the words of the expanded key. The key word to be used is determined by looking at the low-order six bits of R[i-1], and using that as an index into the key array K.

4.4 R-Mashing round

An "r-mashing round" consists of:

```
R-Mash R[3]
R-Mash R[2]
R-Mash R[1]
R-Mash R[0]
```

4.5 Decryption operation

The entire decryption operation can now be described as follows. Here j is a global integer variable which is affected by the mixing operations.

1. Initialize words R[0], ..., R[3] to contain the 64-bit ciphertext value.

2. Expand the key, so that words K[0], ..., K[63] become defined.

3. Initialize j to 63.

4. Perform five r-mixing rounds.

5. Perform one r-mashing round.

6. Perform six r-mixing rounds.

7. Perform one r-mashing round.

8. Perform five r-mixing rounds.

5. Test vectors

Test vectors for encryption with RC2 are provided below.

All quantities are given in hexadecimal notation.

Key length (bytes) = 8
Effective key length (bits) = 63
Key = 00000000 00000000
Plaintext = 00000000 00000000
Ciphertext = ebb773f9 93278eff

Key length (bytes) = 8
Effective key length (bits) = 64
Key = ffffffff ffffffff
Plaintext = ffffffff ffffffff
Ciphertext = 278b27e4 2e2f0d49

Key length (bytes) = 8
Effective key length (bits) = 64
Key = 30000000 00000000
Plaintext = 10000000 00000001
Ciphertext = 30649edf 9be7d2c2

Key length (bytes) = 1
Effective key length (bits) = 64
Key = 88
Plaintext = 00000000 00000000
Ciphertext = 61a8a244 adacccf0

Key length (bytes) = 7
Effective key length (bits) = 64
Key = 88bca90e 90875a
Plaintext = 00000000 00000000
Ciphertext = 6ccf4308 974c267f

Key length (bytes) = 16
Effective key length (bits) = 64
Key = 88bca90e 90875a7f 0f79c384 627bafb2
Plaintext = 00000000 00000000
Ciphertext = 1a807d27 2bbe5db1

Key length (bytes) = 16
Effective key length (bits) = 128
Key = 88bca90e 90875a7f 0f79c384 627bafb2
Plaintext = 00000000 00000000
Ciphertext = 2269552a b0f85ca6

Key length (bytes) = 33
Effective key length (bits) = 129
Key = 88bca90e 90875a7f 0f79c384 627bafb2 16f80a6f 85920584
c42fceb0 be255daf 1e

```
Plaintext = 00000000 00000000
Ciphertext = 5b78d3a4 3dfff1f1
```

6. RC2 Algorithm Object Identifier

The Object Identifier for RC2 in cipher block chaining mode is

```
rc2CBC OBJECT IDENTIFIER
  ::= {iso(1) member-body(2) US(840) rsadsi(113549)
       encryptionAlgorithm(3) 2}
```

RC2-CBC takes parameters

```
RC2-CBCParameter ::= CHOICE {
  iv IV,
  params SEQUENCE {
    version RC2Version,
    iv IV
  }
}
```

where

```
IV ::= OCTET STRING -- 8 octets
RC2Version ::= INTEGER -- 1-1024
```

RC2 in CBC mode has two parameters: an 8-byte initialization vector (IV) and a version number in the range 1-1024 which specifies in a roundabout manner the number of effective key bits to be used for the RC2 encryption/decryption.

The correspondence between effective key bits and version number is as follows:

1. If the number EKB of effective key bits is in the range 1-255, then the version number is given by Table[EKB], where the 256-byte translation table Table[] is specified below. Table[] specifies a permutation on the numbers 0-255; note that it is not the same table that appears in the key expansion phase of RC2.

2. If the number EKB of effective key bits is in the range 256-1024, then the version number is simply EKB.

   The default number of effective key bits for RC2 is 32. If RC2-CBC is being performed with 32 effective key bits, the parameters should be supplied as a simple IV, rather than as a SEQUENCE containing a version and an IV.

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

A. Intellectual Property Notice

RC2 is a registered trademark of RSA Data Security, Inc. RSA’s copyrighted RC2 software is available under license from RSA Data Security, Inc.

B. Author’s Address

Ron Rivest
RSA Laboratories
100 Marine Parkway, #500
Redwood City, CA 94065 USA

Phone: (650) 595-7703
EMail: rsa-labs@rsa.com

C. Full Copyright Statement