The RC5 Encryption Algorithm*

Ronald L. Rivest

MIT Laboratory for Computer Science
545 Technology Square, Cambridge, Mass. 02139
rivest@theory.lcs.mit.edu

Abstract. This document describes the RC5 encryption algorithm. RC5 is a fast symmetric block cipher suitable for hardware or software implementations. A novel feature of RC5 is the heavy use of *data-dependent rotations*. RC5 has a variable word size, a variable number of rounds, and a variable-length secret key.

1 A Parameterized Family of Encryption Algorithms

RC5 is *word*-oriented: all of the primitive operations work on w-bit words as their basic unit of information. Here we assume $w = 32$, although the formal specification of RC5 admits variants for other word lengths, such as $w = 64$ bits. RC5 has two-word (64-bit) input (plaintext) and output (ciphertext) block sizes.

RC5 uses an "expanded key table," S, derived from the user's supplied secret key. The size t of table S depends on the number r of rounds: S has $t = 2(r+1)$ words.

There are thus several distinct "RC5" algorithms, depending on the choice of parameters w and r. We summarize these parameters below:

- w This is the *word size*, in bits; each word contains $u = (w/8)$ 8-bit bytes. The standard value of w is 32 bits; allowable values of w are 16, 32, and 64. RC5 encrypts two-word blocks: plaintext and ciphertext blocks are each $2w$ bits long.
- r This is the number of rounds. Also, the expanded key table S contains $t = 2(r+1)$ words. Allowable values of r are 0, 1, ..., 255.

In addition to w and r, RC5 has a variable-length secret cryptographic key, specified parameters b and K:

- b The number of bytes in the secret key K. Allowable values of b are 0, 1, ..., 255.
- K The b-byte secret key: $K[0]$, $K[1]$, ..., $K[b-1]$.

A particular RC5 algorithm is designated as *RC5-w/r/b* . For example, RC5-32/16/10 has 32-bit words, 16 rounds, a 10-byte (80-bit) secret key variable, and an expanded key table of $2(16+1) = 34$ words, Parameters may be dropped,

* RC5 is a trademark of RSA Data Security. Patent pending.

from last to first, to talk about RC5 with the dropped parameters unspecified. (So, for example: how many rounds should one use in RC5-32?)

All of the parameters given above are packaged together to form a *RC5 control block*, containing the following fields:

v 1 byte version number; `10` (hex) for version 1.0 here.
w 1 byte.
r 1 byte.
b 1 byte.
K b bytes.

A control block is thus represented using $b + 4$ bytes. It is expected that RC5 key-management schemes would typically manage and transmit entire RC5 control blocks. As an example, the control block

```
10 20 10 0A 20 33 7D 83 05 5F 62 51 BB 09 (in hexadecimal)
```

specifies an RC5 algorithm (version 1.0) with 32-bit words, 16 rounds, and a 10-byte (80-bit) key "`20 33 ... 09`".

2 Parameter Values – Philosophy

It is not intended that RC5 be secure for all possible parameter values. For example, $r = 0$ provides essentially no encryption, and $r = 1$ is easily broken. And choosing $b = 0$ clearly gives no security.

On the other hand, choosing the maximum parameter values would be overkill for most applications.

We provide a variety of parameter settings so that users may select an encryption algorithm whose security and speed are optimized for their application, while providing an evolutionary path for adjusting their parameters as necessary in the future.

As an example, consider the problem of replacing DES with an "equivalent" RC5 algorithm, such as RC5-32/16/7. The input/output blocks are $2w = 64$ bits long, just as in DES. The number of rounds is also the same, although each RC5 round is similar to two DES rounds since all data registers, rather than just half of them, are updated in one round. Finally, the number of key bits, $7 * 8 = 56$, is identical.

Unlike unparameterized DES, however, RC5 permits upgrades as necessary. For example, an RC5 user can upgrade the above choice for a DES replacement to an 80-bit key, for example, by moving to RC5-32/16/10. As technology improves, and as the true strength of RC5 algorithms becomes better understood through analysis, the most appropriate parameters can be chosen.

The choice of r affects both encryption speed and security. For some applications, high speed may be the most critical requirement—one wishes for the best security obtainable within a given encryption time requirement. Choosing a smallish value of r (say $r = 6$) may provide some security, albeit modest, within the given speed constraint.

In other applications, such as key-management, security is the primary concern, and speed is relatively unimportant. Here, choosing 32 rounds might be appropriate for such applications. Since RC5 is a new design, the security provided by various values of r is still open to study.

Similarly, the word size w also affects speed and security. For example, choosing a value of w larger than the register size of the CPU can degrade encryption speed. The word size $w = 16$ is primarily for researchers who wish to examine the security properties of a natural "scaled-down" RC5. As 64-bit processors become common, one can move to RC5-64 as a natural extension of RC5-32. It may also be convenient to specify $w = 64$ if RC5 is to be used as the basis for a hash function, in order to have 128-bit input/output blocks.

It may be considered unusual and risky to specify an encryption algorithm that permits insecure parameter choices. We have two responses to this criticism:

1. A fixed set of parameters may be at least as dangerous, since the parameters can not be increased when necessary. Consider the problem DES has now: its key size is too short, and there is no easy way to increase it.
2. It is expected that implementors will provide implementations that ensure that suitably large parameters are chosen. While unsafe choices might be usable in principle, in practice they would be forbidden.

It is not expected that a typical RC5 implementation will work with any RC5 control block. Rather, it may only work for certain parameter values, or parameters in a certain range. The parameters w, r, and b in a received or transmitted RC5 control block are then merely used for *type-checking*—values other than those supported by the implementation will be disallowed. The flexibility of RC5 is thus utilized at the system design stage, when the appropriate parameters are chosen, rather than at run time, when unsuitable parameters might be chosen by an unwary user.

We propose RC5-32/12/16 as providing a "nominal" choice of parameters. Further analysis is needed to analyze the security of this choice.

3 Notation and RC5 Primitive Operations

We use $\lg(x)$ to denote the base-two logarithm of x.

RC5 uses only the following three primitive operations (and their inverses).

1. Two's complement addition of words, denoted by "+". This is modulo-2^w addition. The inverse operation, subtraction, is denoted "−".
2. Bit-wise exclusive-OR of words, denoted by $\oplus$.
3. A left-rotation (or "left-spin") of words: the rotation of word x left by y bits is denoted $x \lll y$. Only the $\lg(w)$ low-order bits of y are used to determine the rotation amount, so that y is interpreted modulo w . The inverse operation, right-rotation, is denoted $x \ggg y$.

These operations are directly and efficiently supported by most processors.

A distinguishing feature of RC5 is that the rotations are rotations by "variable" (plaintext-dependent) amounts. We note that on modern microprocessors, a variable-rotation $x \lll y$ takes an amount of time that is independent of the shift amount y. We also note that rotations are the only non-linear operator in RC5; there are no nonlinear substitution tables or other nonlinear operators. The strength of RC5 depends heavily on the cryptographic properties of data-dependent rotations.

4 The RC5 Algorithm

The plaintext input to RC5 consists of two w-bit words, denoted A and B.

The algorithm uses an *expanded key table*, $S[0...t-1]$, consisting of $t = 2(r+1)$ w-bit words. The key-expansion algorithm initializes S from the user's given secret key parameter K. (Note that S is not used like a DES S-box.)

RC5 consists of three components: a *key expansion* algorithm, an *encryption* algorithm, and a *decryption* algorithm.

4.1 Key Expansion

The purpose of the key-expansion routine is to expand the user's key K to fill the expanded key array S, so S resembles an array of t random binary words determined by the user's secret key K.

Definition of the Magic Constants The key-expansion algorithm uses two word-sized binary constants P_w and Q_w. They are defined for arbitrary w as follows:

$$P_w = \text{Odd}((e-2)2^w) \tag{1}$$

$$Q_w = \text{Odd}((\phi-1)2^w) \tag{2}$$

where

$$e = 2.718281828459... \text{ (base of natural logarithms)}$$

$$\phi = 1.618033988749... \text{ (golden ratio)} ,$$

and where $\text{Odd}(x)$ is the least odd integer greater than or equal to $\lfloor x \rfloor$. For $w = 16$, 32, and 64, each constant is given below in binary and in hexadecimal.

```
P16 = 1011011111100001 = b7e1
Q16 = 1001111000110111 = 9e37

P32 = 10110111111000010101000101100011 = b7e15163
Q32 = 10011110001101110111100110111001 = 9e3779b9

P64 = 1011011111100001010100010110001010001010111011010010101001101011
    = b7e151628aed2a6b
Q64 = 1001111000110111011110011011100101111111010010100111110000010101
    = 9e3779b97f4a7c15
```

Converting the Secret Key from Bytes to Words. The first step of key expansion is to copy the secret key $K[0...b-1]$ into an array $L[0...c-1]$ of $c = \lceil b/u \rceil$ words, where $u = w/8$ is the number of bytes/word. This is done in a natural manner, using u successive key bytes of K to fill up each successive words in L, low-order byte to high-order byte. Unfilled byte positions of L, if any, are zeroed.

On "little-endian" machines such as an Intel '486, the above task can be accomplished merely by zeroing the array L, and then copying the string K directly into the memory positions representing L. The following pseudo-code achieves the same effect, assuming that all bytes are "unsigned" and that array L is initially zeroed.

for $i = b-1$ **downto** 0 **do**
    $L[i/u] = (L[i/u] \lll 8) + K[i]$;

Initializing the Array *S*. The second step of key expansion is to initialize array S using a linear congruential generator modulo 2^w determined by the "magic constants" P_w and Q_w. Since Q_w is odd, the generator has period 2^w.

$S[0] = P_w$;
for $i = 1$ **to** $t-1$ **do**
    $S[i] = S[i-1] + Q_w$;

Mixing in the Secret Key. The third step of key expansion is mix in the user's secret key in three passes over the arrays S and L. Actually, due to the differing sizes of these arrays, the largest array will be processed three times, and the other may be handled more times.

$i = j = 0$;
$A = B = 0$;
do $3 * \max(t, c)$ **times:**
    $A = S[i] = (S[i] + A + B) \lll 3$;
    $B = L[j] = (L[j] + A + B) \lll (A + B)$;
    $i = (i+1) \bmod (t)$;
    $j = (j+1) \bmod (c)$;

The key-expansion function has a certain amount of "one-wayness": it is not so easy to determine K from S.

4.2 Encryption

We assume that the input block is given in two w-bit registers A and B. (We assume standard *little-endian* conventions for packing bytes into input/output blocks: the first byte goes into the low-order bit positions of register A, and so on.) We also assume that key-expansion has already been performed. Here is the encryption algorithm in pseudo-code:

$A = A + S[0];$
$B = B + S[1];$
for $i = 1$ **to** r **do**
    $A = ((A \oplus B) \lll B) + S[2 * i];$
    $B = ((B \oplus A) \lll A) + S[2 * i + 1];$

The output is in the registers A and B.

4.3 Decryption

The decryption routine is easily derived from the encryption routine.

for $i = r$ **downto** 1 **do**
    $B = ((B - S[2 * i + 1]) \ggg A) \oplus A;$
    $A = ((A - S[2 * i]) \ggg B) \oplus B;$
$B = B - S[1];$
$A = A - S[0];$

5 Discussion

A distinguishing feature of RC5 is its heavy use of *data-dependent rotations*—the amount of rotation performed is dependent on the input data, and is not pre-determined.

The encryption/decryption routines are very simple. While other operations (such as substitution operations) could have been included in the basic round operations, the objective here is to focus on the data-dependent rotations as a source of cryptographic strength.

Some of the expanded key from S is initially added to the plaintext, and each round ends by adding expanded key from S to the intermediate values just computed. This assures that each round acts in a potentially different manner, in terms of the shift amounts used.

The xor operations provide some avalanche properties, causing a single-bit change in an input block to cause multiple bit-changes in following rounds.

The encryption algorithm is very compact, and can be coded efficiently in assembly language on most processors. The table S is accessed sequentially, minimizing issues of cache size. The RC5 encryption speeds obtainable are yet to be fully determined. For RC5-32/12/16 on a 90MhZ Pentium, a preliminary C++ implementation compiled with the Borland C++ compiler (in 16-bit mode) performs a key-setup in 220 microseconds and performs an encryption in 22 microseconds (equivalent to 360,000 bytes/sec). These timings can presumably be improved by more than an order of magnitude using a 32-bit compiler and/or assembly language—an assembly-language routine for the '486 can perform each round in eight instructions.

6 Analysis

This section contains some preliminary results on the strength of RC5. Much more work remains to be done. Here we report the results of two experiments studying how changing the number of rounds affects properties of RC5.

The first test involved uniformity of correlation between input and output bits. We found that four rounds sufficed to get very uniform correlations between individual input and output bits in RC5-32.

The second test checked to see if the amount of variable rotations performed depended on every plaintext bit, in 10,000 trials. That is, it checked whether flipping a plaintext bit caused some intermediate rotation to be a rotation by a different amount. We found that seven rounds in RC5-32 were sufficient to cause each message bit to affect some rotation amount.

The number of rounds chosen in practice should always be at least as great (if not substantially greater) than these simple-minded tests would suggest. As noted above, we suggest 12 rounds as a "nominal" choice.

The use by RC5 of variable rotations should help defeat differential cryptanalysis (Biham/Shamir [1]) and linear cryptanalysis (Matsui [2]), since bits are rotated to "random" positions in each round.

There is no obvious way in which an RC5 key can be "weak," other than by being too short.

I invite the reader to help determine the strength of RC5.

7 Acknowledgements

I'd like to thank Burt Kaliski, Lisa Yin, Paul Kocher, and everyone else at RSA Laboratories for their comments and constructive criticisms.

References

1. E. Biham and A. Shamir. *A Differential Cryptanalysis of the Data Encryption Standard.* Springer-Verlag, 1993.
2. Mitsuru Matsui. The first experimental cryptanalysis of the data encryption standard. In Yvo G. Desmedt, editor, *Proceedings CRYPTO 94*, pages 1–11, Springer, 1994. Lecture Notes in Computer Science No. 839.

This article was processed using the LaTeX macro package with LLNCS style