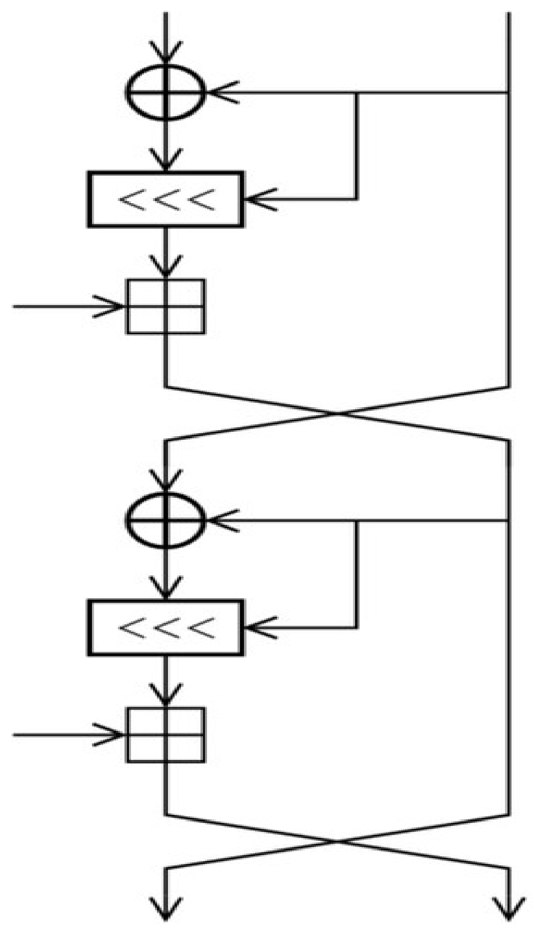
Improving the RC5 Encryption Algorithm

Betty Huang

Computer Systems Lab 2009-2010

Abstract

Since 2005, collision factors have been found for the SHA-1 algorithm. This paper aims to provide several mechanisms to improve the applicability of the algorithm for areas that are not able to adopt a new standard readily. Authors such as Yiqun Lisa Yin, Michael Szydlo, and various members of the National Institute of Standards and Technology have written about the possibilty of modifying the original input in order to reduce the likelihood of collisions/pre-image attacks, but none have tested extensively on how much the randomization would increase the efforts of decrpytion. I hope to be able to test a (likely simplified) version of the techniques against several sophisticated collision-based attacks.



Background and Introduction

A block cipher consists of two paired algorithms, one for encryption, E, and the other for decryption, E-1. Both algorithms accept two inputs: an input block of size n bits and a key of size k bits, yielding an n-bit output block. For any one fixed key, decryption is the inverse function of encryption, so that for any block M and key K. M is termed the plaintext and C the ciphertext.

Since the advent of the encryption algorithms, there have been numerous attempts to reverse engineer the process in order to return the original input. Thus, the response of NIST has been to develop stronger algorithms. The weaker, "obsolete" algorithm is then cycled out of use. However, the implementation of a stronger security standard may take years. However, there is a need for a method that can reduce the chances of breaking the encryption algorithm, without modifying the algorithm itself. This would not be as difficult to implement, and can secure the privacy of individuals while the administrators switch to a newer algorithm. Thus, my project seeks to find out how "efficient" (ie. how much effort a collision attack would require) these modifications are. I decided to test the efficiency of this method against the RC5 encryption algorithm, which is a symmetric block cipher developed by Ronald L. Rivest. The RC5 is distinctive as it uses data-dependent rotations (which simply means that the rotation amounts are random variables that arise from the input, rather than predetermined constants). The strength of RC5 is dependent on the number of "rotations," or repeated algorithmic inputs. Thus, different permutations of the algorithm may arise, which results in a notation of RC5-w/r/b, where w is the word size (in bits), r is the number of rounds, and b is the number of bytes allowed. RC5 utilizes an expanded key table S from secret key K, which size t is determinant on the number of rounds. Generally, t=2(r+1). Common w/r/b choices are RC5-32/16/7, with upgrades easily available (RC5-32/16/10). The algorithm itself is actually three smaller algorithms (key expansion, encryption, and decryption). The key expansion is mainly dependant on "magic constants" and mixing with K and + and left-rotation (<<<) operations.
During the encryption state, the algorithm takes two w-bit registers (A and B) and outputs A and B after modification. Here is Rivest's original outline:

1. Two's complement addition of words, denoted by “+". This is modulo-2w addition. The inverse operation, subtraction, is denoted “-”.

2. Bit-wise exclusive-OR of words, denoted by ⊕.

3. A left-rotation (or “left-spin") of words: the cyclic rotation of word x left by y bits is denoted x << y. Here y is interpreted modulo w, so that when w is a power of two, only the lg(w) low-order bits of y are used to determine the rotation amount. The inverse operation, right-rotation, is denoted x >> y.

```
void RC5_SETUP(WORD *K[4]) { /* secret input key K[0...b-1]    */

          WORD i, j, k, u=w/8, A, B, L[c];

          /* NB: L[] == K[] in this instance */

          for (S[0]=P,i=1; i<t; i++) {
                 S[i] = S[i-1]+Q;
          }

          L[0]=K[0];
          L[1]=K[1];
          L[2]=K[2];
          L[3]=K[3];

          for (A=B=i=j=k=0; k<3*t; k++,i=(i+1)%t,j=(j+1)%c) { /* 3*t
   times */
                 A = S[i] = S[i]+(A+B);
                 B = L[j] = L[j]+(A+B);
          }
   }
```

Discussion

During this quarter, I experimented with coding a break of the RC5. At first, I worked on trying to
identify weak sections of the algorithm by studying the effects of simplifying the round and
rotation number. The first program written simply shifted through all the possible bit combinations
of RC5. This is the algorithm that Yin et al outlined in their paper:

```
For $i = 2r + 1$ down to 3
  Obtain a set $G_i$ of good pairs for $S_i$
  For $s = 0$ to $w - 1$
    Select a pair in $G_i$ that is useful
    Compute $S_i[s]$
```

Results

```
00000000000000000000000000000000,FB6286344868FD57
00000000000000000000000000000001,160EBEBC61C38363
00000000000000000000000000000002,6C8246C81A1EB773
00000000000000000000000000000004,A5B198246521EA6F
00000000000000000000000000000008,9AC017840EEF96F7
00000000000000000000000000000010,B298FBD49DB32717
00000000000000000000000000000020,D66955F4398EDB17
.....
08000000000000000000000000000000,536286340868FD57
10000000000000000000000000000000,AB628634C868FD57
20000000000000000000000000000000,DB6286344868FD57
40000000000000000000000000000000,BB6286344868FD57
80000000000000000000000000000000,7B6286344868FD57
```

```
Key: CCB62876197C388009D0AFA10CF84A99 -> ciphertext: 8CA07222811CDE76
Key: 4CB62876997C388009D0AFA10CF84A99 -> ciphertext: 8CA07222811CDE76
Key: 995308BF2A5150434D90AD23002823AF -> ciphertext: 8CA07222811CDE76
Key: 973F78BF20F5804349100D230F2C1BAF -> ciphertext: 8CA07222811CDE76
Key: 195308BFAA5150434D90AD23002823AF -> ciphertext: 8CA07222811CDE76 …
```

```
Key: B3BF78BF0475804389100D230F2C1BAF -> ciphertext: 8CA07222811CDE76
Key: EC8D89E03BAB57168BD425290115EE0B -> ciphertext: 8CA07222811CDE76
Key: E942B3E0326A4D16855685290C1F5C0B -> ciphertext: 8CA07222811CDE76
```