Groupe de travail Réseau
Request for Comments : 2040
Catégorie : Information
Traduction Claude Brière de L'Isle

R. Baldwin, RSA Data Security, Inc.
R. Rivest, MIT Laboratory for Computer Science
et RSA Data Security, Inc.
octobre 1996

Algorithmes RC5, RC5-CBC, RC5-CBC-Pad, et RC5-CTS

Statut de ce mémoire
Le présent mémoire apporte des informations pour la communauté de l'Internet. Il ne spécifie aucune sorte de norme de l'Internet. La distribution du présent mémoire n'est soumise à aucune restriction.
(Cette traduction incorpore les errata de 2004 mentionnés concernant la Section 8)

Remerciements
Nous tenons à remercier Steve Dusse, Victor Chang, Tim Mathews, Brett Howard, et Burt Kaliski de leurs utiles suggestions.

Table des Matières

1. Résumé

Le présent document définit quatre chiffrements avec assez de détails pour assurer l'interopérabilité entre les différentes mises en œuvre. Le premier chiffrement est le chiffrement de bloc RC5 brut. Le chiffrement RC5 prend une taille fixe de bloc d'entrée et produit un bloc de sortie de taille fixe en utilisant une transformation qui dépend d'une clé. Le second chiffrement, RC5-CBC, est le mode de chaînage de bloc de chiffrement (CGC, *Cipher Block Chaining*) pour RC5. Il peut traiter des messages dont la longueur est un multiple de la taille de bloc RC5. Le troisième chiffrement, RC5-CBC-Pad,

traite du texte source de toute longueur, mais le texte chiffré sera plus long que le texte source d'au moins la taille d'un seul bloc RC5. Le chiffrement RC5-CTS est le mode de soustraction de texte chiffré de RC5, qui traite des textes source de toute longueur et la longueur du texte chiffré correspond à la longueur du texte source.

Le chiffrement RC5 a été inventé par le Professeur Ronald L. Rivest du Massachusetts Institute of Technology en 1994. C'est un algorithme très rapide et simple qui est paramétré par la taille de bloc, le nombre de tours, et la longueur de clé. Ces paramètres peuvent être ajustés pour satisfaire différents objectifs de sécurité, performances, et exportabilité.

RSA Data Security Incorporated a octroyé une licence d'application sur le chiffrement RC5 et pour la protection de la marque commerciale de RC5, RC5-CBC, RC5-CBC-Pad, RC5-CTS et leurs diverses variations.

2. Généralités

Le présent mémoire est une reproduction de matériaux existants déjà publiés. La description de RC5 suit la notation et l'ordre d'explication qui se trouvent dans l'article original sur RC5 du Professeur Rivest [2]. Le mode CBC apparaît en référence dans des travaux comme ceux de Bruce Schneier [6]. Le mode CBC-Pad est le même que dans la norme de chiffrement à clé publique (PKCS, *Public Key Cryptography Standard*) numéro cinq [5]. Un échantillon de code C [8] est inclus à de seules fins de clarté et est équivalent à la description en langue anglaises.

Les chiffrements seront expliqués en mode objet de façon descendante. D'abord les clés RC5 seront présentées avec l'algorithme d'expansion de clé. Ensuite, le chiffrement de bloc RC5 est expliqué, et finalement, les chiffrements RC5-CBC et RC5-CBC-Pad sont spécifiés. Pour faire court, seul le processus de chiffrement est décrit. Le déchiffrement est réalisé en inversant les étapes du chiffrement.

La description en mode objet qu'on trouve ici devrait faciliter le mise en œuvre interopérable des systèmes, bien qu'elle ne soit pas aussi concise que les descriptions fonctionnelles qu'on trouvera dans les références. Il y a deux classes d'objets, les clés et les algorithmes de chiffrement. Les deux classes partagent des opérations qui créent et suppriment ces objets d'une manière qui assure que des informations secrètes ne sont pas retournées au gestionnaire de la mémoire.

Les clés ont aussi une opération "set" qui copie une clé secrète en l'objet. L'opération "set" pour les objets de chiffrement définit le nombre de tours, et la valeur d'initialisation.

Il y a quatre opérations pour les objets de chiffrement décrits dans le présent mémoire. Il y a le lien d'une clé à un objet de chiffrement, établissant une nouvelle valeur d'initialisation (IV, *initialization vector*) pour un objet de chiffrement sans changer la clé, le chiffrement d'une partie d'un message (cela va être effectué de nombreuses fois pour les messages longs) et le traitement de la dernière partie d'un message qui peut ajouter du bourrage ou vérifier la longueur du message.

En résumé, le chiffrement va être expliqué dans les termes de ces opérations :
RC5_Key_Create – Crée un objet de clé.
RC5_Key_Destroy – Supprime un objet de clé.
RC5_Key_Set – Lie une clé d'utilisateur à un objet de clé.
RC5_CBC_Create – Crée un objet de chiffrement.
RC5_CBC_Destroy – Supprime un objet de chiffrement
RC5_CBC_Encrypt_Init – Lie un objet de clé à un objet de chiffrement.
RC5_CBC_SetIV – Établit une nouvelle IV sans changer la clé.
RC5_CBC_Encrypt_Update – Traite une partie d'un message.
RC5_CBC_Encrypt_Final – Traite la fin d'un message.

3. Terminologie et notation

Le terme "mot" se réfère à une chaîne de bits d'une certaine longueur qui peut être traitée comme un entier non signé ou une valeur binaire. Par exemple un "mot" peut être long de 32 ou 64 bits selon la taille de bloc désirée pour le chiffrement RC5. Un mot de 32 bits va produire une taille de bloc de 64 bits. Pour de meilleures performances, la taille du mot RC5 devrait correspondre à la taille de registre de la CPU. Le terme "octet" se réfère à huit bits.

Les variables ci-après seront utilisées dans le présent mémoire avec les significations suivantes :

W : c'est la taille de mot mesuré en bits pour RC5. C'est la moitié de la taille de bloc. Les tailles de mot couvertes par le présent mémoire sont 32 et 64.

WW : c'est la taille de mot mesuré en octets pour RC5.

B : c'est la taille de bloc mesurée en bits pour RC5. C'est deux fois la taille de mot. Lorsque RC5 est utilisé comme chiffrement de bloc de 64 bits, B fait 64 et W fait 32. 0 < B < 257. Dans l'échantillon de code, B est utilisé comme variable au lieu d'un paramètre de système de chiffrement, mais cet usage devrait être évident d'après le contexte.

BB : c'est la taille de bloc pour RC5 mesuré en octets. BB = B / 8.

b : c'est la longueur en octets de la clé secrète. $0 \leq b < 256$.

K : c'est la clé secrète qui est traitée comme une séquence de b octets indexés par : K[0], ..., K[b-1].

R : c'est le nombre de tours de la transformation RC5 interne. $0 \leq R < 256$.

T : c'est le nombre de mots dans le tableau de clés expansées. C'est toujours 2*(R + 1). $1 < T < 513$.

S : c'est le tableau de clés expansé qui est traité comme une séquence de mots indexés par : S[0], ..., S[T-1].

N :c'est la longueur en octets du message en texte source.

P : c'est le message en texte source qui est traité comme une séquence de N octets indexés par : P[0], ..., P[N-1].

C : c'est le résultat en texte chiffré qui est traité comme une séquence d'octets indexée par : C[0], C[1], ...

I : c'est la valeur d'initialisation pour le mode CBC, traitée comme une séquence d'octets indexée par : I[0], ..., I[BB-1].

4. Description des clés RC5

Comme la plupart des chiffrements de bloc, RC5 étend une petite clé d'utilisateur dans un tableau de clés internes. La longueur en octets de la clé d'utilisateur est un des paramètres du chiffrement, de sorte que l'objet clé d'utilisateur RC5 doit être capable de tenir des clés de longueur variable. Une structure possible en langage C pour cela est :

```
/* Définition de l'objet RC5 clé d'utilisateur. */
typedef struct rc5UserKey
  {
    int         keyLength;                  /* en octets. */
    unsigned char   *keyBytes;
  } rc5UserKey;
```

Les opérations de base sur une clé sont créer *(create)*, supprimer *(destroy)* et établir *(set)*. Pour éviter d'exposer le matériel de chiffrement aux autres parties d'une application, l'opération "destroy" remet à zéro la mémoire allouée pour la clé avant de la rendre au gestionnaire de mémoire. Un objet de clé général peut prendre en charge d'autres opérations comme de générer une nouvelle clé aléatoire et déduire une clé d'informations d'accord de clé.

4.1 Création d'une clé RC5

Pour créer une clé, la mémoire pour l'objet de clé doit être allouée et initialisée. Le code C ci-dessous suppose qu'une fonction appelée "malloc" va retourner un bloc de mémoire non initialisée à partir du tas, ou zéro, ce qui indique une erreur.

```
/* Allouer et initialiser une clé d'utilisateur RC5. Retourner 0 en cas de problème. */
  rc5UserKey *RC5_Key_Create ()
  {
    rc5UserKey *pKey;

    pKey = (rc5UserKey *) malloc (sizeof(*pKey));
    if (pKey != ((rc5UserKey *) 0))
    {
       pKey->keyLength = 0;
       pKey->keyBytes = (unsigned char *) 0;
    }
    return (pKey);
```

```
}
```

4.2 Suppression d'une clé RC5

Pour supprimer une clé, la mémoire doit être zéroïsée et livrée au gestionnaire de mémoire. Le code C ci-dessous suppose qu'une fonction appelée "free" va retourner un bloc de mémoire à partir du tas.

```
/* Mise à zéro et libération d'une clé d'utilisateur RC5.  */
 void RC5_Key_Destroy (pKey)
  rc5UserKey      *pKey;
 {
  unsigned char   *to;
  int         count;

  if (pKey == ((rc5UserKey *) 0))
     return;
  if (pKey->keyBytes == ((unsigned char *) 0))
     return;
  to = pKey->keyBytes;
  for (count = 0 ; count < pKey->keyLength ; count++)
     *to++ = (unsigned char) 0;
  free (pKey->keyBytes);
  pKey->keyBytes = (unsigned char *) 0;
  pKey->keyLength = 0;
  free (pKey);

 }
```

4.3 Établissement d'une clé RC5

L'établissement de l'objet de clé fait une copie de la clé secrète dans un bloc de mémoire alloué à partir du tas.

```
/* Régler la valeur d'une clé d'utilisateur RC5. Copier les octets de la clé afin que l'appelant puisse zéroïser et libérer
     l'original. Retourner zéro en cas de problème.  */
 int RC5_Key_Set (pKey, keyLength, keyBytes)
  rc5UserKey  *pKey;
  int         keyLength;
  unsigned char   *keyBytes;
 {
  unsigned char   *keyBytesCopy;
  unsigned char   *from, *to;
  int         count;

  keyBytesCopy = (unsigned char *) malloc (keyLength);
  if (keyBytesCopy == ((unsigned char *) 0))
     return (0);
  from = keyBytes;
  to = keyBytesCopy;
  for (count = 0 ; count < keyLength ; count++)
     *to++ = *from++;
  pKey->keyLength = count;
  pKey->keyBytes = keyBytesCopy;
  return (1);
 }
```

5. Description de l'expansion de clé RC5

Cette section décrit l'algorithme d'expansion de clé. Pour être précis, l'échantillon de code suppose que la taille de bloc est de 64 bits. Plusieurs paramètres de programmation dépendent de la taille de bloc.

```
/* Définitions pour RC5 comme chiffrement de bloc de 64 bits. Le "unsigned int" (entier non signé) sera 32 bits sur tous les
    compilateurs sauf les plus anciens, qui le feront de 16 bits. Sur un DEC Alpha "unsigned long" est 64 bits, et non 32. */
 #define RC5_WORD    unsigned int
 #define W           (32)
 #define WW          (W / 8)
 #define ROT_MASK    (W - 1)
#define BB          ((2 * W) / 8) /* Octets par bloc */
/* Définit les macros utilisées dans plusieurs procédures. Ces macros supposent que ">>" est une opération non signée, et
    que x et s sont du type RC5_WORD. */
 #define SHL(x,s)    ((RC5_WORD)((x)<<((s)&ROT_MASK)))
 #define SHR(x,s,w)  ((RC5_WORD)((x)>>((w)-((s)&ROT_MASK))))
 #define ROTL(x,s,w) ((RC5_WORD)(SHL((x),(s))|SHR((x),(s),(w))))
```

5.1 Définition des constantes d'initialisation

Deux constantes, Pw et Qw, sont définies pour toute taille de mot W par les expressions :

Pw = Odd((e-2)*2**W)

Qw = Odd((phi-1)*2**W)

où e est la base du logarithme naturel (2,71828 ...), et phi est le nombre d'or (1,61803 ...), et 2**W est 2 à la puissance W, et Odd(x) est égal à x si x est impair, ou égal à x plus un si x est pair. Pour W égal à 16, 32, et 64, les constantes Pw et Qw sont les valeurs hexadécimales suivantes :

```
#define P16  0xb7e1
#define Q16  0x9e37
#define P32  0xb7e15163
#define Q32  0x9e3779b9
#define P64  0xb7e151628aed2a6b
#define Q64  0x9e3779b97f4a7c15
#if W == 16
#define Pw   P16                    /* choisit la taille de mot de 16 bits */
#define Qw   Q16
#endif
#if W == 32
#define Pw   P32                    /* choisit la taille de mot de 32 bits */
#define Qw   Q32
#endif
#if W == 64
#define Pw   P64                    /* choisit la taille de mot de  64 bits */
#define Qw   Q64
#endif
```

5.2 Définition d'interface

Le sous programme d'expansion de clé convertit les b octets de la clé secrète, K, en une clé expansée, S, qui est une séquence de T = 2*(R+1) mots. L'algorithme d'expansion utilise deux constantes qui sont déduites des constantes, e, et phi. Elles sont utilisées pour initialiser S, qui est alors modifié en utilisant K. Un en-tête de procédure en code C pour ce sous programme pourrait être :

```
/* Expansion d'une clé d'utilisateur RC5. */
 void RC5_Key_Expand (b, K, R, S)
   int     b;                   /* longueur en octets de la clé secrète */
   char       *K;               /* clé secrète */
   int     R;                   /* nombre de tours */
   RC5_WORD *S;                 /* mémoire tampon de clé expansée, 2*(R+1) mots */
 {
```

5.3 Conversion de clé secrète d'octets en mots

Cette étape convertit la clé de b octets en une séquence de mots mémorisés dans le dispositif L. Sur un processeur petit boutien cela se fait en mettant à zéro le dispositif L et en y copiant les b octets de K. Le code C suivant va réaliser cet effet sur tous les processeurs :

```
   int i, j, k, LL, t, T;
   RC5_WORD   L[256/WW];              /* se fonde sur la taille de clé maximale */
   RC5_WORD   A, B;

/* LL est le nombre d'éléments utilisés dans L. */
   LL = (b + WW - 1) / WW;
   for (i = 0 ; i < LL ; i++)  {
      L[i] = 0;
   }
   for (i = 0 ; i < b ; i++)  {
      t = (K[i] & 0xFF) << (8*(i%4)); /* 0, 8, 16, 24*/
      L[i/WW] = L[i/WW] + t;
   }
```

5.4 Initialiser le tableau de clé expansée

Cette étape remplit le tableau S avec un schéma pseudo aléatoire fixe (indépendant de la clé) en utilisant une progression arithmétique fondée sur Pw et Qw modulo 2**W. L'élément S[i] est égal à i*Qw + Pw modulo 2**W. Ce tableau pourrait être pré calculé et copié en tant que de besoin ou calculé au vol. En code C, il peut être calculé par :

```
T = 2*(R+1);
S[0] = Pw;
for (i = 1 ; i < T ; i++)  {
   S[i] = S[i-1] + Qw;
}
```

5.5 Mixage dans la clé secrète

Cette étape mélange la clé secrète, K, dans la clé étendue, S. D'abord, le nombre d'itérations de la fonction de mixage, k, est réglé à trois fois le maximum du nombre d'éléments initialisés de L, appelé LL, et le nombre d'éléments dans S, appelé T. Chaque itération est similaire à une itération de la boucle de chiffrement interne en ce que ces deux variables A et B sont mises à jour par la première et la seconde moitié de l'itération.

Initialement, A et B sont à zéro comme le sont les indices dans le dispositif S, i, et le dispositif L, j. Dans la première moitié de l'itération, un résultat partiel est calculé en ajoutant S[i], A et B. La nouvelle valeur pour A est ce résultat partiel avec une rotation à gauche de trois bits. La valeur A est alors placée dans S[i]. La seconde moitié de l'itération calcule un second résultat partiel qui est la somme de L[j], A et B. Le second résultat partiel est alors tourné à gauche de A+B positions de bits et réglé à la nouvelle valeur de B. La nouvelle valeur de B est alors placée dans L[j]. À la fin de l'itération, i et j sont incrémentés modulo la taille de leurs dispositifs respectifs. En code C :

```
i = j = 0;
A = B = 0;
if (LL > T)
   k = 3 * LL;                     /* Longueur de clé secrète > clé expansée. */
else
   k = 3 * T;                      /* Longueur de clé secrète > clé expansée. */
for ( ; k > 0 ; k--)  {
   A = ROTL(S[i] + A + B, 3, W);    S[i] = A;
   B = ROTL(L[j] + A + B, A + B, W);
   L[j] = B;
   i = (i + 1) % T;
   j = (j + 1) % LL;
}
retour ;
}                                  /* Fin de RC5_Key_Expand */
```

6. Description du chiffrement de bloc RC5

Cette section décrit le chiffrement de bloc RC5 en expliquant les étapes requises pour effectuer le chiffrement d'un seul bloc d'entrée. Le processus de déchiffrement est l'inverse de ces étapes de sorte qu'il ne sera pas expliqué. Le chiffrement RC5 est paramétré par un numéro de version, V, un compte d'itérations, R, et une taille de mot en bits, W. Cette description correspond à la version d'origine de RC5 (V = 16 en décimal) et couvre toute valeur positive pour R et les valeurs 16, 32, et 64 pour W.

L'entrée à ce procès sont le tableau de clé expansée, S, le nombre de tours, R, le pointeur de mémoire tampon d'entrée, in, et le pointeur de mémoire tampon de sortie, out. Un en-tête possible de procédure en code C pour cela serait :

```
void RC5_Block_Encrypt (S, R, in, out)
 RC5_WORD   *S;
 int  R;
 char   *in;
 char   *out;
{
```

6.1 Chargement des valeurs A et B

Cette étape convertit les octets d'entrée en deux entiers non signés appelés A et B. Lorsque RC5 est utilisé comme un chiffrement de bloc de 64 bits, A et B sont des valeurs de 32 bits. Le premier octet d'entrée devient l'octet de moindre poids de A, le quatrième octet d'entrée devient l'octet de poids fort de A, le cinquième octet d'entrée devient l'octet de moindre poids de B et le dernier octet d'entrée devient l'octet de poids fort de B. Cette conversion peut être très efficace pour les processeurs petits boutiens comme ceux de la famille Intel. En code C, cela serait exprimé par :

```
 int  i;
 RC5_WORD   A, B;

A  =  in[0] & 0xFF;
A += (in[1] & 0xFF) << 8;
A += (in[2] & 0xFF) << 16;
A += (in[3] & 0xFF) << 24;
B  =  in[4] & 0xFF;
B += (in[5] & 0xFF) << 8;
B += (in[6] & 0xFF) << 16;
B += (in[7] & 0xFF) << 24;
```

6.2 Itération de la fonction d'itération

Cette étape mélange la clé expansée avec l'entrée pour effectuer l'opération fondamentale de chiffrement. Les deux premiers mots de la clé expansée sont ajoutés à A et B respectivement, et la fonction d'itération est répétée R fois.

La première moitié de la fonction d'itération calcule une nouvelle valeur pour A sur la base des valeurs de A, B, et du prochain mot non utilisé dans le tableau de clé expansée. Précisément, A est OUixé avec B et ensuite le premier résultat partiel subit une rotation à gauche d'une quantité spécifiée par B pour former le second résultat partiel. La rotation est effectuée sur une limite de bit W (c'est-à-dire, une rotation de 32 bit pour la version de RC5 qui a une taille de bloc de 64 bits). La quantité réelle de rotation dépend seulement des W log base-2 bits de moindre poids de B. Le prochain mot non utilisé du tableau de clé expansée est alors ajouté au second résultat partiel et cela devient la nouvelle valeur de A.

La seconde moitié de la fonction d'itération est identique, sauf que les rôles de A et B sont inversée. Précisément, B est et OUixé avec A et ensuite ce premier résultat partiel subit une rotation à gauche d'une quantité spécifiée par A pour former le second résultat partiel. Le prochain mot non utilisé du tableau de clé expansée est alors ajouté au second résultat partiel et cela devient la nouvelle valeur pour B.

Une façon d'exprimer cela en code C est :

```
A = A + S[0];
B = B + S[1];
for (i = 1 ; i <= R ; i++) {
   A = A ^ B;
```

```
    A = ROTL(A, B, W) + S[2*i];
    B = B ^ A;
    B = ROTL(B, A, W) + S[(2*i)+1];
}
```

6.3 Mémorisation des valeurs A et B

L'étape finale est de reconvertir A et B en une séquence d'octets. C'est l'inverse de l'opération de chargement. Une expression en code C pourrait être :

```
out[0] = (A >>  0) & 0xFF;
out[1] = (A >>  8) & 0xFF;
out[2] = (A >> 16) & 0xFF;
out[3] = (A >> 24) & 0xFF;
out[4] = (B >>  0) & 0xFF;
out[5] = (B >>  8) & 0xFF;
out[6] = (B >> 16) & 0xFF;
out[7] = (B >> 24) & 0xFF;
return;
} /* Fin de RC5_Block_Encrypt */
```

7. Description de RC5-CBC et de RC5-CBC-Pad

Cette section décrit les modes CBC et CBC-Pad du chiffrement RC5. Cette description se fonde sur les objets de clé RC5 et le chiffrement de bloc RC5 décrits précédemment.

7.1 Création des objets de chiffrement

L'objet de chiffrement doit garder trace du mode de bourrage, du nombre de tours, de la clé expansée, de la valeur d'initialisation, du bloc de chaîne CBC, et de la mémoire tampon d'entrée. Une définition de la structure possible pour cela en code C serait :

```
/* Définition de l'objet d'algorithme CBC RC5. */
typedef struct rc5CBCAlg
{
  int         Pad;                    /* 1 = RC5-CBC-Pad, 0 = RC5-CBC. */
  int         R;                      /* Nombre de tours. */
  RC5_WORD       *S;                  /* Clé expansée. */
  unsigned char   I[BB];              /* Valeur d'initialisation. */
  unsigned char   chainBlock[BB];
  unsigned char   inputBlock[BB];
  int         inputBlockIndex;        /* Prochain octet de bloc d'entrée. */
} rc5CBCAlg;
```

Pour créer un objet d'algorithme de chiffrement, les paramètres doivent être vérifiés et ensuite de l'espace doit être alloué pour le tableau de clé expansée. La clé expansée est initialisée en utilisant la méthode décrite précédemment. Finalement, les variables d'état (mode de bourrage, nombre de tours, et la mémoire tampon d'entrée) sont réglés à leur valeur initiale. En code C, ceci pourrait être réalisé avec :

```
/* Allouer et initialiser l'objet d'algorithme CBC RC5. Retourne 0 si il y a des problèmes. */
rc5CBCAlg *RC5_CBC_Create (Pad, R, Version, bb, I)
  int     Pad;                    /* 1 = RC5-CBC-Pad, 0 = RC5-CBC. */
  int     R;                      /* Nombre de tours. */
  int     Version;                /* Numéro de version RC5. */
  int     bb;                     /* Octets par bloc RC5 == longueur de l'IV. */
  char     *I;                    /* IV CBC, long de bb octets. */
{

  rc5CBCAlg    *pAlg;
  int          index;
```

```
  if ((Version != RC5_FIRST_VERSION) ||
     (bb != BB) ||   (R < 0) || (255 < R))
     return ((rc5CBCAlg *) 0);
  pAlg = (rc5CBCAlg *) malloc (sizeof(*pAlg));
  if (pAlg == ((rc5CBCAlg *) 0))
     return ((rc5CBCAlg *) 0);
  pAlg->S = (RC5_WORD *) malloc (BB * (R + 1));
  if (pAlg->S == ((RC5_WORD *) 0))    {
     free (pAlg);
     return ((rc5CBCAlg *) 0);
  }
  pAlg->Pad = Pad;
  pAlg->R = R;
  pAlg->inputBlockIndex = 0;
  for (index = 0 ; index < BB ; index++)
     pAlg->I[index] = I[index];
  return (pAlg);
}
```

7.2 Destruction des objets de chiffrement

Détruire les objets de chiffrement est l'inverse de les créer en prenant soin d'avoir zéro mémoire avant de les retourner au gestionnaire de mémoire. En C cela pourrait être accompli par :

```
/* Zéroisation et libération de l'objet d'algorithme RC5. */
void RC5_CBC_Destroy (pAlg)
  rc5CBCAlg   *pAlg;
{
  RC5_WORD    *to;
  int      count;

  if (pAlg == ((rc5CBCAlg *) 0))
     return;
  if (pAlg->S == ((RC5_WORD *) 0))
     return;
  to = pAlg->S;
  for (count = 0 ; count < (1 + pAlg->R) ; count++)
  {
     *to++ = 0;                              /* Deux mots de clé expansée par tour. */
     *to++ = 0;
  }
 free (pAlg->S);
  for (count = 0 ; count < BB ; count++)

  {
    pAlg->I[count] = (unsigned char) 0;
     pAlg->inputBlock[count] = (unsigned char) 0;
     pAlg->chainBlock[count] = (unsigned char) 0;
  }
  pAlg->Pad = 0;
  pAlg->R = 0;
  pAlg->inputBlockIndex = 0;
  free (pAlg);
}
```

7.3 Réglage de l'IV pour les objets de chiffrement

Pour les objets de chiffrement CBC, l'état de l'algorithme dépend de la clé expansée, du bloc de chaîne CBC, et de toutes les entrées mises en antémémoire en interne. Souvent, la même clé est utilisée avec de nombreux messages qui ont chacun une valeur d'initialisation unique. Pour éviter la surcharge d'avoir à créer un nouvel objet de chiffrement, il y a du sens à

fournir une opération qui permet à l'appelant de changer la valeur d'initialisation pour un objet de chiffrement existant. En C, ceci pourrait être réalisé par le code suivant :

```
/* Établir une nouvelle valeur d'initialisation pour une opération CBC et rétablir l'objet CBC. Ceci peut être invoqué après
    l'opération finale sans avoir besoin d'invoquer à nouveau Init ou Create. Retourner zéro si il y a des problèmes. */
int RC5_CBC_SetIV (pAlg, I)
 rc5CBCAlg   *pAlg;
 char        *I;                         /* Valeur d'initialisation CBC, BB octets. */
{
 int     index;

 pAlg->inputBlockIndex = 0;
 for (index = 0 ; index < BB ; index++)
 {
    pAlg->I[index] = pAlg->chainBlock[index] = I[index];
    pAlg->inputBlock[index] = (unsigned char) 0;
 }
 return (1);
}
```

7.4 Lier une clé à un objet de chiffrement

L'opération qui lie une clé à un objet de chiffrement effectue une expansion de clé. L'expansion de clé pourrait être une opération sur les clés, mais cela ne fonctionnerait pas correctement pour les chiffrements qui modifient la clé expansée lorsque ils opèrent. Après l'expansion de la clé, cette opération doit initialiser le bloc de chaîne CBC à partir de la valeur d'initialisation et préparer la mémoire tampon d'entrée à recevoir le premier caractère. En C, ceci pourrait être fait par :

```
/* Initialiser l'objet de chiffrement avec la clé donnée. Après ce sous programme, l'appelant libère l'objet de clé. L'IV pour
    cet objet CBC peut être changée en invoquant le sous programme SetIV. La seule façon de changer la clé est de détruire
    l'objet CBC et d'en créer un nouveau. Retourner zéro si il y a des problèmes. */
int RC5_CBC_Encrypt_Init (pAlg, pKey)
 rc5CBCAlg      *pAlg;
 rc5UserKey  *pKey;
{
 if ((pAlg == ((rc5CBCAlg *) 0)) ||
    (pKey == ((rc5UserKey *) 0)))
    return (0);
 RC5_Key_Expand (Key->keyLength, pKey->keyBytes, pAlg->R, pAlg->S);
 return (RC5_CBC_SetIV(pAlg, pAlg->I));
}
```

7.5 Traitement d'une partie d'un message

Le processus de chiffrement décrit ici utilise le paradigme Init-Update-Final. L'opération de mise à jour peut être effectuée sur une séquence de parties de message afin de produire le texte chiffré de façon incrémentaire. Après le traitement de la dernière partie, l'opération finale est invoquée pour rassembler tous les octets du texte source ou pour bourrer ceux qui sont en mémoire tampon à l'intérieur de l'objet de chiffrement. Un en-tête de procédure approprié pour cette opération serait :

```
/* Chiffrer une mémoire tampon du texte source. Le texte source et la mémoire tampon de texte chiffré peuvent être les
    mêmes. La longueur en octets du texte chiffré est mise dans *pCipherLen. Invoquer cela plusieurs fois en passant les
    parties successives d'un grand message. Après que la dernière partie est passée à Update, invoquer Final. Retourner zéro
    si il y a des problèmes comme une mémoire tampon de sortie trop petite. */
int RC5_CBC_Encrypt_Update (pAlg, N, P, pCipherLen, maxCipherLen, C)
 rc5CBCAlg   *pAlg;                      /* Objet algorithme de chiffrement. */
 int         N;                          /* Longueur en octets de P. */
 char        *P;                         /* Mémoire tampon du texte source. */
 int         *pCipherLen;                /* Obtient la longueur en octets de C. */
 int         maxCipherLen;               /* Taille de C. */
 char        *C;                         /* Mémoire tampon du texte chiffré. */
{
```

7.5.1 Vérification de la taille de mémoire tampon de sortie

La première étape du traitement du texte source est de s'assurer que la mémoire tampon de sortie est assez grande pour contenir le texte chiffré. Le texte chiffré sera produit en multiples de la taille de bloc et dépend du nombre de caractères du texte source passés à cette opération plus tous les caractères qui sont dans la mémoire tampon interne de l'objet de chiffrement. En code C, ce serait :

```
int     plainIndex, cipherIndex, j;
                                        /* vérifie la taille de la mémoire tampon de sortie. */
 if (maxCipherLen < (((pAlg->inputBlockIndex+N)/BB)*BB))
{
   *pCipherLen = 0;
   return (0);
}
```

7.5.2 Division du texte source en blocs

L'étape suivante est d'ajouter des caractères à la mémoire tampon interne jusqu'à ce qu'un bloc complet ait été construit. Lorsque cela arrive, les pointeurs de la mémoire tampon sont remis à zéro et la mémoire tampon d'entrée est OUixée avec le bloc de chaînage CBC. L'ordre des octets du bloc de chaînage est le même que celui du bloc d'entrée. Par exemple, le neuvième octet d'entrée est OUixé avec le premier octet du texte chiffré. Le résultat est alors passé au chiffrement de bloc RC5 qui a été décrit précédemment. Pour réduire les problèmes de mouvement des données et d'alignement d'octets, le résultat de RC5 peut être écrit directement dans le bloc de chaînage CBC. Finalement, ce résultat est copié dans la mémoire tampon de texte chiffré fournie par l'utilisateur. Avant de retourner, la taille réelle du texte chiffré est repassée à l'appelant. En C, cette étape peut être effectuée par :

```
plainIndex = cipherIndex = 0;
 while (plainIndex < N)
 {
    if (pAlg->inputBlockIndex < BB)
    {
       pAlg->inputBlock[pAlg->inputBlockIndex]
             = P[plainIndex];
       pAlg->inputBlockIndex++;
       plainIndex++;
    }

    if (pAlg->inputBlockIndex == BB)
    {                                   /* Quand on a un bloc d'entré complet, on le traite. */
       pAlg->inputBlockIndex = 0;
       for (j = 0 ; j < BB ; j++)
       {                                /* OUixer dans le bloc de chaîne. */
          pAlg->inputBlock[j] = pAlg->inputBlock[j] ^ pAlg->chainBlock[j];
         }
         RC5_Block_Encrypt(pAlg->S, pAlg->R
                     pAlg->inputBlock,
                     pAlg->chainBlock);
         for (j = 0 ; j < BB ; j++)
         {                              /* Sortir le texte chiffré. */
            C[cipherIndex] = pAlg->chainBlock[j];
            cipherIndex++;
         }
       }
    }
    *pCipherLen = cipherIndex;
    return (1);
  } /* Fin de RC5_CBC_Encrypt_Update */
```

7.6 Traitement du bloc final

Cette étape traite le dernier bloc de texte source. Pour RC5-CBC, cette étape effectue juste la vérification d'erreurs pour s'assurer que la longueur du texte source est bien un multiple de la longueur de bloc. Pour RC5-CBC-Pad, les octets de

bourrage sont ajoutés au texte source. Les octets de bourrage sont tous les mêmes et sont réglés à un octet qui représente le nombre d'octets de bourrage. Par exemple, si il y a huit octets de bourrage, les octets auront tous la valeur hexadécimale de 0x08. Il y aura entre un et BB octets de bourrage, inclus. En code C, ce serait :

```
/* Produire le bloc final de texte chiffré incluant tout bourrage, et ensuite rétablir l'objet d'algorithme. Retourner zéro si il y
     a des problèmes. */
int RC5_CBC_Encrypt_Final (pAlg, pCipherLen, maxCipherLen, C)
  rc5CBCAlg   *pAlg;
  int         *pCipherLen;                  /* Obtient la longueur en octets de C. */
  int          maxCipherLen;                /* Longueur de la mémoire tampon de C. */
  char        *C;                           /* Mémoire tampon du texte chiffré. */
{
  int     cipherIndex, j;
  int     padLength;

/* Pour une erreur de mode non bourrage si des octets d'entrée sont mis en mémoire tampon. */
  *pCipherLen = 0;

  if ((pAlg->Pad == 0) && (pAlg->inputBlockIndex != 0))
     return (0);

  if (pAlg->Pad == 0)
    return (1);
  if (maxCipherLen < BB)
    return (0);

  padLength = BB - pAlg->inputBlockIndex;
  for (j = 0 ; j < padLength ; j++)
  {
     pAlg->inputBlock[pAlg->inputBlockIndex]
          = (unsigned char) padLength;
     pAlg->inputBlockIndex++;
  }
  for (j = 0 ; j < BB ; j++)
  {                                         /* OUixe le bloc de chaîne en bloc de texte source. */
     pAlg->inputBlock[j] = pAlg->inputBlock[j]
                     ^ pAlg->chainBlock[j];
  }
  RC5_Block_Encrypt(pAlg->S, pAlg->R,
                pAlg->inputBlock, pAlg->chainBlock);
  cipherIndex = 0;
  for (j = 0 ; j < BB ; j++)
  {                                         /* Sort le texte chiffré. */
     C[cipherIndex] = pAlg->chainBlock[j];
     cipherIndex++;
  }
  *pCipherLen = cipherIndex;

                                            /* Rétablir l'objet d'algorithme CBC. */
  return (RC5_CBC_SetIV(pAlg, pAlg->I));
} /* Fin de RC5_CBC_Encrypt_Final */
```

8. Description de RC5-CTS

Le mode soustraction du texte chiffré (CTS, *Cipher Text Stealing*) pour les chiffrements de bloc est décrit par Schneier aux pages 195 et 196 de [6]. Ce mode traite toute longueur de texte source supérieure à un bloc et produit un texte chiffré dont la longueur correspond à la longueur du texte source. Le mode CTS se comporte comme le mode CBC en tout, sauf pour les deux derniers blocs de texte source. Les étapes suivantes décrivent comment traiter les deux dernières portions du texte source, appelées Pn-1 et Pn, où la longueur de Pn-1 est égale à la taille de bloc, BB, et la longueur du dernier bloc, Pn, est de Ln octets. On remarque que Ln est compris entre 1 et BB, inclus, de sorte que Pn pourrait en fait être un bloc complet.

1. OUixer Pn-1 avec le bloc de texte chiffré précédent, Cn-2, pour créer Xn-1. Pour les messages courts où Cn-2 n'existe pas, utiliser l'IV.
2. Chiffrer Xn-1 pour créer En-1.
3. Choisir les Ln premiers octets de En-1 pour créer Cn.
4. Bourrer de zéros à la fin pour créer P de longueur BB.
5. OUixer En-1 avec P pour créer Dn.
6. Chiffrer Dn pour créer Cn-1.
7. Les deux dernières parties du texte chiffré sont respectivement Cn-1 et Cn.

Pour mettre en œuvre le chiffrement CTS, l'objet RC5-CTS doit contenir au plus (en antémémoire) 2*BB octets de texte source et les traiter spécialement lorsque le sous programme RC5_CTS_Encrypt_Final est invoqué.

Les étapes suivantes décrivent comment déchiffrer Cn-1 et Cn.
1. Déchiffrer Cn-1 pour créer Dn.
2. Bourrer Cn de zéros en fin pour créer C de longueur BB.
3. OUixer Dn avec C pour créer Xn.
4. Choisir les Ln premiers octets de Xn pour créer Pn.
5. Ajouter la queue (BB moins Ln) des octets de Xn à Cn pour créer En.
6. Déchiffrer En et OUixer avec Cn-2 pour créer Pn-1. Pour les messages courts où Cn-2 n'existe pas, utiliser l'IV.
7. Les deux dernières parties du texte source sont respectivement Pn-1 et Pn.

9. Programme et valeurs d'essai.

Pour aider à confirmer la justesse d'une mise en œuvre, cette section donne un programme d'essais et les résultats d'un ensemble de valeurs d'essai.

9.1 Programme d'essai

Le programme d'essai suivant écrit en C lit les valeurs d'essai à partir de son flux d'entrée et écrit les résultats sur son flux de sortie. Les paragraphes qui suivent donnent un ensemble de valeurs d'essai pour les entrées et les sorties résultantes.

```
#include <stdio.h>

 #define BLOCK_LENGTH     (8 /* octest */)
 #define MAX_KEY_LENGTH   (64 /* octets */)
 #define MAX_PLAIN_LENGTH (128 /* octets */)
 #define MAX_CIPHER_LENGTH(MAX_PLAIN_LENGTH + BLOCK_LENGTH)
 #define MAX_ROUNDS       (20)
 #define MAX_S_LENGTH     (2 * (MAX_ROUNDS + 1))

 typedef struct test_vector
 {
  int padding_mode;
  int rounds;
  char    keytext[2*MAX_KEY_LENGTH+1];
  int key_length;
  char    key[MAX_KEY_LENGTH];
  char    ivtext[2*BLOCK_LENGTH+1];
  int iv_length;
  char    iv[BLOCK_LENGTH];
  char    plaintext[2*MAX_PLAIN_LENGTH+1];
  int plain_length;
  char    plain[MAX_PLAIN_LENGTH];
  char    ciphertext[2*MAX_CIPHER_LENGTH+1];
  int cipher_length;
  char    cipher[MAX_CIPHER_LENGTH];
  RC5_WORD    S[MAX_S_LENGTH];
 } test_vector;

 void show_banner()
 {
```

```
    (void) printf("RC5 CBC Tester.\n");
    (void) printf("Chaque ligne d'entrée devrait contenir les \n");
    (void) printf("paramètres d'essai suivants séparés par une seule espace:\n");
    (void) printf("- Fanion de mode bourrage. Utiliser 1 pour RC5_CBC_Pad, autrement 0.\n");
    (void) printf("- Nombre de tours pour RC5.\n");
    (void) printf("- Octets de clé en hexadécimal. Deux caractères par octet comme '01'.\n");
    (void) printf("- Octets d'IV en hexadécimal. Doit être de 16 caractères hex.\n");
    (void) printf("- Octets du texte source en hexadécimal.\n");
    (void) printf("Une fin de fichier ou une erreur de format termine le testeur.\n");
    (void) printf("\n");
  }

/* Convertit une mémoire tampon de ascii hex en octets. Régler pTo_length à la longueur en octets du résultat. Retourner 1
     si tout s'est bien passé. */
int hex_to_bytes (from, to, pTo_length)
  char    *from, *to;
  int     *pTo_length;
{
  char    *pHex;                        /* Pointeur sur le prochain caractère hex. */
  char    *pByte;                       /* Pointeur sur le prochain octet résultant. */
  int  byte_length = 0;
  int  value;

  pByte = to;
 for (pHex = from ; *pHex != 0 ; pHex += 2)  {
     if (1 != sscanf(pHex, "%02x", &value))
         return (0);
     *pByte++ = ((char)(value & 0xFF));
     byte_length++;
  }
  *pTo_length = byte_length;
  return (1);
}

/* Convertit une mémoire tampon d'octets en ascii hex. Retourne 1 si tout s'est bien passé. */
  int bytes_to_hex (from, from_length, to)
    char    *from, *to;
    int from_length;
  {
    char    *pHex;                      /* Pointeur sur le prochain caractère hex.. */
    char    *pByte;                     /* Pointeur sur le prochain octet résultant. */
    int  value;

    pHex = to;
    for (pByte = from ; from_length > 0 ; from_length--)  {
       value = *pByte++ & 0xFF;
       (void) sprintf(pHex, "%02x", value);
       pHex += 2;
    }
    return (1);
  }

/* Retourne 1 si on obtient une valeur d'essai valide. */
  int get_test_vector(ptv)
    test_vector *ptv;
  {

    if (1 != scanf("%d", &ptv->padding_mode))
       return (0);
    if (1 != scanf("%d", &ptv->rounds))
       return (0);
    if ((ptv->rounds < 0) || (MAX_ROUNDS < ptv->rounds))
       return (0);
```

```
  if (1 != scanf("%s", &ptv->keytext))
     return (0);
  if (1 != hex_to_bytes(ptv->keytext, ptv->key, &ptv->key_length))
     return (0);
  if (1 != scanf("%s", &ptv->ivtext))
     return (0);
  if (1 != hex_to_bytes(ptv->ivtext, ptv->iv, &ptv->iv_length))
     return (0);
  if (BLOCK_LENGTH != ptv->iv_length)
     return (0);    if (1 != scanf("%s", &ptv->plaintext))
     return (0);
  if (1 != hex_to_bytes(ptv->plaintext, ptv->plain, &ptv->plain_length))
     return (0);
  return (1);
}

void run_test (ptv)
  test_vector *ptv;
{
  rc5UserKey  *pKey;
  rc5CBCAlg       *pAlg;
  int          numBytesOut;

  pKey = RC5_Key_Create ();
  RC5_Key_Set (pKey, ptv->key_length, ptv->key);

  pAlg = RC5_CBC_Create (ptv->padding_mode,
               ptv->rounds,
               RC5_FIRST_VERSION,
               BB,
               ptv->iv);
  (void) RC5_CBC_Encrypt_Init (pAlg, pKey);
  ptv->cipher_length = 0;
  (void) RC5_CBC_Encrypt_Update (pAlg,
               ptv->plain_length, ptv->plain, &(numBytesOut),
               MAX_CIPHER_LENGTH - ptv->cipher_length,
               &(ptv->cipher[ptv->cipher_length]));

  ptv->cipher_length += numBytesOut;
  (void) RC5_CBC_Encrypt_Final (pAlg, &(numBytesOut),
               MAX_CIPHER_LENGTH - ptv->cipher_length,
               &(ptv->cipher[ptv->cipher_length]));
  ptv->cipher_length += numBytesOut;
  bytes_to_hex (ptv->cipher, ptv->cipher_length, ptv->ciphertext);
  RC5_Key_Destroy (pKey);
  RC5_CBC_Destroy (pAlg);
}

void show_results (ptv)
  test_vector *ptv;
{
  if (ptv->padding_mode)
     printf ("RC5_CBC_Pad ");
  else
     printf ("RC5_CBC     ");
  printf ("R = %2d ", ptv->rounds);
  printf ("Key = %s ", ptv->keytext);
  printf ("IV = %s ", ptv->ivtext);
  printf ("P = %s ", ptv->plaintext);
  printf ("C = %s", ptv->ciphertext);
  printf ("\n");
}
```

```
int main(argc, argv)
  int argc;
  char *argv[];
{
  test_vector tv;
  test_vector *ptv = &tv;

  show_banner();
  while (get_test_vector(ptv))  {
     run_test(ptv);
     show_results(ptv);
  }
  return (0);
}
```

9.2 Valeurs d'essai

Le texte qui suit est un fichier d'entrée du programme d'essai présenté au paragraphe précédent. Le résultat est donné au paragaphe qui suit.

```
0 00 00             0000000000000000 0000000000000000
0 00 00             0000000000000000 ffffffffffffffff
0 00 00             0000000000000001 0000000000000000
0 00 00             0000000000000000 0000000000000001
0 00 00             0102030405060708 1020304050607080
0 01 11             0000000000000000 0000000000000000
0 02 00             0000000000000000 0000000000000000
0 02 00000000          0000000000000000 0000000000000000
0 08 00             0000000000000000 0000000000000000
0 08 00             0102030405060708 1020304050607080
0 12 00             0102030405060708 1020304050607080
0 16 00             0102030405060708 1020304050607080
0 08 01020304          0000000000000000 ffffffffffffffff
0 12 01020304          0000000000000000 ffffffffffffffff
0 16 01020304          0000000000000000 ffffffffffffffff
0 12 0102030405060708   0000000000000000 ffffffffffffffff
0 08 0102030405060708   0102030405060708 1020304050607080
0 12 0102030405060708   0102030405060708 1020304050607080
0 16 0102030405060708   0102030405060708 1020304050607080
0 08 0102030405060708102030405060708 0   0102030405060708 1020304050607080
0 12 0102030405060708102030405060708 0   0102030405060708 1020304050607080
0 16 0102030405060708102030405060708 0   0102030405060708 1020304050607080

0 12 0102030405         0000000000000000 ffffffffffffffff
0 08 0102030405         0000000000000000 ffffffffffffffff
0 08 0102030405         7875dbf6738c6478 0808080808080808
1 08 0102030405         0000000000000000 ffffffffffffffff

0 08 0102030405         0000000000000000 0000000000000000
0 08 0102030405         7cb3f1df34f94811 1122334455667701

1 08 0102030405         0000000000000000 ffffffffffffffff7875dbf6738c647811223344556677
```

9.3 Résultats d'essai

Le texte qui suit est le résultat produit par le programme d'essai qui fonctionne sur les entrées données au paragraphe précédent.

```
Testeur C5 CBC.
Chaque ligne d'entrée devrait contenir les paramètres d'essai suivants séparés par une seule espace :
- Fanion de mode bourrage. Utiliser 1 pour RC5_CBC_Pad, autrement 0.
```

- Nombre de tours pour RC5.
- Octets de clé en hexadécimal. Deux caractères par octet comme '01'.
- Octets d'IV en hexadécimal. Doit être de 16 caractères hexadécimaux.
- Octets de texte source en hexadécimal.

Une fin de fichier ou une erreur de format termine le testeur.

RC5_CBC R = 0 Key = 00 IV = 0000000000000000
P = 0000000000000000 C = 7a7bba4d79111d1e
RC5_CBC R = 0 Key = 00 IV = 0000000000000000
P = ffffffffffffffff C = 797bba4d78111d1e
RC5_CBC R = 0 Key = 00 IV = 0000000000000001
P = 0000000000000000 C = 7a7bba4d79111d1f
RC5_CBC R = 0 Key = 00 IV = 0000000000000000
P = 0000000000000001 C = 7a7bba4d79111d1f
RC5_CBC R = 0 Key = 00 IV = 0102030405060708
P = 1020304050607080 C = 8b9ded91ce7794a6
RC5_CBC R = 1 Key = 11 IV = 0000000000000000
P = 0000000000000000 C = 2f759fe7ad86a378
RC5_CBC R = 2 Key = 00 IV = 0000000000000000
P = 0000000000000000 C = dca2694bf40e0788
RC5_CBC R = 2 Key = 00000000 IV = 0000000000000000
P = 0000000000000000 C = dca2694bf40e0788
RC5_CBC R = 8 Key = 00 IV = 0000000000000000
P = 0000000000000000 C = dcfe098577eca5ff
RC5_CBC R = 8 Key = 00 IV = 0102030405060708
P = 1020304050607080 C = 9646fb77638f9ca8
RC5_CBC R = 12 Key = 00 IV = 0102030405060708
P = 1020304050607080 C = b2b3209db6594da4
RC5_CBC R = 16 Key = 00 IV = 0102030405060708
P = 1020304050607080 C = 545f7f32a5fc3836
RC5_CBC R = 8 Key = 01020304 IV = 0000000000000000
P = ffffffffffffffff C = 8285e7c1b5bc7402
RC5_CBC R = 12 Key = 01020304 IV = 0000000000000000
P = ffffffffffffffff C = fc586f92f7080934
RC5_CBC R = 16 Key = 01020304 IV = 0000000000000000
P = ffffffffffffffff C = cf270ef9717ff7c4
RC5_CBC R = 12 Key = 0102030405060708 IV = 0000000000000000
P = ffffffffffffffff C = e493f1c1bb4d6e8c
RC5_CBC R = 8 Key = 0102030405060708 IV = 0102030405060708
P = 1020304050607080 C = 5c4c041e0f217ac3
RC5_CBC R = 12 Key = 0102030405060708 IV = 0102030405060708
P = 1020304050607080 C = 921f12485373b4f7
RC5_CBC R = 16 Key = 0102030405060708 IV = 0102030405060708
P = 1020304050607080 C = 5ba0ca6bbe7f5fad
RC5_CBC R = 8 Key = 0102030405060708102030405060708 0
IV = 0102030405060708
P = 1020304050607080 C = c533771cd0110e63
RC5_CBC R = 12 Key = 0102030405060708102030405060708 0
IV = 0102030405060708
P = 1020304050607080 C = 294ddb46b3278d60
RC5_CBC R = 16 Key = 0102030405060708102030405060708 0
IV = 0102030405060708
P = 1020304050607080 C = dad6bda9dfe8f7e8
RC5_CBC R = 12 Key = 0102030405 IV = 0000000000000000
P = ffffffffffffffff C = 97e0787837ed317f
RC5_CBC R = 8 Key = 0102030405 IV = 0000000000000000
P = ffffffffffffffff C = 7875dbf6738c6478
RC5_CBC R = 8 Key = 0102030405 IV = 7875dbf6738c6478
P = 0808080808080808 C = 8f34c3c681c99695
RC5_CBC_Pad R = 8 Key = 0102030405 IV = 0000000000000000
P = ffffffffffffffff C = 7875dbf6738c64788f34c3c681c99695
RC5_CBC R = 8 Key = 0102030405 IV = 0000000000000000
P = 0000000000000000 C = 7cb3f1df34f94811

RC5_CBC R = 8 Key = 0102030405 IV = 7cb3f1df34f94811
P = 1122334455667701 C = 7fd1a023a5bba217
RC5_CBC_Pad R = 8 Key = 0102030405 IV = 0000000000000000
P = ffffffffffffffff7875dbf6738c647811223344556677
C = 7875dbf6738c64787cb3f1df34f948117fd1a023a5bba217

10. Considérations sur la sécurité

Le chiffrement RC5 est relativement nouveau de sorte que des révisions critiques sont encore effectuées. Cependant, la structure simple du chiffrement rend facile l'analyse et donc plus facile d'assurer sa force. Les révisions ont été assez prometteuses jusqu'à présent.

Les premiers résultats [1] suggèrent que pour RC5 avec une taille de bloc de 64 bits (mots de 32 bits) 12 tours vont suffire pour résister aux cryptanalyses linéaires et différentielles. La version en bloc de 128 bits n'a pas encore été étudiée autant que la version à 64 bits, mais il apparaît que 16 tours serait un minimum approprié. Les tailles de bloc de moins de 64 bits sont d'un intérêt académique mais ne devraient pas être utilisées pour la sécurité cryptographique. Une sécurité supérieure peut être réalisée en augmentant le nombre de tours au prix d'une diminution du débit du chiffrement.

La longueur de la clé secrète aide à déterminer la résistance du chiffrement aux attaques de recherche de clé en force brute. Une longueur de clé de 128 bits devrait donner une protection adéquate contre les recherches de clé en force brute par un attaquant disposant de moyens pour plusieurs décennies [7]. Pour RC5 avec 12 tours, le temps d'établissement de la clé et le temps de chiffrement des données sont les mêmes que pour toutes les longueurs de clé de moins de 832 bits, de sorte qu'il n'y a pas de raisons liées aux performances pour choisir des clés courtes. Pour de plus grandes clés, l'étape d'expansion de clé va fonctionner plus lentement à cause du tableau de clé d' usager, L, qui sera plus longue que le tableau de clé expansée, S. Cependant, le temps de chiffrement sera inchangé car c'est seulement une fonction du nombre de tours.

Pour se conformer aux réglementations sur l'exportation, il peut être nécessaire de choisir des clés qui n'ont que 40 bits inconnus. Une mauvaise façon de le faire serait de choisir une simple clé de 5 octets. Ceci devrait être évité parce qu'il serait aisé pour un attaquant de pré calculer les informations de recherche de clé. Une autre mécanisme courant est de prendre une clé de 128 bits et de publier les 88 premiers bits. Cette méthode révèle un grand nombre des entrées dans le tableau de clé de l'usager, L, et la question de savoir si l'expansion de clé RC5 fournit une sécurité adéquate dans cette situation n'a pas été étudiée, bien que cela puisse être acceptable. Une façon prudente de se conformer à cette limitation à 40 bits est de prendre une valeur de germe de 128 bits, de publier 88 bits de ce germe, de faire tourner le germe entier à travers une fonction de hachage comme MD5 [4], et d'utiliser le résultat de 128 bit de la fonction de hachage comme clé RC5.

Dans le cas d'une clé de 40 bits inconnus avec 88 bits de clé connus (c'est-à-dire, 88 bits de sel) il devrait toujours y avoir 12 tours ou plus pour la version de bloc de 64 bits de RC5, autrement, la valeur de l'ajout des bits de sel à la clé sera probablement perdue.

La durée de vie de la clé influence aussi la sécurité. Pour des applications de haute sécurité, la clé pour tout bloc de 64 bits devrait être changée après le chiffrement de 2**32 blocs (2**64 blocs pour un chiffrement de bloc de 128 bits). Cela aide à se garder contre les cryptanalyses linéaires et différentielles. Pou le cas de blocs de 64 bits, cette règle recommanderait de changer la clé après le chiffrement de 2**40 (c'est-à-dire 10**12) octets. Voir Schneier [6] page 183 pour une discussion plus approfondie.

11. Identifiants ASN.1

Pour les applications qui utilisent des descriptions ASN.1, il est nécessaire de définir l'identifiant d'algorithme pour ces chiffrements ainsi que pour leurs formats de bloc de paramètres. La définition ASN.1 d'un identifiant d'algorithme existe déjà et est donnée ci-dessous pour référence.

```
AlgorithmIdentifier ::= SEQUENCE {
  algorithm    OBJECT IDENTIFIER,
  parameters   ANY DEFINED BY algorithm OPTIONAL
}
```

Les valeurs pour le champ Algorithme sont :

RC5_CBC OBJECT IDENTIFIER ::= { iso (1) member-body (2) US (840) rsadsi (113549) encryptionAlgorithm (3)

RC5CBC (8) }

RC5_CBC_Pad OBJECT IDENTIFIER ::= { iso (1) member-body (2) US (840) rsadsi (113549) encryptionAlgorithm (3) RC5CBCPAD (9) }

La structure du champ Paramètres pour ces algorithmes est donnée ci-dessous.
Note : si le champ iv n'est pas inclus, la valeur par défaut de la valeur d'initialisation est un bloc de zéros dont la taille dépend du champ blockSizeInBits.

```
C5_CBC_Parameters ::= SEQUENCE {
 version          INTEGER (v1_0(16)),
 rounds           INTEGER (8..127),
 blockSizeInBits   INTEGER (64, 128),
 iv               OCTET STRING OPTIONAL
}
```

Références

[1] Kaliski, Burton S., and Yinqun Lisa Yin, "On Differential and Linear Cryptanalysis of the RC5 Encryption Algorithm", In Advances in Cryptology - Crypto '95, pages 171-184, Springer-Verlag, New York, 1995.

[2] Rivest, Ronald L., "The RC5 Encryption Algorithm", dans Proceedings of the Second International Workshop on Fast Software Encryption, pages 86-96, Leuven Belgium, décembre 1994.

[3] Rivest, Ronald L., "RC5 Encryption Algorithm", In Dr. Dobbs Journal, number 226, pages 146-148, janvier 1995.

[4] Rivest, Ronald L., "The MD5 Message-Digest Algorithm", RFC 1321.

[5] RSA Laboratories, "Public Key Cryptography Standards (PKCS)", RSA Data Security Inc. Voir ftp.rsa.com.

[6] Schneier, Bruce, "Applied Cryptography", Second Edition, John Wiley and Sons, New York, 1996. Errata : page 195, ligne 13, le numéro de référence devrait être [402].

[7] Business Software Alliance, Matt Blaze et al., "Minimum Key Length for Symmetric Ciphers to Provide Adequate Commercial Security", http://www.bsa.org/bsa/cryptologists.html.

[8] RSA Data Security Inc., "RC5 Reference Code in C", Voir le site : www.rsa.com. Non disponible avec le premier projet de ce document.

Adresse des auteurs

Robert W. Baldwin
RSA Data Security, Inc.
100 Marine Parkway
Redwood City, CA 94065
téléphone : (415) 595-8782
Fax : (415) 595-1873
mél : baldwin@rsa.com, or baldwin@lcs.mit.edu

Ronald L. Rivest
Massachusetts Institute of Technology
Laboratory for Computer Science
NE43-324
545 Technology Square
Cambridge, MA 02139-1986
téléphone : (617) 253-5880
mél : rivest@theory.lcs.mit.edu