Estudio comparativo de los algoritmos de cifrado de flujo RC4, A5 y SEAL.

Silvana Bravo Hernández
Avance de Proyecto
Curso: Códigos y Criptografía
Profr. Dr. Francisco Rodríguez Henríquez
Departamento de Ingeniería Eléctrica
Sección Computación
CINVESTAV-IPN

6 de Agosto de 2002

Resumen

Actualmente el cifrado de flujo es el método utilizado para proteger la transmisión de datos en redes inalámbricas debido a la importancia que esto representa surge la necesidad de conocer a fondo los algoritmos más utilizados. Este documento contiene la descripción de la implementación de los algoritmos de cifrado de flujo RC4, A5 y SEAL, pruebas comparativas de ejecución de los algoritmos y conclusiones.

Introducción

Existe un esquema de encriptación perfecta, que se conoce como *One-time pad* y fue inventado en 1917 por Joseph Mauborgne y Gibert Vernam de AT&T [6], este esquema consiste en emplear una secuencia aleatoria de igual longitud que el mensaje, la cual se utiliza una sola vez, combinandolá con una operación simple y reversible con el texto original bit a bit. Este esquema carece de utilidad práctica por el inconveniente de que la llave es tan grande como el mensaje original, sin embargo existe un esquema que se deriva de éste, que es más práctico, en él se usa un generador pseudoaleatorio capaz de generar secuencias aleatorias, de forma que la longitud de los posibles ciclos sea extremadamente grande, y así utilizar la semilla del generador pseudoaleatorio como una llave privada para generar una secuencia de bits y aplicarle un XOR con el texto original para generar el mensaje encriptado, así todo aquel que conozca la semilla pódra reconstruir la secuencia aleatoria y con ella desencriptar el mensaje. Actualmente este esquema llamado cifrado en flujo es uno de los métodos más usados a la hora de encriptar información. Podemos encontrar aplicaciones de este cifrado en sistemas tales como el *DVD* que usa un cifrado de flujo denominado *CSS* para encriptar la información que contiene en MPEG, el protocolo SSL (que proporciona una comunicación segura a través de Internet), y el protocolo WEP (para proteger la transmisión de datos en redes inalámbricas) entre otras apllicaciones, por eso es de gran importancia conocer los algoritmos de cifrado de flujo más utilizados en estas aplicaciones, los cuales son RC4,A5 y SEAL, para así establecer las ventajas de cada uno. El documento contiene la sección de Análisis del Problema, que presenta la generación de una secuencia aleatoria, en la segunda sección se presenta los algoritmos RC4,A5 y SEAL y sus implementaciones, las cuales se utilizarón para ejecutar las pruebas que se presentan en la siguiente sección y por último la sección de conclusiones.

Análisis del Problema

La seguridad de un criptosistema con cifrado de flujo depende de la generación de una secuencia aleatoria, pero sabemos que la generación de una secuencia realmente aleatoria en una computadora es imposible ya que son totalmente deterministas. Así que para el uso de un criptosistema basado en el de Vernam, utilizaremos secuencias pseudoaleatorias generadas por una computadora. Estas secuencias serán finitas y tendrán, por tanto, un periodo. La generación de estas secuencias se intentará que sean lo más parecido a una secuencia verdaderamente aleatoria. Así, en primer lugar, se utilizarán secuencias cuyo periodo sea lo más grande posible (como mínimo, la longitud del texto original). Y en segundo lugar, que no haya las mínimas relaciones entre los números de la secuencia generada para que se considere prácticamente aleatoria. Esta segunda condición da lugar a distintos niveles de aleatoriedad en las secuencias, que son:

Secuencias estadísticamente aleatorias Secuencias que superan los tests estadísticos de aleatoriedad. Un generador congruencial lineal que es de la forma:

$$X(i+1) = (a * X(i) + b)(mod n)$$

cumple con este requisito, pero plantea un problema para su uso en Criptografía. Para el cálculo de cada valor de la secuencia se utiliza como semilla el valor anterior, con lo que dado un valor de la secuencia se obtiene toda ella. Además es relativamente fácil atacar la secuencia, de forma similar a los cifradores afines de la criptografa clásica.

Secuencias criptográficamente aleatorias Secuencias que han de cumplir la propiedad de ser impredecibles. Esto surge como alternativa a las secuencias estadísticamente aleatorias. Y cuando se hablamos de impredecibles queremos decir que no es computacionalmente tratable el averiguar un número de la secuencia aún en el caso de que se tengan

todos los valores anteriores de dicha secuencia y el algoritmo utilizado. Existen generadores de este tipo, pero de nuevo plantean un problema parecido. Utilizan una semilla inicial del que parte el algoritmo. Si esta semilla es conocida, el sistema está comprometido.

Secuencias totalmente aleatorias Dado que no se puede producir secuencias verdaderamente aleatorias en una comutadora, consideraremos que una secuencia es totalmente aleatoria cuando no puede ser reproducida de manera fiable.

Así pues, para nosotros será suficiente con generar secuencias a partir de semillas que sean impredecibles.

Generación de Secuencias Aleatorias

Para la generación de secuencias impredecibles, es necesario, en primer lugar, la inicialización de una semilla valida y, en segundo lugar, la generación propiamente dicha.

Obtención de una Semilla aleatoria

En una computadora todas las operaciones aritméticas y lógicas son totalmente deterministas por lo que habrá que buscar elementos menos deterministas que ayuden a la generación de estas semillas, en la mayoría de los casos, se utilizan diferentes métodos basados en el Hardware:

- Lectura del reloj interno
- Lectura de los números de serie de los componentes de la computadora
- Lectura de tarjetas digitalizadoras de audio o video

Generalmente, se utilizan diferentes métodos en conjunción mediante algoritmos adecuados que aseguren una aleatoriedad en la semilla. Además, dependiendo de los bits que se necesitan para la semillase tiene que reducir el número generado convenientemente o buscar métodos adicionales para la generación de bits adicionales.

Generadores Aleatorios Criptográficamente Seguros

Ahora veremos diferentes generedores criptográficos. En primer lugar, describiremos dos generadores aleatorios ampliamente utilizados (X9.17 y Blum

Blum Shub) y después haremos un análisis más exhaustivo de los generadores con registros de desplazamiento.

Generador X9.17 El ANSI propuso el siguiente método para la genración de una secuencia de llaves:

- s0 es la semilla inicial de 64 bits
- Gn es la secuencia de números de 64 bits generados
- K es la clave aleatoria reservada para generar esta secuencia
- t es el tiempo en el que un número es generado tan pronto como se ha encontrado una solución (de hasta 64 bits)
- DES (K, Q) es la encriptaciósn DES del número Q con la clave K

$$g_n = DES(k, DES(k, t).xor.s_n)$$

$$s_{n+1} = DES(k, DES(k, t).xor.g_n)$$

Blum Blum Shub Este es el generador que ha soportado mayores pruebas de resistencia en la actualidad. Debe su nombre a sus creadores [BBS]. Es muy simple y está basado en residuos cuadráticos. Su única desventaja es que tiene un costo computacional mucho mayor que el algoritmo $X9.17$. Es por ello que se usa para propósitos de uso no demasiado frecuente.

Este algoritmo simplemente escoge 2 números primos relativos grandes, a los que llamaremos p y q, tales que ambos tienen un resto de 3 al dividirlos por 4. Tenemos $n = p * q$. Entonces escogemos un número aleatorio X primo con n. La semilla inicial y el algoritmo para calcular los siguientes valores de la secuencia serán entonces:

$$s_0 = (x^2)(Mod n)$$

$$s_{i+1} = (s^2)(Mod n)$$

Hay que tener cuidado en usar solo los bits menos significativos de s. Siempre es seguro usar solo el bit menos significativo. Si no se usan más de

$$log_2(log_2(s_i))$$

bits bajos, entonces predecir cualquier bit adicional de una secuencia generada por este método está probado que es tan difícil como factorizar n. Dado que la X inicial es secreta, se puede hacer público n.

Una característica interesante de este generador es que se puede calcular directamente cualquiera de los valores s. En particular

$$s_i = (s_0^{((2^i)(Mod((p-1)*(q-1))))})(Modn)$$

Esto implica que no es necesario almacenar todas las llaves generadas siguiendo este método. Todas las llaves pueden ser ordenadas y recuperadas a partir de su índice y la s y la n inicial.

Generadores basados en registros de desplazamiento Existen dos tipos:

- Realimentados linealmente (LFSR)
- Realimentados No linealmente (NLFSR)

La ventaja de estos generadores se basa en que mediante simples operaciones ofrecen excelente resultados estadísticos y son fáciles de implementar mediante circuitos.

LFSR

Sean $a_0, ..., a_k - 1$ elementos de F. Una sucesión $s_0, s_1, ..., s_n...$ de elementos de F que satisfacen

$$s(n+k) = a(k-1)s(n+k-1)+a(k-2)s(n+k-2)++a(0)s(n)paran = 0, 1, ...$$

se dice que es una sucesión de recurrencia lineal homogénea de orden k. Un LFSR es un dispositivo que implementa una sucesión de recurrencia lineal. Un reloj externo controla las entradas y salidas de los registros, de manera que en cada pulso del mismo las unidades de registro proporcionan una salida.

Inicialmente los registros contienen los valores $s(0), ..., s(k-1)$, de manera que el primer pulso del reloj proporciona s0 como salida y los registros pasan a contener los elementos $s(1), ..., s(k)$, siendo $s(k) :=$

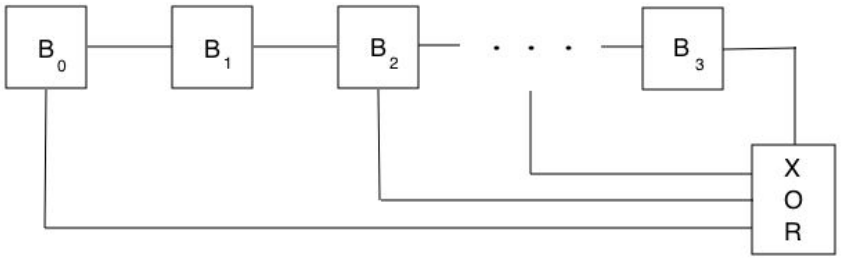

Figura 1:

$sum(a(i)s(i))$. La sucesión lineal lineal recurrente del comienzo está generada por el LFSR.

El estado inicial $s(0) = (s(0), ..., s(k-1))$ junto con el LFSR determinan completamente la sucesión recurrente. Codificaremos el LFSR mediante su polinomio caracterstico

$$f(X) = X^(k) - a_(k-1)X^(k-1) - \ - a(1)X - a(0)$$

Los estados posibles, $s(n) = (s(n), ..., s(n+k-1))$ son a lo sumo $2^k - 1$ (el estado (0, . . . , 0) se omite, ya que produce la sucesión idénticamente nula). Como consecuencia cualquier SLR es preperiódica .

LNFSR

Como medida de la complejidad de una secuencia cifrante se usa la llamada complejidad lineal, que es esencialmente el tamaño mínimo de un LFSR capaz de generarla (el orden k en el caso de los LFSR) en lugar de su periodo (que puede ser 2k - 1 para un LFSR).

Para resolver la debilidad de los LFSR (baja complejidad lineal), aprovechando al mismo tiempo su sencillez de manejo y de implementacin, se utilizan diversas de tcnicas de combinación no lineales de varios de ellos.

Puesto que queremos formar SLR lo más aleatorias posibles, es claro que una primera condición es que su periodo sea lo más grande posible. De hecho si se usan como una llave de cifrado su periodo debería ser al menos tan largo como el mensaje a cifrar. De ahí el interés de conocer el periodo de las SLR y como se pueden conseguir periodos largos.

Implementación

RC4

El Algoritmo *RC4* fue diseñado por Ron Rivest en 1987 para la compañía *RSA Data Security*. Su implementación es extremadamente sencilla y rápida, y está orientado a generar secuencias en unidades de un byte, además de permitir claves de diferentes longitudes.

El código del algoritmo no se ha publicado nunca oficialmente, pero en 1994 alguien difundió en los grupos de noticias de Internet una descripción que, como posteriormente se ha comprobado, genera las mismas secuencias.

Forma una parte vital del sistema de cifrado en capas *SSL*, ampliamente utilizado en navegadores de Internet tales como Netscape Navigator y Microsoft Internet Explorer. Por desgracia, la versión exportable de *RC4* tiene una llave de solamente 40 bits, lo que lo hace altamente vulnerable a ataques por fuerza bruta. Algoritmo:

1. A partir de la clave secreta se elige una permutación del grupo simétrico S256 , es decir, una forma particular de ordenar los números del 0 al 255.

2. Se toma el primer byte del mensaje y se realiza el cifrado efectuando una xor con una parte de la permutación, este proceso se sigue para cada byte del mensaje.

A la primera parte se la conoce como la extensión de la llave, la segunda describe el proceso de cifrado. Cabe hacer notar que el proceso de descifrado es igual al cifrado por lo tanto solo nos dedicaremos al primero.

1. Extensión de la llave

   (a) Se inicializan las siguientes variables

```
for(i = 0; i < 256; i++)
            state[i] = i;
```

```
x = 0;
y = 0;
index1 = 0;
index2 = 0;
```

Es decir el array state de 256 entradas se inicializa con state[i] = i.

(b) A partir de la llave privada, se reordena el array state, de la siguiente manera

```
for(i= 0; i < 256; i++)
{
   index2 = (key[index1] + state[i] + index2) mod 256;
   swapbyte(state[i], state[index2]);
   index1 = (index1 + 1) mod keylen;
}
```

Recorriendo cada elemento del array state, primero se recalcula la variable index2.

Posteriormente se realiza una transposición (swap), es decir, se intercambia el elemento en state[i] por state[index2].

Finalmente se recalcula la variable index1.

Lo anterior da un nuevo orden a los elementos del array state, que no es nada ms que una permutación calculada a partir de la llave privada, es decir se elige un elemento del grupo simétrico S256 .

2. Proceso de cifrado

(a) El proceso de cifrado es ahora muy simple

```
for(i = 0; i < menlen; i++)
{
x = (x + 1) mod 256;
y = (state[x] + y) mod 256;
swapbyte(state[x], state[y]);
xorIndex = (state[x] + state[y]) mod 256;
men[i] ^= state[xorIndex];
}
```

Para cada byte del mensaje, es decir desde i=0 hasta i=longitud del mensaje (menlen), se calculan las variables x, y con las formulas expuestas.

Se realiza una transposición (swap) del elemento state[x] y state[y]. Se calcula la variable xorIndex. Y finalmente se cifra el byte men[i] con la formula men[i] xor state[xorIndex].

A5

El algoritmo de cifrado por flujo *A5* es un generador binario de secuencia cifrante utilizado para cifrar el enlace entre el télefono móvil y la estación base en el sistema de telefonía móvil *GSM (Global Systems for Mobile communications)*. Una conversación en el sistema de telefonía *GSM* entre dos comunicantes A y B se transmite como una sucesión de tramas. Cada trama consta de 228 bits de los cuales los 114 primeros representan la comunicación digitalizada en el sentido A $\longrightarrow$ B, mientras que los restantes 114 bits representan la comunicación digitalizada en el sentido contrario B $\longrightarrow$ A. Cada 4.6 milisegundos se envía una nueva trama. Previamente a la transmisión de cualquier se realiza un proceso de sincronización entre comunicantes que incluye la sincronización entre comunicantes que incluye la criptosincronización. El generador *A5* produce 228 bits pseudoaleatorios de cada trama que se sumarán, bit a bit, mediante la operación XOR con los 228 bits de conversación en claro dando lugar a los 228 bits de conversación cifrada[5]. A5 consiste de 3 LFSR's, las longitudes de los registros son 19,22 y 23, todos los polinomios feedback son dispersos, la salida es el XOR de los tres LFSR's. A5 usa variables de control de reloj.

```
void a5_key(a5_ctx *c, char *k)
{
   c->r1 = k[0]<<11|k[1]<<3 | k[2]>>5;                /* 19 */
   c->r2 = k[2]<<17|k[3]<<9 | k[4]<<1 | k[5]>>7;      /* 22 */
   c->r3 = k[5]<<15|k[6]<<8 | k[7];                   /* 23 */
}

int a5_step(a5_ctx *c)
{

   int control;
   control = threshold(c->r1,c->r2,c->r3);
   c->r1 = clock_r1(control,c->r1);
   c->r2 = clock_r2(control,c->r2);
```

```
   c->r3 = clock_r3(control,c->r3);
   return((c->r1^c->r2^c->r3)&1);
}

/* cifra un bufer de longitud len */
void a5_encrypt(a5_ctx *c, char *data, int len)
{
   int i,j;
   char t;

   for(i=0;i<len;i++)
   {
      for(j=0;j<8;j++)
         t = t << 1 | a5_step(c);
      data[i]^=t;
   }
}
```

SEAL

SEAL es un generador de secuencia diseñado en 1993 para IBM por Phil Rogaway y Don Coppersmith, cuya estructura está especialmente pensada para funcionar de manera eficiente en computadoras con una longitud de palabra de 32 bits. Su funcionamiento se basa en un proceso inicial en el que se calculan los valores para unas tablas a partir de la llave, de forma que el cifrado propiamente dicho puede llevarse a cabo de una manera realmente rápida. Una característica muy útil de este algoritmo es que no se basa en un sistema lineal de generación, sino que define una familia de funciones pseudo aleatorias, de tal forma que se puede calcular cualquier porción de la secuencia suministrando únicamente un número entero n de 32 bits. La idea es que, dado ese número, junto con la llave k de 160 bits, el algoritmo genera un bloque $k(n)$ de L bits de longitud. De esa forma, cada valor de k da lugar a una secuencia total de $L/232$ bits, compuesta por la yuxtaposición de los bloques $k(0); k(1); :::; k(232-1)$. *SEAL* se basa en el empleo del algoritmo SHA para generar las tablas que usa internamente.

Descripción de la implementación del Algoritmo

Como se muestra en la figura las 3 tablas derivadas de la llave **a** llamadas **R S** y **T**, manejan el algoritmo,la taba **T** es una caja S de 9 x 32 bits. En

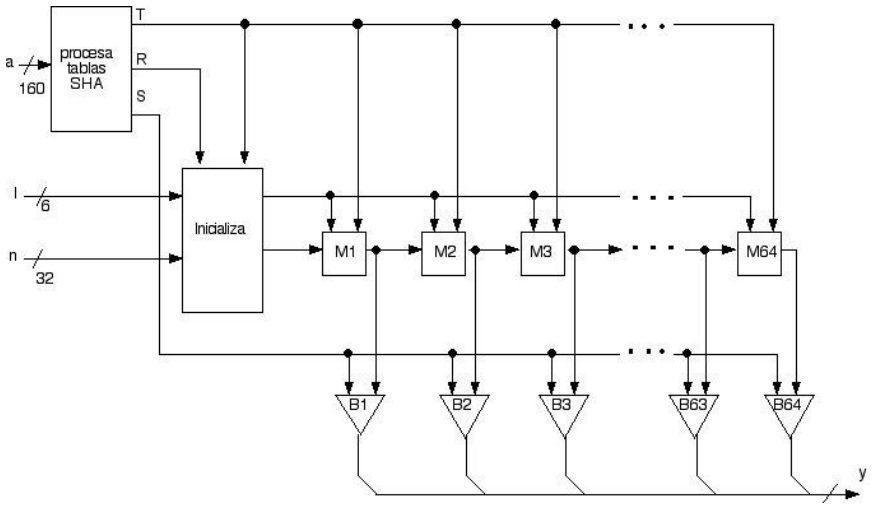

Figura 2:

un paso de preprocesamiento se mapea la llave a estas tablas usando un procedimiento basado en SHA, que se realiza en las siguientes funciones:

```
int g(unsigned char *in,int i,unsigned long *h)
{
unsigned long h0;
unsigned long h1;
unsigned long h2;
unsigned long h3;
unsigned long h4;
unsigned long a;
unsigned long b;
unsigned long c;
unsigned long d;
unsigned long e;
unsigned char *kp;
unsigned long w[80];
unsigned long temp;

   kp = in;
   h0 = WORD(kp); kp += 4;
   h1 = WORD(kp); kp += 4;
   h2 = WORD(kp); kp += 4;
```

```
h3 = WORD(kp); kp += 4;
h4 = WORD(kp); kp += 4;

w[0] = i;
for(i=1;i<16;i++)
   w[i]=0;
for(i=16;i<80;i++)
   w[i] = w[i-3]^w[i-8]^w[i-14]^w[i-16];

a = h0;
b = h1;
c = h2;
d = h3;
e = h4;

for(i=0;i<20;i++)
{
  temp = ROT27(a) + F1(b,c,d) + e + w[1] + 0x5a827999;
  e = d;
  d = c;
  c = ROT2(b);
  b = a;
  a = temp;
}

for(i=20;i<40;i++)
{
  temp = ROT27(a) + F2(b,c,d) + e + w[1] + 0x6ed9eba1;
  e = d;
  d = c;
  c = ROT2(b);
  b = a;
  a = temp;
}

for(i=40;i<60;i++)
{
  temp = ROT27(a) + F3(b,c,d) + e + w[1] + 0x8f1bbcdc;
  e = d;
  d = c;
```

```
    c = ROT2(b);
    b = a;
    a = temp;
  }

  for(i=60;i<80;i++)
  {
    temp = ROT27(a) + F1(b,c,d) + e + w[1] + 0xca62c1d6;
    e = d;
    d = c;
    c = ROT2(b);
    b = a;
    a = temp;
  }
  h[0]=h0+a;
  h[1]=h1+b;
  h[2]=h2+c;
  h[3]=h3+d;
  h[4]=h4+e;

  return (ALG_OK);
}

int seal_init(seal_ctx *result,unsigned char *key)
{
int i;
unsigned long h[5];

  for(i=0;i<510;i+=5)
      g(key,i/5,&(result->t[i]));

  g(key,510/5,h);
  for(i=510;i<512;i++)
      result->t[i] = h[i-510];

  g(key,(-1+0x1000)/5,h);
  for(i=0;i<4;i++)
      result->s[i] = h[i+1];
  for(i=4;i<254;i+=5)
      g(key,(i+0x1000)/5, &(result->s[i]));
```

```
    g(key,(254+0x1000)/5,h);
    for(i=254;i<256;i++)
        result->s[i] = h[i-254];

    g(key,(-2+0x2000)/5,h);
    for(i=0;i<3;i++)
        result->r[i] = h[i+2];
    for(i=3;i<13;i+=5)
        g(key,(i+0x2000)/5, &(result->r[i]));

    g(key,(13+0x2000)/5,h);
    for(i=13;i<16;i++)
        result->r[i] = h[i-13];

    return (ALG_OK);
}
```

Se utilizan 4 registros de 32 bits, su valor inicial es determinado por **n** y las tablas **R** y **T**. Estos registros obtinen moificaciones en varas iteraciones, cada una envuelve 8 rondas, en cada ronda 9 bits de uno de los registros son usados para indexar la tabla **T**, con el valor obtenido y el contenido de otro de los registro se realiza un XOR, después de 8 roundas de estas, los registros son sumados a la keystream, cada registro es enmascarado primero por realizar un XOR con cierta palabra de la tabla **S**. Las interacciones son completadas por sumar a los registros **A** y **C** los valores en n, n_1, n_2, n_3, n_4 exactamente con uno de ellos dependiendo de la paridad del número de iteración.

```
int seal(seal_ctx *key, unsigned long in, unsigned long *out)
{
int i;
int j;
int l;
unsigned long a;
unsigned long b;
unsigned long c;
unsigned long d;
unsigned short p;
unsigned short q;
```

```
unsigned long n1;
unsigned long n2;
unsigned long n3;
unsigned long n4;
unsigned long *wp;

  wp = out;

  for(l=0;l<4;l++)
  {
     a = in ^ key->r[4*l];
     b = ROT8(in) ^ key->r[4*l+1];
     c = ROT16(in) ^ key->r[4*l+2];
     d = ROT24(in) ^ key->r[4*l+3];

     for(j=0;j<2;j++)
     {
         p = a & 0x7fc;
         b += key->t[p/4];
         a = ROT9(a);

         p = b & 0x7fc;
         c += key->t[p/4];
         b = ROT9(b);

         p = c & 0x7fc;
         d += key->t[p/4];
         c = ROT9(c);

         p = d & 0x7fc;
         a += key->t[p/4];
         d = ROT9(d);
     }
     n1 = d;
     n2 = b;
     n3 = a;
     n4 = c;

     p = a & 0x7fc;
     b += key->t[p/4];
```

```
 a = ROT9(a);

 p = b & 0x7fc;
 c += key->t[p/4];
 b = ROT9(b);

 p = c & 0x7fc;
 d += key->t[p/4];
 c = ROT9(c);

 p = d & 0x7fc;
 a += key->t[p/4];
 d = ROT9(d);

/* Esto genera 64 32-bit words, o 256 bytes de keystream */

 for(i=0;i<64;i++)
 {
   p = a & 0x7fc;
   b += key->t[p/4];
   a = ROT9(a);
   b ^= a;

   q = b & 0x7fc;
   c ^= key->t[q/4];
   b = ROT9(b);
   c ^= b;

   p = (p+c) & 0x7fc;
   d += key->t[p/4];
   c = ROT9(c);
   d ^= c;

   q = (q+d) & 0x7fc;
   a ^= key->t[q/4];
   d = ROT9(d);
   a ^= d;

   p = (p+a) & 0x7fc;
   b ^= key->t[p/4];
```

```
        a = ROT9(a);

        q  = (q+b) & 0x7fc;
        c += key->t[q/4];
        b = ROT9(b);

        p = (p+c) & 0x7fc;
        d ^= key->t[p/4];
        c = ROT9(c);

        q = (q+d) & 0x7fc;
        a ^= key->t[q/4];
        d = ROT9(d);

        *wp = b + key->s[4*i];
        wp++;
        *wp = c ^ key->s[4*i+1];
        wp++;
        *wp = d + key->s[4*i+2];
        wp++;
        *wp = a ^ key->s[4*i+3];
        wp++;

        if(i&1)
        {
           a+=n3;
           c+=n4;
         }else{
           a+=n1;
           c+=n2;
         }
   }}
 return (ALG_OK);
}
```

Pruebas

Las pruebas se realizarón en una maquina SP2 con 4 procesadores Intel Xeon de 750MHz con 1Gb de Memoria RAM.

Las tablas muestran el desempenõ de los tres algoritmos para cifrar archivos de 7,39,131y 170 Megabytes:

Algoritmo	Megabytes	Segundos
SEAL	7	1
RC4	7	1
A5	7	18
SEAL	39	6
RC4	39	6
A5	39	94
SEAL	131	19
RC4	131	20
A5	131	322
SEAL	170	25
RC4	170	26
A5	170	420

En la siguiente figura se muestra una gráfica comparativa de la ejecución en segundos de los 3 algoritmos, y claramente podemos ver como los algoritmos SEAL y RC4, son más eficientes que el algoritmo A5.

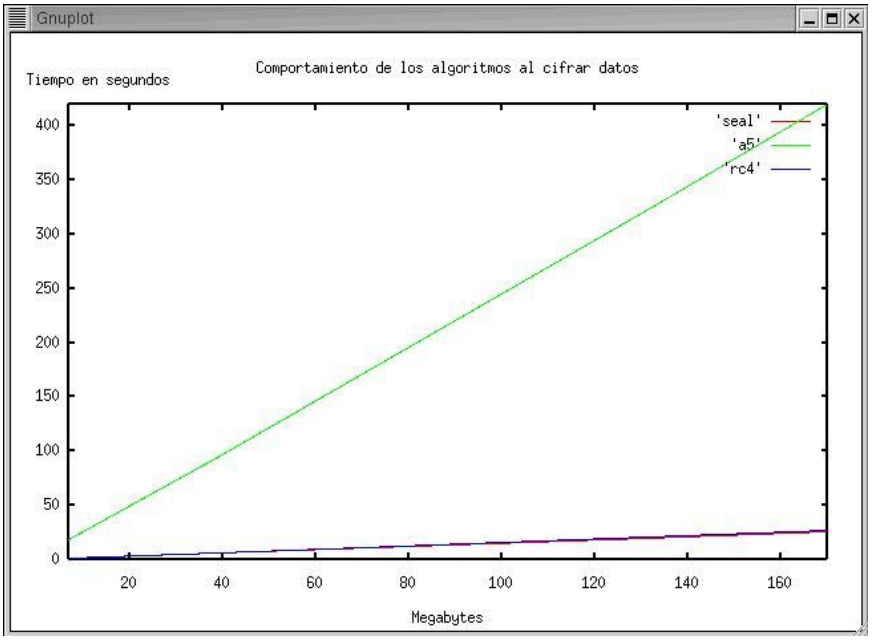
Gnuplot
Comportamiento de los algoritmos al cifrar datos
Tiempo en segundos
'seal'
'a5'
'rc4'
400
350
300
250
200
150
100
50
0
20
40
60
80
100
120
140
160
Megabytes

Figura 3:

Conclusiones

- SEAL requiere alrededor de 5 operaciones para encriptar cada byte de texto, lo que lo hace muy rapido, sin embargo es necesario preprocesar la llave en tablas iternas. Estas tablas tienen un tamaño de alrededor de 3 Kilobytes, así que no es apropiado para cituaciones cuando no se tiene tiempo de preprocesar la llave o no se tiene memoria para almacenar las tablas.

- RC4 presenta casi la misma efectividad en tiempo de ejecución que el algoritmo SEAL, sin embargo si existen patrones en la llave, se propagan también al estado interno del algoritmo, debilitndolo considerablemente.

- A5 es el más lento de los 3 algoritmos, sin embargo pasa todas las pruebas estadisticas de aleatoriedad, su debilidad es que sus registros son suficientemente pequeños para hacer factible una busqueda exhaustiva, por lo que variando el tamaño de los registros se puede obtener más seguridad.

Bibliografía

[1] http://cryptome.org/gsm-a512.html

[2] Introduction to Cryptography with Coding Theory
W. Trappe & L. C. Washington
Prentice Hall 2002

[3] Applied Cryptography
Bruce Schneier
John Wiley & Sons, Inc. 1996

[4] Contemporary Cryptologya: The Science of intformation Integrity,
G.J. Simmons
IEEE Press 1990
pp 65-134

[5] "A Software-Optimized Encryption Algorithm"
Phillip Rogaway and Don Coppersmith
Cambridge Security Workshop
Springer Verlag 1994
pp 56-63

[6] "The Codebreakers:The Story of Secret Writing"
New York: Macmillan Publishing Co. 1967