Universidade de São Paulo – USP
Instituto de Ciências Matemáticas e de Computação – ICMC
SCC - Departamento de Sistemas de Computação
SSC0547 - Engenharia de Segurança – Profa. Dra. Kalinka Castelo Branco

Trabalho 2

ESTUDO COMPARATIVO DE ALGORITMOS DE CRIPTOGRAFIA SIMÉTRICA E ASSIMÉTRICA

Nome : João Carlos Batista NºUSP :6792197

Data da entrega: 21/11/2012

Sumário

Seção 1 : Criptografia simétrica e assimétrica

O objetivo desta monografia é de fazer uma analise comparativa de algoritmos de criptografia simétrica (chave secreta) e assimétrica (chave pública). Especificamente trataremos com dois dos algoritmos mais conhecidos na literatura: por um lado DES e AES, e por outro lado Diffie Hellman e RSA respectivamente. Nóte-se que foram empregados diversas fontes de criptografia na literatura como são [2],

1. 1 Tipos de criptografia

A criptografia pode ser classificada quanto ao número de chaves utilizadas, (se elas utilizam uma mesma chave para criptografar e descriptografar -simétrica ou se utiliza duas chaves diferentes –assimétrica), e forma de processamento (bloco ou *stream* – fluxo).

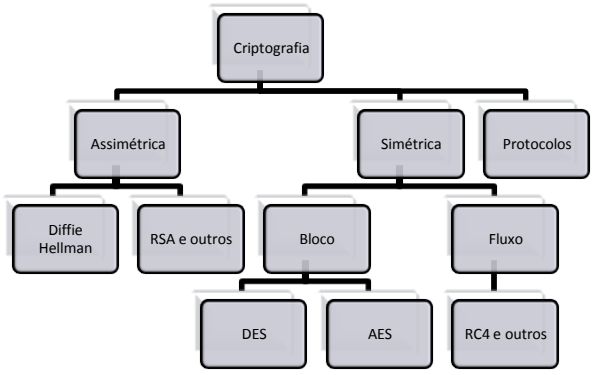

Figura 1. Tipos de criptografia segundo o tipo da chave

1. 1.1 Criptografia Simétrica:

A criptografia simétrica é um método criptográfico de chave única, onde o emissor e o destinatário possuem a mesma chave. Assim esta chave é utilizada tanto para criptografar a mensagem como para descriptografar. Exemplos: DES e AES.

1. 1.2 Criptografia Assimétrica:

Também chamados de algoritmos de chave pública. Utiliza duas chaves: Uma para criptografar e outra para descriptografar a mensagem. A geração das chaves (problema direto) é rápida e eficiente, já o problema reverso (a partir da chave pública definir a chave privada), é a parte complexa da técnica de chave assimétrica. Exemplos: Diffie Hellman e RSA.

1. 2 Funcionamento DES: *Data Encryption Standard*

O DES é um algoritmo de cifragem que foi mais utilizado no mundo. O Algoritmo DES é composto de operações simples, como: permutações, substituições, XOR e deslocamentos.

No processo de criptografar dados, o DES divide a mensagem em blocos de 64 bits e retorna blocos de texto cifrado do mesmo tamanho. O algoritmo parametrizado por uma chave de 56 bits, tem 19 estágios distintos, sendo que o primeiro é uma transposição independente da chave no texto simples de 64 bits, e no ultimo estágio acontece o processo inverso dessa transposição. O penúltimo estágio troca os 32 bits mais à esquerda por 32 bits mais à direita.

Os 16 estágios restantes são iterações que cada bloco de 64 bits sofrerá. Porem cada uma delas com uma chave diferente. Antes de cada iteração, a chave é particionada em duas unidades de 28 bits, sendo cada uma delas girada à esquerda um número de bits que depende do número de iteração. Utiliza se ainda um parâmetro k que é derivada da girada, pela aplicação de mais uma transposição de 56 bits sobre ela. E em cada rodada, um subconjunto de 48 bits dos 56 bits é extraído e permutado.

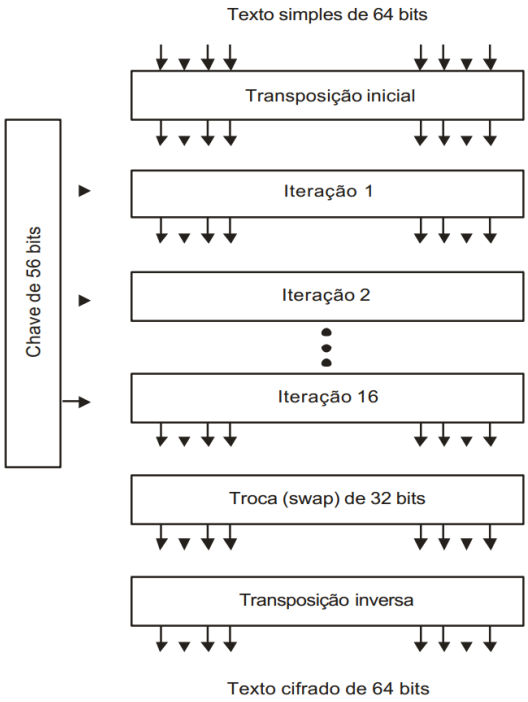

Figura 2. Visão geral do funcionamento do DES.

Sobre a direção de John Gilmore do EFF, uma equipe construiu uma maquina que podia analisar todo o espaço de chaves de 56 bits de DES, e em 17 de julho de 1998, foi anunciado que havia conseguido quebrar uma chave de 56 bits em 48 horas. Logo depois surgiu o TripleDES.

1. 3 Funcionamento AES: *Advanced Encryption Standard*

Como a chave de 56 bits do DES foi quebrado, um órgão dos Estados Unidos chamado NIST patrocinou um concurso de criptografia, com o intuito de conseguir um algoritmo para ser utilizado como o novo padrão. O algoritmo vencedor foi o Rijndael e a partir do ano 2000 o cifrador passou a se chamar de AES.

O AES utiliza a substituição, permutação e rodadas assim com o DES, porem o numero de rodadas depende do tamanho da chave e do tamanho do bloco, já que o tamanho da chave é variável entre 128, 192 e 256 bits, o que significa que se pode ter tamanho de blocos com tamanhos de chaves diferentes. Por exemplo são 10 rodadas para cada chave de 128 bits com blocos de 128 bits, passando para 14 no caso da maior chave ou do maior bloco. No entanto, diferente do DES, todas as operações envolvem bytes inteiros, para permitir implementações eficientes, tanto em hardware como em software.

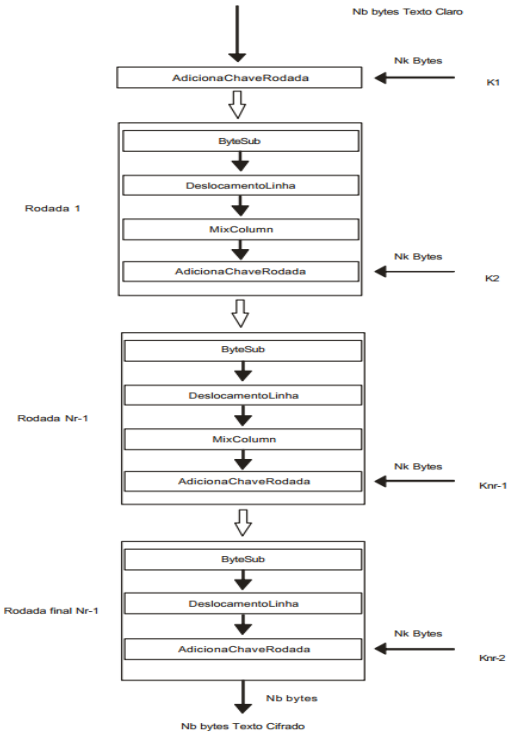

Figura 3 . Algoritmo de Cifragem do AES

O algoritmo se baseia em quatro operações

1. SubByte: os bytes de cada bloco são substituídos por seus equivalentes em uma tabela de substituição (S-BOX).
2. ShiftRow: ou deslocamento de linha: nesta etapa, os bytes são rotacionados em grupos de quatro bytes.
3. MixColumn: cada grupo de quatro bytes sujeita-se a uma multiplicação modular, o que proporciona a cada byte do grupo influenciar todos os outros bytes.
4. AddRoundKey: ou adição da chave de rodada: nesta fase, o bloco de dados é alterado por meio da subchave da rodada, a qual possui o mesmo tamanho do bloco, que realiza uma operação XOR com o bloco inteiro.

1.4 Funcionamento Diffie Hellman

No ano de 1976, dois pesquisadores da University of Stanford, Diffie e Hellman, propuseram um algoritmo de criptografia radicalmente novo, no qual as chaves de criptografia e de descriptografia eram diferentes, os algoritmos utilizados para a troca de chave secreta em canal público abriu as portas para a Criptografia de chave pública. Sua patente expirou em 1997.

O algoritmo de troca de chaves Diffie-Hellman constitui em uma série de operações matemáticas feitas por ambas as partes, sendo que ambos chegarão a um mesmo valor, a um mesmo número. Ele usa a exponenciação e a aritmética modular como base de seus cálculos; sendo bastante seguro, mas envolve números muito grandes e cálculos relativamente lentos.

O princípio da troca de chaves idealizada por Diffie Hellman tinha com base três requisitos básicos:

1. **D(E(P)) = P**: neste primeiro requisito, fala se aplicarmos **D** a uma mensagem criptografada, **E(P),** obteremos outra vez a mensagem de texto simples original **P**. Este requisito permite ao destinatário legitimo decodificar o texto cifrado.
2. É extremamente difícil deduzir **D** a partir de **E**.
3. **E** não pode ser decifrado por um ataque de texto simples escolhido.

1.5 Funcionamento RSA: *Rivest, Chamir e Adleman*

Criado em 1977 por Ron Rivest, Adi Shamir e Len Adleman, é um dos algoritmos de chave assimétrica mais utilizados. Sua patente expirou no ano 2000. Este sistema de criptografia consiste em gerar uma chave pública, geralmente utilizada para cifrar dados e uma chave privada, utilizada para decifrar os dados, por meio de números primos grandes, o que dificulta a obtenção de uma chave a partir da outra.

Seu funcionamento consiste na multiplicação de 2 números primos muito grandes para a geração de um terceiro número. A chave privada são os dois números primos e a pública é o terceiro número. A segurança proporcionada deste algoritmo de criptografia depende do tamanho dos números primos fornecidos, ou seja, quanto maior os números primos fornecidos maior segurança. Atualmente os números primos que são utilizados têm geralmente 512 bits de comprimento e combinados formam chaves de 1.024 bits.

Para usarmos o método RSA, devemos converter uma mensagem em uma sequência de números.

Geração das chaves

1. Escolha de forma aleatória dois números primos grandes p e q, da ordem de 10^{100} no mínimo.
2. Compute $n = pq$
3. Compute a função totiente em n: $\phi(n) = (p-1)(q-1)$.
4. Escolha um inteiro e tal que $1 < e < \phi(n)$, de forma que e e $\phi(n)$ sejam primos entre si. Compute d de forma que $de \equiv 1 \pmod{\phi(n)}$, ou seja, d seja o inverso multiplicativo de e em $\pmod{\phi(n)}$.

Por final temos:

A chave pública: o par de números n e e
A chave privada: o par de números n e d

Cifração: $c = m^e \mod n$

Descifração: $m = c^d \mod n$

1.6 Resumo comparativo

Fator de Analise	AES	DES	RSA	Diffie Hellman
Tipo de cifrador	Simétrico	Simétrico	Assimétrico	Assimétrico
Desenvolvido	2000	1977	1978	1976
Velocidade	Rápido	Rápido	Rápido	Lento
Tamanho da chave	128-192-256	56	> 1024 bits	Key exchange management
Seguridade	Considerado seguro	Foi quebrado		
Seguridade contra ataques	Chosen Plain, Knonwn plaintext	Força bruta	Timing attacks	Eavedrop ping

Tabela 1. Resumo comparativo dos algoritmos estudados. Baseado em [4, 5]

Seção 2 : Algoritmos implementados de criptografia baseado em mapas caóticos

2.1 Introdução

A teoria do caos apresenta características importantes que a tornam viável para seu uso na criptografia: padrões complexos, imprevisíveis, dependência das condições iniciais, baseado em equações simples e determinismo.

A motivação de se usar mapas caóticos apóia-se na simplicidade para a sua geração, pois podem ser criados por regras simples. Isso fez com que surgissem vários trabalhos sobre a sua aplicação em sistemas de criptografia e geradores de números aleatórios, cuja idéia básica é o emprego de sequências caóticas geradas por esses mapas, tornando a quebra das mensagens um trabalho bastante complexo. Em 1998, Baptista [3] propus um dois primeiros resultados aplicando a teoria do caos na criptografia.

O objetivo deste trabalho é de construir um método para criptografar e/ou descriptografar mensagens baseado na teoria do caos. Aqui propomos otimizar o algoritmo proposto por Baptista e além disso propomos o uso de um outro mapa caótico (Mapa Tent) para poder comparar ambos algoritmos de criptografia.

2.2 Requisitos

Para o desenvolvimento dos algoritmos não foram usados nenhuma biblioteca pronta ou framework e o código foi implementado a partir do artigo do Baptista [3] o qual não possui código disponível na internet. Foi desenvolvido em C++ usando o CodeBlocks v 10.05 com compilador GNU GCC Compiler versão para Windows. O qual pode se descarregar no link: http://sourceforge.net/projects/codeblocks/files/Binaries/10.05/Windows/codeblocks-10.05-setup.exe. As bibliotecas usadas foram: cstdlib, iostream, stdio, string, fstream, cmath, ctype.

Note-se que o algoritmo proposto consegue criptografar e descriptografar com arquivos binários como por exemplo: .txt, .jpg, .pdf, .exe, etc. Não importando os acentos e pontuação.

Exemplo de execução

```
>main 1 C chave.txt entrada.txt saida.txt
>main 1 D chave.txt entrada.txt saida.txt
```

2.3 Metodologia

A seguir, serão descritos os mapas caóticos empregados, as funções necessárias, a geração da chave entre outros com o respectivo código fonte.

Mapa Logístico: intervalo [0,1]

O mapa logístico é dado através da equação:

$$x_{n+1} = r x_n (1 - x_n)$$

onde "r" é um parâmetro. Este mapa possui diferentes comportamento segundo distintos valores deste parâmetro. Quando o valor de $r = 4$ o mapa logístico é caótico.

```
11  //valores constantes para chegar ao estado caotico de cada mapa
12  double r = 4;//estado caotico mapa Logistico

31  //funçãoo do mapa logistico
32  double mapaLogistico(double xAnt)
33  {
34      double xNovo;
35      xNovo = r * xAnt * (1 - xAnt);
36      return xNovo;
37  }
```

Mapa Tent: intervalo [0,1]

$$x_{n+1} = \begin{cases} \mu x_n, & x_n < 0.5 \\ \mu(1 - x_n), & x \geq 0.5 \end{cases}$$

Quando o valor de $\mu = 1.97$ o mapa Tent é caótico.

```
11  //valores constantes para chegar ao estado caotico de cada mapa
15  double rho = 1.97; // mapa Tent

52  //http://en.wikipedia.org/wiki/Tent_map
53  // [0,1]
54  double mapaTent(double xAnt)
55  {
56      double xNovo;
57      if(xAnt >= 0.5)
58          xNovo = rho*(1-xAnt);
59      else if(xAnt < 0.5 ) // caso contrario
60          xNovo=rho * xAnt;
61
62      return xNovo;
63  }
```

Senha:

A senha tem um espaço de 64 bits (*long*), é dizer pode ser qualquer número desde 0 até 2^{64}. A senha é então normalizada para um valor de]0,1[como condição inicial do mapa.

```
25  double min_permitido = 0.0001;//0.0
```

```
26  double max_permitido = 0.9999;//1.0

93  //F : função para converter uma senha no valor inicial do sistema
94  double geraChave(long senha)
95  {
96      double value;// por defeito, nao pode ser 0
97      value = (max_permitido - min_permitido) * abs(sin(senha));
98      // utiliza a função seno.. Pode ser qualquer uma outra
99      value = value + min_permitido;
100
103      if(value <=0)
104          value = min_permitido;
105      else if(value>=1)
106          value = max_permitido;
107
108      return value;
109  }
```

Blocos:

A mensagem é dividida em blocos de 8 bytes, é dizer que o espaço dos blocos é de 2^8, é dizer desde 0- 255, sendo os caracteres ASCII.

```
28  int S; // units alphabet //2^8
150  void codificador(long senha, int opcaoAlg)
151  {
152      double dois=2;
153      double pot = 8;
154      S = pow(dois, pot);
155      x_zero= geraChave(senha);
156      x_init = x_zero;
157
158      escolherMapa(opcaoAlg);
159  }
```

Iterar:

Os mapas caóticos precisam ser iterados constantemente. Estes valores serão usados como um gerador de números aleatórios, por tanto precisam ser normalizados e mapeados para os valores de 0-255 ASCII para depois ser combinados ao criptografar/descriptografar a mensagem.

```
78  int iterar(double x, int iter)
79  {
80      double temp = 0;
81      temp = (*MAPA)(x);
82      for (int i = 0; i < iter - 1; i++)
83      {
84          temp = (*MAPA)(temp); //iterar exemplo: MapaLogistico(temp)
85      }
86      x_zero = temp;
87      double key =((x_zero -min_permitido)/(max_permitido-min_permitido))*S;
88
89      return (int)key;
90
91  }
```

Criptografar:

```
131  int* criptografar(int *texto)
```

```
132  {
133
134      int *cifrado= new int[8];
135      int mapa;
136      int iter= primoA*transiente%primoC;
137      mapa = iterar(x_zero,iter);
138      cifrado[0] = texto[0] ^ mapa; // ^ XOR
139
140      for(int i=1; i< 8; i++)
141      {
142          iter= primoA*iter%primoC;
143          mapa = iterar(x_zero,iter);
144          cifrado[i]=texto[i] ^ mapa ^ cifrado[i-1];
145
146      }
147      return cifrado;
148  }
```

Descriptografar:

```
111  int* descriptografar(int *cifrado)
112  {
113      int *texto= new int[8];
114      int mapa;
115
116      int iter= primoA*transiente%primoC;
117      mapa = iterar(x_zero,iter);
118      texto[0] = cifrado[0] ^ mapa ; // ^ XOR
119
120      for(int i=1; i< 8; i++)
121      {
122          iter= primoA*iter%primoC;
123          mapa = iterar(x_zero,iter);
124          texto[i]= cifrado[i] ^ mapa^ cifrado[i-1];
125      }
126      return texto;
127
128
129  }
```

Padding:

Quando o tamanho da mensagem não um múltiplo de 64 bits, é preciso efetuar uma operação chamada *padding*, que consiste em acrescentar alguns bits até que a mensagem total contenha um múltiplo de 64 bits. Nestes algoritmos adicionamos zeros até completar o último bloco, que é o mais comum nas implementações atuais.

```
82   for(i=controlador+1; i<8; i++)
83   {
84       dados[i]=0;
85   }
```

Menu de entrada:

```
60  if(*argv[2]=='C')
61  {
62      int *cifrado=criptografar(dados) ;
63      for(int i=0; i< 8; i++)
64      {
```

```
65          fprintf(arq,"%c",cifrado[i]);
66      }
67  }
68  else if(*argv[2]=='D')
69  {
70      int *descifrado=descriptografar(dados) ;
71      for(int i=0; i< 8; i++)
72      {
73          fprintf(arq,"%c",descifrado[i]);
74      }
75  }
```

A figura 4 exemplifica a arquitetura dos algoritmos a partir de qualquer mapa caótico.

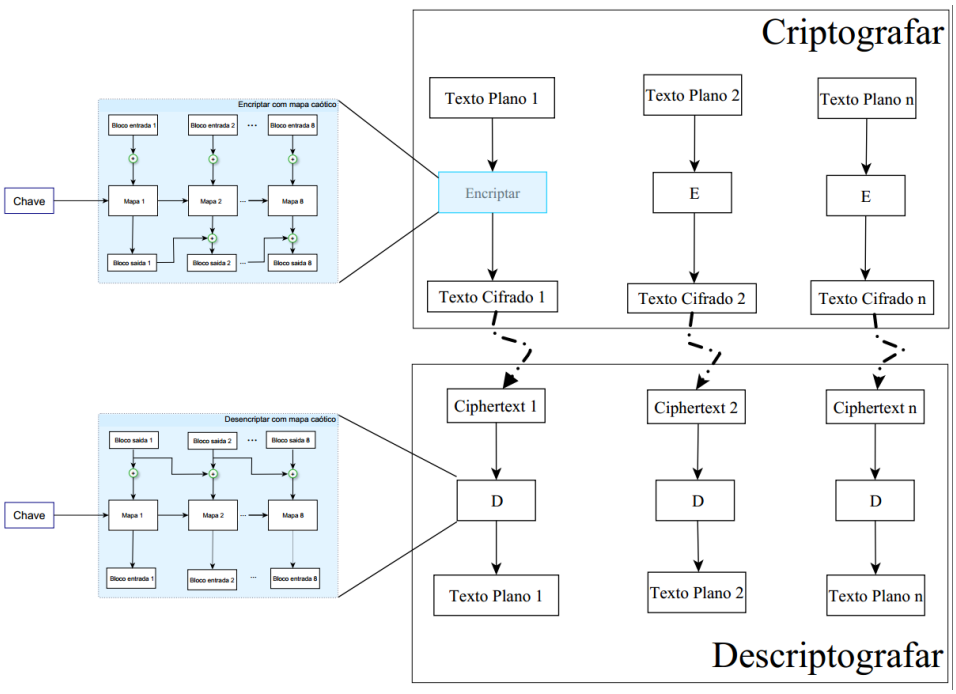

Figura 4: Diagrama de blocos do algoritmo de criptografia baseado em mapas caóticos

2.4 Testes de desempenho

Foram realizados vários testes de desempenho para medir a velocidade de processamento dos dois algoritmos. Estes testes foram realizados em um computador Intel Core i7 de 3.4 Ghz, memória RAM de 8GB e sistema operacional Windows 7 de 64-bits.

	Algoritmo 1 Mapa Logístico	**Algoritmo 2 Mapa Tent**
Tamanho	tempo segundos	tempo segundos

100	KB	0.029	0.031
200	KB	0.058	0.064
300	KB	0.085	0.095
400	KB	0.113	0.126
500	KB	0.140	0.157
600	KB	0.167	0.187
700	KB	0.194	0.218
800	KB	0.228	0.257
900	KB	0.253	0.280
1	MB	0.288	0.319
2	MB	0.567	0.633
3	MB	0.870	0.940
4	MB	1.146	1.259
5	MB	1.445	1.575
6	MB	1.690	1.903
7	MB	1.966	2.197
8	MB	2.257	2.536
9	MB	2.589	2.866
10	MB	2.877	3.165
20	MB	5.641	6.272
30	MB	8.336	9.477
40	MB	11.246	12.490
50	MB	13.975	15.864

Tabela 2: Resultados de velocidade para criptografar arquivos Kb/segundos

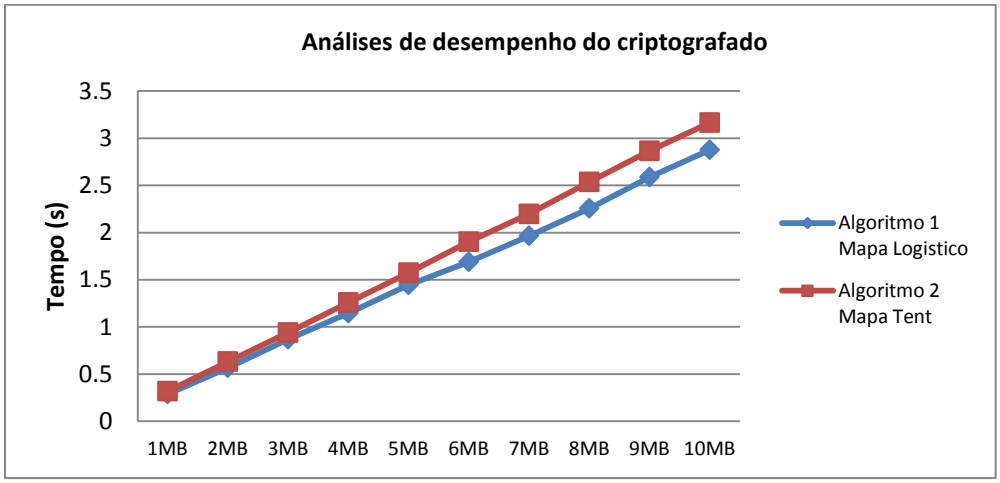

Figura 5: Análises de desempenho do criptografado MB/s

Analise de histogramas

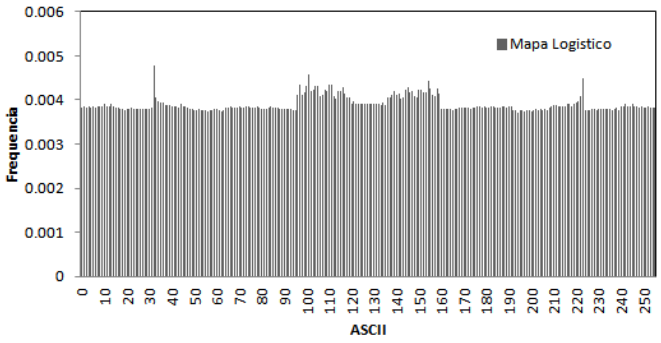

Figura 6: Histograma de arquivo de 20Mb com Mapa Logístico

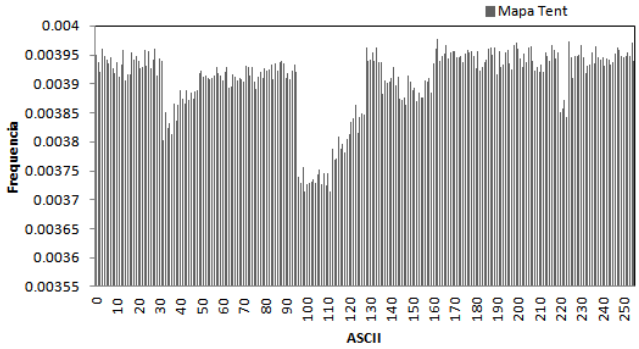

Figura 7: Histograma de arquivo de 20Mb com Mapa Tent

Observação:

Com respeito do algoritmo proposto por Baptista, conseguimos melhorar em vários aspectos de programação, já que o algoritmo original está baseado em tabelas de valores recorrendo a processos ineficientes de procura.

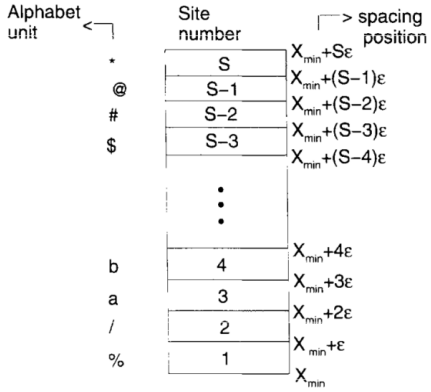

Figura 8: Algoritmo proposto por Baptista [1] baseado em tabelas

Em lugar de usar tabelas, usamos a normalização dos dados, por tanto é muito mais rápido e eficiente. Além disso, usamos valores ASCII de 255 bytes em lugar de um número reduzido como mostrado na Figura 8.

2.5 Conclusões

- O algoritmo do mapa logístico possui um histograma mais uniforme que o mapa Tent, o qual indicaria maior segurança para criptografar.
- Adicionalmente se fez um teste com arquivos de letras iguais: "AAAA....A", e "BAAAA...A", com um bit de diferença sendo os cifrados altamente diferentes, já que os algoritmos propostos evitam que haja sequências iguais para dados diferentes. Por tanto o cifrado não pode ser quebrado tão facilmente.
- O algoritmo do mapa Logístico é mais rápido em processamento que o mapa Tent, em razão de 0.029 KB/s a comparação de 0.031 KB/s respectivamente.
- O tamanho da chave de 2^{64} foi baseado em standard da literatura.
- A leitura de arquivos em binário permite o uso do algoritmo para qualquer tipo de arquivos.
- Pomos afirmar que o mapa Logístico é melhor que o mapa Tent.
- Precisam-se mais testes de segurança para conferir os nossos resultados.
- Precisa-se melhorar o algoritmo de geração de chave para não permitir a quebra da chave.

2.6 Referências

[1] Antonio Cândido Faleiros, "Criptografia", Centro de Matemática, Computação e Cognição Universidade Federal do ABC, 2010

[2] Jocimar Fernandes, "Criptografia e modelo criptográfico do sistema informatizado de eleições do Brasi"l, trabalho de conclusão de curso de pósgraduação na ESAB, 2007

[3] M. S. Baptista, "Cryptography with chaos" *Physics Letters A*, vol. 9601, no. 98, pp. 50–54, 1998.

[4] Al Jeeva et al,"Comparative analysis of performance efficiency and security measures of some encryption algorithms", International Journal of Engineering Research and Applications (IJERA), 2012

[5] Hamdan Alanazi et al, "New Comparative Study Between DES, 3DES and AES within Nine Factors", Journal of computing, volume 2, issue 3, 2010.