CLEFIA

Miguel Gaiowski e Douglas Santos

beamerinstitute

2012-06-27 Wed

Introdução

- CLEFIA é um algoritmo de cifragem de blocos, desenvolvido pela SONY, apresentado em 2007
- O tamanho do bloco é de 128 bits, enquanto que o tamanho da chave pode ser de 128, 192 ou 256 bits, fazendo com que ele seja compatível com a interface do cifrador AES.
- CLEFIA foi criado devido ao aumento da demanda por *lightweight cryptography*. *Lightweight cryptography* consiste de mecanismos criptográficos que fornecem segurança para dispositivos com capacidades limitadas, como por exemplo, sensores wireless.
- Portanto, CLEFIA fornece um alto level de segurança e ainda tem uma elevada perfomance tanto em software, quanto em hardware.

Introdução

- Nosso trabalho consiste de implementar o CLEFIA em software, na linguagem C.
- Foi implementado todas as versões de tamanho de chave (128, 192 e 256 bits)
- Foi implementado o modo de operação CBC (Cipher-block chaining)
- Foram realizados testes de cifragens e decifragens de arquivos grandes, utilizando o modo CBC.
- Os mesmos testes foram realizados com o cifrador AES também no modo CBC, disponível na biblioteca OpenSSL. Dessa forma, foi possível comparar os resultados.

Generalized Feistel Network 4-branch

$$\mathbf{GFN}_{4,r}(\mathbf{RK}_0, \ldots, \mathbf{RK}_{2r-1}, \mathbf{X}_0, \mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3)$$

Entrada:	32-bit round keys	$\mathrm{RK}_0, \ldots, \mathrm{RK}_{2r-1}$
	32-bit data	$\mathrm{X}_0, \mathrm{X}_1, \mathrm{X}_2, \mathrm{X}_3$
Saída:	32-bit data	$\mathrm{Y}_0, \mathrm{Y}_1, \mathrm{Y}_2, \mathrm{Y}_3$

Passo 1.	$\mathrm{T}_0 \mid T_1 \mid T_2 \mid T_3 \leftarrow X_0 \mid X_1 \mid X_2 \mid X_3$
Passo 2.	Para i $= 0$ até $r - 1$ faça:
Passo 2.1.	$\mathrm{T}_1 \leftarrow T_1 \oplus \mathrm{F0}(\mathrm{RK}_{2\mathrm{i}}, \mathrm{T}_0)$,
	$\mathrm{T}_3 \leftarrow T_3 \oplus \mathrm{F1}(\mathrm{RK}_{2\mathrm{i}+1}, \mathrm{T}_2)$
Passo 2.2.	$\mathrm{T}_0 \mid T_1 \mid T_2 \mid T_3 \leftarrow T_1 \mid T_2 \mid T_3 \mid T_0$
Passo 3.	$\mathrm{Y}_0 \mid Y_1 \mid Y_2 \mid Y_3 \leftarrow T_3 \mid T_0 \mid T_1 \mid T_2$

Generalized Feistel Network 8-branch

$$\mathbf{GFN_{8,r}(RK_0, \ldots, RK_{4r-1}, X_0, X_1, \ldots, X_7)}$$

Entrada :	32-bit round keys	$RK_0, \ldots, RK_{4r-1}$,
	32-bit data	$X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7$,
Saída:	32-bit data	$Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, Y_7$

Passo 1. $T_0 \mid T_1 \mid \ldots \mid T_7 \leftarrow X_0 \mid X_1 \mid \ldots \mid X_7$

Passo 2. Para i $= 0$ até $r - 1$ faça:

Passo 2.1.
$T_1 \leftarrow T_1 \oplus F0(RK_{4i}, T_0)$,
$T_3 \leftarrow T_3 \oplus F1(RK_{4i+1}, T_2)$,
$T_5 \leftarrow T_5 \oplus F0(RK_{4i+2}, T_4)$,
$T_7 \leftarrow T_7 \oplus F1(RK_{4i+3}, T_6)$

Passo 2.2. $T_0 \mid T_1 \mid \ldots \mid T_6 \mid T_7 \leftarrow T_1 \mid T_2 \mid \ldots \mid T_7 \mid T_0$

Passo 3. $Y_0 \mid Y_1 \mid \ldots \mid Y_6 \mid Y_7 \leftarrow T_7 \mid T_0 \mid \ldots \mid T_5 \mid T_6$

Generalized Feistel Network 4-branch inversa

GFNINV$_{4,\mathrm{r}}$**(RK**$_0$**, . . . , RK**$_{2\mathrm{r}-1}$**, X**$_0$**, X**$_1$**, X**$_2$**, X**$_3$**)**

Entrada:	32-bit round keys	$\mathrm{RK}_0, \ldots, \mathrm{RK}_{2\mathrm{r}-1}$,
	32-bit data	$\mathrm{X}_0, \mathrm{X}_1, \mathrm{X}_2, \mathrm{X}_3$,
Saída:	32-bit data	$\mathrm{Y}_0, \mathrm{Y}_1, \mathrm{Y}_2, \mathrm{Y}_3$

Passo 1.	$\mathrm{T}_0 \mid T_1 \mid T_2 \mid T_3 \leftarrow X_0 \mid X_1 \mid X_2 \mid X_3$
Passo 2.	Para i $= 0$ até $r - 1$ faça:
Passo 2.1.	$\mathrm{T}_1 \leftarrow T_1 \oplus \mathrm{F0}(\mathrm{RK}_{2(\mathrm{r}-\mathrm{i}}, \mathrm{T}_0)$,
	$\mathrm{T}_3 \leftarrow T_3 \oplus \mathrm{F1}(\mathrm{RK}_{2(\mathrm{r}-\mathrm{i}}, \mathrm{T}_2)$
Passo 2.2.	$\mathrm{T}_0 \mid T_1 \mid T_2 \mid T_3 \leftarrow T_3 \mid T_0 \mid T_1 \mid T_2$
Passo 3.	$\mathrm{Y}_0 \mid Y_1 \mid Y_2 \mid Y_3 \leftarrow T_1 \mid T_2 \mid T_3 \mid T_0$

Função F0

F0(RK, x)

Entrada : 32-bit round key RK, 32-bit data x,
Saída: 32-bit data y

Passo 1. $T \leftarrow RK \oplus x$

Passo 2. Let $T = T_0 \mid T_1 \mid T_2 \mid T_3$, com T_i de 8-bits,
$T_0 \leftarrow S0(T_0)$,
$T_1 \leftarrow S1(T_1)$,
$T_2 \leftarrow S0(T_2)$,
x

Passo 3. Let $y = y_0 \mid y_1 \mid y_2 \mid y_3$, com y_i de 8-bits,
$y \leftarrow M0trans((T_0, T_1, T_2, T_3))$

Função F1

F1(RK, x)

Entrada : 32-bit round key RK, 32-bit data x,
Saída: 32-bit data y

Passo 1. $T \leftarrow RK \oplus x$

Passo 2. Let $T = T_0 \mid T_1 \mid T_2 \mid T_3$, com T_i de 8-bits,
$T_0 \leftarrow S1(T_0)$,
$T_1 \leftarrow S0(T_1)$,
$T_2 \leftarrow S1(T_2)$,
$T_3 \leftarrow S0(T_3)$

Passo 3. Let $y = y0 \mid y1 \mid y2 \mid y3, comy_i$ de 8-bits,
$y \leftarrow M1trans((T_0, T_1, T_2, T_3))$

S-box S0

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.a	.b	.c	.d	.e	.f
0.	57	49	d1	c6	2f	33	74	fb	95	6d	82	ea	0e	b0	a8	1c
1.	28	d0	4b	92	5c	ee	85	b1	c4	0a	76	3d	63	f9	17	af
2.	bf	a1	19	65	f7	7a	32	20	06	ce	e4	83	9d	5b	4c	d8
3.	42	5d	2e	e8	d4	9b	0f	13	3c	89	67	c0	71	aa	b6	f5
4.	a4	be	fd	8c	12	00	97	da	78	e1	cf	6b	39	43	55	26
5.	30	98	cc	dd	eb	54	b3	8f	4e	16	fa	22	a5	77	09	61
6.	d6	2a	53	37	45	c1	6c	ae	ef	70	08	99	8b	1d	f2	b4
7.	e9	c7	9f	4a	31	25	fe	7c	d3	a2	bd	56	14	88	60	0b
8.	cd	e2	34	50	9e	dc	11	05	2b	b7	a9	48	ff	66	8a	73
9.	03	75	86	f1	6a	a7	40	c2	b9	2c	db	1f	58	94	3e	ed
a.	fc	1b	a0	04	b8	8d	e6	59	62	93	35	7e	ca	21	df	47
b.	15	f3	ba	7f	a6	69	c8	4d	87	3b	9c	01	e0	de	24	52
c.	7b	0c	68	1e	80	b2	5a	e7	ad	d5	23	f4	46	3f	91	c9
d.	6e	84	72	bb	0d	18	d9	96	f0	5f	41	ac	27	c5	e3	3a
e.	81	6f	07	a3	79	f6	2d	38	1a	44	5e	b5	d2	ec	cb	90
f.	9a	36	e5	29	c3	4f	ab	64	51	f8	10	d7	bc	02	7d	8e

S-box S1

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.a	.b	.c	.d	.e	.f
0.	6c	da	c3	e9	4e	9d	0a	3d	b8	36	b4	38	13	34	0c	d9
1.	bf	74	94	8f	b7	9c	e5	dc	9e	07	49	4f	98	2c	b0	93
2.	12	eb	cd	b3	92	e7	41	60	e3	21	27	3b	e6	19	d2	0e
3.	91	11	c7	3f	2a	8e	a1	bc	2b	c8	c5	0f	5b	f3	87	8b
4.	fb	f5	de	20	c6	a7	84	ce	d8	65	51	c9	a4	ef	43	53
5.	25	5d	9b	31	e8	3e	0d	d7	80	ff	69	8a	ba	0b	73	5c
6.	6e	54	15	62	f6	35	30	52	a3	16	d3	28	32	fa	aa	5e
7.	cf	ea	ed	78	33	58	09	7b	63	c0	c1	46	1e	df	a9	99
8.	55	04	c4	86	39	77	82	ec	40	18	90	97	59	dd	83	1f
9.	9a	37	06	24	64	7c	a5	56	48	08	85	d0	61	26	ca	6f
a.	7e	6a	b6	71	a0	70	05	d1	45	8c	23	1c	f0	ee	89	ad
b.	7a	4b	c2	2f	db	5a	4d	76	67	17	2d	f4	cb	b1	4a	a8
c.	b5	22	47	3a	d5	10	4c	72	cc	00	f9	e0	fd	e2	fe	ae
d.	f8	5f	ab	f1	1b	42	81	d6	be	44	29	a6	57	b9	af	f2
e.	d4	75	66	bb	68	9f	50	02	01	3c	7f	8d	1a	88	bd	ac
f.	f7	e4	79	96	a2	fc	6d	b2	6b	03	e1	2e	7d	14	95	1d

Matrizes de Difusão M0

- O sinal * indica multiplicação em GF(2^8) onde o polinômio primitivo é $z^8 + z^4 + z^3 + z^2 + 1$.

y = M0 trans((T_0, T_1, T_2, T_3)):

$$
\begin{aligned}
y0 &= T_0 \oplus (0x02 * T_1) \oplus (0x04 * T_2) \oplus (0x06 * T_3) \\
y1 &= (0x02 * T_0) \oplus T_1 \oplus (0x06 * T_2) \oplus (0x04 * T_3) \\
y2 &= (0x04 * T_0) \oplus (0x06 * T_1) \oplus T_2 \oplus (0x02 * T_3) \\
y3 &= (0x06 * T_0) \oplus (0x04 * T_1) \oplus (0x02 * T_2) \oplus T_3
\end{aligned}
$$

Matrizes de Difusão M1

$y = M1\ \text{trans}((T_0, T_1, T_2, T_3))$:

$$
\begin{array}{lllllllll}
y0 = & T_0 & \oplus & (0x08 * T_1) & \oplus & (0x02 * T_2) & \oplus & (0x0a * T_3) \\
y1 = & (0x08 * T_0) & \oplus & T_1 & \oplus & (0x0a * T_2) & \oplus & (0x02 * T_3) \\
y2 = & (0x02 * T_0) & \oplus & (0x0a * T_1) & \oplus & T_2 & \oplus & (0x08 * T_3) \\
y3 = & (0x0a * T_0) & \oplus & (0x02 * T_1) & \oplus & (0x08 * T_2) & \oplus & T_3
\end{array}
$$

Algoritmo de encriptação

Entrada:	bloco plain :	$P = \{P_0, P_1, P_2, P_3\}$
	numero de rounds :	r tem 32 bits
	rounds keys :	$RK = \{RK_0, \ldots, RK_{2r-1}\}$
	whitening keys :	$WK = \{WK_0, WK_1, WK_2, WK_3\}$
Saída:	bloco cifrado :	$C = \{C_0, C_1, C_2, C_3\}$

Passo 1. $T_0 \mid T_1 \mid T_2 \mid T_3 \leftarrow P0 \mid (P1 \oplus WK0) \mid P2 \mid (P3 \oplus WK1)$

Passo 2. $T_0 \mid T_1 \mid T_2 \mid T_3 \leftarrow GFN_{4,r}(RK_0, \ldots, RK_{2r-1}, T_0, T_1, T_2, T_3)$

Passo 3. $C0 \mid C1 \mid C2 \mid C3 \leftarrow T_0 \mid (T_1 \oplus WK2) \mid T_2 \mid (T_3 \oplus WK3)$

Algoritmo de desencriptação

Entrada:	bloco plain :	$C = \{C_0, C_1, C_2, C_3\}$
	numero de rounds :	r tem 32 bits
	rounds keys :	$RK = \{RK_0, \ldots, RK_{2r-1}\}$
	whitening keys :	$WK = \{WK_0, WK_1, WK_2, WK_3\}$
Saída:	bloco cifrado :	$P = \{P_0, P_1, P_2, P_3\}$

Passo 1. $T_0 \mid T_1 \mid T_2 \mid T_3 \leftarrow C_0 \mid (C_1 \oplus WK2) \mid C_2 \mid (C_3 \oplus WK3)$

Passo 2. $T_0 \mid T_1 \mid T_2 \mid T_3 \leftarrow GFNINV_{4,r}(RK_0, \ldots, RK_{2r-1}, T_0, T_1, T_2, T$

Passo 3. $P_0 \mid P_1 \mid P_2 \mid P_3 \leftarrow T_0 \mid (T_1 \oplus WK0) \mid T_2 \mid (T_3 \oplus WK1)$

Geração das chaves

- Agora apresentaremos como são geradas as rounds keys e whitening keys usadas para encriptar e desencriptar.
- Essas chaves são geradas a partir da chave K, com os seguintes passos:
- Gera L a partir de K
- Expande K e L, gerando WK e RK
- Para gerar L a partir de K, é usado um 4-branch GFN com 12 rounds se K for de 128 bits, ou é usado um 8-branch GFN com 10 rounds se K for de 192 ou 256 bits.

Função de dupla Troca (SIGMA)

$$
\begin{aligned}
Y &= \text{Sigma}(X) \\
&= X[7\text{-}63] \mid X[121-127] \mid X[0-6] \mid X[64-120]
\end{aligned}
$$

Geração de chaves para o modo 128 bits

Entrada: chave : $K = \{k_0, k_1, K_2, k_3\}$
Saida: rounds keys : $RK = \{RK_0, \ldots, RK_{35}\}$
whitening keys : $WK = \{WK_0, WK_1, WK_2, WK_3\}$

Passo 1. $L \leftarrow GFN_{4,12}(CON_{128}\{0\}, \ldots, CON_{128}\{23\}, K0, \ldots, K3)$
Passo 2. $WK0 \mid WK1 \mid WK2 \mid WK3 \leftarrow K$
Passo 3. For i = 0 to 8 do the following:
$T \leftarrow L \oplus (CON_{128}[24 + 4i] \mid CON_{128}[24 + 4i + 1] \mid CON_{128}[24 +$
$L \leftarrow Sigma(L)$
if i is odd: $T \leftarrow T \oplus K$
$RK_{4i} \mid RK_{4i+1} \mid RK_{4i+2} \mid RK_{4i+3} \leftarrow T$

Geração de chaves para o modo 192 e 256 bits

Entrada para K = 192:	chave : k = $\{k_0, k_1, K_2, k_3, k_4, k_5\}$
Saida para K = 192:	rounds keys : RK = $\{RK_0, \ldots, RK_{43}\}$
	whitening keys : WK = $\{WK_0, WK_1, WK_2, WK_3\}$
Entrada para K = 256:	chave : k = $\{k_0, k_1, K_2, k_3, k_4, k_5, k_6, k_7\}$
Saida para k = 256:	rounds keys : RK = $\{RK_0, \ldots, RK_{51}\}$
	whitening keys : WK = $\{WK_0, WK_1, WK_2, WK_3\}$
Passo 1.	k = 192 ou k = 256
Passo 2.	If k = 192:
	else if k = 256 :

Geração de chaves para o modo 192 e 256 bits

Passo 3. Let KL = KL0 | $KL1$ | $KL2$ | $KL3$
KR = KR0 | $KR1$ | $KR2$ | $KR3$
LL | $LR \leftarrow GFN_{8,10}(\text{CON}_k\{0\}, \ldots, \text{CON}_k\{39\}, \text{KL0}, \ldots, \text{KL3}, \text{K}$

Passo 4. WK0 | $WK1$ | $WK2$ | $WK3 \leftarrow KL \oplus$ KR

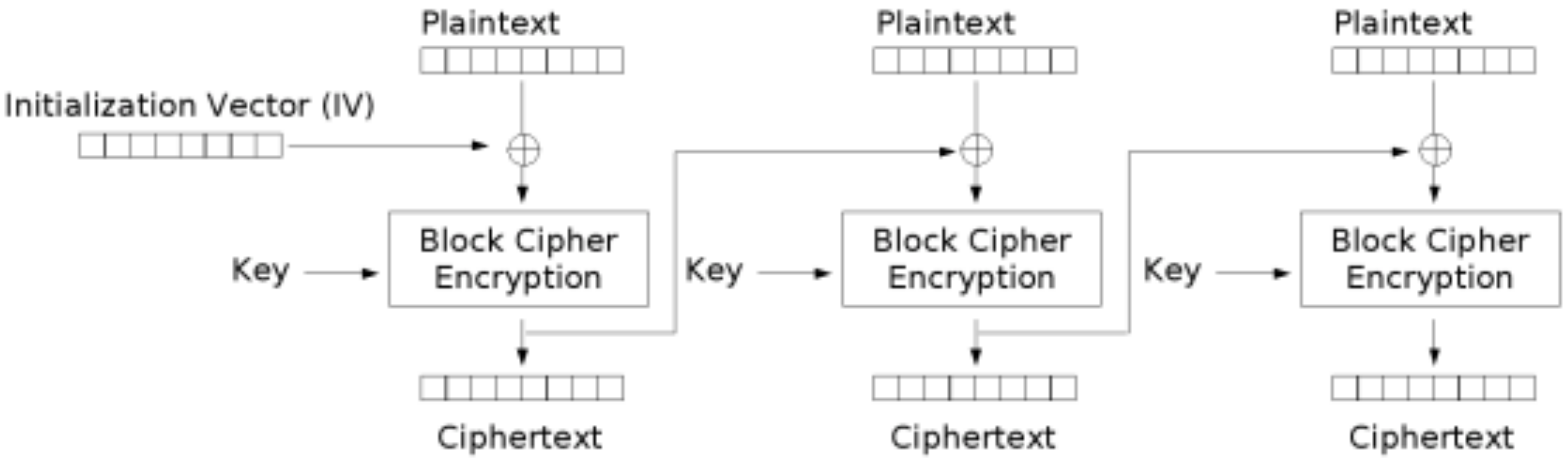
Passo 5. For i = 0 to 10 (if k = 192),
or 12 (if k = 256) do the following:
If (i mod 4) = 0 or 1:
T $\leftarrow LL \oplus$ (CON_k[40 + 4i] | CON_k[40 + 4i + 1] | CON_k[40 +
LL $\leftarrow Sigma(LL)$
if i is odd:
T $\leftarrow T \oplus$ KR
else:
T $\leftarrow LR \oplus$ (CON_k[40 + 4i] | CON_k[40 + 4i + 1] | CON_k[40 +
LR $\leftarrow Sigma(LR)$
if i is odd: T $\leftarrow T \oplus$ KL

$$\text{RK}_{4\text{i}} \mid RK_{4\text{i}+1} \mid RK_{4\text{i}+2} \mid RK_{4\text{i}+3} \leftarrow T$$

Modo de operação CBC

- Como CLEFIA é um cifrador de blocos de comprimento fixo, é necessário utilizar um modo de operação para que cifre mensagens de qualquer compriment
- Para realizar testes cifrando mensagens longas, foi necessário implementar um modo de operação
- Optamos pelo CBC, devido a sua simplicidade tanto para encriptar quanto para desencriptar.
- CBC é bastante utilizado, tendo inclusive na biblioteca openSSL com o cifrador AES.

Modo de Operação CBC



Cipher Block Chaining (CBC) mode encryption

Figura: Cifragem no modo CBC

Modo de Operação CBC

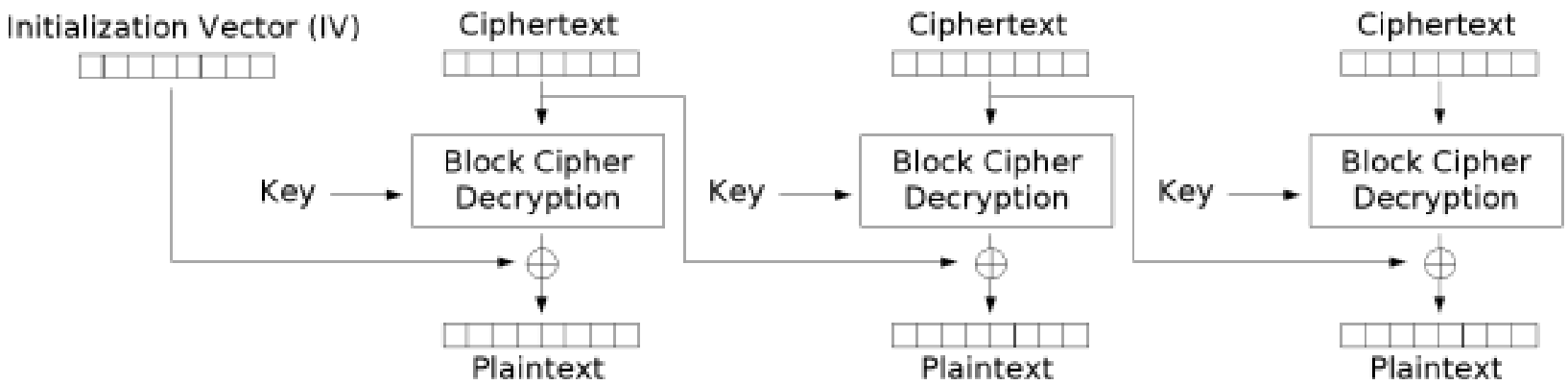


Figura: Decifragem no modo CBC

Resultados experimentais

- Com o intuito de compararar o algoritmo Clefia com o AES, fizemos vários testes de cifragem e decifragem. Os tempos obtidos aparecem nas tabelas abaixo
- É óbvio que a implementação do AES da biblioteca OpenSSL é muito rápida, já que o código aberto vem sendo testado e otimizado há bastante tempo
- Nosso código, por outro lado, foi escrito por duas pessoas apenas e sem o tempo necessário para fazer otimizações mais profundas no código
- Apesar disso, a primeira implementação era 10% mais lenta. O que fizemos para acelerar um pouco a execução foram pequenas otimizações de código, como *loop unrolling* de alguns laços e trocar funções por macros, que são substituídas em tempo de compilação. Acreditamos que com mais tempo e mais otimizações poderíamos deminuir drasticamente os tempos apresentados

Resultados experimentais

- O arquivo cifrado é uma imagem de uma distribuição Linux, de 200278016 bytes.

Cifrar	AES-128 (s)	Clefia-128 (s)
Teste 1	1.996	31.778
Teste 2	1.933	31.783
Teste 3	2.030	31.773
Teste 4	1.933	31.782
Teste 5	2.026	31.782
Teste 6	2.059	31.786
Teste 7	2.078	31.791
Teste 8	2.124	31.785
Teste 9	1.948	31.904
Teste 10	1.981	31.775
Média	2.0108	31.7939
Desvio Padrão	0.0644	0.0390

Resultados experimentais

Decifrar	AES-128 (s)	Clefia-128 (s)
Teste 1	2.081	33.233
Teste 2	2.025	33.256
Teste 3	2.059	33.261
Teste 4	2.065	33.236
Teste 5	2.051	33.268
Teste 6	2.021	33.499
Teste 7	2.044	33.13
Teste 8	2.029	33.034
Teste 9	2.034	33.335
Teste 10	2.049	33.243
Média	2.0458	33.2495
Desvio Padrão	0.0191	0.1207

Referências

www.faqs.org/rfcs/rfc6114.html#
www.sony.net/Products/cryptography/clefia/?j-short=clefia
www.sony.net/Products/cryptography/clefia/download/data/clefia-d
William Stallings - Cryptography and Network Security,
4th Edition