Universidade de Brasília

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Modelagem de Sistemas em Silício

Documento do Projeto
IDEA

Grupo:
Lucas Avelino - 13/0013072
Lucas Santos - 14/0151010

Professor:
Ricardo Jacobi

29 de junho de 2017

1 Introdução

1.1 Contexto

O *SystemC* é uma biblioteca de classes e macros para *C++* que fornecem uma interface de simulação baseada em eventos. Permite uma modelagem em nível de sistema, mas com mecanismos que também se aproximam de uma linguagem de descrição de *hardware*.

Através da ferramenta *SystemC* sistemas de um alto grau de complexidade onde existe a presença de diversos módulos distintos que se comunicam a taxas diferentes podem ser modelados em um nível de abstração que permite descrevê-los de forma clara e simples, mas ao mesmo tempo, que simulam todo o seu comportamento.

1.2 Níveis de abstração de um projeto de hardware

Os níveis de abstração são basicamente o vocabulário que o projetista de *hardware* utiliza para descrever o projeto em construção. No projeto de *hardware* existem vários níveis de abstração distintos: nível de transistor, nível de portas lógicas, *Register-Transfer Level (RTL)*, comportamental, transacional, entre outros.

O nível RTL é um dos mais utilizados por vários motivos, inclusive por ser possível de realizar o processo de síntese lógica em um software compilador e transformá-lo em uma implementação em termos de portas lógicas. Entretanto, é notável que a modelagem RTL tem se mostrado muito baixo nível para o tamanho dos sistemas que estão sendo fabricados.

1.2.1 O que é uma descrição em nível transacional - TLM ?

O nível de abstração transacional (*Transaction Level Modeling - TLM*) é uma abordagem alto-nível para modelagem de sistemas digitais onde os detalhes de comunicação entre os módulos são separados dos detalhes de implementação dos blocos funcionais e da arquitetura de comunicação.

Enquanto o nível RTL é caracterizado por operações aritméticas, estruturas de controle condicionais, registradores, sinais e por atividades a cada ciclo de clock bem definidas, no nível transacional, existem os objetos transacionais que nada mais são do que uma abstração de interfaces de comunicação entre módulos, que permite uma abstração maior do sistema como um todo.

1.3 Sistemas em Sílicio

Um sistema em silício é um circuito integrado (CI) que contém um ou mais núcleos de processamento, tais como: microprocessadores (MPUs), microcontroladores (MCUs) e processadores digitais de sinais (DSPs) bem como memória, aceleradores de funções por hardware, funções de periféricos, dentre outros, em uma única pastilha de silício.

Tais sistemas são difícilmente modelados em nível RTL devido a sua complexidade elavada. Entretanto, através da ferramenta *SystemC* é possível modelar esses sistemas em nível transacional sem se preocupar em definir todas as operações a serem executadas em termos de ciclos de clock.

Este trabalho visa exercitar os conceitos de modelagem de sistemas em silício aprendidos ao longo do semestre através da implementação de um módulo de criptografia e sua integração a uma *Network-on-Chip* no ambiente de simulação do *SystemC*.

2 Motivação

A importância de realizar este projeto se deve à familiarização dos membros do grupo tanto com um algoritmo de criptografia/decriptografia (*IDEA*) quanto com uma *Network on Chip*, que é uma tecnologia com nótaveis melhorias em relação à conexão por barramento, comumente utilizada, melhorando também a escalabilidade e a eficiência de energia dos *System on Chip*. O aprendizado que será obtido pela elaboração deste projeto é relevante nas áreas de *System on Chip*, segurança de dados, comunicações *on-chip* e teoria de redes.

3 Descrição do *IDEA*

IDEA (*International Data Encryption Algorithm*) é um algoritmo com chave secreta de 128 bits e tanto o texto legível (entrada) quanto o texto ilegível (saída) de 64 bits. Este algoritmo é conhecido publicamente desde 1991 e até agora não foram reveladas vulnerabilidades, mesmo após anos de criptoanálise feita por especialistas. O *IDEA* serve tanto para criptografar quanto para decriptografar, alterando apenas a forma de geração das *sub-chaves*.

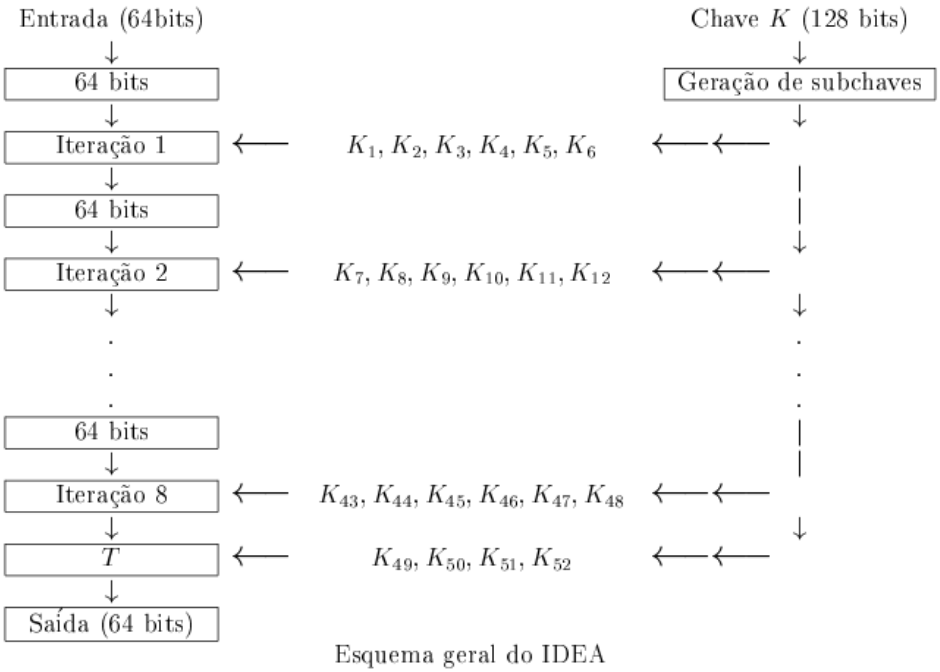

Figura 1: Esquema geral do *IDEA*.

A figura acima representa as fases de execução do *IDEA* de uma forma geral, onde são geradas 52 sub-chaves de 16 bits, que são utilizadas durante 8 iterações de um *round* e uma última iteração de um *half-round*. Após tais fases a saída pode ser tanto legível quanto ilegível, dependendo da forma de geração das sub-chaves, como explicado anteriormente.

3.1 Estrutura do *IDEA*

A estrutura do *IDEA* possui um número fixo de *rounds* de uma mesma função que utiliza sub-chaves distintas. Estes rounds são compostos por três operações distintas de três grupos algébricos sobre dois bytes (16 bits):

1. *Ou-exclusivo (XOR)*, sobre dois bytes (16 bits), representado pelo símbolo $\oplus$;
2. *Soma módulo* 2^{16}, que é equivalente à soma habitual entre 16 bits sem *overflow*, representada pelo símbolo $\boxplus$;
3. *Multiplicação módulo* $2^{16}+1$, que realiza a multiplicação de dois operandos de dois bytes, representada pelo símbolo $\odot$, seguindo os seguintes passos:
   (a) Ao multiplicar tais operandos, é obtido no máximo um valor de 33 bits, sendo que se algum operando for 0, o mesmo deve ser alterado para 2^{16} e vice-versa;
   (b) Após a obtenção deste valor, deve-se calcular o resto da divisão por $2^{16}+1$;
   (c) Se o resultado for 2^{16}, ou seja, a operação causou um *overflow*, o resultado deve ser 0.

As três operações explicitadas acima são inversíveis, o que proporciona a decriptografia da mensagem anteriormente criptografada, possibilitando a utilização do *IDEA* tanto para criptografia quanto para decriptografia.

3.2 Geração das sub-chaves

A partir da chave secreta K de 128 bits são geradas 52 sub-chaves de 16 bits, denotadas por K_1, K_2, K_3, ..., K_{52}. A geração de todas as sub-chaves, para o processo de criptografia, serão ilustradas a seguir.

128 bits →	16	16	16	16	16	16	16	16
de K	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8

Figura 2: Geração das 8 primeiras sub-chaves.

As primeiras 8 sub-chaves são geradas simplesmente considerando os primeiros 16 bits da esquerda para a direita da chave secreta como sendo K_1 e assim por diante.

128 bits →	25	16	16	16	16	16	16	7
de K		K_9	K_{10}	K_{11}	K_{12}	K_{13}	K_{14}	parte K_{15}

9	16	16	16	16	16	16	16	7	← 128 bits
parte K_{15}	K_{16}								de K

Figura 3: Geração das 8 próximas sub-chaves.

As 8 sub-chaves seguintes são geradas de forma semelhante, porém com K_9 começando após 25 bits à direita do extremo esquerdo da chave secreta e assim por diante. Dessa forma, K_{15} terá apenas 7 bits por enquanto, os bits restantes de K_{15} começam no primeiro bit da chave secreta, o que equivale a deslocar circularmente para a esquerda a chave secreta de 25 bits, antes de iniciar a geração de cada uma das 8 sub-chaves, e K_9 começar no primeiro bit

128 bits →	50	16	16	16	16	14
K		K_{17}	K_{18}	K_{19}	K_{20}	parte K_{21}

2	16	16	16	16	16	16	16	14	← 128 bits
parte K_{21}	K_{22}	K_{23}	K_{24}						de K

As subchaves K_{25} a K_{32} são geradas como na figura a seguir:

128 bits →	75	16	16	16	5
de K		K_{25}	K_{26}	K_{27}	parte K_{28}

11	16	16	16	16	16	16	16	5	← 128 bits
parte K_{28}	K_{29}	K_{30}	K_{31}	K_{32}					de K

As subchaves K_{33} a K_{40} são geradas como na figura a seguir:

128 bits →	100	16	12
de K		K_{33}	parte K_{34}

4	16	16	16	16	16	16	16	12	← 128 bits
parte K_{34}	K_{35}	K_{36}	K_{37}	K_{38}	K_{39}	K_{40}			de K

As subchaves K_{41} a K_{48} são geradas como na figura a seguir:

128 bits →	125	3
de K		parte K_{41}

13	16	16	16	16	16	16	16	3	← 128 bits
parte K_{41}	K_{42}	K_{43}	K_{44}	K_{45}	K_{46}	K_{47}	K_{48}		de K

Figura 4: Geração das sub-chaves intermediárias.

As 8 sub-chaves seguintes são geradas de forma análoga mas K_{17} começando após 50 bits à direita do extremo esquerdo da chave secreta, K_{18} começando 66 bits à direita do extremo esquerdo da chave secreta e assim por diante. Isto equivale a ter deslocado circurlamente para a esquerda a chave secreta de 25 bits, além das 25 anteriores.

128 bits →	22	16	16	16	16	16	16	10
de K		K_{49}	K_{50}	K_{51}	K_{52}			

Figura 5: Geração das últimas sub-chaves.

3.3 Iterações

O *IDEA* possui 8 iterações ou *rounds* e uma última transformação, chamada de T ou *half-round*. Cada iteração utiliza seis sub-chaves e é divida em duas partes:

1. A primeira parte utiliza quatro sub-chaves K_a, K_b, K_c, K_d e uma entrada de 64 bits tratada como quatro sub-entradas de 16 bits X_a, X_b, X_c, X_d para obter as saídas X'_a, X'_b, X'_c, X'_d.
2. A segunda parte utiliza duas sub-chaves K_e e K_f que são operadas com as quatro saídas de 16 bits da primeira parte X_a, X_b, X_c, X_d para formar novos X'_a, X'_b, X'_c, X'_d.

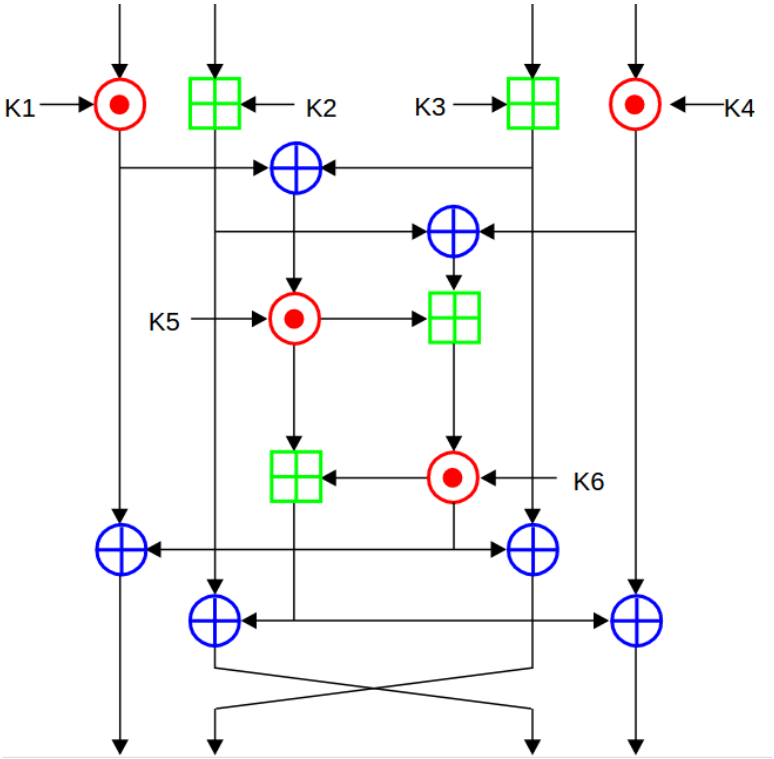

Figura 6: Um round do *IDEA*.

Após 8 repetições de um *round*, o resultado X'_a, X'_b, X'_c, X'_d é fornecido como entrada para a última transformação T, também conhecida como *half-round*, que utiliza as últimas quatro sub-chaves. As operações realizadas são idênticas às da primeira parte de cada iteração, com uma pequena diferença na ordem das chaves para as operações.

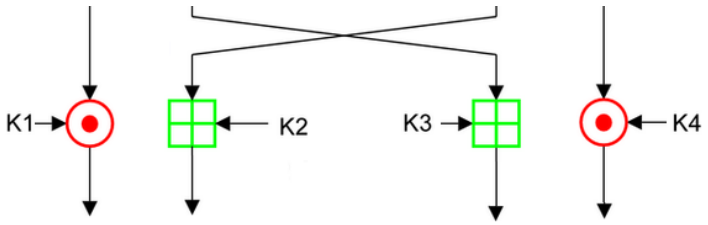

Figura 7: Round final (half-round) do *IDEA*.

3.4 Decriptografia

O IDEA foi feito de forma que o mesmo circuito ou software serve para criptografar ou decriptografar 64 bits de entrada. As três operações básicas do *IDEA* são facilmente inversíveis, e portanto não é complicado de se obter as subchaves inversas, Para realizar a decriptação é necessário:

1. Calcular previamente as sub-chaves inversas para a primeira parte de cada iteração do algorítmo;
2. A segunda parte de cada iteração é exatamente a mesma para encriptação e decriptação e portanto pode ser executada normalmente;
3. Inverter a ordem em que as chaves são utilizadas.

Para exemplificar o método descrito, as sub-chaves geradas para realizar a decriptografia de uma mensagem de 64 bits são explicitadas adiante. Para o primeiro *round* da decriptação as sub-chaves são:

- $K_{d1} = K_{49} - 1$
- $K_{d2} = -K_{50}$
- $K_{d3} = -K_{51}$
- $K_{d4} = K_{52} - 1$
- $K_{d5} = K_{47}$
- $K_{d6} = K_{48}$

Para a segundo *round* da decriptação, as sub-chaves são:

- $K_{d7} = K_{43} - 1$
- $K_{d8} = -K_{44}$
- $K_{d9} = -K_{45}$
- $K_{d10} = K_{46} - 1$
- $K_{d11} = K_{41}$
- $K_{d12} = K_{42}$

O processo demonstrado é repetido mais seis vezes, uma para cada *round*, adicionando 6 para cada índice de sub-chave de decriptografia e subtraindo 6 para cada índice de sub-chave de criptografia.

4 Interface com a *Network on Chip*

O modelo de interface de rede usado na *NoC* utiliza dois módulos para cada módulo conectado, os módulos *Kernel* e *Shell*. O módulo Kernel implementa o empacotamento e o desempacotamento dos dados provenientes do módulo e da NoC, respectivamente (O empacotamento é realizado pelo codificador e o desempacotamento é realizado pelo decodificador). O módulo Kernel é comum para todos os módulos da NoC, enquanto o módulo Shell é específico de cada módulo, adaptando os sinais de dados, controle e endereços do módulo para um formato padrão que será transmitido para outro módulo por meio da NoC.

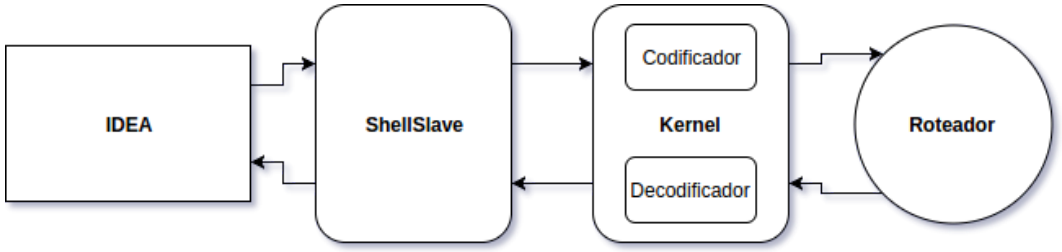

Figura 8: Representação da conexão do módulo *IDEA* com a *NoC*.

4.1 Endereços para acesso ao *IDEA*

- Saída do módulo *IDEA* (64 bits):
  - 0xFFFF0040 - *cifra_out_h* (32 a 63);
  - 0xFFFF0044 - *cifra_out_l* (0 a 31);
- Entrada do módulo *IDEA* (64 bits):
  - 0xFFFF0048 - *cifra_in_h* (32 a 63);
  - 0xFFFF004C - *cifra_in_l* (0 a 31);
- Chave secreta para geração de sub-chaves (128 bits):
  - 0xFFFF0050 - *chave_128_3* (96 a 127);
  - 0xFFFF0054 - *chave_128_2* (64 a 95);
  - 0xFFFF0058 - *chave_128_1* (32 a 63);
  - 0xFFFF005C - *chave_128_0* (0 a 31);
- Registrador de comandos (32 bits):
  - 0xFFFF0060 – *DMA_CMD*.

4.2 Comandos para o *IDEA*

- **IDEA_IDLE**: módulo *IDEA* disponível;
- **IDEA_BKEYS**: gerar as chaves a partir da chave secreta de 128 bits. O módulo deve gerar as chaves diretas e inversas (decifragem);
- **IDEA_CYPHER**: cifrar uma palavra de 64 bits com os parâmetros fornecidos, colocando o resultado em *cifra_out_h* e *cifra_out_l*;
- **IDEA_DECYPHER**: decifrar uma palavra de 64 bits com os parâmetros fornecidos, colocando o resultado em *cifra_out_h* e *cifra_out_l*.

5 Metodologia

O módulo será implementado em *C++* utilizando a biblioteca do *SystemC*, para controle de versão do projeto, o *GitHub* será utilizado. Conforme demonstrado na figura acima, o módulo terá um conjunto de nove registradores, sendo eles:

1. *CMD*: Registrador de 32 bits que armazenará os comandos dados ao módulo;
2. *W0*: Registrador de 16 bits que possui o quarto menos significante da palavra a ser operada;
3. *W1*: Registrador de 16 bits que possui o segundo quarto menos significante da palavra a ser operada;
4. *W2*: Registrador de 16 bits que possui o segundo quarto mais significante da palavra a ser operada;

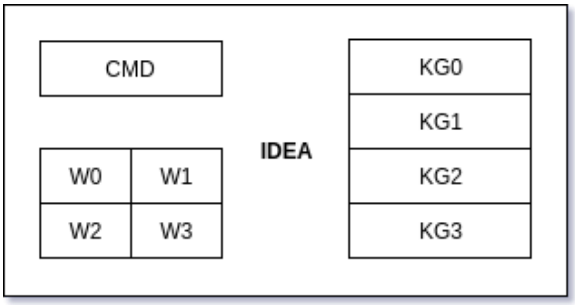

Figura 9: Módulo que representa o *IDEA* implementado como um módulo em *SystemC*.

5. *W3*: Registrador de 16 bits que possui o quarto mais significante da palavra a ser operada;
6. *KG0*: Registrador de 32 bits que possui o quarto mais significante da chave secreta que será utilizada para a geração das sub-chaves;
7. *KG1*: Registrador de 32 bits que possui o quarto mais significante da chave secreta que será utilizada para a geração das sub-chaves;
8. *KG2*: Registrador de 32 bits que possui o quarto mais significante da chave secreta que será utilizada para a geração das sub-chaves;
9. *KG3*: Registrador de 32 bits que possui o quarto mais significante da chave secreta que será utilizada para a geração das sub-chaves.

Tendo o módulo descrito com os registradores apresentados acima, as principais tarefas que seguem são:

1. Implementação da estrutura do módulo em *SystemC* (Início do projeto);
2. Implementação da geração das sub-chaves, tanto para criptografia quanto para decriptografia;
3. Implementação das operações dos *rounds*;
4. Implementação dos *rounds* em si;
5. Realização de testes no módulo completo;
6. Implementação da *Shell*;
7. Conexão com a *NoC* (Fim do projeto).

Após a implementação das estrutura do módulo estar completa, as seguintes tarefas relacionadas com a funcionalidade do módulo serão concebidas por meio de funções da classe do módulo. Os membros do grupo definiram a separação de tarefas da seguinte forma: O membro Lucas Avelino é responsável pelas tarefas 2,5,6,7; O membro Lucas Santos é responsável pelas tarefas 1,3,4,5,6,7. Ambos os membros irão trabalhar na etapa final do projeto, onde serão realizados os testes do módulo desenvolvido e também a realização da conexão com a *NoC*.

6 Verificação do Sistema

Para verificar a funcionalidade do módulo, as duas tabelas a seguir serão utilizadas para garantir a corretude das operações realizadas pelo *IDEA*. além disso, durante a própria implementação de testes temporários serão realizados por meio de *defines* no código. Os passos que serão executados para a execução dos testes serão os seguintes:

1. Testes no decorrer da implementação das operações e iterações;
2. Ao final da implementação, os valores de entrada descritos nas duas tabelas serão utilizados para a verificação do módulo;
3. A cada iteração so valores serão comparados aos valores das tabelas, certificando a igualdade dos valores comparados;
4. Finalmente, após o sucesso dos testes, o sistema será comprovado como funcional.

	Chave $K = (1, 2, 3, 4, 5, 6, 7, 8)$						Entrada $(0, 1, 2, 3)$			
	(128 bits)						(64 bits)			
iter.	K_a	K_b	K_c	K_d	K_e	K_f	X_a	X_b	X_c	X_d
1	0001	0002	0003	0004	0005	0006	00f0	00f5	010a	0105
2	0007	0008	0400	0600	0800	0a00	222f	21b5	f45e	e959
3	0c00	0e00	1000	0200	0010	0014	0f86	39be	8ee8	1173
4	0018	001c	0020	0004	0008	000c	57df	ac58	c65b	ba4d
5	2800	3000	3800	4000	0800	1000	8e81	ba9c	f77f	3a4a
6	1800	2000	0070	0080	0010	0020	6942	9409	e21b	1c64
7	0030	0040	0050	0060	0000	2000	99d0	c7f6	5331	620e
8	4000	6000	8000	a000	c000	e001	0a24	0098	ec6b	4925
T	0080	00c0	0100	0140	–	–	11fb	ed2b	0198	6de5

Figura 10: Tabela de resultados esperados 1.

	Chave $K = (1, 2, 3, 4, 5, 6, 7, 8)$						Entrada de 64 bits			
	(128 bits)						$(11fb, ed2b, 0198, 6de5)$			
iter.	K_a	K_b	K_c	K_d	K_e	K_f	X_a	X_b	X_c	X_d
1	fe01	ff40	ff00	659a	c0000	e001	d98d	d331	27f6	82b8
2	fffd	8000	a000	cccc	0000	2000	bc4d	e26b	9449	a576
3	a556	ffb0	ffc0	52ab	0010	0020	0aa4	f7ef	da9c	24e3
4	554b	ff90	e000	fe01	0800	1000	ca46	fe5b	dc58	116d
5	332d	c800	d000	fffd	0008	000c	748f	8f08	39ds	45cc
6	4aab	ffe0	ffe4	c001	0010	0014	3266	045e	2fb5	b02e
7	aa96	f000	f200	ff81	0800	0a00	0690	050a	00fd	1dfa
8	4925	fc00	fff8	552b	0005	0006	0000	0005	0003	000c
T	0001	fffe	fffd	c001	–	–	0000	0001	0002	0003

Figura 11: Tabela de resultados esperados 2.

Para a verificação da decriptografia, uma mensagem será encriptada e logo após decriptada e o resultado obtido deverá ser o mesmo da entrada inicial a este teste.

Referências

1. ivansarno - IDEA-cipher;
2. Roudo Terada (DCC-USP) 1999 - IDEA;
3. Quadibloc - IDEA;
4. Aldeia Numaboa - O algoritmo IDEA ilustrado;
5. Material disponibilizado pelo professor Ricardo Jacobi.