INSTITUTO MILITAR DE ENGENHARIA

Cap JORGE DE ALBUQUERQUE LAMBERT

CIFRADOR SIMÉTRICO DE BLOCOS: PROJETO E AVALIAÇÃO

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção de título de Mestre em Ciências em Sistemas e Computação.

Orientador: Prof. José Antônio Moreira Xexéo, D. Sc.
Co-orientador: Prof. Alex de Vasconcellos Garcia, D. Sc.

Rio de Janeiro

2004

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80 – Praia Vermelha
Rio de Janeiro - RJ CEP: 22290-270

652.8 Lambert, Jorge de Albuquerque.
L 222 Cifrador simétrico de blocos: projeto e avaliação / Jorge de Albuquerque Lambert – Rio de Janeiro : Instituto Militar de Engenharia, 2004.
385 p. : il., tab.

Dissertação (mestrado): Instituto Militar de Engenharia Rio de Janeiro, 2004.

1. Criptografia. 2. Criptografia simétrica. 3. Algoritmo. 4. Cifra de blocos. 5. Algoritmos, projeto e avaliação. I Instituto Militar de Engenharia. II. Título.

CDD 652.8

INSTITUTO MILITAR DE ENGENHARIA

Cap JORGE DE ALBUQUERQUE LAMBERT

CIFRADOR SIMÉTRICO DE BLOCOS: PROJETO E AVALIAÇÃO

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção de título de Mestre em Ciências em Sistemas e Computação.

Orientador: Prof. José Antônio Moreira Xexéo, Ten Cel R/1, D. Sc.

Co-orientador: Prof. Alex de Vasconcellos Garcia, D. Sc.

Aprovada em 5 de março de 2004 pela seguinte Banca Examinadora:

_____

Prof. José Antônio Moreira Xexéo, D. Sc. do IME – Presidente

_____

Prof. Alex de Vasconcellos Garcia, D. Sc. do IME

_____

Prof. Paulo Cesar Coelho Ferreira, Ph. D. do PRODERJ

Rio de Janeiro

2004

Aos meus familiares, pela formação que me proporcionaram e pelo incentivo que sempre me deram, com especial destaque para a minha avó, Carmelina Albuquerque, pelo exemplo de determinação e força de vontade que sempre foi para os irmãos, filhas e netos.

AGRADECIMENTOS

Ao Exército Brasileiro, pelas oportunidades que são oferecidas a todos os brasileiros nas suas principais escolas de formação, em especial na Escola Preparatória de Cadetes, na Academia Militar das Agulhas Negras e no Instituto Militar de Engenharia.

Ao professor Almir Paz de Lima, pela disponibilidade e paciência constantes e, "pela orientação segura e competente" para todos seus alunos desde há muitos anos (XEXEO, 1983), apesar de não estar oficialmente vinculado a esta dissertação de mestrado.

Ao meu principal orientador, Professor Xexéo, em primeiro lugar pela confiança que depositou aceitando ser meu orientador, apesar de não me conhecer e, principalmente, pela firmeza, competência, objetividade e tranqüilidade com que conduziu a orientação de todos os trabalhos realizados.

Ao professor Alex, por ter aceitado ser co-orientador deste trabalho e, principalmente, pelo exemplo de capacidade profissional demonstrado em todas as vezes que se pronunciou durante este período de dois anos de convivência.

Aos oficiais, coordenadores e professores do DE-9 por terem tornado possível, apesar de todas as dificuldades encontradas, a realização do estudo deste tema de interesse do Exército neste Departamento.

À minha namorada, Gisele, pela postura de compreensão, carinho e disponibilidade que sempre manteve durante o desenvolvimento deste trabalho.

"A segurança de um sistema criptográfico deve residir unicamente no segredo da chave, e não no sigilo do algoritmo."

Auguste Kerckhoffs
("*La Criptographie Militaire"*, 1883)

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

ABA	*American Bankers Association*
AES	*Advaced Encryption Standard*
ANSI	*American National Standards Institute*
ATM	*Automatic Teller Machine*
CAE	Critério de avalanche estrito
CD	Criptoanálise diferencial
CL	Criptoanálise linear
CG	Corpo de Galois
CPI	*Cyberspace Policy Institute*
DES	*Data Encryption Standard*
DPA	*Differential PowerAanalysis*
FIPS	*Federal Information Processing Standard*
HNC	*Hipótese do não cancelamento das dependências*
IBM	*International Business Machines*
KUL	Universidade Católica de Louvain – Bélgica
LIPA	Limite inferior do poder de avalanche
LSPA	Limite superior do poder de avalanche
MDE	Matriz(es) de dependências estatísticas
MDR	Matriz(es) de dependências recursivas
MN	Método dos nós
MQL	Método dos quadrados latinos
NBS	*National Bureau of Standards*
NESSIE	*New European Schemes for Signatures, Integrity and Encription*
NIST	*National Institute of Standards and Technology*

NSA	*National Security Agency*
QL	Quadrado(s) latino(s)
RACE	*Research and Development in Advanced Communication Technologies in Europe*
RIPE	*RACE Integrity Primitives Evaluation*
RS/R	Relação sinal/ruído
SLTN	*Substitution - linear transformation network*
SPN	*Substitution - permutation network*
TL	Transformação(ões) linear(es)
TNL	Transformação(ões) não-linear(es)

RESUMO

Neste trabalho é apresentado um estudo dos processos de seleção de padrões criptográficos realizados pelo Ministério do Comércio Norte Americano para os padrões de criptografia *Data Encryption Standard (DES)* e *Advanced Encryption Standard (AES)*. São estudadas as exigências feitas aos algoritmos candidatos, bem como os aspectos positivos e negativos de três destes algoritmos: o DES, o AES (Rijndael) e o Serpent;

Com base nos estudos supracitados, é apresentada uma filosofia para projeto de cifradores de blocos traduzida em cinco princípios básicos: *segurança, eficiência, simplicidade, flexibilidade, credibilidade.*

É apresentada uma descrição do Rijndael baseada apenas em operações lógicas, de modo que o algoritmo pode ser estudado e implementado de maneira mais simples, sem a necessidade explícita da álgebra de polinômios em corpos finitos.

É apresentada uma técnica para avaliação de transformações lineares sob a ótica do *efeito avalanche* – as *matrizes de avalanche*.

É apresentada uma técnica para construção de transformações lineares para cifradores de blocos capaz de produzir a *completude* da função criptográfica em um número de iterações menor que o necessário para os algoritmos DES, Rijndael e Serpent.

É apresentada uma técnica para construção e utilização de tabelas de substituição em cifradores tipo Feistel capaz de produzir boa resistência contra as criptoanálises *diferencial, linear* e *criptoanálise por entradas invariantes.*

É apresentado o cifrador *Alpha*, um cifrador de blocos de 128 bites tipo Feistel modificado construído dentro da filosofia de projeto apresentada. O cifrador *Alpha* utiliza chaves a partir de 128 bites.

São construídas as *matrizes de correlação linear* e *matrizes de distribuição de diferenças* para as tabelas de substituição apresentadas, com o objetivo de mostrar a sua resistência contra as criptoanálises *linear* e *diferencial*.

São executados no cifrador *Alpha* os principais testes realizados pelo *National Institute of Standards and Technology (NIST)* e pelo *New European Schemes for Signatures, Integrity and Encryption (NESSIE)* em seus processos de seleção de cifras de blocos.

ABSTRACT

In this work a study of the US Department of Commerce selection process for the standards DES and AES is presented. The design criteria specified by the National Institute of Standards and Technology are exploited as well as positive and negative features of three algorithms: *Data Encryption Standard(DES)*, *Advanced Encryption Standard (AES - Rijndael)* and *Serpent*.

Based on the study above, a block-cipher design philosophy is presented, which is expressed by a set of five basic design principles: *safety, efficiency, simplicity, flexibility, and credibility.*

A simple logical description of Rijndael is shown which allows the study and implementation of the algorithm without the usual finite field structures approach.

A technique to evaluate the *avalanche effect* of linear transformations is presented – the *avalanche matrix*.

A technique to build linear transformations for block ciphers is presented. This technique focuses the *completeness* of the linear layer of the cryptographic transformation in a number of rounds less than the required by the DES, Rijndael and Serpent algorithms.

A technique to build non-linear transformations for Feistel block ciphers is presented. The technique provides good resistance *against differential and linear cryptanalysis* and *invariant entries cryptanalysis.*

A 128 bits modified Feistel cipher, designed according to the suggested philosophy is presented: the block-cipher *Alpha*, with 128 bits key size (or greater).

The *correlation matrix* and the *differences distribution matrix* for the non-linear transformations are built to evaluate the *resistance against differential and linear cryptanalysis.*

Some of the basic tests adopted by the *National Institute of Standards and Technology (NIST-USA)* e pelo *New European Schemes for Signatures, Integrity and Encryption (NESSIE)* are executed on the cipher *Alpha*.

1 INTRODUÇÃO

1.1 CONSIDERAÇÕES INICIAIS

A arte e a ciência[1] de manter as mensagens seguras utilizando cifras ou códigos[2] é a *criptografia*. Seus praticantes são *criptólogos*. A arte e ciência de buscar o conteúdo de uma mensagem criptografada sem autorização chama-se *criptoanálise* e seus praticantes são os *criptoanalistas* . A área da matemática que abrange a criptografia e a criptoanálise é a *criptologia* (SCHNEIER, 1996, p. 1).

Informalmente, um *algoritmo criptográfico* é uma função que transforma uma mensagem (*texto em claro*) em um *criptograma* (*texto cifrado*). Esta transformação é dependente de uma chave que pode ser de conhecimento público ou não, dependendo da natureza do algoritmo em questão. A transformação inversa é necessariamente dependente de uma chave que só deve ser conhecida pela parte autorizada a tomar conhecimento da informação cifrada.

Os algoritmos criptográficos podem ser classificados quanto ao segredo da chave que é utilizada para cifrar. Alguns utilizam a mesma[3] chave para cifrar e decifrar. Por esta razão, são classificados como algoritmos *simétricos* ou algoritmos de *chave secreta*, uma vez que a chave destinada a cifrar deve ser obviamente mantida em segredo. Outros algoritmos utilizam duas chaves diferentes: uma para cifrar e outra para decifrar, sendo que a chave para decifração não pode ser obtida em tempo oportuno a partir da chave utilizada para cifrar. A chave que é utilizada para cifrar não pode ser utilizada para decifrar criptograma e, por isso, pode ser de conhecimento público (*chave pública*). A segurança está associada ao segredo da chave que é utilizada para decifrar, que é chamada *chave privada*. Estes sistemas são classificados como *sistemas de chave pública ou sistemas assimétricos*.

---

[1] Os autores têm definições um tanto distintas para *criptografia* e *criptoanálise*. Singh refere-se à criptografia como *arte*, Schneier como *arte e ciência* e Menezes define *criptografia* como estudo de métodos matemáticos relacionados à segurança da informação (pag. 4) e posteriormente declara que a criptografia iniciou-se como uma *arte* e tornou-se *ciência* ao longo dos últimos 25 anos (pag 6). É correto portanto afirmar que ambas se iniciaram como arte e têm adquirido caráter cada vez mais científico ao longo do tempo.

[2] A definição dada por Schneier não faz referência a "cifras ou códigos". A definição foi complementada a bem da precisão e clareza, de acordo com o emprego observado da palavra criptografia nos diversos contextos estudados na bibliografia.

[3] Rigorosamente, é suficiente que a chave utilizada para decifrar possa ser obtida de forma eficiente a partir da chave utilizada para cifrar.

Os algoritmos podem ser divididos em duas classes quanto à maneira como atuam sobre as mensagens: i) *cifradores de fluxo*, que cifram a mensagem símbolo a símbolo, acompanhando o fluxo de entrada dos dados; ii) *cifradores de blocos,* que dividem a informação (*texto em claro*) em blocos e cifram cada um destes blocos separadamente. O criptograma do texto completo é obtido pela concatenação dos criptogramas de cada um dos blocos. É comum encontrar cifradores que utilizam blocos de 64, 128 e 256 bites, por exemplo.

Os cifradores simétricos de blocos são os mais importantes elementos em muitos sistemas criptográficos. Por serem versáteis, podem ser utilizados na construção de geradores de números pseudo-aleatórios, cifradores de fluxo, códigos de autenticação de mensagens, e outras aplicações importantes (MENEZES, 1996, p.223). Este trabalho não tem por objetivo tratar do emprego de cifradores. Os objetivos traçados neste trabalho estão voltados apenas para o projeto e avaliação dos cifradores de blocos. Portanto, deste ponto em diante, todas as referências feitas a algoritmos criptográficos dizem respeito a cifradores simétricos de blocos.

1.2 BREVE HISTÓRICO SOBRE A CRIPTOGRAFIA

Durante milhares de anos, reis, rainhas, generais e outras autoridades dependeram de comunicações eficientes para governar seus países, comandar seus exércitos e exercer as suas funções. Enquanto isso, todos estavam conscientes das conseqüências de suas mensagens caírem em mãos erradas, revelando segredos preciosos a nações rivais ou divulgando informações comprometedoras ao inimigo. A ameaça de interceptação pelo inimigo ou oponente tem motivado o desenvolvimento de códigos e cifras, técnicas para tornar o conteúdo de uma mensagem acessível apenas para um destinatário autorizado, desde a antigüidade até hoje em dia (SINGH, 2001, p. 11). Os primeiros registros sobre a criptografia datam de 4000 anos atrás, quando esta *arte* era utilizada de maneira bastante incipiente pelos egípcios. Desde aquela época até a década de 1960, a criptografia foi um assunto essencialmente restrito aos militares, à Igreja (KAHN, 1976, p.106-114, 126-127, 141, 148-151), aos serviços diplomáticos e às atividades governamentais em geral (MENEZES, 1996, p. 1).

Durante a 2ª Guerra Mundial a Inglaterra, por exemplo, acreditava que a quebra de códigos era importante o suficiente para merecer a alocação de 30.000 pessoas neste tipo de

atividade. Estima-se que a criptoanálise tenha antecipado o fim da Guerra do Pacífico em um ano (KAHN, 1976, ix) e a 2ª Guerra Mundial em 3 anos (SINGH, 2001).

1.3 PADRÕES

No início da década de 70, o uso da criptografia por não militares era pouco difundido, e praticamente nenhum resultado de pesquisa era publicado neste campo. As pessoas sabiam que os militares e o governo utilizavam códigos especiais, mas a Agência de Segurança Nacional norte-americana (NSA) sequer admitia a existência desse conhecimento.

Durante este período, os produtos criptográficos eram vendidos e utilizados sem que os usuários tivessem qualquer certeza à cerca da segurança dos mesmos. Não havia nenhuma entidade independente que pudesse certificar tais produtos.

Em 1972, o Departamento Nacional de Padrões dos Estados Unidos (NBS) entendeu que mesmo os não militares necessitavam de um sistema para proteger suas informações sigilosas (SCHNEIER, 1996, 265). Muitas agências e órgãos governamentais, bem como empresas privadas utilizavam sistemas de criptografia para proteger seus dados. Um dos sistemas utilizados era o *Lucifer*, desenvolvido no início da década de 70 por um engenheiro da IBM chamado Horst Feistel (SINGH, 2001, p. 272-73). A dificuldade no trâmite das informações codificadas por diferentes sistemas levava à necessidade de um algoritmo padrão para a criptografia de dados e comunicações (SINGH, 2001, p. 272). Para que o sistema fosse confiável, deveria ser de domínio público, pois um sistema se torna tão mais confiável quanto mais tempo resistir submetido às avaliações da comunidade científica (SCHNEIER, 1996, p. 214-15). O conhecimento público do sistema criptográfico está de acordo também com o princípio de *Auguste Kerckhoffs* ("*La Criptographie Militaire"*, 1873): *"a segurança de um sistema criptográfico deve residir unicamente no segredo da chave, e não no sigilo do algoritmo"* (BUCHMANN, 1998, p. 88) (SCHNEIER, 1996, p. 7, 265, 266) (LANDAU, 2000, 116)(KAHN, 1976, p. 230). Deve-se admitir que aquisição de um exemplar do equipamento ou *software* que executa a criptografia, bem como a execução da engenharia reversa para obtenção do algoritmo, é apenas uma questão de tempo e dinheiro.

A NSA, que era o órgão governamental com mais experiência em sistemas criptográficos, não parecia estar disposta a desenvolver criptografia para uso público (LANDAU, 2000, p. 115). O crescimento da demanda por sistemas criptográficos comerciais,

entretanto, conduzia à necessidade de fomentar o aprimoramento do conhecimento público nessa área.

Em 1974, a IBM, que tinha um programa de pesquisa para o desenvolvimento de sistemas criptográficos comerciais, apresentou ao NBS um candidato a sistema criptográfico padrão promissor (SCHNEIER, 1996, p. 266). O novo algoritmo, baseado em parte nas idéias de Feistel, tornou-se o padrão para criptografia de dados (*Data Encryption Standard* - DES) (FIPS-46-3, 1999)(DIFFIE,1977). O enorme impacto econômico causado pela adoção do DES viria a ser estudado e relatado de forma mais ampla recentemente, em (LEECH, 2001).

Durante muito tempo, houve desconfiança de que a NSA tivesse induzido a IBM a colocar um *alçapão*[4] no DES. Havia também críticas quanto ao tamanho da chave, insinuando que aquela Agência talvez pudesse pesquisar todas as combinações possíveis e decifrar, sem autorização, mensagens criptografadas usando aquele padrão (LANDAU, 2000, p. 115) (SCHNEIER_96, 267). Parte de toda esta desconfiança surgiu, aparentemente, pelo fato de a IBM ter alegado a necessidade do uso de bites de paridade na chave, o que reduz o tamanho efetivo inicial da chave de 64 para 56 bites (LANDAU, 2000, p. 115) (DIFFIE, 1977, p. 74) (SCHNEIER, 1996, p. 266, 267).

O governo americano usava todos os recursos legais disponíveis para evitar a exportação de material criptográfico e a discussão na indústria girava em torno da quantidade de bites que o governo autorizaria para o tamanho da chave nos sistemas exportáveis (LANDAU, 2000, p. 115-16). Foi estabelecido em 1992 o limite de 40 bites e, com raras exceções, os produtos incorporando o DES não podiam ser exportados (LANDAU, 2000, p. 115-16). Tais restrições só foram reduzidas em 1998, quando uma mensagem cifrada usando o DES foi decifrada (LANDAU, 2000, p. 115-16). Àquela altura, o DES já precisava de um substituto.

Em fevereiro de 1998, o Instituto Nacional de Padrões e Tecnologia americano (*National Institute of Standards and Technology* - *NIST*, antigo *NBS*) lançou uma competição para um proposto Padrão Avançado de Criptografia (AES). Seriam avaliados os parâmetros *segurança*, *custo* e *flexibilidade de implementação*. Os algoritmos deveriam ser capazes de operar com chaves de 128, 192 e 256 bites. Em 1996, um artigo publicado por criptólogos respeitados na comunidade científica estimava que o tamanho de 90 bites era recomendado como mínimo para a década 1996-2005 (LANDAU, 2000, p. 116) (BLAZE, 1995).

[4] Uma informação privilegiada capaz de reduzir o esforço computacional em relação ao esperado na busca exaustiva.

Ainda em 1998, quinze candidatos ao AES que atendiam às exigências mínimas do NIST se apresentaram para a primeira etapa do concurso, e, em 1999, foram anunciados os cinco finalistas (LANDAU, 2000, p. 115-16)**,** que apresentavam diferentes aspectos positivos e vulnerabilidades potenciais (NECHVATAL, 1999) (LANDAU, 2000, p. 118). O algoritmo vencedor foi o belga Rijndael, desenvolvido por Joan Daemen e Vincent Rijmen, dois pesquisadores da Universidade Católica de Louvain (KUL). Em 2001, o Rijndael foi anunciado pelo NIST como o *Advanced Encryption Standard - AES* (FIPS-197, 2001).

Na Europa, também sob a coordenação da KUL, foi criado um projeto de pesquisa multinacional voltado para a Tecnologia da Informação, *o New European Schemes for Signatures, Integrity and Encription* (projeto NESSIE). O NESSIE é composto pelas seguintes entidades: *Ecole Normale Supérieure* (França*), Royal Holloway-University of London*, *Siemens Aktiengesellschaft* (Alemanha), *Technion – Israel Institute of Technology*, *Universitet i Bergen* (Noruega). O projeto, apesar de não ser um instituto de padrões, se propõe a ser uma interface entre a comunidade de pesquisa e a comunidade de usuários, testando e comparando algoritmos antes que estes sejam propostos como padrões (PRENEEL, 2003, p. 1).

Em setembro de 2000, pesquisadores de mais de dez países diferentes submeteram ao NESSIE quarenta e dois algoritmos de criptografia para avaliação (PRENEEL, 2003, p. 1-2). Desde então, estes algoritmos têm sido testados quanto à segurança e à eficiência. Em setembro de 2001, como resultado de uma primeira etapa de seleção, o universo de quarenta e dois algoritmos foi reduzido a um conjunto de vinte e quatro (PRENEEL, 2003, p. 1-2). A segunda etapa da seleção terminou em fevereiro de 2003, quando o NESSIE anunciou uma lista de dezessete algoritmos, nos quais não havia encontrado ponto fraco que comprometesse a segurança (PRENEEL, 2003, p.1-2). Este conjunto de algoritmos é composto da seguinte forma: doze foram selecionados dos quarenta e dois submetidos e cinco, entre eles o Rijndael, foram adicionados à lista por serem considerados padrões já estabelecidos. Quatro dos algoritmos constantes da lista do NESSIE são cifradores de bloco (PRENEEL, 2003, p. 1-2).

O projeto NESSIE pretende introduzir os algoritmos selecionados e ainda não padronizados em entidades de padronização tais como a *International Organization for Standardisation (ISO)* e a *Internet Engineering Task Force(IETF).* (PRENEEL_03, p. 2).

1.4 ALGORITMOS PÚBLICOS x ALGORITMOS RESTRITOS

Apesar das vantagens da padronização, sob o pretexto de aumentar a segurança, alguns fabricantes e usuários dão preferência à utilização de algoritmos secretos, também chamados algoritmos *restritos* ou *algoritmos proprietários*. Muitos algoritmos públicos, como por exemplo o *CRYPTO-MECCANO,* publicado em 1990, o algoritmo de *Rao-Nam* e suas variantes, uma variante do algoritmo de Mc Eliece, e outros, são considerados inseguros pela comunidade criptográfica (SCHNEIER, 1996, p. 346). Quatro japoneses publicaram um algoritmo baseado em transformações caóticas no *Eurocrypt*[5] de 1991 que foi criptoanalisado por Eli Biham[6] na mesma conferência. O MacGuffin também foi criptoanalisado na mesma conferência em que foi apresentado. Existem vários algoritmos que foram construídos com base na teoria dos códigos de correção de erros que também são inseguros. Mesmo algoritmos considerados seguros podem passar a ser inseguros se utilizados com um número de iterações menor que o recomendado.

O algoritmo IDEA (*International Data Encryption Algorithm*) é o resultado de um aprimoramento feito em um algoritmo anterior, o PES (*Proposed Encryption Standard*), que se tornou IPES (*Improved PES*) e foi renomeado IDEA. O aprimoramento do algoritmo foi resultado da demonstração da criptoanálise diferencial após a publicação do PES. Segundo Bruce Schneier, o IDEA era o melhor algoritmo disponível em 1996. O algoritmo RC3, de Ron Rivest, da RSA, foi quebrado antes do término do seu desenvolvimento.

Todos os algoritmos citados foram desenvolvidos por criptólogos respeitados e suas fraquezas não teriam aparecido se estes algoritmos fossem restritos. Os algoritmos considerados inseguros, em sua maioria, não se tornaram importantes do ponto de vista da sua utilização, mas são extremamente importantes porque ilustram de maneira incontestável o perigo de se confiar na própria avaliação, sobretudo quando se trata de um modelo não convencional de cifra.

O DES e o AES têm sido exaustivamente estudados desde que foram divulgados. Em toda a bibliografia[7] consultada por este autor, referente aos últimos 25 anos de pesquisa em busca de um suposto alçapão no DES, não foi encontrada nenhuma evidência da sua existência. Ao contrário, diversas das fontes mais recentes fazem referência ao fato de

[5] Um dos mais importantes seminários internacionais sobre criptografia que ocorrem anualmente na Europa.
[6] Criptólogo reconhecido internacionalmente e um dos criadores da criptoanálise diferencial.
[7] Inclusive nos trabalhos (LIMA, 1999), (XEXEO, 1983) e (XAVJR, 1999), desenvolvidos no Instituto Militar de Engenharia.

ninguém ter publicado qualquer resultado que mostre a existência de tal alçapão (BIHAM, 1993, p. 2) (CARVALHO, 2001, p. 103-104) (LOUDON, 2000, p. 457) (MENEZES, 1996, p. 256 a 259) (SCHNEIER, 1996, p. 278-280).

O DES mostrou-se teoricamente vulnerável à criptoanálise diferencial, embora esta forma de criptoanálise tenha sido demonstrada apenas em um modelo reduzido. Não foi encontrado na literatura nenhum registro de implementação prática desta técnica que tenha sido feita no DES completo (16 passos). As decifrações[8] de criptogramas do DES registradas até o ano de 2000 foram feitas pela *força bruta*, uma utilizando uma máquina dedicada e outra, uma associação de dezenas de milhares de computadores em paralelo (SCHNEIER, 2002, p. 1)(LANDAU, 2000, p.. 115). Não foi encontrado nenhum resultado mais significativo publicado até a presente data. A principal vulnerabilidade do DES permanece o tamanho da chave.

O AES permanece considerado seguro. O resultado mais importante contra o Rijndael encontrado na bibliografia até o presente momento utiliza um sistema de equações superdefinido para cuja solução não se conhece método eficiente (COURTOIS, 2002).

A credibilidade dos algoritmos públicos cresce à proporção que o tempo passa e não surge nenhum ataque capaz de ameaçá-los.

O princípio de *Kerckhoffs* foi considerado quando da utilização da máquina Enigma[9] pelos alemães. Apesar disso, após a captura de uma máquina e conseguidos os seus detalhes construtivos, foi possível quebrar a cifra utilizando uma grande quantidade de matemáticos e lingüistas trabalhando em conjunto. A quebra da Enigma permaneceu em sigilo por mais de 30 anos, até 1974. Naquele ano, como nenhum país da Comunidade Econômica Européia utilizava mais a máquina, foi divulgado que a máquina já não era mais segura (SINGH, 2001, Cap 4). Se o algoritmo da Enigma fosse colocado à disposição da comunidade criptográfica nos dias atuais, mesmo que o tamanho da chave fosse aumentado o suficiente para inviabilizar a busca exaustiva, provavelmente as fraquezas do algoritmo ante um *ataque com textos conhecidos* seriam rapidamente exploradas e o algoritmo seria considerado fraco. Entretanto, se algum fabricante de produto criptográfico atual quiser utilizar o algoritmo da Enigma - ou alguma variante dele - como algoritmo restrito em seus produtos, as suas fraquezas não aparecerão, mas nem por isso o sistema poderá ser considerado seguro.

[8] Em geral, o termo *decifração* não se aplica à técnica de "*criptoanálise*". Entretanto, no caso particular da força bruta (busca exaustiva), o que efetivamente ocorre é uma sucessão de *tentativas de decifração* com chaves supostas.

As principais vantagens dos algoritmos públicos são a sua constante avaliação pela comunidade científica e o fato de terem sido desenvolvidos por gente capacitada. Os algoritmos públicos mais importantes são os padrões (AES e DES, por exemplo). A possível desvantagem destes algoritmos é a possibilidade de sua segurança ser limitada quando o oponente é uma agência de inteligência, como por exemplo a Agência de Segurança Nacional norte-americana (NSA), pela qual os algoritmos são analisados antes de serem adotados como padrão.

Do acima exposto, pode-se concluir que algoritmos de domínio público são, em princípio, mais confiáveis que algoritmos privados, desde que as informações a serem protegidas não sejam importantes para a segurança nacional nem para decisões comerciais internacionais que envolvam grandes quantias (bilhões de dólares). Para estes casos, é prudente desenvolver o conhecimento científico na área de criptografia. O ponto de partida para tal conhecimento é o estudo dos algoritmos públicos.

1.5 MOTIVAÇÃO

Os países detentores de tecnologia em criptografia têm leis que restringem as exportações dos produtos que contenham criptografia forte (SINGH, 2001, p.327,328) (DIFFIE,2001). Em junho de 1999, o *Cyberspace Policy Institute* (CPI), sediado na Universidade George Washington, publicou um relatório intitulado *O Crescente Desenvolvimento de Produtos Criptográficos Estrangeiros em Virtude das Restrições de Exportação dos EUA* (HOFFMAN, 1999, 6). Neste documento são relacionados oitocentos e cinco produtos não norte-americanos de trinta e cinco países diferentes envolvendo criptografia (*hardware* ou *software*). O Brasil não está entre estes países.

Nos Estados Unidos, na Europa e mesmo no Brasil, o assunto criptografia é tratado oficialmente como assunto relacionado ao desenvolvimento e à segurança nacionais (DECRETO, 1998)(DECRETO, 2000)(LEI, 1991)(DIFFIE, 2001) (SING, 2001, p. 331, 332) (LANDAU, 2000) (RSAS, 1989, p. 73-81) (SCHNEIER, 1996, p. 214-16, 610-18). Como ilustração, segue uma transcrição parcial do projeto da Lei Geral Anticrime do Senado Norte Americano de 1991:

[9] Máquina de cifrar e decifrar utilizada pelos alemães na 2ª GM. Há quem atribua à sua quebra a antecipação do final da guerra em 3 anos.

"*É opinião do Congresso que os fornecedores de serviços de comunicação eletrônica e os fabricantes de equipamentos de comunicação eletrônica devem garantir que esses sistemas de comunicações permitam a obtenção por parte do governo do conteúdo completo de texto, voz, dados e outros tipos de comunicação, quando autorizado pela lei.*"

Apesar de não ter sido apresentada nenhuma prova de existência de alçapão nos algoritmos aprovados pelo NIST ou NESSIE, a história nos tem mostrado que a ausência de tal prova não é prova de sua inexistência (SINGH, 2001, p. 314-318, 327, 328 ,331, 332, 333) (DIFFIE, 2001) e que é prudente moderar a confiança em tais sistemas (SCHNEIER, 1996, p. 215-16).

No Brasil, com base na bibliografia consultada até o presente momento, não existe padronização de sistema criptográfico com algoritmo nacional de domínio público. As entidades que necessitam de criptografia utilizam sistemas comerciais estrangeiros ou de domínio próprio. Vários registros mostram a inconveniência de se utilizar produtos criptográficos prontos (sobretudo estrangeiros) para proteger informações de considerável valor comercial ou estratégico. Como exemplos, podemos citar os casos da *Lotus Notes* e da *Cripto AG*, dois grandes fabricantes de *software* cujos produtos criptográficos enviavam, juntamente com a mensagem de correio eletrônico cifrada, 24 dos 64 bites da chave. Tal informação reduzia por um fator 2^{24} o custo computacional da busca exaustiva da chave, tornando possível para a NSA pesquisar o universo de 2^{40} chaves restantes e acessar a informação em um curto espaço de tempo (BOWMAN, 1995) (DER SPIEGEL, 1996) (LAURIN, 1997) (STOA, 1999). Outros três artigos, intitulados *Por que espionamos nossos aliados?* (WOOLSEY, 2000), *Aliança Sagrada da Espionagem* (LE MONDE, 2000) e *A Exportação de Criptografia no Século 20* (DIFFIE, 2001) também sugerem que é importante o desenvolvimento de criptografia própria quando se trata de proteger informações de valor estratégico. O desafio é, portanto, desenvolver conhecimento para produzir cifradores fortes dentro da premissa de *Kerckhoffs*: *o inimigo conhece todos os detalhes do algoritmo*. Utilizar uma criptografia fraca é pior do que não utilizar criptografia alguma, pois a falsa sensação de segurança tende aumentar os níveis de comprometimento. Na bibliografia, encontram-se registros de diversos testes realizados em diversos tipos de algoritmos, sobretudo em cifradores de bloco. Apesar de haver na comunidade científica um conjunto consensual de *condições necessárias* para que um algoritmo seja confiável, não foi encontrado e não parece ser viável um teste ou método de avaliação que seja *suficiente* para *garantir* a segurança de um dado algoritmo criptográfico (COPPERSMITH, 2000c) (DIFFIE, 1977) (VAUDENAY,

1996). Dentro desta realidade, torna-se útil um documento que contribua para definir um conjunto de critérios para nortear o projeto e a avaliação da qualidade de um algoritmo criptográfico.

1.6 OBJETIVOS DESTE TRABALHO

i) Definir e apresentar uma filosofia de projeto (conjunto de princípios) para cifradores simétricos iterativos de blocos que estejam de acordo com os padrões internacionais atuais, com base no estudo dos padrões criptográficos já estabelecidos (DES e AES);
ii) Definir e sugerir um conjunto de técnicas para projeto de cifradores simétricos iterativos de blocos que estejam de acordo com a filosofia apresentada;
iii) Definir e sugerir um conjunto de técnicas para avaliar a conformidade de um cifrador simétrico iterativo de blocos com a filosofia apresentada; e
iv) Apresentar um algoritmo cifrador simétrico iterativo de blocos que esteja em conformidade com a filosofia de projeto apresentada.

1.7 SOBRE ESTA DISSERTAÇÃO

A metodologia adotada neste trabalho consistiu dos seguintes passos:

- estudo dos requisitos de projeto apresentados pelo Ministério do Comércio norte-americano para os algoritmos candidatos a padrão criptográfico DES (1973-1977) e do AES (1996-2001);
- estudo[10] das cifras de bloco DES, Rijndael (AES) e Serpent, com base nas suas descrições e nos seus pontos elogiados e criticados pela comunidade científica, inclusive as formas mais importantes de criptoanálise;
- sugestão de uma filosofia de projeto, baseada nos princípios que foram observados como fundamentais nos estudos realizados: *segurança, eficiência, simplicidade, flexibilidade e credibilidade;*
- desenvolvimento e sugestão de técnicas para avaliação e projeto de cifras de bloco dentro dos princípios preconizados;

[10]Foram utilizados trabalhos já realizados no IME ((XEXEO, 1983), (CASTAÑO, 1998), (LIMA,1999), (XAVJR, 1999)) e outros trabalhos realizados na comunidade científica internacional;

- desenvolvimento e apresentação de um cifrador de blocos desenvolvido com o auxílio das técnicas sugeridas;
- realização de testes de avaliação com a cifra apresentada.

As técnicas apresentadas são as seguintes:

- uma técnica para avaliação de transformações lineares sob a ótica do efeito avalanche, a técnica das matrizes de avalanche;
- uma técnica para construção de transformações lineares capazes de produzir o efeito avalanche em um número de iterações menor do que as transformações lineares do Rijndael, do Serpent e do DES;
- uma técnica para construção de transformações não-lineares baseada nas propriedades dos *quadrados latinos* e dos *corpos finitos*, visando a propiciar boa resistência contra as formas de criptoanálise mais importantes.

Ainda no início do trabalho, é feita uma descrição do Rijndael (AES) que visa a tornar suas operações mais claras para aqueles que estão habituados ao DES. Esta descrição é considerada uma contribuição importante desta dissertação.

Cada passo da metodologia acima descrita é considerado uma contribuição desta dissertação.

A metodologia adotada encontra-se ilustrada na FIG. 1.1.

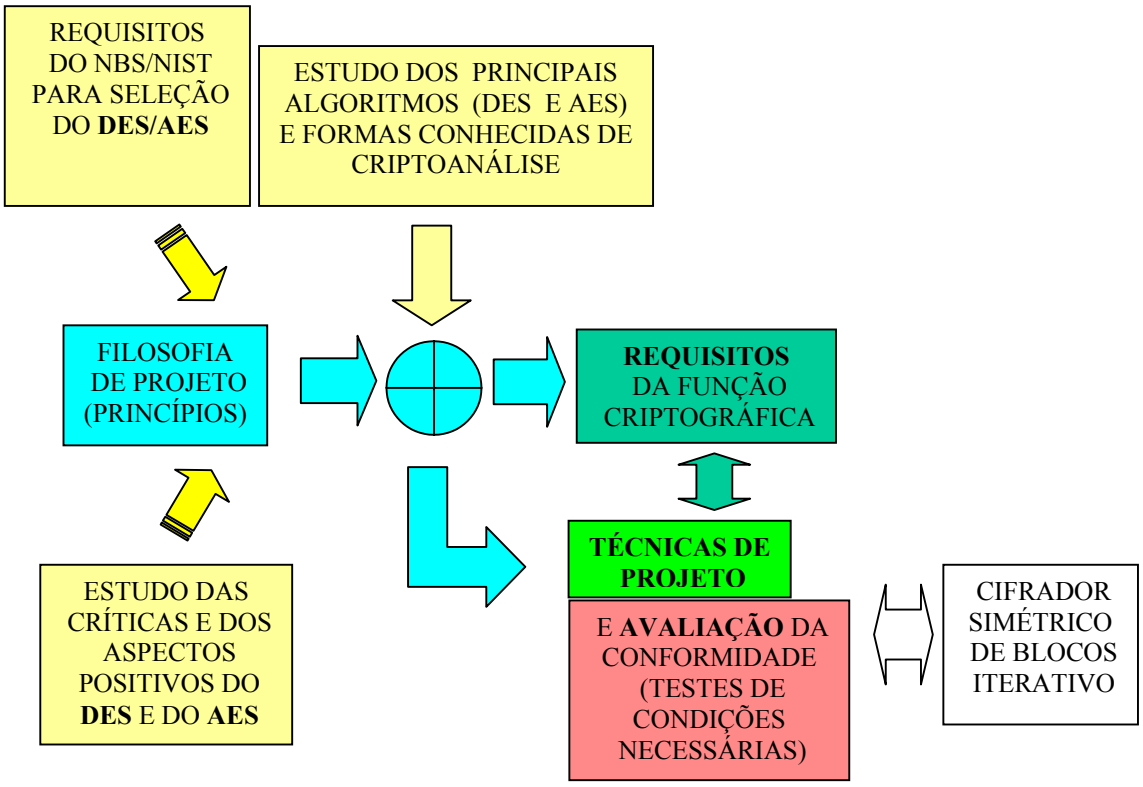


FIG. 1.1 Metodologia

2 TERMINOLOGIA, NOTAÇÃO E CONCEITUAÇÃO BÁSICA

2.1 TERMINOLOGIA

Ao longo deste trabalho será utilizada a seguinte terminologia:

Bloco: É uma seqüência finita de bites. Os blocos serão representadas por uma letra maiúscula. A quantidade de bites que forma o bloco é chamada **tamanho do bloco**. Quando o tamanho do bloco for diferente de 128, este tamanho será especificado. Nos demais casos, neste documento, a palavra *bloco* refere-se a um bloco de 128 bites.

Palavra: É um bloco de trinta e dois bites. Cada palavra se divide em quatro octetos. Os bites são designados da esquerda para a direita por **letras minúsculas** seguidas de um número (*0* a *n-1, n = tamanho do bloco*) que representa a posição do bite no bloco (ordem de entrada). Os baites são designados da esquerda para a direita por **letras maiúsculas** seguidas de um número (*0* a *n/8-1*, *n=0 mod 8*) que representa a posição do baite no bloco. Assim, o bloco de 128 bites

B = (*b0*|*b1*|*b2...b7*| *b8*|...*b15*| *b16*|...|*b23*| ....| *b120*|...*b127*), onde (*b0...b7*) é o baite *B0*, (*b8...b15*) é o baite *B1,* e assim por diante, até o baite *B15*.

2.2 OPERAÇÕES: SIMBOLOGIA

A operação lógica "*ou exclusivo*" (XOR) será representada pelo símbolo "⊕" ou pelo sinal "+" quando o contexto deixar claro que trata-se de soma módulo 2.

A operação lógica "*e*" será representada pelo símbolo " ^ ".

A operação de concatenação será representada pelo símbolo "|". Quando não houver perda de clareza, a concatenação poderá ser também representada pela simples justaposição dos blocos.

2.3 CONCEITUAÇÃO BÁSICA

i) Premissa de Kerckhoffs: *O adversário tem conhecimento detalhado do algoritmo criptográfico bem como da sua implementação* (SCHNEIER, 1996, p. 5, 7, 265, 266) (BUCHMANN, 1998, p. 88) (LANDAU, 2000, p. 116) (KAHN, 1976, p. 230) (STINSON,1995, 24).

ii) Alfabeto de definição α: É um conjunto finito de elementos. Neste caso, será utilizado o alfabeto binário $\alpha=\{0,1\}$. Qualquer alfabeto pode ser codificado em termos de alfabeto binário. (MENEZES, 1996, p. 6-13)

iii) Espaço das mensagens μ : É um conjunto de seqüências de símbolos de α**.** Um elemento de μ é chamado *mensagem em claro* ou *texto em claro*. (MENEZES, 1996, p. 6-13)

iv) Espaço dos criptogramas ξ: É um conjunto de seqüências de símbolos de α. Um elemento de ξ é chamado *texto cifrado* ou *criptograma*. (MENEZES, 1996, p. 6-13)

v) Espaço das Chaves Ω: É um conjunto de seqüências de símbolos de α. Um elemento de Ω é chamado *chave*. (MENEZES, 1996, p. 11-21)

vi) Função de cifrar: Cada elemento K de Ω determina de maneira única uma bijeção de μ em ξ representada por E_k. E_k é chamada *função de cifrar* ou *transformação de cifrar*. (MENEZES, 1996, p. 6-13)

vii) Função de decifrar: Cada elemento K de Ω determina de maneira única uma bijeção de μ em ξ representada por D_k. D_k é chamada *função de decifrar* ou *transformação de decifrar* (MENEZES, 1996, p. 6-13).

viii) Função criptográfica (MENEZES, 1996, p. 6-13)

No caso dos cifradores simétricos, para cada chave K, temos um par de transformações (E_k, D_k), onde $D_k = E_k^{-1}$. Seja M um elemento de μ e C um elemento de ξ tais que $C = E_k(M)$. Temos que $D_k(C) = D_k(E_k(M)) = M$. D_k e E_k são exemplos de funções criptográficas.

Uma vez que este trabalho tem como objeto apenas cifradores simétricos, será utilizada a seguinte definição para *função criptográfica*.

Definição: Sejam $M \in \mu$, $K \in \Omega$. Se existe um único $C \in \xi$ e uma função *f* tal que $f(M, K) = C$ e $f^{-1}(C, K) = M$, então *f* é uma função criptográfica[11].

ix) Função de sentido único[12] - *Definição:* Uma função *f* de um conjunto μ em um conjunto ξ é dita *função de sentido único* se existe um algoritmo eficiente para computar *f(M)* para todo *M* pertencente a μ mas é "computacionalmente impraticável"[13] encontrar *M* pertencente a μ tal que $f(M)=C$ para "quase todo"[14] *C* pertencente a ξ. (MENEZES, 1996, p. 8)

x) Função de sentido único com alçapão[15] - Definição: Uma *função de sentido único com alçapão* é uma função de sentido único com a propriedade adicional de que, dada alguma informação adicional (chamada *informação alçapão*[16]), torna-se viável encontrar para qualquer *C* pertencente a *Im(f)* um *M* pertencente a μ tal que $f(M)=C$. (MENEZES, 1996, p. 9)

xi) Cifrador produto - *Definição:* Um cifrador produto combina duas ou mais transformações de tal forma que a cifra resultante seja mais segura do que cada transformação individualmente. (MENEZES, 1996, p. 251)

xii) Cadeia de substituições e permutações[17] – *Definição:* Cadeia de substituições e permutações é um cifrador produto composto por mais de um estágio, cada um deles envolvendo substituições e permutações. (MENEZES, 1996, p. 251)

[11] Rigorosamente, *f* é, neste caso, uma função criptográfica simétrica, pois utiliza a mesma chave *K* para cifrar e decifrar.
[12] Função de mão-única ou *one-way function*.
[13] O autor utiliza "*computationally infeasible*". O conceito é o mesmo de *problema intratável(item 2.15)*.
[14] O autor utiliza *"essentially all"*. Este conceito indica que a probabilidade de se encontrar um *y* pertencente a *Y* que não satisfaça a condição é pequena o suficiente para ser negligenciada.
[15] *Trapdoor One-way function*
[16] *Trapdoor information*
[17] *substitution-permutation network (SPN)*

xiii) Cifrador de blocos iterado – *Definição:* Um cifrador de blocos iterado é um cifrador de blocos que inclui uma repetição seqüencial de uma função interna chamada *função de passo*. Os parâmetros de um cifrador de blocos iterados incluem o número de passos *Nr*, o tamanho dos blocos *n*, o tamanho *k* da chave *K* a partir da qual as *Nr* sub-chaves K_i de passo são geradas. Cada valor de K_i deve determinar uma bijeção de um passo (para permitir a decifração).

xiv) Cifrador simétrico de blocos - *Definição:* Um cifrador simétrico de blocos é um algoritmo capaz de cifrar e decifrar blocos utilizando uma mesma[18] chave para ambas as operações (SCHNEIER, 1996, p. 4)(DENNING, 1983, p. 8).

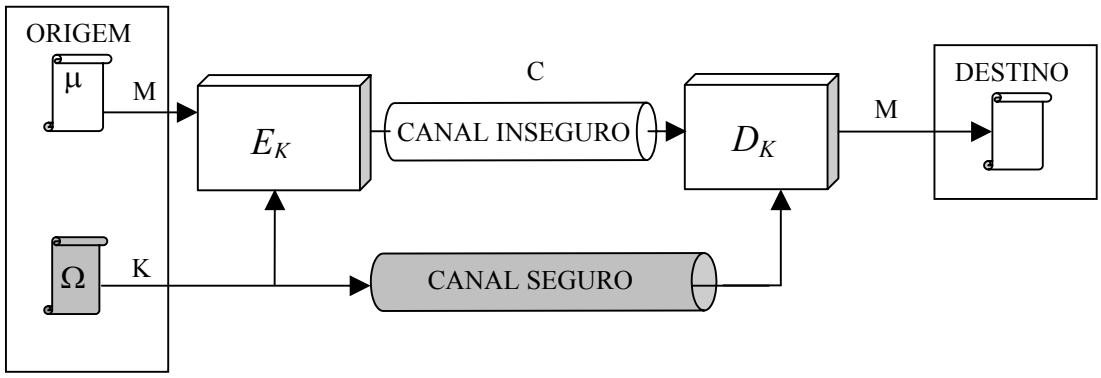


FIG. 2.1 Sistema simétrico

xvi) *Definição: Criptoanálise*[19] é a ciência que trata da busca de vulnerabilidades em um sistema criptográfico objetivando a recuperação do texto em claro de uma mensagem sem o conhecimento da chave ou a recuperação da chave em si (SCHNEIER,1996, p. 5).

[18] Rigorosamente, a chave não precisa ser a mesma. É suficiente que a chave necessária para decifrar possa ser obtida a partir da chave utilizada para cifrar de maneira eficiente (DENNING, 1983).

[19] A obtenção não autorizada da chave por meio diferente da criptoanálise e chamada *comprometimento* (SCHNEIER, 1996, p. 5). O mesmo autor considera que estamos em uma era de criptoanálise teórica e que a simples redução do fator trabalho estimado para quebrar uma cifra também é considerada criptoanálise, mesmo que a solução do novo problema não seja realizável. Nesta concepção, a criptoanálise pode ter por objetivo apenas reduzir, por exemplo, a complexidade de um problema de 2^{512} para 2^{490}, mesmo sabendo-se de que a execução de 2^{490} operações é irrealizável na prática.

Existe uma sutileza no conceito de *ataque criptoanalítico* que por vezes causa dúvida e merece ser comentada. Nas palavras de Bruce Schneier: "...para um teórico, um ataque criptoanalítico é qualquer técnica teoricamente capaz de quebrar um algoritmo com um esforço menor do que a força bruta, mesmo que este ataque seja impraticável. Para um engenheiro, uma técnica só pode ser considerada um ataque se for exeqüível no presente ou num futuro próximo (alguns anos)." (SCHNEIER, 2002, 1) A criptoanálise está vivendo uma época extremamente teórica. Alguns dos ataques que se têm desenvolvido (OSWALD, 2002), como os ataques quadráticos (LUCKS, 2000) (FERGUSON, 2000), XSL (COURTOIS,2002) e a "criptoanálise diferencial impossível" (CHEON, 2001), são impraticáveis, sendo mais argumentos teóricos em modelos reduzidos do que demonstrações computacionais de decifração da cifra completa.

xvii) *Definição:* Uma tentativa de criptoanálise é chamada ataque (SCHNEIER, 1996, p. 5).

xviii) Classificação geral dos ataques

De acordo com as possibilidades supostas do criptoanalista de obter informação sobre os textos em claro e estatísticas da linguagem utilizada, os ataques são classificados em cinco[20] categorias principais (SCHNEIER, 1996, p. 6):

- Ataque *com texto cifrado*: o criptoanalista dispõe de tantos criptogramas quantos ele deseje, tendo sido todos eles cifrados utilizando o mesmo algoritmo criptográfico e a mesma chave. O objetivo do criptoanalista é obter a maior quantidade possível de textos em claro ou a chave, o que lhe permitirá decifrar todos os criptogramas;
- Ataque *com texto em claro conhecido*: o criptoanalista tem acesso não somente aos criptogramas de diversas mensagens, mas também aos respectivos textos em claro. O seu objetivo é descobrir a(s) chave(s) ou um algoritmo capaz de decifrar qualquer novo criptograma cifrado com a(s) mesma(s) chave(s);

- Ataque *com texto em claro escolhido*: o criptoanalista tem acesso não somente aos criptogramas de diversas mensagens e aos respectivos textos em claro, como também pode escolher qual o texto em claro que será criptografado. Este tipo de ataque é mais

poderoso que o anterior, pois o criptoanalista pode escolher blocos de texto a serem cifrados capazes de revelar mais informações sobre a chave. O seu objetivo é descobrir a(s) chave(s) ou um algoritmo capaz de decifrar qualquer novo criptograma cifrado com a(s) mesma(s) chave(s);

- Ataque *com texto em claro escolhido adaptado*: Este é um caso especial do anterior. O criptoanalista além de ter acesso aos criptogramas de diversas mensagens, aos respectivos textos em claro e poder escolher qual o texto em claro que será criptografado, ele pode adaptar a sua escolha de acordo com o resultado da criptografia do texto que havia escolhido;

- Ataque *com texto cifrado escolhido*[21]: O criptoanalista escolhe o criptograma a ser decifrado e tem acesso ao respectivo texto em claro decifrado. Apesar de principalmente indicado contra algoritmos de chave pública, este tipo de ataque também pode ser realizado contra algoritmos simétricos.

xix) Força bruta[22]: É a técnica de criptoanálise que realiza a busca da chave testando um a um todos os elementos do espaço de chaves (SCHNEIER, 1996, p. 8). O fator trabalho associado a este tipo de ataque (que é o tamanho do espaço das chaves) é um limite superior natural da segurança um algoritmo (BIHAM, 1993, p. 178).

xx) Complexidade (SCHNEIER, 1996, p. 9)

Uma função criptográfica deve possibilitar uma computação fácil para um usuário autorizado (que conhece a chave) e difícil[23] para um usuário não autorizado. Esta dificuldade é traduzida na complexidade da criptoanálise, que pode ser subdividida em *complexidade de dados, complexidade de processamento*[24] ou *fator trabalho*[25] e *necessidade de memória.*

---

[20] Schneier faz referência também a duas outras categorias de ataque: *ataque com chave escolhida* e criptoanálise *rubber-hose* (cacetete). Estas duas categorias foram excluídas por não se aplicarem ao estudo proposto nesta dissertação.

[21] A combinação do *ataque com texto em claro escolhido* com o *ataque com criptograma escolhido* é normalmente chamada *ataque com texto escolhido*.

[22] É comum referir-se à força bruta com a expressão *busca exaustiva*.(BIHAM, 1993, p. 178)

[23] Por *difícil* entende-se um trabalho que não seja computacionalmente realizável com a tecnologia disponível atualmente em tempo oportuno.

[24] As complexidades em criptografia estão relacionadas e são definidas por ordens de grandeza. A expressão da complexidade em notação *O* (MARKEZON, 1994, p. 15) deve ser observada com cautela. Observe que se, como

- Complexidade de processamento: é a quantidade de tempo[26] necessária para a implementação de um ataque.
- Complexidade de dados: é a quantidade de dados necessária como entrada para a implementação de um ataque.

xxi) A notação *O*

Genericamente, a complexidade computacional de um algoritmo é expressa segundo o que chamamos *notação O*: a ordem de grandeza da complexidade computacional. Seja *n* um número que representa o tamanho da entrada. A *notação O* leva em consideração apenas o termo que cresce mais significativamente com *n*, os demais termos são desconsiderados. Por exemplo, se a complexidade de processamento de um algoritmo é $4n^2 + 7n + 12$, sua complexidade computacional é da ordem de n^2, expressa como $O(n^2)$. A medição da complexidade desta forma é independente do sistema computacional que executará o algoritmo(SCHNEIER, 1996, p. 238, 239).

xxii) Classificação dos algoritmos segundo a complexidade computacional

Geralmente, os algoritmos são classificados segundo a sua complexidade de tempo (processamento) ou espaço (memória requerida). Um algoritmo é dito *constante* se sua complexidade é independente de *n*: *O(1).* Um algoritmo é dito *linear* se sua complexidade é *O(n)*. Por extensão deste raciocínio, algoritmos podem ser *quadráticos*, *cúbicos* e assim por diante. Todos estes algoritmos são ditos *polinomiais;* suas complexidades são $O(n^m)$, onde *m* é alguma constante. Os algoritmos polinomiais formam uma classe de algoritmos: a classe dos algoritmos de *tempo polinomial*. (SCHNEIER, 1996, p. 238, 239)

Algoritmos cujas complexidades são $O(t^{f(n)})$, onde t é uma constante maior do que *1* e *f* é uma função polinomial de *n* são ditos algoritmos *exponenciais*[27]. O subconjunto dos algoritmos exponenciais cujas complexidades são $O(c^{f(n)})$, onde *c* é uma constante e *f(n)* é

---

de costume, o tamanho da chave está definido por *n* e é constante, o fator trabalho para a criptoanálise está limitado superiormente pela constante 2^n. Entretanto, esta constante não pode ser desconsiderada.

[25] O fator trabalho é chamado também esforço computacional e está relacionado com o número de operações esperado para realizar uma tarefa, em particular, a criptoanálise.

[26] Apesar de Schneier utilizar a palavra tempo em sua definição, é conveniente utilizar o fator trabalho e estimar o tempo necessário levando em consideração a tecnologia disponível.

[27] Também chamados algoritmos de tempo exponencial(SCHNEIER, 1996, p. 238).

mais que constante mas sub-linear[28], é o conjunto dos algoritmos *superpolinomiais* (SCHNEIER, 1996, p. 238, 239).

xxiii) Complexidade dos problemas

A teoria da complexidade classifica os problemas que podem ser resolvidos com algoritmos de tempo polinomial como *tratáveis*[29]. Problemas que não podem ser resolvidos em tempo polinomial são chamados *intratáveis*[30], pois a obtenção da sua solução se torna impraticável mesmo para valores relativamente pequenos de *n* (SCHNEIER, 1996, p. 238, 239). Para dar um significado mais objetivo a estes termos, em nosso contexto serão classificados como *tratáveis* problemas para cuja solução o fator trabalho seja até 2^{40} e *intratáveis* quando este fator trabalho for maior que 2^{90} (BLAZE, 1995)(SCHNEIER, 1996, cap. 7)(SCHNEIER, 2002). Quanto aos valores intermediários, as fontes consultadas por este autor não permitem classificar com segurança.

xxiv) A construção da complexidade

Considerando o atual[31] estado da arte na teoria da complexidade computacional, que ocorre na prática é que, rigorosamente, a afirmação mais forte que se pode fazer quanto à complexidade da criptoanálise de um sistema criptográfico é "*todos os algoritmos conhecidos para quebrar tal sistema são de complexidade superpolinomial*" (SCHNEIER, 1996, p. 238).

As técnicas mais simples para cifrar um bloco de símbolos são *substituição* e *transposição*. A utilização de uma combinação apropriada de *substituições*[32] e *permutações* costuma ser mais forte que a soma das forças de cada uma destas operações. Elas são a base do DES[33], do AES, e da maioria dos outros sistemas de criptografia de blocos simétricos iterativos conhecidos (LANDAU, 2000, p. 116)(DAVIDA, 1979).

Transformações *não-lineares* estão na base da construção de um algoritmo criptográfico. Uma regra básica de projeto é que nenhuma função linear de bites da saída deve ser função linear de bites de entrada. Mas funções criptográficas devem ser invertíveis, fáceis de computar, e não devem exigir muita memória. Essa necessidade faz com que funções lineares também exerçam um papel importante dentro da transformação criptográfica. A não-

[28] Para todo par de constantes *k, k'* existe um valor *n0* tal que para todo $n>n0$, $k<f(n)<k'n$.
[29] Isto porque estes problemas podem usualmente ser resolvidos em um tempo razoável para tamanhos de entrada razoáveis. O conceito de "razoável" depende das circunstâncias (SCHNEIER, 1996, p. 239).
[30] Ou simplesmente *difíceis*.
[31] Na data da edição, 1996.
[32] Aí incluídas as *somas módulo 2 (XOR)*.

linearidade freqüentemente é obtida usando *tabelas de substituição* (*s-boxes*). Um outro recurso básico é a *iteração*. Apesar de haver algoritmos não iterativos para codificação de blocos, a iteração é uma tendência natural, pois pode oferecer boa complexidade utilizando apenas operações simples dentro do laço que será executado várias vezes. A idéia é que mesmo transformações criptográficas relativamente simples podem se tornar difíceis de criptoanalisar se repetidas um número suficientemente grande de vezes. Nos algoritmos adotados como padrão de criptografia existe a tendência de manter o sistema simples, a fim de facilitar a avaliação da segurança do algoritmo e a sua certificação (LANDAU, 2000, p. 116).

A complexidade para a criptoanálise também pode ser obtida utilizando funções matemáticas de difícil inversão (de sentido único), mas este tipo de construção não é objeto de estudo deste trabalho[34](DIFFIE, 1976).

xxv) Confusão e difusão

Uma operação de *substituição* em uma etapa de transformação criptográfica adiciona *confusão* à cifragem, ao passo que uma operação de *permutação*[35] adiciona *difusão*. O conceito de *confusão* está associado a tornar a relação entre o texto em claro, o criptograma e a chave tão complexa quanto possível. O conceito de *difusão* está relacionado com mudança de posição dos bites dentro da mensagem de tal forma que qualquer tipo de redundância ou padrão existente na mensagem seja espalhado por todo o criptograma[36] (SCHNEIER, 1996, p. 237).

xxvi) Alçapão[37]

É uma propriedade de um sistema criptográfico, presumidamente intencionalmente introduzida, que permite a um oponente conhecedor de uma informação ou um caminho secreto (informação alçapão), tornar tratável o problema da criptoanálise.

---

[33] O *Data Encryption Standard* e o *Advanced Encryption Standard* são apresentados no capítulo 2.
[34] No caso do AES, a tabela de substituição é baseada nas operações com polinômios sobre corpos finitos. Considerações sobre este aspecto do AES estão apresentadas no capítulo 3.
[35] *Permutação* tem, no contexto de criptologia o mesmo sentido que *transposição*.
[36] Os conceitos de difusão e confusão estão intimamente ligados ao de *avalanche*.

xxvii) Atalho[38]

Um recurso (intencionalmente introduzido ou não) capaz de reduzir o esforço da criptoanálise a uma complexidade significativamente menor que o da força bruta.

xxviii) Vetor *complemento unitário-k*

Definição: Vetor complemento unitário-k de tamanho *t* é o vetor $U_k^t = u_0|u_1|u_2|u_3|...|u_{t-1}$, $U_k^t \in Z_2^t$, onde $u_i = 1$ para $i = k$ e $u_i = 0$ para $i \neq k$ ($0 \leq i < t$; $0 \leq k < t$).

Exemplos: $U_1^8 = 0100\ 0000$; $U_5^{16} = 0000\ 0100\ 0000\ 0000$;

O vetor unitário U_k^t , na maioria das vezes, aparecerá em um contexto onde é somado a outro vetor de tamanho *t* ou representa a diferença (soma módulo dois) entre dois vetores de tamanho *t*. Nestes casos, poderá ser omitido *t* e a notação fica simplificada para U_k.

xxix) Dependência

Seja $f: Z_2^m \rightarrow Z_2^m$ uma transformação criptográfica.

Sejam $X = x_0|x_1|x_2|x_3|...|x_{m-1}$, $j \in Z_m$ e $i \in Z_m$.

Dizemos que na transformação *f* existe uma relação de dependência do *i-ésimo* bite da saída em relação *ao j-ésimo* bite da entrada se e somente se existe pelo menos um par de vetores de entrada (*X, X'*) que diferem apenas pelo *j-ésimo* bite e cujas respectivas saídas *f(X)* e *f(X')* diferem pelo menos pelo *i-ésimo* bite.

Seja $D_{(f,i,j)}$ o valor binário da *dependência do bite i (da saída) em relação ao bite j(da entrada) na transformação f* e seja $Y = y_0|y_1|y_2|y_3|...|y_{m-1} = f(X) \oplus f(X \oplus U_j)$. $D_{(f,i,j)}$ é igual a *1* se, e somente se, existe pelo menos um *X* tal que o *i-ésimo* bite y_i de *Y* é igual a *1*. Caso contrário, $D_{(f,i,j)} = 0$.

Quando $D_{(f,i,j)} = 1$ diz-se que o *j-ésimo* bite (da entrada) *afeta* o *i-ésimo* bite (da saída) e que o *i-ésimo* bite *depende* do *j-ésimo* bite *(da entrada)*. Contrariamente, quando $D_{(f,i,j)} = 0$ dizemos que o *j-ésimo* bite *não afeta* o *i-ésimo* bite e que o *i-ésimo* bite *independe* do *j-ésimo* bite.

A notação $D_{(f,i,j)}$ poderá ser simplificada para $D_{i,j}$ sempre que, dentro do contexto, tal simplificação não gere imprecisão.

---

[37] Este conceito é uma síntese do que é apresentado como *trapdoor* em (MENEZES, 1996, p. 9, 26) e do sentido da expressão nos contextos de toda a bibliografia estudada.

[38] Um exemplo de atalho (*shortcut*) é o caso da substituição monoalfabética, que apesar de apresentar um espaço de chaves com 26! = 4 x 10^{26} chaves possíveis, pode ser quebrado manualmente utilizando a técnica da análise de freqüências (DIFFIE, 1977, p. 2). Outros exemplos podem ser vistos em (OSWALD, 2002).

xxx) Hipótese do não-cancelamento das dependências (HNC)

Seja $f: Z_2^m \rightarrow Z_2^m$ uma transformação criptográfica. Dizemos *que f satisfaz a hipótese do não cancelamento das dependências* se e somente se o *i-ésimo* bite *(da entrada) afeta* o *i-ésimo* bite *(da saída) para todo* $i \in Z_m$. Logo, se o *i-ésimo* bite de *g(X)* depende do *j-ésimo* bite de *X* e *f* satisfaz a HNC, o *i-ésimo* bite de *f (g (X))* também depende *do j-ésimo* bite de *X* $(f:Z_2^m \rightarrow Z_2^m, g:Z_2^m \rightarrow Z_2^m, X \in Z_2^m, m=0,1,2...)$.[39]

xxxi) Avalanche

Nas palavras de Horst Feistel:

"À medida que o texto em claro atravessa os diversos níveis de transformação ao longo da função criptográfica, o efeito da modificação de um bite na entrada é amplificado em uma avalanche imprevisível. Ao final do processo, em média, metade dos bites da saída deve ser igual 0 e metade igual a 1" (FEISTEL, 1973, p.15-23).

O *efeito avalanche* se caracteriza em uma função criptográfica iterativa quando o número de bites da entrada que afetam cada bite da saída cresce a cada iteração, tendo como limite o tamanho do bloco de entrada. A fim de apresentar formalmente o conceito de *avalanche*, considere-se o seguinte:

Seja $f_r : Z_2^m \rightarrow Z_2^m$ uma transformação criptográfica iterativa composta da forma $f(X)=h^r(X) \cdot g(X)$, onde *h(X)* é uma transformação realizada *r* vezes *(r>0)*. Seja $W_i(f_r)$ o número de bites da entrada que afetam o *i-ésimo* bite da saída para a transformação f_r $(0 \leq i < m)$. Temos então que $W_i(f_r)$ é dado pela expressão 2-I.

$$W_i(f_r) = \sum_{j=0}^{j=m-1} D_{(f\,r,\ i,\ j)} \quad , (0 \leq i \leq m) \qquad (2\text{-I})$$

Note-se[40] que $0 \leq W_i(f_r) \leq m \ \forall \ i$.

Definição: Uma transformação criptográfica f : $Z_2^m \rightarrow Z_2^m$ apresenta o *efeito avalanche* se e somente se o número $W_i(f_r)$ de bites que afetam o *i-ésimo* bite da saída é uma função crescente[41] de r para todos os valores de i $(0 \leq i < m)$ e é estritamente crescente para pelo menos um *i* em algum intervalo $[r_1, r_2]$ $(0 \leq i < m , 0 < r_1 < r_2)$.

O conceito de avalanche está intimamente relacionado com o conceito de *completude*.

[39] Obs.: a função identidade $f(X)=X$ satisfaz à HNC.

[40] Rigorosamente, $Wi>0$. $Wi=0$ implicaria que o *i-ésimo* bite da saída *independe* da entrada e este tipo de transformação não é objeto de nosso estudo.

xxxii) Critério de avalanche estrito (CAE)

Sempre que for mudado um bite da entrada, cada bite da saída deve mudar com probabilidade ½ (MENEZES, 1996, p. 277).

xxxiii) Completude

Uma transformação criptográfica $f : Z_2^m \rightarrow Z_2^n$ é dita *completa* se e somente se para qualquer par de posições (i, j) existe pelo menos um par de blocos de entrada X, X' tais que X, X' diferem apenas pelo *i-ésimo* bite e seus respectivos blocos de saída Y', Y' diferem pelo *j-ésimo* bite, pelo menos. Ou seja, uma função é *completa* se cada um dos bites de saída depende de todos os bites da entrada (MENEZES, 1996, p. 277). Formalmente, temos:

Definição: Uma transformação criptográfica $f : Z_2^m \rightarrow Z_2^n$ *é dita* ***completa*** *se e somente se*

$$\boldsymbol{D_{(f,i,j)} = 1} \quad \forall\ (i, j) \in (Z_m \times Z_n)$$

xxxiv) Perturbação

O conceito de perturbação foi criado para quantificar o *efeito avalanche*. Seja a função criptográfica $f : Z_2^m \rightarrow Z_2^n$. Sejam $X \in Z_2^m$ *e* Ui o *vetor complemento unitário-i* , $i \in Z_m$.

Definição: *Perturbação* $P_{X,i}$ é o peso *Hamming*[42] do vetor $Y=f(X) \oplus f(X \oplus U_i)$.

Se a função analisada for *completa*, espera-se que o valor médio[43] da perturbação seja *n/2*. O conceito de perturbação é útil para quantificar a *difusão* em função do número de iterações. Este conceito está intimamente relacionado com o CAE.

xxxv) Segurança

A *segurança* é o principal atributo de um algoritmo criptográfico. Sempre relacionado ao *fator trabalho* estimado para a criptoanálise, o aspecto segurança está ligado basicamente à inexistência (aparente) de *atalhos* e à resistência às formas de criptoanálise conhecidas(OSWALD, 2002). Desde que tais condições sejam atendidas, a segurança passa a ser uma função do tamanho da chave, uma vez que a busca exaustiva não explora propriedades estruturais do cifrador de blocos. Naturalmente, a existência ou não de um

[41] Obs: rigorosamente, uma função *constante* é *crescente*.
[42] O peso Hamming de um bloco X é o número de bites iguais a *um* no bloco X.
[43] Escolhendo-se aleatoriamente diversos pares $(X, i) \in Z_2^m \times Z_m$.

atalho em um algoritmo é um indicador de qualidade amplamente aceito pela comunidade internacional.

O único tipo de sistema criptográfico comprovadamente seguro encontrado na bibliografia é o *One Time Pad*[44], cuja utilização não e prática. Como, para fins práticos, não é possível garantir a segurança neutralizando os riscos de ataque, resta tentar identificar, classificar e mensurar os riscos. A exigência de segurança é satisfeita quando não é conhecida nenhuma forma de tornar o esforço da criptoanálise significativamente menor que o da força bruta (nenhum atalho ou alçapão) e o tamanho da chave torna a busca exaustiva impraticável ou não compensadora (SCHNEIER, 1996, p. 8) (BLAZE, 1995, p. 2). Diz-se que a criptoanálise é não-compensadora quando o seu custo estimado é maior do que o valor da informação a ser protegida. Atualmente[45], um espaço de chaves com 2^{90} combinações possíveis é considerado computacionalmente seguro (BLAZE, 1995) (SCHNEIER_02, p. 3).

Observe-se que o desenvolvimento ou o conhecimento de uma forma teórica de criptoanálise que reduza o esforço estimado para a quebra do algoritmo em relação à busca exaustiva, embora seja um fato que depõe contra a qualidade da cifra, só representa uma ameaça real se o fator trabalho resultante for menor do que 2^{90} (LEECH, 2001, 14)**.** Um aspecto que também deve ser observado é que um algoritmo para proteger informações deve permanecer seguro ao longo de toda a vida útil da informação, e que a despesa para a construção de máquinas dedicadas à criptoanálise deve cair ao longo do tempo (DIFFIE, 1977). O custo em *milhões x horas* destas máquinas é limitado superiormente pelo tamanho da chave e inferiormente (critério prático para *estimativa*) pela complexidade das formas de criptoanálise conhecidas. Para tanto, deve-se supor que o surgimento de uma nova técnica de criptoanálise contra o algoritmo é suficientemente improvável. Uma discussão interessante sobre custo de criptoanálise pode ser vista em (SCHNEIER, 1996, p. 300).

A maioria dos ataques por atalho descritos na literatura não pode ser implementada na prática (OSWALD, 2002, p. 2). Apesar disso, a existência de um atalho compromete seriamente a *credibilidade* do algoritmo. Ataques em modelos reduzidos dos algoritmos podem auxiliar na estimativa da margem de segurança[46] que tal algoritmo oferece (BIHAM, 1999, p. 4), embora tal afirmação seja contestada pelo pessoal do NESSIE (PRENEEL, 2000, p. 3).

---

[44] Ou *chave de uma só vez*. Neste tipo de sistema, cuja criação e atribuída a Gilbert Vernam, cada chave é do mesmo tamanho do texto em claro e é utilizada uma única vez (SCHNEIER, 1996, p. 15).

[45] Os autores estimaram em 1995, considerando os avanços esperados até 2005.

[46] Para um algoritmo de R iterações a existência de um ataque para quebrar o algoritmo em R-r iterações revela uma margem de segurança absoluta de r iterações ou uma margem relativa de r/R. É importante lembrar que a

3 ALGORITMOS DES, AES E SERPENT: UMA BASE PARA A UMA FILOSOFIA DE PROJETO DE CIFRADORES DE BLOCOS

Em 15 de maio de 1973, o então NBS[47] requisitou à comunidade (indústrias e comunidade científica) propostas de algoritmos para que um deles fosse adotado como o padrão oficial de criptografia de dados. O padrão de criptografia seria utilizado por órgãos do governo para proteger informações *sensíveis mas não classificadas*[48], podendo também ser adotado por instituições não governamentais (LEECH, 2001, p. 1 e 10) (SCHNEIER, 1996, p. 612). O NBS estabeleceu que os candidatos deveriam atender às exigências apresentadas em 3.2. Inicialmente, nenhum candidato reuniu as condições exigidas. Apenas após um segundo anúncio, em 27 de agosto de 1974, um único algoritmo, baseado no *Lucifer*[49], parecia ser capaz de atender a todas as condições impostas [SCHNEIER_96, 266]. O algoritmo sofreu algumas modificações e foi adotado como DES em 1977, permanecendo como padrão de criptografia de dados até hoje. Em 1998 um concurso para selecionar um novo padrão foi concluído e uma versão do algoritmo Rijndael foi anunciado como AES. O conjunto de exigências e critérios de seleção adotados para escolher o AES encontra-se comentado no item 3.3.

3.1 OBJETIVOS DO CAPÍTULO

Os objetivos deste capítulo são:

i) Comentar as exigências feitas pelo NIST (NBS[1]) aos algoritmos candidatos nos processos de adoção do DES (1973) e seleção do AES (1998), deixando claras as semelhanças e diferenças entre os dois conjuntos de exigências

---

dificuldade de criptoanálise normalmente cresce exponencialmente com o número de iterações (exceto no ataque algébrico teórico XSL, em (COURTOIS, 2002),(OSWALD, 2002)).

[47] *National Bureau of Standards*, setor do Ministério do Comércio norte-americano que posteriormente viria a se chamar *National Institute of Standards and Technology (NIST)*, uma agência do *U.S. Department of Commerce*.

[48] *unclassified* – termo que se refere a informações que não comprometem a segurança nacional, de acordo com a lei de segurança nacional norte-americana. A expressão *sensitive unclassified* diz respeito, por exemplo, a transações comerciais e mensagens de *e-mai* (BIHAM, 1993, p. 1).

[49] O *Lucifer* foi um algoritmo desenvolvido por Horst Feistel, na IBM, no início dos anos 70. Era considerado na época um dos mais poderosos sistemas comercialmente disponíveis e consequentemente utilizado por uma grande variedade de organizações (SINGH, 2002, p. 273) (SCHNEIER, 1996, p. 265,303,304).

ii) Mostrar que este conjunto de exigências baseia-se em cinco princípios que devem ser observados quando do projeto de um cifrador: SEGURANÇA, EFICIÊNCIA, FLEXIBILIDADE, SIMPLICIDADE, CONFIABILIDADE

iii) Comentar os aspectos positivos observados em cada um dos três algoritmos, sob a ótica dos princípios de projeto

iv) Comentar as críticas e aspectos negativos dos três algoritmos em estudo, sob a ótica dos princípios de projeto

v) Justificar a escolha dos algoritmos DES, AES e Serpent para o trabalho relatado nesta dissertação

Em seguida aos critérios de aceitação e escolha dos algoritmos, encontram-se comentadas as principais críticas sofridas por cada algoritmo e os principais aspectos favoráveis (positivos) encontrados nos respectivos projetos.

No capítulo 5 mostra-se que estes três importantes algoritmos criptográficos apresentam semelhanças[50] em suas estruturas, o que sugere que o estudo de tais características pode contribuir para fundamentar uma base para um conjunto de técnicas de projeto para cifradores simétricos de blocos.

3.2 O PROCESSO DE SELEÇÃO DO DES, E OS ASPECTOS POSITIVOS E NEGATIVOS DA CIFRA

3.2.1 EXIGÊNCIAS QUE TIVERAM QUE SER ATENDIDAS PELO DES

3.2.1.1 O ALGORITMO DEVERIA PROVER ALTO[51] NÍVEL DE SEGURANÇA

Este é o aspecto mais importante de um algoritmo criptográfico padrão. Em 1977, estimava-se um custo de 20 milhões de dólares para realizar uma criptoanálise exaustiva com 2^{56} chaves possíveis em 12 horas[52] (DIFFIE, 1977, p. 1).

[50] Em suas descrições originais estas semelhanças podem não ser evidentes.

[51] O conceito de *alto nível* de segurança é subjetivo (vide capitulo 2) e está relacionado à confiança que um algoritmo criptográfico adquire ao longo do tempo enquanto não é *quebrado*. A expressão foi utilizada apenas para manter a conformidade com a fonte.

[52] O raciocínio para este custo é baseado na quantidade *milhões x horas*, neste caso igual a 240, que indica que uma máquina de 10 milhões poderia realizar o mesmo trabalho em 24 horas. A cada bite acrescido ao tamanho da chave o custo estimado da busca exaustiva em *milhões x horas* dobra, pois está diretamente associado ao *fator trabalho*.

3.2.1.2 O ALGORITMO DEVERIA SER COMPLETAMENTE ESPECIFICADO E DE FÁCIL COMPREENSÃO

Este aspecto é fundamental para a aplicação prática do algoritmo e a sua efetiva utilização pela comunidade. Para que esta condição possa ser satisfeita o algoritmo deve ser forte o suficiente para satisfazer o princípio de *Auguste Kerckhoffs,* reforçando a exigência de ***segurança*** do algoritmo. Além disso, esta exigência sugere a ***simplicidade*** como um dos princípios de projeto. No DES, a simplicidade[53] de cada uma das operações realizadas (permutações e substituições realizadas em seqüências de até 64 bites) contribui significativamente para a facilidade de compreensão do algoritmo. A combinação dos princípios de ***segurança*** e ***simplicidade*** conduz à ***credibilidade*** do algoritmo, pois é esperado que a simplicidade das operações evidencie qualquer fraqueza potencial da cifra.

A satisfação desta condição também tem forte influência sobre o tempo que o padrão leva para ser popularizado, tendo grande importância do ponto de vista comercial. Este aspecto sugere que a ***credibilidade*** do algoritmo é um dos princípios que devem nortear o seu projeto. No caso do DES, a indústria levou de três a seis anos para chegar a um consenso sobre a sua utilização comercial (LEECH, 2001, p. 2).

3.2.1.3 A SEGURANÇA DO ALGORITMO DEVERIA RESIDIR UNICAMENTE NO SIGILO DA CHAVE, E NÃO NO SIGILO DO ALGORITMO

Esta exigência é o próprio enunciado do princípio de *Kerckhoffs* e é praticamente um complemento da exigência anterior. É uma decorrência da necessidade de o algoritmo poder ser amplamente divulgado e completamente definido, sem que seja comprometida a ***segurança***.

3.2.1.4 O ALGORITMO DEVERIA ESTAR DISPONÍVEL PARA TODOS OS USUÁRIOS

Esta condição, dependente da liberação dos direitos sobre a propriedade intelectual, é muito importante do posto de vista comercial, devido à isenção de *royalties* ser fundamental para estimular o desenvolvimento de produtos contendo o DES pela indústria em geral (LEECH, 2001, p.1). Desde que o princípio da ***segurança*** seja satisfeito, a ampla utilização do algoritmo e a sua ***credibilidade*** se realimentam mutuamente, de tal forma que o aumento da *credibilidade* estimula a utilização do algoritmo e *vice-versa*.

3.2.1.5 O ALGORITMO DEVERIA SER ADAPTÁVEL PARA USO EM DIVERSAS APLICAÇÕES

Esta exigência sugere a ***flexibilidade*** do algoritmo como um dos princípios a serem observados em seu projeto.

Assim como a ***credibilidade*** e a liberdade em relação aos *royalties*, a ***flexibilidade*** em relação à plataforma de utilização também é de grande importância comercial. Isto porque um padrão criptográfico é utilizado em diversos tipos de componentes, como máquinas tipo de atendimento eletrônico[54], cartões inteligentes, terminais de computadores, equipamentos de criptografia de voz, etc...

Apesar de não haver uma relação rígida entre ***simplicidade*** e ***flexibilidade***, a *simplicidade* do projeto tende a torná-lo mais flexível, uma vez que, sendo melhor compreendido, pode ser mais fácil e eficientemente adaptado para diferentes aplicações.

3.2.1.6 A IMPLEMENTAÇÃO EM DISPOSITIVOS ELETRÔNICOS DEVERIA SER ECONÔMICA

Esta exigência se fazia necessária tendo em vista a perspectiva de utilização do padrão em dispositivos eletrônicos comerciais e inclusive em cartões inteligentes. Para atender tal exigência de maneira competitiva dois princípios devem ser seguidos: ***simplicidade*** e ***flexibilidade*** em relação às plataformas de utilização.

Diversos componentes eletrônicos foram desenvolvidos incorporando o DES e encontram-se comercialmente disponíveis até hoje a custos relativamente baixos. O baixo custo de implementação é fundamental, sobretudo em produtos como cartões inteligentes, que precisam ser utilizados em grande quantidade e não podem ter o preço final muito alto (LEECH, 2001, p.14) (SCHNEIER_96, p. 278). No caso do DES, por exemplo, a quantidade de bites a ser manipulada foi limitada pela tecnologia dos *chips* de então[55], nitidamente com vistas a tornar mais simples e eficientes as implementações em *hardware* (SCHNEIER, 1996, p. 293-4).

---

[53] Vide comentário sobre *simplicidade* nos critérios de julgamento do AES em 3.4.5.
[54] *Automatic Teller Machine*
[55] A quantidade de bites na entrada das tabelas de substituição foi adotada levando-se em consideração que 6 era a quantidade de bites acomodável em um único *chip* com tecnologia de 1974.

A análise deste tópico sugere que, no projeto de um algoritmo criptográfico, deve-se primar por estruturas simples e flexíveis, levando em consideração a tecnologia disponível.

3.2.1.7 A UTILIZAÇÃO DO ALGORITMO DEVERIA SER EFICIENTE

A necessidade de satisfação desta exigência nos conduz à adoção da eficiência como um dos princípios de projeto. Este conceito está relacionado ao aspecto da velocidade ou desempenho do algoritmo nas plataformas em que se pretende utilizá-lo, sendo de grande importância técnica e comercial. A importância fica clara sobretudo nas aplicações que exigem maior velocidade como por exemplo máquinas tipo ATM. Uma apreciação sobre a velocidade dos componentes com o DES embutido pode ser encontrada em (SCHNEIER, 1996, p. 278).

Entretanto, a *eficiência* de um algoritmo tem uma relação intrínseca com o nível de *segurança* do mesmo. Esta relação fica clara quando levamos em consideração, por exemplo, a forma como estes dois aspectos se relacionam com o número de iterações. O custo da criptoanálise diferencial, por exemplo, cresce exponencialmente[56] com o número de iterações, enquanto a velocidade diminui de forma aproximadamente linear com o mesmo parâmetro.

Atualmente, mais evidentemente do que ocorria no final dos anos 70, a tendência ao aumento de velocidade dos computadores torna boa prática dar ênfase à segurança mesmo que, possivelmente, em detrimento de alguma eficiência (velocidade). Esta consideração deve contribuir para aumentar a expectativa de vida útil de um padrão criptográfico e está de acordo com o que sugeriu Lars Knudsen em (KNUDSEN, 2000).

Mais uma vez, os princípios de ***segurança***, ***simplicidade***, e ***flexibilidade*** ficam evidentes. Estes princípios devem orientar o projeto no sentido de produzir uma estrutura de cifrador ***simples*** e ***flexível*** o suficiente para fornecer o nível de ***segurança*** desejado mesmo com uma quantidade de iterações pequena o suficiente para que a ***eficiência*** do algoritmo seja satisfatória.

As técnicas de projeto devem, portanto, conduzir a estruturas que produzam em poucas iterações os efeitos[57] desejados.

A comparação da performance de novas funções criptográficas com a performance (eficiência) das operações no DES é um recurso para auxiliar no projeto de novas transformações criptográficas neste trabalho.

[56] (OSWALD, 2002)

3.2.1.8 O ALGORITMO DEVERIA SER VALIDÁVEL

Este aspecto tem grande importância do ponto de vista da ***segurança*** e também comercial. A ***simplicidade*** do projeto tem grande importância na simplicidade dos processos de validação do algoritmo. Deveria ser possível certificar produtos contendo o DES. Deveria ser possível, portanto, definir testes capazes de determinar se o produto em questão realizava a função do DES. Esta característica tem grande impacto na produção e comercialização de produtos e foi muito importante para o sucesso do DES. De fato, diversos produtos incorporando o DES foram validados[58], tanto em *hardware* quanto em *software*, conforme pode ser visto em (LEECH, 2001).

As técnicas de projeto, sobretudo as técnicas para construção das operações utilizadas na função de passo, devem portanto levar em consideração o princípio da ***simplicidade*** aplicado também à validação dos produtos incorporando o algoritmo, a fim de que a ***segurança*** e a ***credibilidade*** do algoritmo sejam extensíveis aos produtos que o incorporam.

3.2.1.9 O ALGORITMO DEVERIA SER EXPORTÁVEL

A satisfação desta exigência estava relacionada às restrições para a exportação de produtos criptográficos impostas pela legislação norte-americana na época e se relacionava fortemente com restrições em relação ao tamanho da chave[59](LANDAU, 2000)(SINGH, 2001). Este aspecto é importante sob o ponto de vista da *segurança* e sob o aspecto comercial, quando existem leis ou regulamentos específicos que estabeleçam limites inferiores e/ou superiores para a segurança[60].

---

[57] Avalanche, completude, boas características diferencial e linear, por exemplo.
[58] Em (DIFFIE, 1977) é discutida a dificuldade de se provar a unicidade da chave para um determinado par texto em claro-criptograma, o que nos sugere que para satisfazer a condição de validação o algoritmo deve ser simples.
[59] É um fato que esta exigência não está explícita no conjunto de condições impostas aos algoritmos que, em 1998 candidatar-se-iam ao AES. Aparentemente, os resultados apresentados em (HOFFMAN, 1999), que trata do desenvolvimento da criptografia no mundo em face das restrições de exportação norte-americanas tiveram influência sobre esta mudança de filosofia.
[60] Uma discussão interessante sobre as questões envolvendo privacidade e segurança pode ser encontrada em (SINGH, 2001, p. 321) e uma nota recente foi publicada em *O Globo – Informática*, pg 4, de 08 Dez 2003.

3.2.2 CRÍTICAS E ATAQUES CONHECIDOS AO DES

Todas as críticas ao DES diziam respeito ao aspecto ***segurança.***

3.2.2.1 O TAMANHO DA CHAVE SERIA PEQUENO, DEIXANDO O DES VULNERÁVEL À FORÇA BRUTA

Esta crítica está ligada ao limite superior da segurança do algoritmo, que é medido pelo custo da busca exaustiva. O tamanho considerado pequeno da chave (56 bites) era a principal crítica ao DES e foi bastante abordada por Diffie e Hellman em (DIFFIE, 1977), quando o custo para quebrar o DES foi estimado em *20 milhões de dólares x 12 horas = 240 milhões de dólares x horas*. O tamanho da chave é um aspecto facilmente modificável do algoritmo e é um problema que pode ser contornado quando da especificação do algoritmo. Uma maneira simples de fazer isso no DES é utilizar uma única chave de *16 x 48 = 768* bites em vez de utilizar a função de expansão de chave original apresentada no algoritmo padrão. É importante notar que embora o aumento do tamanho da chave aumente exponencialmente o custo da busca exaustiva, o aumento do tamanho da chave em *m* bites pode[61] não significar a multiplicação do custo da criptoanálise por 2^m. Esta relação depende de como a técnica de criptoanálise empregada se relaciona com o tamanho da chave, como pode ser visto em (BIHAM, 1993).

O respeito ao princípio da segurança quando do projeto de um algoritmo criptográfico deve portanto ter como principal variável a quantidade de bites que produz um espaço de chaves suficientemente grande para ser considerado seguro com a tecnologia disponível (BLAZE, 1995).

3.2.2.2 ALEGAÇÃO DE QUE O DES É UMA VERSÃO ENFRAQUECIDA DO *LUCIFER*

Esta crítica surgiu principalmente pela redução do tamanho da chave de 128 bites (no Lucifer) para 56 bites (64 bites ao todo, menos 8 bites de paridade) (SINGH, 2001, p. 274)(SCHNEIER, 1996, p. 267).

[61] No caso do DES, é possível quebrar o a cifra com 16 chaves independentes (16x48=768 bites) em até 2^{61} passos em um ataque com textos conhecidos, o que é muito menos que 2^{768}, mas continua bastante superior a 2^{56} (BIHAM, 1993, p. 8).

A criptoanálise diferencial[62] aplicada às duas[63] versões do *Lucifer* e ao DES mostrou que, pelo menos em relação a este tipo de criptoanálise, tal crítica não procede (SCHNEIER, 1996, p.303, 304)(MENEZES,1996, p. 276).

Em respeito aos princípios de *segurança* e *credibilidade*, é conveniente que as técnicas de projeto de um algoritmo criptográfico possibilitem *comprovação*[64] da resistência do algoritmo contra as principais formas de criptoanálise conhecidas. De fato, como ficou claro posteriormente no processo de seleção do AES, a resistência às criptoanálises diferencial e linear é um parâmetro importante na estimativa da segurança de um algoritmo.

3.2.2.2 EXISTÊNCIA DE CHAVES FRACAS E SEMIFRACAS

Uma chave fraca para o DES é uma chave tal que $E_K(E_K(x))=x$, para todo x, definindo assim uma *involução*. Um par de chaves *semifracas* para o DES é um par de chaves (K_1, K_2) tal que $E_{K1}(E_{K2}(X))=X$. A operação de cifrar com uma das chaves de um par de chaves *semifracas* do DES é idêntica à operação de decifrar utilizando a outra chave *semifraca* do referido par. O DES tem quatro chaves *fracas* e seis pares de chaves *semifracas* conhecidas. A existência e as características das chaves *fracas* e *semifracas* do DES encontram-se apresentadas em (MENEZES, 1996, p. 257-258) e (SCHNEIER, 1996, p. 282-283), por exemplo. A quantidade de chaves fracas e *semifracas* no DES, entretanto, é muito pequena em relação ao tamanho do espaço de chaves. A probabilidade de se utilizar uma destas chaves ao acaso é pequena o suficiente para poder ser desprezada e, em última análise, tais chaves são facilmente evitáveis. A existência de tais chaves decorre da forma como as sub-chaves são geradas a partir da chave inicial no DES, em particular das simetrias apresentadas pelo algoritmo de expansão de chaves (vide capítulo 5, 5.4). A expansão da chave no DES é alvo de muitas críticas, pois define cada sub-chave como uma subseqüência de bites *escolhidos diretamente da chave fornecida pelo usuário*. Esta deficiência pode e deve ser contornada quando do projeto de um algoritmo, com técnicas que utilizem chaves completamente independentes ou utilizem transformações lineares e não-lineares no processo de expansão de chaves, quando este for usado. É também recomendável, neste caso, que estas transformações apresentem o *efeito avalanche*.

---

[62] Esta forma de criptoanálise foi desenvolvida por Adir Shamir e Eli Biham (BIHAM, 1993) (LIMA, 1999) (CASTAÑO, 1998).

[63] Shamir e Biham publicaram resultados de testes com criptoanálise diferencial em duas versões do *Lucifer* , às quais Schneier se refere como primeira e segunda "encarnações".

[64] Ou mensuração.

A clareza da forma como as transformações utilizadas na expansão das chaves são projetadas e empregadas deve ser suficiente para dar boa credibilidade a esta função.

3.2.2.3 SUSPEITAS DE QUE A NSA TIVESSE INSERIDO UM ALÇAPÃO NO DES

Estas suspeitas devem-se principalmente à pouca informação no que diz respeito ao projeto das tabelas de substituição do DES. Um comitê do Senado americano investigou a matéria, mas os resultados foram considerados material classificado, exceto uma parte, que foi publicada isentando a NSA de qualquer envolvimento "impróprio" com o projeto do DES, apesar de aquela agência ter modificado o conteúdo das tabelas de substituição (SCHNEIER, 1996, p. 280). Tais modificações provavelmente se devem ao fato de a IBM já ter conhecimento da criptoanálise diferencial[65] na época do projeto do DES, embora isso não fosse divulgado (BIHAM, 1994, p. iv). Depois que a criptoanálise diferencial foi publicada, a IBM publicou os critérios de projeto utilizados para a construção das tabelas de substituição (SCHNEIER, 1996, p. 293-4).

Em toda a bibliografia pesquisada referente aos últimos 25 anos de pesquisa em busca deste suposto alçapão, inclusive nos trabalhos (LIMA, 1999), (XEXEO, 1983) e (XAVIER JR, 1999), desenvolvidos no Instituto Militar de Engenharia, não foi encontrada nenhuma evidência da sua existência. Ao contrário, diversas das fontes mais recentes fazem referência ao fato de ninguém ter publicado nenhum resultado que mostre a existência de tal alçapão (BIHAM, 1993, p. 2) (CARVALHO, 2001, p.103-104)(LOUDON, 2000, p. 457) (MENEZES, 1996, p. 256-259) (SCHNEIER, 1996, p. 278-280).

Uma tabela de substituição tem conteúdo aparentemente aleatório e é preciso que fique claro como este conteúdo foi produzido, respeitando o princípio da ***simplicidade***. Se tal princípio não for empregado de maneira transparente, a ***credibilidade*** do cifrador pode ficar comprometida limitando a sua utilização. As técnicas de projeto utilizadas nas transformações lineares e nas tabelas de substituição devem levar isso em consideração, procurando gerar os padrões pseudo-aleatórios de forma "confiável" para o usuário ou avaliador do algoritmo.

[65] Um forma de criptoanálise importante, que é descrita no Anexo II.

3.2.2.4 VULNERABILIDADE ÀS CRIPTOANÁLISES DIFERENCIAL E LINEAR

Em 1990, Eli Biham e Adi Shamir apresentaram[66] o conceito *de criptoanálise diferencial*. O melhor ataque por criptoanálise diferencial contra o DES completo (16 passos) encontrado na literatura requer 2^{47} textos escolhidos (ou 2^{55} textos conhecidos) e 2^{37} operações com o DES. Este ataque é altamente teórico e impraticável para a maioria das instituições. Segundo Schneier (1996), para acumular os dados necessários a este ataque, cifrando 1.5 megabites de *textos escolhidos* por segundo, seria necessário um tempo de quase três anos (SCHNEIER, 1996, p. 285-90). A execução da criptoanálise diferencial foi demonstrada apenas em um modelo reduzido do DES e não foi encontrado na literatura nenhum registro de implementação prática que tenha sido feita no DES completo. As decifrações[67] de criptogramas do DES registradas até o ano de 2000 foram feitas pela força bruta, uma utilizando uma máquina dedicada, e outra utilizando uma associação de dezenas de milhares de computadores em paralelo (SCHNEIER, 2002, p. 1)(LANDAU, 2000, p. 115). Não foi encontrado nenhum resultado mais significativo publicado até a presente data. Uma descrição desta forma de criptoanálise pode ser vista em (DAEMEN, 1995) e (CASTAÑO, 1998).

A criptoanálise linear, criada por Mitsuru Matsui (MATSUI, 1994), utiliza aproximações lineares[68] para descrever a ação de um cifrador de blocos. Uma apresentação um pouco mais detalhada desta forma de criptoanálise pode ser vista em (LIMA, 1999) e (CASTAÑO, 1998). Nos capítulos 4 e 7 é tratado o aspecto da resistência contra sa criptoanálises diferencial e linear. Segundo relatado por Schneier e também por Menezes, a criptoanálise linear pode recuperar uma chave de DES (16 passos) com uma média de 2^{43} *textos conhecidos*. Um implementação em *software* deste ataque contra o DES completo recuperou uma chave em 50 dias usando 12 estações de trabalho HP9000/735 e este foi o mais efetivo ataque contra o DES até 1996 (SCHNEIER, 1996, p. 289)(MENEZES, 1996).

Estas formas de ataque ganharam destaque sobretudo pelo desempenho em modelos reduzidos do DES.

---

[66] Os mesmos autores citam que a criptoanálise diferencial já era conhecida pela IBM quando do projeto do DES, mas foi mantida em segredo por ser considerada uma forma poderosa de criptoanálise (BIHAM, 1993, p. vi).

[67] Em geral, o termo *decifração* não se aplica à técnica de "criptoanálise". Entretanto, no caso particular da força bruta (busca exaustiva), o que efetivamente ocorre é um conjunto de *tentativas de decifração* com chaves supostas.

[68] Aproximações lineares são equações lineares de operações OU-EXCLUSIVO de bites do texto em claro, do criptograma e da chave que valem com alguma probabilidade $p \neq 1/2$.

A criptoanálise diferencial e a criptoanálise linear são fortemente dependentes sobretudo da estrutura das tabelas de substituição, e é possível, aplicando os princípios da ***simplicidade*** e da ***segurança*** no projeto do algoritmo, determinar limites inferiores para as complexidades teóricas de dados e de processamento para ambas (diferencial e linear).

3.2.3 ASPECTOS POSITIVOS DO DES

Na época que antecedeu à adoção do DES, o NBS solicitou que a NSA[69] avaliasse a segurança do DES e a conveniência de sua utilização como padrão. O DES esteve submetido às análises das agências de segurança e ao público em geral desde 17 de março de 1975 (SCHNEIER, 1996, p. 266). Ainda em 1976, houve duas oficinas conduzidas pelo NBS para analisar o DES. A primeira oficina analisou matematicamente o algoritmo a fim de verificar a possível existência de algum alçapão. O segundo destinou-se a estudar a possibilidade de aumentar o tamanho da chave. Foram convidados os criadores, os avaliadores, implementadores, usuários e críticos. Apesar de haver criticas[70], a adoção do DES como padrão oficial foi publicada em 15 de janeiro de 1977 (SCHNEIER, 1996, p. 267)(FIPS 46, 1977).

O DES também foi aprovado pelo *American National Standards Institute (ANSI)* como padrão de criptografia para o setor privado em 1981 (ANSI X3.92). O ANSI refere-se ao algoritmo como *Data Encryption Algorithm (DEA)*. Além de publicar um padrão para os modos de operação do DEA (ANSI X3.106) similar ao documento do NBS, o ANSI também publicou um padrão criptográfico a ser utilizado em redes que utilizam o DES (ANSI X3.105).

Diversos outros[71] padrões baseados no DES foram publicados pelo ANSI para serem utilizados em transações comerciais e financeiras e no trâmite de informações sigilosas em geral (SCHNEIER, 1996, p. 267) . O DES foi reafirmado como padrão em 1983, 1988 e 1993, quando foi anunciada a intenção de promover a sua substituição na revisão seguinte, o que aconteceu em 1996, quando do início do processo de seleção do AES (LEECH, 2001, p. 19).

[69] Agência de Segurança Nacional norte-ameriacana - *National Security Agency*.
[70] A única crítica consistente encontrada na bibliografia refere-se ao tamanho da chave, aspecto este que pode ser facilmente modificado(vide item 3.2.2 - Criticas ao DES).
[71] ANSI X9.8, ANSI X9.19, ANSI X9.24, ANSI X9.9, ANSI X9.17, ANSI X9.26, ANSI X9.23

A Associação Americana de Banqueiros[72], que desenvolve voluntariamente padrões para a indústria financeira, publicou um padrão recomendando a utilização do DES para criptografia (SCHNEIER, 1996, p. 268).

Segundo Carvalho, o que se pensa atualmente[73] é que a vulnerabilidade do DES não está no algoritmo em si, mas no tamanho da chave (CARVALHO, 2001, p. 104).

Estruturalmente, o DES apresenta as seguintes qualidades úteis como referência:

i) É uma estrutura tipo *rede de substituição e permutação* Feistel, que tem um potencial de avalanche intrínseco e apresenta boa relação entre segurança e eficiência;

ii) Por ser uma estrutura Feistel, possibilita a utilização de uma função criptográfica não invertível na função de passo, o que dá mais liberdade em relação ao projeto desta função e permite que a mesma estrutura seja utilizada para cifrar e decifrar;

iii) Foi projetado levando em consideração a tecnologia disponível na época, limitando assim a quantidade de bites operados às possibilidades de acomodação dos circuitos integrados comerciais;

Também Knudsen concorda que o DES é um bom padrão de referência (KNUDSEN, 2000, p. 1). As qualidades do DES são reconhecidas e as suas vulnerabilidades foram exaustivamente estudadas por mais de 25 anos, sendo muito pouco provável que exista alguma forma de ataque ainda desconhecida (MENEZES, 1996, p. 1, 256-259). Este aspecto contribui para a ***credibilidade*** de um projeto que contemple uma estrutura que seja suficientemente semelhante ao DES para se possa acreditar que os ataques aplicáveis ao novo cifrador também seriam aplicáveis ao DES.

O DES foi e é tão amplamente divulgado, estudado e conhecido que constitui uma base do conhecimento de criptologia. Pode-se estender o conceito de ***simplicidade*** à propriedade de um cifrador ser semelhante ao DES em sua descrição.

3.2.4 CONCLUSÃO QUANTO À IMPORTÂNCIA DO PROCESSO DE SELEÇÃO DO DES E DE SUA HISTÓRIA PARA A DEFINIÇÃO DE UMA FILOSOFIA DE PROJETO DE CIFRADORES DE BLOCOS

O DES foi o mais importante e, até hoje, o mais estudado algoritmo de criptografia. A sua influência está presente em um grande número de algoritmos criptográficos modernos

[72] *American Bankers Association*
[73] 2001

(DAEMEN, 2002, p. 81). Pelo acima exposto, e também com base nos estudos realizados no IME ((LIMA, 1999)(XEXEO, 1983)(XAVIER JR, 1999)), o DES foi considerado um dos pontos de partida para este trabalho.

Os aspectos abordados acima foram fundamentais para a definição da filosofia de projeto apresentada no capítulo 6.

3.3 EXIGÊNCIAS MÍNIMAS E CRITÉRIOS DE JULGAMENTO PARA A SELEÇÃO OS CANDIDATOS FINALISTAS PARA O AES, ENTRE ELES O RIJNDAEL E O SERPENT (NECHVATAL_99)

Na publicação do Registro Federal dos EUA (U.S. Federal Register) de 2 de janeiro de 1997, o NIST requisitou à comunidade internacional informações a fim definir o processo para escolher o algoritmo que substituiria o DES. A compilação destas informações, juntamente com a experiência adquirida com o DES[74] levou o NIST, naquele mesmo ano, a anunciar o conjunto de condições (exigências mínimas) para a aceitação de algoritmos candidatos à adoção no AES. Cada uma das condições está apresentada e comentada a seguir.

3.3.1 O ALGORITMO DEVERIA SER PUBLICAMENTE DEFINIDO

São válidas as considerações feitas em relação ao DES quanto aos princípios de ***segurança*** e ***simplicidade***. Além disso, no processo de seleção do AES, destacou-se o interesse pela transparência nos critérios de projeto, no sentido de tornar público também os métodos de construção das tabelas de substituição e das transformações lineares. Tais aspectos se sobressaem principalmente na proposta do Rijndael, onde a preocupação com a ***credibilidade*** é um ponto que se destaca (DAEMEN, 1999) e parece ter sido um fator importante na escolha do Rijndael como vencedor do concurso AES (NIST, 1999, p. 3) (COPPERSMITH, 2000a) (COPPERSMITH, 2000, b).

3.3.2 O ALGORITMO DEVERIA SER UM CIFRADOR SIMÉTRICO DE BLOCOS COM SUPORTE PARA BLOCOS DE, NO MÍNIMO, 128 BITES

Aparentemente, esta exigência baseia-se no seguinte raciocínio. A bem da *eficiência*, é conveniente que as operações possam ser realizadas em quantidades de 32 bites

[74] Na experiência com o DES, deve-se incluir também os resultados apresentados em (LEECH, 2001) e (HOFFMAN, 1999).

(processadores Pentium atuais) e 64 bites (tendência de popularização de processadores de 64 bites). O bloco do mesmo tamanho que a chave torna simples e práticas as transformações internas nas cifras, que normalmente envolvem somas módulo dois entre o bloco que está sendo transformado e a sub-chave de fase.

O tamanho mínimo considerado seguro para a chave atualmente é 90 bites[75]. O primeiro múltiplo de 64 (e 32) bites acima de 90 é 128. Esta exigência portanto é favorável à *segurança* e à *eficiência* e traz implícita a idéia de *flexibilidade*, uma vez que estabelece apenas um tamanho mínimo (BAUDRON, 199X, p. 1-2) (SCHNEIER, 1999)(NIST, 1999) (COPPERSMITH, 2000a) (COPPERSMITH, 2000b).

3.3.3 TAMANHO DA CHAVE

Em relação aos tamanhos da chave e dos blocos, o algoritmo deveria ser projetado de tal forma que comportasse, no mínimo, chaves de 128, 192 e 256 bites e o tamanho da chave pudesse ser aumentado conforme o necessário

Cabe o mesmo raciocínio feito no item anterior. Esta exigência parece estar relacionada com os resultados que posteriormente seriam apresentados em (HOFFMAN, 1999) e (LEECH, 2001), que tratam do desenvolvimento da criptografia no mundo em face das restrições de exportação impostas pelos EUA e ressaltam a importância comercial da vida útil do algoritmo.

3.3.4 O ALGORITMO DEVERIA SER IMPLEMENTÁVEL EM *HARDWARE* E *SOFTWARE*

Aqui se ressalta o princípio da ***flexibilidade*** em relação à plataforma de utilização. É importante também o aspecto da velocidade (desempenho). Uma vez que não ficou bem definida nenhuma plataforma prioritária para avaliação e o fator desempenho é fortemente dependente da plataforma utilizada, alguns algoritmos tiveram melhor desempenho em *hardware* e outros em *software*. É importante portanto levar em consideração as características da plataforma a que se destina o algoritmo.

[75] Em (BLAZE, 1995) este tamanho de chave é considerado suficiente para a tecnologia atual.

3.3.5 O ALGORITMO DEVERIA: A) SER DISPONÍVEL GRATUITAMENTE E B) ESTAR DISPONÍVEL DE ACORDO COM OS TERMOS DE POLÍTICA DE PATENTES DO ANSI

Esta exigência é de cunho comercial, e contribui para o emprego do algoritmo em produtos comerciais, sobretudo os de baixo preço final. Isso é fundamental para a difusão da cifra e teve grande importância para a popularização do DES, como pode ser visto ne bibliografia (LEECH, 2001).

3.4 CRITÉRIOS PARA SELEÇÃO DOS FINALISTAS DO AES

Os algoritmos que estivessem de acordo com as exigências acima, seriam comparados segundo os fatores relacionados a seguir.

3.4.1 SEGURANÇA

Assim como no caso do DES, a segurança[76] foi o aspecto mais importante considerado na seleção do AES. A equipe de avaliação do AES inicialmente excluiu parte dos candidatos observando apenas este aspecto, antes de prosseguir com a análise dos demais fatores (NIST, 1999).

3.4.2 EFICIÊNCIA COMPUTACIONAL

No AES o princípio da eficiência ficou expresso como grandeza de comparação. Este aspecto está relacionado à complexidade (de processamento e de dados) do algoritmo. Quando utilizado como critério de comparação, os resultados podem variar, dependendo da plataforma utilizada como alvo (BIHAM, 1999).

No caso do AES, o NIST adotou a plataforma *Pentium Pro 200MHz* como referência, por ser a plataforma que mais provavelmente estaria na maioria dos computadores, embora os testes tenham sido realizados em diferentes combinações de processadores, sistemas

[76] Vide conceito de segurança no item 2.36, capítulo 2.

operacionais e compiladores (BAUDRON, 199X)(SCHNEIER, 2000)(NIST, 1999, p.2). Mesmo com a tendência dos processadores de 64 bites se tornarem mais populares, é provável que os resultados comparativos de desempenho dos algoritmos permaneçam válidos (SCHNEIER, 1999, p. 2). Em (SCHNEIER, 1999), (NECHVATAL, 1999), (HINZ, 2000), (PRENEEL, 2000) e (DRAY, 1999) são feitas comparações entre as performances dos diferentes candidatos ao AES.

Uma análise rigorosa da eficiência deve levar em conta, além da velocidade nas plataformas de interesse, o tamanho do código e a margem de segurança estimada. Estão portanto relacionados os princípios de ***eficiência***, ***simplicidade, flexibilidade*** e ***segurança***.

3.4.3 NECESSIDADE DE MEMÓRIA

Este aspecto refere-se especificamente à *complexidade de dados* do algoritmo e rigorosamente está incluído na abordagem sob a ótica da *eficiência*. São particularmente importantes o tamanho do código e a quantidade de memória requisitada durante o processamento. Em situações específicas, pode ser significativa a diferença entre os tamanhos das tabelas de substituição e a forma como a função de expansão de chaves atua. É interessante que a função de expansão de chaves, quando existir, permita uma implementação em que cada sub-chave possa ser gerada recursivamente a partir da sub-chave anterior, sem que haja necessidade de armazenar todas as sub-chaves simultaneamente.

3.4.4 ADEQUAÇÃO À IMPLEMENTAÇÃO EM *HARDWARE* E *SOFTWARE*

Esta necessidade foi bem ilustrada pela popularização do DES, que foi introduzido em diversos produtos, tanto em *hardware* quanto em *software* (LEECH, 2001)(SCHNEIER, 1996). Analisando as propostas dos cinco algoritmos finalistas do AES e sobretudo a do Rijndael, podem-se observar operações que necessitam relativamente poucos ciclos de *clock*. Estas operações são as somas módulo dois e os deslocamentos de baites ou palavras. O Rijndael, por exemplo, opera grupos de 8 e 32 bites, que são tamanhos bastante convenientes à plataforma *Pentium*, o que foi de grande importância no processo de avaliação do NIST (DAEMEN, 1999) (DAEMEN, 2002) (NECHVATAL, 1999) (MASSEY, 199X) (COPPERSMITH, 2000c)(KNUDSEN, 1999).

3.4.5 SIMPLICIDADE

Nas palavras de Daemen e Rijmen "*algumas pessoas acreditam que um algoritmo criptográfico não deve apenas produzir saídas sem estrutura aparente, mas deve também esconder sua própria estrutura utilizando componentes complexas. Esta abordagem é diferente da nossa*[77]*. Nossa forma de abordagem não deve ser vista entretanto como um sinal de fraqueza no projeto do algoritmo.*"(DAEMEN, 2000, p. 2,3).

O princípio da simplicidade é de grande utilidade quando se pretende analisar a estrutura de um algoritmo em busca de atalhos[78] ou compreender os efeitos das operações sobre os blocos. A perfeita compreensão das transformações dá uma boa idéia da margem de segurança presente no cifrador (BIHAM, 1999, 4) e contribui para a sua ***credibilidade***.

O conceito de simplicidade está associado aos critérios de projeto, à fácil especificação e compreensão do algoritmo, às operações utilizadas e à semelhança do algoritmo com outros já amplamente estudados. São consideradas operações simples, pela sua eficiente implementação em *hardware* e *software*, as seguintes operações: deslocamentos, somas módulo dois[79], permutações, busca em tabelas de substituição (estas operações necessitam uma quantidade pequena de ciclos de clock, quando comparadas, por exemplo, com as operações de divisão e multiplicação, como pode ser visto em (PATTERSON, 2000).

3.4.6 FLEXIBILIDADE

Este aspecto está especialmente relacionado à possibilidade de utilização de diferentes tamanhos de chave e blocos e em diferentes ambientes (processadores de 8 bites, redes ATM, comunicações por voz e satélite, HDTV, etc.). No caso do AES o NIST também valorizou a performance dos algoritmos em cartões inteligentes[80] (SCHNEIER_1996, p.1, 10) (CHARI, 1999) (FOTI, 1998).

---

[77] Referência à filosofia de projeto do Rijndael, que torna públicas todas as motivações e estruturas envolvidas no projeto.

[78] Do inglês *shortcuts*

[79] Ou operações *ou-exclusivo*. Note-se que as multiplicações por matrizes apresentadas neste trabalho são úteis para uma melhor visualização da transformação. A implementação neste formato não é eficiente, tendo em vista a grande quantidade de operações sem efeito. As multiplicações por matriz devem ser implementadas utilizando as expressões *booleanas* que elas representam.

[80] No caso dos cartões inteligentes, especial atenção deve ser dada quanto à possibilidade de análise diferencial de potência.

3.4.7 O ALGORITMO DEVERIA SER LIVRE DE PATENTES

Cabem aqui os mesmos comentários feitos em relação ao DES.

3.5 ASPECTOS NEGATIVOS DO RIJNDAEL: CRÍTICAS E ATAQUES CONHECIDOS

A principal crítica ao Rijndael deve-se à origem puramente algébrica[81] da sua estrutura.

3.5.1 A ORIGEM ALGÉBRICA DO RIJNDAEL PODE POSSIBILITAR ATAQUES ALGÉBRICOS

A estrutura das transformações existentes no Rijndael baseia-se na álgebra de polinômios sobre corpos finitos, o que é uma filosofia de construção relativamente recente. Sendo assim, novas formas algébricas de ataque deverão surgir, às quais o DES, em princípio, não seria vulnerável. Os principais ataques conhecidos contra o Rijndael foram apresentados em (CHEON, 2001), (COURTOIS, 2002), (FERGUSON, 2000), (GILBERT, 2000), (LUCKS, 2000), (DAEMEN, 2000) e (DAEMEN, 2002). A mais importante delas é a técnica XSL, apresentada em (COURTOIS, 2002). A característica que chama mais atenção nesta técnica é que, ao contrário do que ocorre com a criptoanálise diferencial, a complexidade da criptoanálise XSL não cresce exponencialmente com o número de iterações. A técnica XSL, entretanto, transforma o problema da criptoanálise em um outro problema matemático para o qual também não existe solução eficiente.

Nenhuma forma de criptoanálise capaz de ameaçar o Rijndael foi apresentada na prática até o presente momento. Apesar disso, existe uma expectativa na comunidade científica do surgimento e aprimoramento de técnicas como a XSL e isso pode vir a abalar a *credibilidade* da cifra. No que diz respeito a credibilidade, parece ser conveniente utilizar estruturas conservadoras, para as quais se considere menos provável o surgimento de uma forma nova de ataque, como por exemplo estruturas parecidas com o DES (SCHNEIER, 2002, p. 1) (OSWALD, 2002, p. 6).

[81] A descrição algébrica do projeto do Rijndael também é considerada um aspecto positivo do projeto, pois permite demonstrações matemáticas das construções de sua estrutura e facilita a demonstração da resistência contra a criptoanálise, como pode ser visto em (CASTAÑO, 1998), (DAEMEN, 2002) e (DAEMEN, 1995).

3.5.2 O RIJNDAEL UTILIZA UMA ÚNICA E BIUNÍVOCA TABELA DE SUBSTITUIÇÃO

Esta característica diminui a incerteza na saída da transformação de substituição, o que é ruim. Uma boa técnica de projeto deve contemplar as características positivas da tabela de substituição do Rijndael (comprovadamente boas características diferencial e linear com a utilização de uma única tabela de substituição relativamente pequena) e, se possível, um aumento da incerteza na transformação utilizando uma tabela não invertível em um sistema tipo Feistel e/ou tabelas dependentes de chave**.**

3.6 ASPECTOS POSITIVOS DO RIJNDAEL

3.6.1 ELEGÂNCIA E SIMPLICIDADE DA DEFINIÇÃO MATEMÁTICA DO RIJNDAEL

A construção do Rijndael foi baseada na álgebra de polinômios em corpos finitos. A utilização das propriedades dos corpos finitos na construção do algoritmo permitiu aos projetistas dar grande credibilidade ao algoritmo no que diz respeito à ausência de alçapões e à resistência contra as criptoanálises diferencial e linear(OSWALD, 2002). Além disso, a forma de construção das tabelas de substituição é comprovadamente uma operação não linear. Estas propriedades podem ser vistas em (CASTAÑO, 1998), (DAEMEN, 2000, p. 3) e (DAEMEN, 2002).

O Rijndael foi aprovado pelo NIST e pelo NESSIE e está submetido ao escrutínio da comunidade científica desde agosto de 1998[82] (NECHVATAL, 1999, p. 3)

3.6.2 TRANSFORMAÇÃO LINEAR MAIS FORTE QUE UMA SIMPLES TRANSPOSIÇÃO

A transformação linear definida para o Rijndael, resultante da aplicação uma permutação seguida de uma multiplicação[83], atribui a cada bite da saída a soma módulo dois de diversos bites do argumento. Esta propriedade, que também se verifica no Serpent, contribui para o

[82] Nesta data o NIST anunciou os 15 algoritmos aceitos como candidatos ao AES, entre eles o Rijndael e o Serpent

aumento *do efeito avalanche* e torna mais complexas as equações lineares para a criptoanálise, sobretudo quando estas equações envolvem mais de uma iteração[84]. Em (FERREIRA, 2003) é feita uma comparação entre as transformações lineares dos algoritmos DES, Rijndael e Serpent. Nesta comparação fica clara superioridade da TL do Rijndael em relação às TL do DES.

3.6.3 RESISTÊNCIA DO RIJNDAEL ÀS FORMAS DE CRIPTOANÁLISE CONHECIDAS

A estrutura do Rijndael permite determinar limites inferiores de resistência contra as principais formas de criptoanálise, que são a CL e a CD. Isso se deve às boas características diferenciais e lineares da tabela de substituição e à boa difusão produzida pelas TL, que pode ser verificada observando as transformações de multiplicação por matriz (*MixColumns*) e permutação (*ShiftRows*) separadamente. Esse assunto é tratado mais profundamente em (DAEMEN, 1995) e (CASTAÑO, 1998).

No capítulo 4 é apresentada uma técnica que auxilia na observação do poder de difusão de uma TL, permitindo observar o seu efeito final.

3.6.4 AUSÊNCIA DE CHAVES FRACAS E SEMIFRACAS NO RIJNDAEL

Diferentemente do que ocorreu com o DES, até o presente momento não foi encontrada na bibliografia nenhuma evidência de existência de chaves fracas ou semifracas em relação ao Rijndael. Além disso, a função de expansão de chaves do Rijndael utiliza rotações combinadas com a transformação não-linear (tabela de substituição). Tal construção torna improvável a existência de tais chaves, pois não existem simetrias facilmente identificáveis que possam ser exploradas, como ocorre no DES. Esta característica de não-linearidade na função de expansão de chaves deve ser um dos requisitos da filosofia de projeto para cifradores.

---

[83] Vide a descrição do algoritmo no capítulo 5.

[84] A importância desta diferença em relação ao DES fica clara quando são observadas as *matrizes de avalanche*(cap. 4).

3.7 ASPECTOS NEGATIVOS DO SERPENT: CRÍTICAS E ATAQUES CONHECIDOS

3.7.1 LENTIDÃO

A alegada lentidão do Serpent deve-se principalmente a: i)transformação linear descrita com muitas operações; ii) grande número de iterações utilizado a fim de aumentar a sua margem de segurança; iii) o modo *bitslicing* não é muito eficiente para implementação em *sofware* (NECHVATAL, 1999, p. 32).

Segundo Lars Knudsen[85] , a lentidão da cifra deve ser encarada como parte do custo da sua grande margem de segurança e deve ser lembrado que o algoritmo é mais rápido que o Triplo-DES (FIPS 46-3, 1999) (KNUDSEN, 2000, p.2). No caso do Serpent, os autores especificaram um número de iterações duas vezes maior do que consideraram seguro.

3.8 ASPECTOS POSITIVOS DO SERPENT

3.8.1 O SERPENT FOI APROVADO PELO NIST E ESTUDADO NO IME E ESTÁ SUBMETIDO AO ESCRUTÍNIO DA COMUNIDADE CIENTÍFICA DESDE AGOSTO DE 1998

Na avaliação feita pelo NIST, e no estudo da cifra feito no IME, não foi encontrado nenhum ponto fraco importante[86] no Serpent. O resultado da sua avaliação geral no Relatório da primeira fase do concurso do AES foi positivo e o cifrador foi um dos cinco finalistas que passaram à última fase do processo de seleção (NECHVATAL, 1999, p. 32, 36).

Alguns ataques em modelos reduzidos do Serpent foram apresentados em (KOHNO, 2000). Até a presente data, não foi encontrado por este autor nenhum registro de ameaça à segurança do Serpent.

[85] Lars Knudsen é um dos desenvolvedores do Serpent.

[86] Foi apontada a importância de defesa contra DPA, especificamente em relação à função de expansão de chaves nas implementações em cartões inteligentes. A mesma crítica é feita em relação ao Rijndael. Entretanto, os ataques por DPA tem grande enfoque em hardware e na função de expansão de chaves, que não constituem objeto primordial de estudo nesta dissertação (CHARI, 1999) (SHAMIR, 1999) (NECHVATAL, 1999).

3.8.2 MARGEM DE SEGURANÇA

Os projetistas do Serpent optaram por usar o dobro do número de passos que eles acreditaram ser suficiente para a resistência da cifra contra as formas de criptoanálise conhecidas. Na proposta do algoritmo não fica claro qual a maior ameaça a que a cifra está submetida sob o ponto de vista dos autores, embora seja apresentada uma motivação baseada nas complexidades de dados para a criptoanálise diferencial. Entretanto, o seguinte raciocínio, que favorece a *credibilidade* da cifra, pode ser montado com base na apreciação da proposta e de (KNUDSEN, 2000): i) para o DES, a quantidade de iterações que torna a criptoanálise diferencial mais complexa que a busca exaustiva é dezesseis (SCHNEIER, 1996, p. 284, 289); ii) o fator trabalho para a criptoanálise diferencial é função das tabelas de substituição é das transformações lineares; iii) o Serpent, inicialmente[87], incorporava as mesmas tabelas de substituição usadas pelo DES e houve uma preocupação explícita de aumentar a *difusão* da transformação linear do Serpent em relação à do DES, o que deveria tornar o Serpent no mínimo tão resistente contra a criptoanálise diferencial quanto o DES; iv) os autores resolveram colocar uma margem de segurança conservadora, especificando um número de passos duas vezes maior que o considerado suficientemente seguro. Devido a este fato, o Serpent foi considerado o algoritmo com maior *margem de segurança* dentre os finalistas do AES. Este é um aspecto positivo em relação à segurança, embora comprometa, na opinião de alguns autores, a eficiência do algoritmo. Levando-se em consideração a rápida[88] avalanche produzida pela transformação linear do Serpent, provavelmente[89] é possível a redução do número de passos do algoritmo sem comprometer a sua segurança. Esta é uma diferença importante em relação ao DES e que ficou evidente em (FERREIRA, 2003). Segundo Knudsen, "o Serpent é sem dúvida o mais seguro dos finalistas do AES e a diferença de performance não é significativa, sobretudo considerando que, com a evolução da tecnologia, a segurança tende a decrescer[90] e a velocidade a aumentar" (KNUDSEN, 2000, p. 2).

[87] Posteriormente, as tabelas de substituição foram aprimoradas (KNUDSEN, 1999, p. 7).
[88] Essa característica também pode ser definida como *boa difusão*. Os autores do Serpent fizeram referência a esta propriedade da transformação linear em (KNUDSEN, 1999, p. 6)
[89] Para isso, seria necessário um estudo mais profundo do algoritmo com um número reduzido de passos. O importante para este trabalho no entanto é observar as características positivas do Serpent para aproveitá-las na filosofia de projeto sugerida no capítulo 6.
[90] Com o aumento de velocidade e barateamento do *hardware*, o custo da busca exaustiva deve diminuir.

Também é bastante presente no Serpent a preocupação com a credibilidade. A atenção dos autores em relação à credibilidade é confirmada em (KNUDSEN, 2000), quando o autor assegura que o Serpent é, no mínimo, tão seguro quanto o Triplo-DES (FIPS 46-3, 1999).

3.8.3 RESISTÊNCIA DO SERPENT CONTRA AS PRINCIPAIS FORMAS DE CRIPTOANÁLISE

Em (CASTAÑO, 1998) foi analisada a resistência do Serpent contra às criptoanálises diferencial e linear, concluindo que a cifra apresenta boas características de resistência contra ambas as formas de criptoanálise. Em (CHARI, 1999, p. 12) recomenda-se a proteção contra DPA, sobretudo em cartões inteligentes, levando em consideração a função de expansão de chaves. Alguns ataques em modelos reduzidos do Serpent foram apresentados em (KOHNO, 2000), sem qualquer resultado ameaçador.

3.8.4 SIMPLICIDADE

A descrição original do Serpent é relativamente[91] simples, o que facilita a sua compreensão e avaliação. Os autores do projeto tiveram a preocupação de utilizar nas tabelas de substituição (inicialmente) os mesmos valores encontrados nas tabelas de substituição do DES.

3.8.5 CHAVES FRACAS E SEMIFRACAS

Até o presente momento, não foi encontrada na bibliografia nenhum registro de existência de chaves fracas ou semifracas em relação ao Serpent. Além disso, assim como no Rijndael, a função de expansão de chaves utiliza transformações lineares e não lineares.

[91] Quando comparada, por exemplo, à do MARS, algoritmo submetido ao NIST pela IBM (BURWICK, 1999).

3.8.6 TRANSFORMAÇÃO LINEAR MAIS FORTE QUE UMA SIMPLES TRANSPOSIÇÃO

A transformação linear definida para o Serpent, resultante da aplicação de diversas rotações, deslocamentos e somas módulo dois[92], assim como no Rijndael, atribui a cada bite da saída a soma módulo dois de diversos bites do argumento. Esta construção foi adotada com a finalidade de aumentar o efeito avalanche (KNUDSEN, 1999, p. 6) e o resultado foi positivo, conforme pode ser visto em (FERREIRA, 2003), onde a transformação linear do Serpent é comparada à do Rijndael e do DES.

3.9 IMPORTÂNCIA DO PROCESSO DE SELEÇÃO DO AES E DA HISTÓRIA DO SERPENT E RIJNDAEL

O Rijndael foi utilizado como referência neste trabalho por apresentar uma filosofia de projeto essencialmente diferente do DES, por ter vencido a competição realizada pelo NIST (FIPS 197, 2001) e também ter sido aprovado pelo NESSIE[93] no relatório divulgado em 27 de fevereiro de 2003 (PRENEEL, 2003, p. 2). O *Serpent* foi utilizado por ter sido considerado o mais seguro[94] dos candidatos finalistas ao AES, não sendo escolhido aparentemente pela sua lentidão na plataforma *Pentium* (NIST, 1999, p.3-4) (KNUDSEN, 2000, p. 2) (NECHVATAL, 1999, p.7) (SCHNEIER, 2002, p. 2) (HINZ, 2000, p. 49-54) (KOHNO, 2000). Apesar de não ser um cifrador Feistel, o projeto do Serpent baseou-se em grande parte nas estruturas do DES. Os três algoritmos foram objetos de estudo em trabalhos anteriores no IME em (CASTAÑO, 1998), (XEXEO, 1983), (XAVIER JR, 1998), (LIMA, 1998).

Pelo acima exposto, o Rijndael e o Serpent foram considerados pontos de partida para este trabalho.

[92] Vide a descrição do algoritmo e item 5.7.1.
[93] *New European Schemes for Signatures, Integrity and Ecryption* (2000-2003)
[94] O conceito de seguro neste caso está associado ao número de iterações do algoritmo.

3.10 CONCLUSÃO DO CAPÍTULO

A análise dos processos de seleção do DES e do AES sugere uma série de características desejáveis em um algoritmo criptográfico. Esta análise, juntamente com as apreciações feitas nos capítulos 4 e 5, são a base para a filosofia e de projeto apresentada no capítulo 6.

4 TÉCNICAS SUGERIDAS PARA AVALIAÇÃO DE CIFRAS DE BLOCOS

4.1 INTRODUÇÃO

As técnicas de avaliação de algoritmos encontradas na bibliografia não permitem claramente comparar de maneira objetiva cifradores de blocos que apresentem estruturas descritas de forma diferentes. Neste capítulo é apresentada uma técnica para avaliar os aspectos relacionados à *difusão: efeito avalanche, completude e critério de avalanche estrito.*

No final do capítulo trata-se da relação destas técnicas com a avaliação da resistência contra a criptoanálise diferencial (CD) e a criptoanálise linear (CL).

No capítulo 9 são relacionados os principais testes estatísticos utilizados pelo NIST e pelo NESSIE para avaliação de cifras de blocos.

4.2 AVALIAÇÃO DO *PODER DE AVALANCHE* E DA *COMPLETUDE* DE UMA FUNÇÃO CRIPTOGRÁFICA

A cada iteração, a associação das transformações lineares e não-lineares presentes em um algoritmo deve fazer com que cada vez mais bites da saída *dependam*[95] de mais bites da entrada produzindo o chamado ***efeito avalanche***, que culmina quando cada um dos bites da saída é afetado por todos os bites da entrada e a função é então dita ***completa***. Os conceitos de *efeito avalanche* e *completude* são amplamente[96] difundidos e aceitos pelos especialistas como características básicas necessárias a uma função criptográfica. A seguir é apresentada uma técnica relativamente simples para a avaliação do efeito avalanche da transformação linear de uma transformação criptográfica que associa uma TL e uma TNL. Para este fim, é apresentado e utilizado o conceito de ***matriz de dependências***[97].

[95] O conceito de dependência é apresentado no capítulo 2.
[96](FEISTEL, 1973), (MENEZES, 1996, p. 277), (SCHNEIER, 1996, p. 273), (DAVIDA, 1979), (TAVARES, 1996), (BIHAM, 1993, p. 28, 34, 56), (LANDAU, 2000), (MASSEY_199X).
[97] Este conceito é bastante intuitivo e foi desenvolvido em conjunto com os autores de (FERREIRA, 2003) ao longo deste trabalho, independentemente. Posteriormente o mesmo conceito foi encontrado em (PRENEEL, 2000, p. 24).

4.2.1 PODER DE AVALANCHE NAS TRANSFORMAÇÕES SEPARADAS

As transformações presentes em uma função criptográfica são projetadas para produzirem os efeitos desejados (*não-linearidade, complexidade*, *avalanche*, *completude,* etc...) **quando funcionando em conjunto e com o número de iterações especificado em projeto**. Podem ser realizados ensaios úteis em modelos reduzidos (simplificados) dos algoritmos, desde que as conclusões tiradas destes testes sejam utilizadas de maneira prudente e criteriosa, levando em consideração as fragilidades inseridas, seja pela substituição de transformações por outras mais simples[98], seja pela pela redução do número de iterações (DAEMEN, 2000). Análises de modelos reduzidos podem ser vistas por exemplo em (XEXEO, 1983), (BIHAM, 1993), (CHEON, 2001) e (GILBERT, 2000).

No tópico seguinte é apresentada uma técnica para analisar separadamente o poder de avalanche das transformações lineares, sem levar em consideração o efeito das tabelas de substituição[99].

4.2.2 MATRIZ DE DEPENDÊNCIAS DE UMA TRANSFORMAÇÃO LINEAR

Para compreender os conceitos aqui apresentados, considere o cifrador de blocos hipotético *Beta*, cujo fluxograma apresenta-se na FIG. 4.1. As especificações do algoritmo *Beta* são as seguintes:

Beta é um cifrador de blocos simétrico do tipo SPN, que cifra um bloco *M* de seis bites segundo uma chave *K,* de mesmo tamanho, com sete iterações.

A função de passo *f* de *Beta* é composta por três transformações[100]: uma *adição módulo dois* (XOR) com a chave *K*, uma transformação linear *L* e uma substituição *N*. Portanto, *f* é dada pela seguinte expressão:

$$f(M, K) = N(L(M+K))$$

Por simplicidade didática a mesma chave é utilizada em todas as iterações. Ou seja, a chave de passo $K_i = K\ (i=0,1,2,...6)$.

[98] Esta discussão pode ser mais bem apreciada em (DAEMEN, 2000) (MURPHY, 2000).

[99] É interessante ressaltar que as tabelas de substituição também são projetadas para produzir efeito avalanche, de modo que na saída da transformação de substituição mais de um bite seja *afetado* por cada bite da entrada (SCHNEIER, 1996, p. 349, 350). Isto se verifica, por exemplo no DES, no Rijndael e no Serpent.

O criptograma C (bloco cifrado) referente à mensagem M e a chave K é obtido da seguinte maneira

$$C = f(f(f(f(f(f(f(M, K), K), K), K), K), K), K)$$

Seja $E = e_0|e_1|e_2|e_3|e_4|e_5$ um vetor de seis bites. A transformação L pode ser definida da seguinte forma:

$L(E) = S = s_0|s_1|s_2|s_3|s_4|s_5$, onde

$s_0 = e_1 \oplus e_2 \quad s_1 = e_0 \oplus e_4 \quad s_2 = e_1 \oplus e_3$

$s_3 = e_0 \oplus e_2 \oplus e_4 \quad s_4 = e_3 \oplus e_5 \quad s_5 = e_1 \oplus e_4$

Em relação à transformação de substituição N, é suficiente dizer N satisfaz a hipótese do não cancelamento das dependências (HNC).

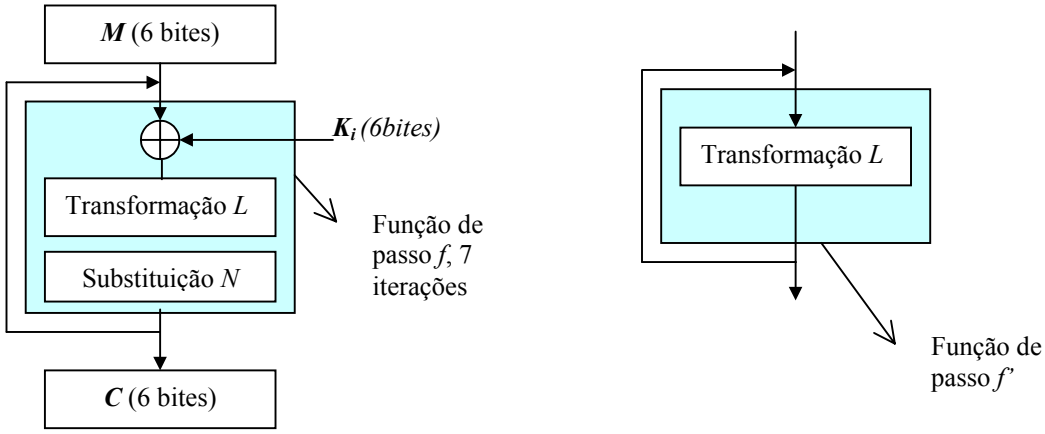


FIG. 4.1 Fluxograma do cifrador Beta

Note que, do conjunto de equações que define L, e considerando a propriedade comentada da transformação não linear N, podemos concluir (sobre as *dependências*) que

$D_{(f1,0,1)} = D_{(f1,0,2)} = D_{(f1,1,0)} = D_{(f1,1,4)} = D_{(f1,2,1)} = D_{(f1,2,3)} \ldots = D_{(f1,5,1)} = D_{(f1,5,4)} = 1$.

[100] Note-se que, se for utilizado um sistema Feistel, nenhuma destas tranformações precisa ser invertível, a exemplo do que ocorre com as tabelas de substituição (*s-boxes*) do DES (SCHNEIER, 1996, p. 347).

Para entender e utilizar o conceito de ***matriz de dependências***, analisaremos o poder de avalanche da camada linear do algoritmo *Beta* . Será utilizada para a análise a função *f'* apresentada na FIG 4.1. *f'* é a própria transformação linear *L*.

Definição: Sejam $E = e_0|e_1|...|e_{(n-1)}$ e $S = s_0|s_1|...|s_{(n-1)} = L(E)$ dois vetores de *n* bites, respectivamente a entrada e a saída de uma dada *transformação linear L*. A **matriz de dependências da transformação linear *L*** é a matriz binária quadrada $[D_1]_{nxn}$ em que o elemento $D_{i,j}$ na linha *i* e na coluna *j* ($i, j \in \{0,1,2...n-1\}$) é igual a ***1*** se, por ação da transformação linear *L*, o bite e_j afeta o bite s_i e igual a ***0*** nos demais casos.

Em outras palavras, o elemento $D_{i,j}$ da linha *i* na coluna *j* da matriz $[D_1]$ é o valor da dependência $D_{(L,i,j)}$, conforme definido no capítulo 2, item 2.3. Temos então que, para o algoritmo *Beta* apresentado, a **matriz de dependências da transformação linear *L*** é a matriz $[D_1]$, apresentada a seguir:

$$[D_1] = \begin{vmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{vmatrix}$$

A matriz $[D_1]$ é a matriz dos coeficientes das expressões booleanas para os bites s_0 a s_5 que definem a transformação *L*. Ou, de outra forma, $L(E) = [S]_{1x6} = [D_1] \cdot [E]_{1x6}^T$, onde $[E]_{1x6}^T$ é o vetor entrada $[E]_{1x6}$ transposto.

Ressalta-se que a matriz de dependências de uma TL reflete o efeito final da TL sobre o bloco que está sendo criptografado. A visualização deste efeito final não é simples quando a TL é descrita na forma de uma sucessão de deslocamentos e somas módulo dois, como ocorre nas descrições do DES (FIPS 46, 1977)(FIPS, 46-3, 1999), do Rijndael (DAEMEN, 1999)(LEECH, 2001) e do Serpent (KNUDSEN, 1999)(CASTAÑO, 1998). Na forma de *matrizes de dependências,* é possível comparar os efeitos finais das TL dos três algoritmos mencionados.

4.2.3 MATRIZES DE AVALANCHE DE UMA TRANSFORMAÇÃO LINEAR (OU MATRIZES DE DEPENDÊNCIAS RECURSIVAS)

As matrizes de dependências recursivas têm por finalidade representar quais os bites da entrada *E* que, **por ação apenas da transformação linear presente em *f***, *afetam* cada bite da saída após uma determinada quantidade de aplicações sucessivas da função *f*, utilizando como entrada para cada repetição da operação a saída da execução anterior. Para explicar o conceito de ***matriz de dependências recursivas (MDR) ou matriz de avalanche***, de uma transformação linear *L*, é útil adotar a seguinte convenção:

$S^r = s^r_0|s^r_1|...|s^r_{n-1}$ representa o vetor de saída da função *f'* na iteração número *r* (na primeira execução de *f'* $r = 1$);

$E^r = e^r_0|e^r_1|...|e^r_{n-1}$ é o vetor entrada (argumento) de *f'* na repetição número *r*.

Para explicar o conceito de *dependência recursiva* será utilizada a *indução*. Suponha que a transformação *L* do algoritmo beta foi executada uma vez ($r=1$). A saída S^1 é dada pelas equações I a VI:

$s^1_0 = e^1_1 \oplus e^1_2$ (I) $\quad s^1_1 = e^1_0 \oplus e^1_4$ (II) $\quad s^1_2 = e^1_1 \oplus e^1_3$ (III)

$s^1_3 = e^1_0 \oplus e^1_2 \oplus e^1_4$ (IV) $\quad s^1_4 = e^1_3 \oplus e^1_5$ (V) $\quad s^1_5 = e^1_1 \oplus e^1_4$ (VI)

Note-se que $E = E^1$.

Agora suponha que a transformação *L* é executada uma segunda vez (duas iterações, $r=2$). A entrada desta operação é a saída da primeira execução, ou seja, $E^2 = S^1 = L(E)$. A partir das equações que definem *L*, e lembrando que $E^2 = S^1$ temos que S^2 é dada pelas equações (VII) a (XII)

$s^2_0 = s^1_1 \oplus s^1_2$ (VII) $\quad s^2_1 = s^1_0 \oplus s^1_4$ (VIII) $\quad s^2_2 = s^1_1 \oplus s^1_3$ (IX)

$s^2_3 = s^1_0 \oplus s^1_2 + s^1_4$ (X) $\quad s^2_4 = s^1_3 \oplus s^1_5$ (XI) $\quad s^2_5 = s^1_1 \oplus s^1_4$ (XII)

Substituindo* nas equações VII a XII cada um dos bites s^1_i $(i \in Z_6)$ de S^1 pelos respectivos conjuntos de bites de E^1 (lados direitos das equações I a VI) que *afetam* s^1_i, temos que os bites da saída S^2, após a segunda iteração da transformação *L,* são relacionados aos

bites da entrada pelos conjuntos de bites apresentados nas expressões XIII a XVIII, lembrando que $E=E^1$:

Bites de E que afetam $s^2_0 : \{e^1_0 , e^1_4 , e^1_1 , e^1_3 \} = \{e_0 , e_1 , e_3 , e_4\}$ *(XIII)*

Bites de E que afetam $s^2_1 : \{e^1_1 , e^1_2, e^1_3, e^1_5\} = \{e_1 , e_2, e_3 , e_5\}$ *(XIV)*

Bites de E que afetam $s^2_2 : \{e^1_0 , e^1_4 , e^1_0 , e^1_2 , e^1_4\} = \{e_0 , e_2, e_4\}$ *(XV)*

Bites de E que afetam $s^2_3 : \{e^1_1 , e^1_2, e^1_1 , e^1_3 , e^1_3 , e^1_5\} = \{ e_1 , e_2, e_3 , e_5\}$ *(XVI)*

Bites de E que afetam $s^2_4 : \{e^1_0 , e^1_2 , e^1_4 , e^1_1, e^1_4\} = \{e_0 , e_1, e_2 , e_4\}$ *(XVII)*

Bites de E que afetam $s^2_5 : \{e^1_0 , e^1_4, e^1_3, e^1_5\} = \{e_0 , e_3 , e_4 , e_5\}$ *(XVIII)*

Os conjuntos apresentados nas expressões XIII a XVIII definem os elementos não nulos na ***segunda MDR*** ou ***segunda matriz de avalanche [D₂]***, apresentada a seguir(os conjuntos σ_i que aparecem à direita de cada linha só serão importantes para o tópico 6.4 e devem ser desconsiderados, por enquanto).

$$[D_2] = \left|\begin{array}{cccccc} 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 \end{array}\right| \quad \begin{array}{l} \sigma_0 = \{1,2\} \\ \sigma_1 = \{0,4\} \\ \sigma_2 = \{1,3\} \\ \sigma_3 = \{0,2,4\} \\ \sigma_4 = \{3,5\} \\ \sigma_5 = \{1,4\} \end{array}$$

* É importante ressaltar que, nas MDR, as equações VII a XVIII e XXV a XXX indicam que os bites do primeiro membro *dependem* dos bites do segundo membro. Este aspecto é muito relevante, pois quando algum bite aparece "somado" duas[101] vezes (como o bite e^1_0 , na expressão XV, por exemplo) a segunda ocorrência não anula a primeira. Esta anulação ocorreria em uma soma módulo dois. Isto se dá porque parte-se da premissa que, quando esta transformação linear estiver operando em um algoritmo criptográfico, entre uma e outra execução da transformação linear haverá uma transformação não-linear que pode modificar o valor do bite considerado. A relação de dependência, entretanto, continua válida (HNC).

Agora considere-se que a transformação *L* é executada uma terceira vez (três iterações, $r=3$). A entrada desta operação é a saída da segunda execução, ou seja, $E^3 = S^2 = L(E^2)$. A partir das equações que definem *L*, temos que S^3 é dada pelas equações XIX a XXIV

$s^3_0 = s^2_1 \oplus s^2_2$ (XIX) $s^3_1 = s^2_0 \oplus s^2_4$ (XX) $s^3_2 = s^2_1 \oplus s^2_3$ (XXI)

$s^3_3 = s^2_0 \oplus s^2_2 \oplus s^2_4$ (XXII) $s^3_4 = s^2_3 \oplus s^2_5$ (XXIII) $s^3_5 = s^2_1 \oplus s^2_4$ (XXIV)

Procede-se de maneira semelhante ao que já foi feito. Substituindo nas equações XIX a XXIV cada bite de S^2 pelos respectivos conjuntos encontrados nas expressões XIII a XVIII, podemos estabelecer recursivavente as dependências da saída S^3 da terceira iteração de L em relação aos bites da entrada E, conforme pode ser visto nas expressões XXV a XXX, e montar a ***terceira matriz de avalanche [D_3].***

Bites de E que afetam $s^3_0 : \{e_1, e_2, e_3, e_5, e_0, e_2, e_4\} = \{e_0, e_1, e_2, e_3, e_4, e_5\}$ *(XXV)*

Bites de E que afetam $s^3_1 : \{e_0, e_1, e_3, e_4, e_0, e_1, e_2, e_4\} = \{e_0, e_1, e_2, e_3, e_4\}$ *(XXVI)*

Bites de E que afetam $s^3_2 : \{e_1, e_2, e_3, e_5, e_1, e_2, e_3, e_5\} = \{e_1, e_2, e_3, e_4, e_5\}$ *(XXVII)*

Bites de E que afetam $s^3_3 : \{e_0, e_1, e_3, e_4, e_0, e_2, e_4, e_0, e_1, e_2, e_4\} = \{e_0, e_1, e_2, e_3, e_4\}$ *(XXVIII)*

Bites de E que afetam $s^3_4 : \{e_1, e_2, e_3, e_5, e_0, e_3, e_4, e_5\} = \{e_0, e_1, e_2, e_3, e_4, e_5\}$ *(XXIX)*

Bites de E que afetam $s^3_5 : \{e_1, e_2, e_3, e_5, e_0, e_1, e_2, e_4\} = \{e_0, e_1, e_2, e_3, e_4, e_5\}$ *(XXX)*

$$[D_3] = \left|\begin{array}{cccccc} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right| \quad \begin{array}{l} \sigma_0 = \{1,2\} \\ \sigma_1 = \{0,4\} \\ \sigma_2 = \{1,3\} \\ \sigma_3 = \{0,2,4\} \\ \sigma_4 = \{3,5\} \\ \sigma_5 = \{1,4\} \end{array}$$

Esse procedimento prossegue até que cada bite da saída S^r seja *afetado* por todos os bites da entrada E para algum valor de r. A partir deste limite, a transformação linear é *completa*.

Como foi visto, as MDR denotadas por $[D_r]$ $(r>1)$, da transformação linear L são construídas com base na seguinte idéia recursiva: se o bite e^r_j afeta o bite s^r_i, todos os bites que afetaram s^{r-1}_j afetam s^r_i $(i,j \in \{0,1,...n-1\}, r \in \{2,3,...m\}$, onde m é o menor r para o qual função L^r é *completa*). Note-se que $s^{r-1}_i = e^r_i$. Portanto, na construção da matriz $[D_r]$ o elemento $D_{r,i,j}$ será igual a *1* sempre que $D_{1,i,w}$ for igual a *1* e $D_{r-1,w,j}$ for igual a *1* $(w \in \{0,1,2...,n-1\})$.

A partir das matrizes $[D_1]$ e $[D_3]$ constrói-se $[D_4]$ para a transformação L. Observe-se que, para $r = 4$ a função é ***completa*** e, por construção, para $r>4$, $[D_r] = [D_4]$.

---

[101] Ou qualquer outro número par.

$$[D_4] = \left|\begin{array}{cccccc} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right| \begin{array}{c} \sigma_0 = \{1,2\} \\ \sigma_1 = \{0,4\} \\ \sigma_2 = \{1,3\} \\ \sigma_3 = \{0,2,4\} \\ \sigma_4 = \{3,5\} \\ \sigma_5 = \{1,4\} \end{array}$$

4.2.4 MATRIZES DE DEPENDÊNCIAS ESTATÍSTICAS (MDE)

As MDR apresentam a vantagem de possibilitar a avaliação das transformações lineares das cifras de blocos iterativas de maneira precisa e sem a necessidade de implementar a função ou executar qualquer cifragem. Nenhum elemento aleatório é empregado. Entretanto, mais importante do que avaliar o *poder de avalanche* da transformação linear isolada, é avaliar o poder de avalanche *da função de passo* ou avaliar o poder de avalanche *da cifra*[102] com um número $r>0$ de passos. Em ambos os casos, é avaliada a combinação do *poder de avalanche da transformação linear* com *o poder de avalanche da transformação não-linear*[103].

Existem três diferenças fundamentais e importantes entre as MDE e as MDR. São elas:

i) As MDE são úteis para a avaliação do efeito avalanche em qualquer função criptográfica, seja ela uma TL[104] ou uma função que inclua uma TNL.

ii) Uma vez que a transformação criptográfica para a qual pretende montar a MDE pode não ser definida por um conjunto de equações lineares, a matriz de dependências é montada estatisticamente (em geral, não há 100% certeza em relação às dependências nulas).

iii) A função em avaliação é suficiente para definir[105] unicamente a MDE *D'*, mas a matriz de dependências (em geral) não é suficiente para definir a função que está sendo analisada.

---

[102] É comum a cifra ser da forma $C = g(f^r(M))$, onde f é a função de passo, r é o número de iterações e g é uma transformação final, realizada após a última iteração da função de passo. Isso ocorre no Rijndael, por exemplo.

[103] A principal finalidade da TNL é extinguir toda correlação existente entre a saída e a entrada da função criptográfica (produzir *confusão*). A principal função da TL é produzir o *efeito avalanche* (*difusão*) e a *completude* da função criptográfica. Entretanto, cada uma das duas transformações deve ser projetada de modo a maximizar o efeito da outra.

[104] Se for uma TL deve-se considerar que pode haver cancelamento das dependências entre as iterações.

Em razão dessas diferenças, será utilizada a notação *[D']* para representar uma MDE.

4.2.4.1 CONSTRUÇÃO DAS MDE PARA UMA FUNÇÃO CRIPTOGRÁFICA

A construção das MDE é baseada simplesmente no conceito de *dependência* apresentado no capítulo 2. A seguir é apresentado um algoritmo para construção da matriz de dependências para uma transformação $h : Z_2^n \times Z_2^n \rightarrow Z_2^n$. O algoritmo gera a matriz de dependências dos bites de *h(M, K)* em relação[106] aos bites de *M*.

Seja $h : Z_2^n \times Z_2^n \rightarrow Z_2^n$ uma transformação criptográfica e seja *w* um inteiro maior que *100*.

i) Construa uma matriz $[D']_{nxn}$ em que todo elemento $D'_{i,j}$ de *[D']* é inicializado com o valor *0*;

ii) Para cada par $(i,j) \in Z_n \times Z_n$ {

para $a = 1$ até **$(a = w)$** faça {

Teste de dependência {

gere aleatoriamente um bloco $M = m_0 | m_1 | m_2 | m_3 | ... | m_{n-1}$;

gere aleatoriamente uma chave $K = k_0 | k_1 | k_2 | k_3 | ... | k_{n-1}$

gere o bloco[107] $T = t_0|t_1|t_2|t_3|...|t_{n-1} = h(M, K) \oplus h(M \oplus U_j, K)$;

se $t_i = 1$ então incremente $D'_{i,j}$;

faça $a = a+1$;

}

}

Os elementos $D'_{i,j}$ diferentes de *0* indicam que o bite *i* da saída *depende* do bite *j* de *M*. Entretanto, os elementos $D'_{i,j}$ que permanecerem iguais a zero indicam que *provavelmente*[108] o bite *i* da saída *independe* do bite *j* de *M*.

---

[105] A definição é estocástica, mas podem ser atingidos níveis de confiança suficientemente próximos de 100%.

[106] Se a expressão $T = h(M, K) \oplus h(M \oplus U_j, K)$ for substituída por $T = h(M, K) \oplus h(M, K \oplus U_j)$ as dependências serão de *h(M,K)* em relação a *K*. Por simplicidade didática, foram adotados chave e bloco do mesmo tamanho.

[107] U_j é o *vetor complemento unitário-j de tamanho n (*definido no apêndice A, item A.28*)*.

[108] A experiência tem mostrado que quando o *i-ésimo* bite da saída é afetado pelo *j-ésimo* bite da entrada, a complementação do *j-ésimo* bite da entrada muda o valor do *i-ésimo* da saída com probabilidade próxima de 0,5 (usualmente maior que 0,2). Portanto, se for realizado o teste de dependência no laço interno do algoritmo acima

Para utilizar as MDE na avaliação da *avalanche* e da *completude* de uma função criptográfica *f*, procede-se da seguinte maneira: Seja f_r a função com *r* passos e *p* o número máximo de iterações que se pretende analisar[109].

para *r = 1* até **(r = p)** faça {

gere *[D'], utilizando o algoritmo apresentado;*

Faça [D'$_r$] = [D'], onde [D'$_r$] é a r-ésima MDE da função h;

Armazene [D'$_r$];

faça *r = r+1*;

}

4.2.5 ANÁLISE DAS MDR

É interessante que o efeito avalanche seja balanceado. Isso se verifica quando o numero de elementos não nulos em cada linha da matriz de dependências é aproximadamente o mesmo[110]. Ou seja, as diferentes linhas da matriz de dependências têm aproximadamente o mesmo peso *Hamming*. Observe que o peso *Hamming* de uma dada linha da matriz de avalanche reflete o poder de avalanche daquela linha.

O crescimento do peso *Hamming* de cada linha à proporção que as iterações ocorrem dá uma idéia da taxa de crescimento da avalanche em direção à *completude* da função. Quanto maior este crescimento, menor o número de iterações necessário para a *completude* da função. Portanto, é útil definir o seguinte:

Seja h_i o número de elementos não nulos da linha *i* da matriz de dependências $[D_1]$. Observando que h_i é igual ao número de bites da entrada que afetam o bite s_i da saída, chegamos às seguintes definições:

---

com w>500 vetores escolhidos aleatoriamente, a probabilidade esperada de $D_{i,j}$ permanecer igual a zero indevidamente é muito pequena.

[109] Note que a expressão *função criptográfica com r passos* pode ter significado diferente de *função criptográfica executada r vezes* (f_r pode ser diferente de f^r), pois *f* pode ser da forma $f(X)=g(h^z(X))$, onde *h* é a função de passo e *z* o numero de passos. Nesse caso (que ocorre no AES) $f_r(X)=g(h^r(X))$ e $f^r(X)=g(h^z(g(h^z(g(h^z(g(h^z(...X))..)$.

[110] Idealmente, é desejável que todas as equações booleanas que definem a transformação linear tenham a mesma quantidade de bites no lado direito e estes bites sejam oriundos de baites distintos.

Definição: O ***poder de avalanche h_i*** de uma expressão *booleana*[111] para um bite s_i de saída de uma transformação linear $L : Z_2^m \rightarrow Z_2^n$ é igual ao número de bites distintos que, por força desta dada expressão, *afetam* s_i. Ou seja, h_i é o peso *Hamming* da linha i $(i \in Z_n)$ da matriz que representa a referida transformação L.

Definição: O ***limite inferior do poder de avalanche LIPA (ou hmin)*** de uma transformação linear $L : Z_2^m \rightarrow Z_2^n$ representada por uma matriz $[L]_{m \times n}$ é o poder de avalanche da linha de $[L]$ com menor valor de $\boldsymbol{h_i}$ $(0 \leq i < n)$. ***Limite Inferior do Poder de Avalanche de uma transformação (f) com r iterações - $LIPA_r(f)$*** *é o peso Hamming da linha de menor peso Hamming na r-ésima MDR de f* $(f : Z_2^m \rightarrow Z_2^n)$.

Definição: O ***limite superior do poder de avalanche LISA (ou hmáx)*** de uma transformação linear $L : Z_2^m \rightarrow Z_2^n$ representada por uma matriz $[L]_{m \times n}$ é o poder de avalanche da linha de $[L]$ com maior valor de $\boldsymbol{h_i}$ $(0 \leq i < n)$. ***Limite Superior do Poder de Avalanche de uma transformação (f) com r iterações - $LIPA_r(f)$*** *é o peso Hamming da linha de maior peso Hamming na r-ésima MDR de f* $(f : Z_2^m \rightarrow Z_2^n)$.

Assim, para o algoritmo exemplo *Beta*, temos que $hmin = 2$ e $hmáx = 3$.

Foi observado que, pelo menos para valores de *hmin* e *hmax* menores[112] que 17, quanto maiores os valores de *hmin* e *hmax*, maior tende a ser *o poder de avalanche* da transformação e existe uma tendência de que a completude da função seja atingida com um número menor de iterações.

Entre duas transformações lineares que são *completas* para um mesmo número de passos, aquela que apresenta menor LSPA na matriz de dependências $[D_1]$ tende a ser mais eficiente, pois provavelmente realiza um número menor de operações em cada iteração.

Entre duas transformações lineares que são *completas* para um mesmo número de passos, aquela que apresenta as primeiras MDR com maiores pesos *Hammig* deve produzir relações mais complexas entre os textos em claro (e chave) e os respectivos textos cifrados.

---

[111] Ou da linha que a representa na forma matricial da referida transformação.

[112] Não foi avaliado o efeito de TL em que o número de coeficientes não nulos nas equações que a definem é maior que 16, pois tais casos não pareceram atraentes para aplicação prática uma vez que o aumento de difusão não é significativo para blocos de até 128 bites. Deve-se notar que se, em um caso limite, todas as equações tivessem todos os coeficientes iguais a um, todos os bites da saída teriam o mesmo valor e só existiriam duas saídas possíveis: todos os bits iguais a 0 ou todos os bits iguais a um.

Outra informação muito importante que pode ser obtida nas MDR é o número de baites[113] que é afetado por cada bite para uma ou mais iterações da TL. Esse número é usado para definir limites mínimos de resistência contra a CD e a CL (o assunto é tratado no item 6.5 e no capítulo 8).

4.2.6 ANÁLISE DAS MDE

A análise das MDE é feita de maneira análoga à análise das MDR. Os conceitos de *hmin* e *hmax* permanecem válidos e úteis para acompanhar o efeito avalanche e verificar a completude da função criptográfica.

Se forem definidas matrizes $[\Delta_r] = [D'_r]-[D_r]$ (r = 1,2,3...p, onde p é o menor número de passos em que a TL é completa) para um algoritmo com estrutura S-TL, valem as seguintes propriedades:

i) *Cada elemento de* $[\Delta_r]$ $\Delta r_{i,j} \geq 0 \; \forall \; (i, j) \in Z_n \; x \; Z_n$. Vide nota.[114]

ii) *Os pesos Hamming das linhas da matriz* $[\Delta_r]$ *são uma medida da contribuição da transformação não-linear para o poder de avalanche do algoritmo.*

Entre duas transformações criptográficas que são *completas* para um mesmo número de passos, aquela que apresenta as primeiras MDE com maiores pesos *Hammig* deve produzir relações mais complexas entre a *entrada (texto em claro)* e a *saída(texto cifrado)*.

As MDE também são úteis para analisar o CAE. Se a função criptográfica atende ao critério de avalanche estrito, cada bite da saída deve ser complementado com probabilidade 0,5 caso seja complementado qualquer bite na entrada. Assim, se forem feitos w testes de dependência, à medida que w cresce, o número de vezes que cada bite da saída é complementado tende para $w/2$.

4.3 AVALIAÇÃO DA RESISTÊNCIA CONTRA CRIPTOANÁLISE

A criptoanálise diferencial (CD) e a criptoanálise linear (CL) são consideradas as principais formas de criptoanálise a que estão sujeitas as cifras de bloco atualmente

[113] Supondo que a TNL opere por baites.
[114] Caso contrário, *D'i,j* deveria ser igual a *1* mas está preenchido com *0* por imprecisão do teste de dependência na construção da MDE.

(OSWALD, 2002). As duas formas de criptoanálise encontram-se descritas, por exemplo, em (CASTAÑO, 1998) e (DAEMEN, 1995).

Na CD uma grande quantidade de pares *texto em claro-criptograma* é utilizada para determinar bites da chave. Informações estatísticas da chave são obtidas de textos cifrados gorados cifrando pares de texto em claro com uma diferença específica bite a bite *E'* sob a chave alvo. O fator trabalho do ataque depende criticamente da maior probabilidade *P (S'|E')* onde *S'* é a diferença entre dois blocos em algum estágio intermediário pré-determinado da transformação, como na entrada da última iteração, por exemplo. A informação sobre a chave é extraída da seguinte maneira: Para cada par, é suposto um valor de diferença intermediária *S'*. Estes valores supostos impõem restrições sobre os valores de uma quantidade *v* de bites da sub-chave do último passo. Diz-se que um par sugere valores de bites da chave que são compatíveis com estas restrições. Enquanto para alguns pares muitas chaves são sugeridas, nenhuma chave é sugerida para outros pares, o que implica que os valores de saída são incompatíveis com *S'*. Para cada valor de sub-chave sugerido uma entrada correspondente em uma tabela de freqüências é incrementada. O ataque tem sucesso se o valor certo da sub-chave é sugerido com uma freqüência significativamente maior do que qualquer outro valor. Pares com uma diferença intermediária diferente de *S'* são chamados *pares errados*. Sub-chaves sugeridas por estes pares são, em geral, erradas. Pares certos, com uma diferença intermediária igual a *S'*, sugerem o valor certo da chave e também valores errados. Para o DES, os valores errados podem ser considerados uniformemente distribuídos sobre todos os valores de chave possíveis se o valor da probabilidade *P(S'|E')* é significativamente maior do que *P(C'|E')*, para todo *C'* diferente de *S'*.

Sob estas condições, faz sentido calcular a razão entre o número de vezes que o valor correto é sugerido e o número médio de sugestões por entrada. Esta razão é chamada *relação sinal/ruído(RS/R)*. A *RS/R* afeta fortemente o número de pares corretos necessários para identificar unicamente a chave e consequentemente o número[115] de cifragens que têm que ser realizadas (DAEMEN, 1995, cap 5). Se o número de textos necessários é maior do que o espaço de chaves, o ataque por CD é inútil[116] e se a quantidade de textos é maior que o espaço de textos em claro, o ataque é impossível (BIHAM, 1993, p. 31).

---

[115] Fator trabalho.
[116] Pois o fator trabalho é maior que o da busca exaustiva.

4.3.1 RESISTÊNCIA CONTRA CRIPTOANÁLISE DIFERENCIAL

Os testes de imunidade contra CD e CL não são possíveis em blocos de tamanho prático, como por exemplo 128 bites (DICHTL, 1999). Entretanto, é possível analisar as tabelas de substituição quanto à resistência a estas formas de criptoanálise de maneira objetiva e estender os resultados ao cifrador utilizando as técnicas apresentadas em (CASTAÑO, 1998, Cap 5).

Os desvios de probabilidade que são exploráveis criptoanaliticamente são calculados em função das distribuições de diferenças nas tabelas de substituição e de como estas distribuições se propagam pelos diversos passos da cifragem. Portanto, é preciso especificar adequadamente a TNL, para que estes desvios de porbabilidade sejam os menores possíveis, e a TL, a fim de difundir o efeito da TNL por todo o bloco, dificultando a CD. Como pode ser visto em (CASTAÑO, 1998)(DAEMEN, 1995, cap. 5) e (DAEMEN, 2003), para construir a prova de resistência contra CD os autores do Rijndael basearam-se, em última análise, nas características de distribuição diferencial da tabela de substituição e na propriedade de difusão (propagação das diferenças) da TL.

A *RS/R* será utilizada como parâmetro para definir a resistência da TNL contra a CD e será considerada referência a máxima[117] *RS/R* da TNL do AES (vide capítulo 4, item 4.5.7). Será considerada aceitável,[118] portanto, uma tabela de substituição com ***RS/R menor ou igual a 4***, desde que sejam atendidas as condições de difusão por parte da TL.

É importante observar que a difusão é analisada no nível dos baites porque este é o elemento que é operado em cada operação de substituição, para a qual foram calculadas as *RS/R*.

A propagação das diferenças segue o mesmo padrão de propagação das dependências[119], uma vez que, usualmente, as transformações responsáveis pela difusão (avalanche) em uma cifra são lineares (CASTAÑO, 1998, cap. 5). Pode-se então, analisando as colunas das MDR, observar qual a menor quantidade de baites da saída de uma (ou mais) fase(s) que são afetados por cada bite da entrada.

---

[117] Esta é uma consideração pessimista que, a favor da segurança, supõe que todas as RS/R são tão grandes quanto a maior (pior) delas.

[118] Para tanto é necessário que a TL e o número de iterações sejam especificados de forma que todo o conjunto seja tão resistente quanto o AES.

[119] Isso explica porque as duas tabelas de pesos Hamming (uma relacionando entradas com saidas e outra relacionando diferencas na entrada com diferencas na saída) apresentadas em (CASTAÑO, 1998) são exatamente iguais, não havendo necessidade de gerar as duas. Essa foi uma questão que tinha ficado pendente naquele trabalho.

A complexidade da criptoanálise cresce com o número de vezes que as substituições têm que ser computadas nas equações. Analisando a matriz de dependências *[D_1]*, que corresponde a uma iteração, pode-se calcular quantas vezes a operação de substituição terá de ser computada em uma equação de uma fase. Analogamente, analisando também a matriz de dependências *[D_2]*, que corresponde a duas iterações, pode-se calcular quantas vezes a operação de substituição terá de ser computada em uma equação de duas fases. O número de vezes que a operação de substituição é computada é igual ao número de baites[120] afetados quando é inserida uma diferença na entrada. É comum referir-se a esta quantidade de baites como o *número de tabelas de substituição ativas*.

Em relação à TL do AES, o argumento dos autores do Rijndael é que a diferença de um baite no bloco de entrada de uma fase afetará quatro baites na saída da respectiva fase. Isso se deve à difusão da operação *MixColumns* (multiplicação pela matriz *M*, capítulo 2) que mistura os quatro baites de uma mesma palavra. Isso pode ser visto na matriz de dependências [D_1] (apêndice 3) e na MDE (1 iteração) do Rijndael (apêndice 4). As operações *ShiftRows* (PERMUTAÇÃO, item 5.2.7) e *ByteSub* (SUBSTITUIÇÃO, item 5.2.8) não produzem difusão entre os baites, uma vez que operam baite a baite. A operação de permutação, por sua vez, mistura quatro *palavras*. A combinação das duas operações faz com que a alteração de um baite em uma fase *r* provoque a alteração de quatro baites na fase *r+1* e que estes quatro baites entrem em quatro palavras distintas na fase *r+2*(isso pode ser visto na MDR [D_2], no apêndice 4). Um conseqüência disso é a completude da função criptográfica com apenas 2 iterações.

Este poder de difusão é o limite sugerido neste trabalho para uma TL ser considerada satisfatória.

Se for adotada uma combinação de TL e TNL dentro dos padrões de aceitação aqui sugeridos e adotado um número[121] de iterações compatível, a cifra de blocos resultante deverá ser, no mínimo, tão resistente contra a CD quanto o AES.

[120] Supondo que a substituição ocorre por baites.
[121] Considerando o exposto sobre a difusão numa estrutura Feistel no capítulo 9, convém acrescentar pelo menos 2 iterações às 10 que são previstas no AES (128 bites).

4.3.2 RESISTÊNCIA CONTRA CRIPTOANÁLISE LINEAR

O raciocínio utilizado para demonstrar a resistência contra a CL é análogo ao que foi utilizado em relação à CD. A condição sugerida para aceitação da TL é que esta seja tão resistente contra a CL quanto a TL do AES.

Em relação à TNL, o que importa são os valores $|p_i\text{-}1/2|$, onde p_i é a probabilidade[122] de a equação linear e_i que relaciona bites da entrada da *substituição* com bites da saída *valer* $(i=1,2...2^{n+m}$*, n e m são os tamanhos da entrada e da saída na TNL)*. Para o AES o valor máximo de |p-1/2| é 16/128=1/8.

4.4 CONCLUSÃO DO CAPÍTULO

A resistência contra CL e a CD é dependente basicamente dos seguintes fatores:

i) das características de distribuição diferencial da TNL;
ii) das características de distribuição linear da TNL;
iii) do poder de difusão da transformação linear (associada à não-linear);
iv) do número de iterações.

O primeiro fator contribui principalmente para diminuir as correlações que favorecem a obtenção de equações que relacionem diferenças na entrada com diferenças na saída da transformação não-linear, que normalmente atua em uma seqüência de bites muito menor do que um bloco. Este fator é medido pela RS/R. É desejável que a RS/R tenha valores próximos da unidade.

O segundo fator reflete a probabilidade de uma equação linear envolvendo bites da entrada e da saída da transformação não-linear ser válida. Este fator é medido pelo valor do |p - 1/2|, onde *p* é a probabilidade de a equação mais favorável valer. É desejável que este valor seja o mais próximo possível de 0.

A incerteza em uma equação de várias fases depende do número de vezes que a incerteza da transformação não-linear tiver que ser computada nesta equação. O terceiro fator está relacionado com a forma como a transformação linear propaga a incerteza na saída de uma TNL para todo o bloco, fazendo com que se estabeleçam cada vez mais relações de dependência. Esse fator é quantificado pelo poder de avalanche e pelo número de passos que a

[122] Vide capítulo 4, AES (item 4.5.7)

função necessita para ser completa. É desejável que esse valor seja o menor possível. Para avaliar o efeito avalanche podem ser utilizadas as matrizes de dependências.

O número de iterações determina o número de fases que terão de ser computadas nas equações para a tentativa de criptoanálise. Na prática, o fator trabalho esperado para a criptoanálise tende a crescer exponencialmente com o número de iterações (OSWALD, 2002).

Se os projetos da TL e da TNL forem bem coordenados é possível atingir a *completude* da função criptográfica (128 bites) em apenas duas iterações, mesmo que sejam utilizadas tabelas de substituição relativamente pequenas (8 bites) e TL com equações que têm no máximo 8 coeficientes não nulos (a exemplo do que foi observado no AES).

Da observação das MDR, pode-se concluir que a TL do AES faz com que cada baite[123] em uma iteração *r* afete exatamente os 4 baites da respectiva *palavra* na iteração *r+1* e que cada um destes 4 baites afetem, cada um, uma das 4 *palavras* do bloco na iteração *r+2*(16 baites, portanto). Isso garante um número mínimo de vezes que a TNL operará e que, portanto, contribuirá para a incerteza nas equações utilizadas para a criptoanálise. Este foi um dos dois principais argumentos utilizados pelos projetistas do AES para demonstrar a resistência contra a CL e a CD. Os autores do Rijndael construiram uma TL composta de duas tansformações (Shift Rows e Mix Columns) separadas a fim de tornar evidente o excelente efeito de difusão produzido e isso foi aparentemente um fator importante para que o Rijndael fosse escolhido para o AES. Utilizando as MDR, o poder de difusão da TL do Rijndael poderia ser verificado independentemente de como a TL tivesse sido construída.

O outro argumento importante dos autores do Rijndael foi o RS/R e as características de distribuição linear da tabela de substituição (CASTAÑO, 1998, cap. 5)(DAEMEN, 1995, cap. 5).

Pelo acima exposto, se for construída uma cifra de 128 bites com uma tabela de substituição que apresente características diferenciais e lineares tão boas quanto as do AES e uma TL com o mesmo poder de difusão (ou superior), espera-se que a cifra tenha no mínimo, as mesmas características, de resistência contra a CL e a CD que o Rijndael (AES). Uma vez que a resistência do AES à criptoanálise é satisfatória, os valores de |p-1/2| e RS/R deste padrão são boas referências para a aceitação de uma TNL.

As MDR podem ser utilizadas para analisar uma TL e concluir sobre o seu poder de difusão, possibilitando quantificar a propagação das dependências e a avalanche. As MDE

[123] ou bite.

podem ser usadas para avaliar a avalanche produzida pela ação iterada da TL com a TNL de uma cifra, concluir sobre a completude da função criptográfica e verificar o CAE[124].

[124] Obs.: Uma função que atende o CAE é necessariamente completa.

5 CONSIDERAÇÕES SOBRE AS ESTRUTURAS DOS ALGORITMOS DES, RIJNDAEL E SERPENT

5.1 INTRODUÇÃO

Neste capítulo são analisados aspectos considerados importantes no projeto e nas estruturas dos três algoritmos escolhidos para serem a base da definição da filosofia de projeto sugerida no capítulo 6.

Inicialmente, no item 4.2, é apresentada uma descrição alternativa para o AES.

As descrições originais dos algoritmos analisados estão disponíveis na Internet e em (FIPS 46, 1977), (FIPS 46-3, 1999), (FIPS 197, 2001), (KNUDSEN, 2000), (MENEZES, 1996, Cap. 7). Suas estruturas estão baseadas em substituições e transformações lineares[125] (SLTN). O conceito de SLTN apresenta-se bem estudado na bibliografia (KELSEY, 1996) (TAVARES, 1996) (PATTERSON, 1999) (DAVIDA, 1979) (CASTANO, 1998) (XAVIER JR, 1999) (SCHNEIER, 1996).

5.2 DESCRIÇÃO ALTERNATIVA DO RIJNDAEL

O Rijndael cifra blocos de 128, 192 ou 256 bites de texto em claro com uma chave que também pode ter qualquer dos três tamanhos. O número de iterações pode ser 10, 12 ou 14, de acordo com os tamanhos da chave e do bloco adotados. Aqui é tratado, sem perda de generalidade, o caso em que são adotados bloco e chave de 128 bites.

As duas principais transformações criptográficas realizadas no Rijndael têm sido descritas como sendo operações com polinômios em corpos finitos. Este tópico apresenta uma descrição baseada apenas em operações lógicas e substituições com tabelas, a exemplo do que ocorre no DES. Com esta descrição, espera-se que o entendimento do AES seja bastante simplificado para quem está familiarizado com o DES. Além disso, conforme será visto no capítulo 6, a forma matricial da transformação linear (uma permutação seguida de

[125] É mais utilizada a expressão SPN. No caso do Serpent e Rijndael, por exemplo, esta expressão está inadequada, pois tais algoritmos empregam transformações lineares que não são rigorosamente *permutações*.

uma multiplicação) permite a construção das matrizes de dependências para avaliação do poder de avalanche[126] desta transformação.

Esta descrição alternativa pode não ser eficiente para implementação em *software*, mas é de grande utilidade para análise do funcionamento da cifra e comparação do poder de difusão da sua transformação linear com o poder de difusão das transformações lineares presentes em outras cifras.

5.2.1 NOTAÇÃO

As transformações realizadas pelo AES estão referidas com nomenclatura em português. As transformações são descritas a partir do tópico 4.2.2. A TAB. 4.1 relaciona a nomenclatura proposta aos termos do (FIPS 197, 2001):

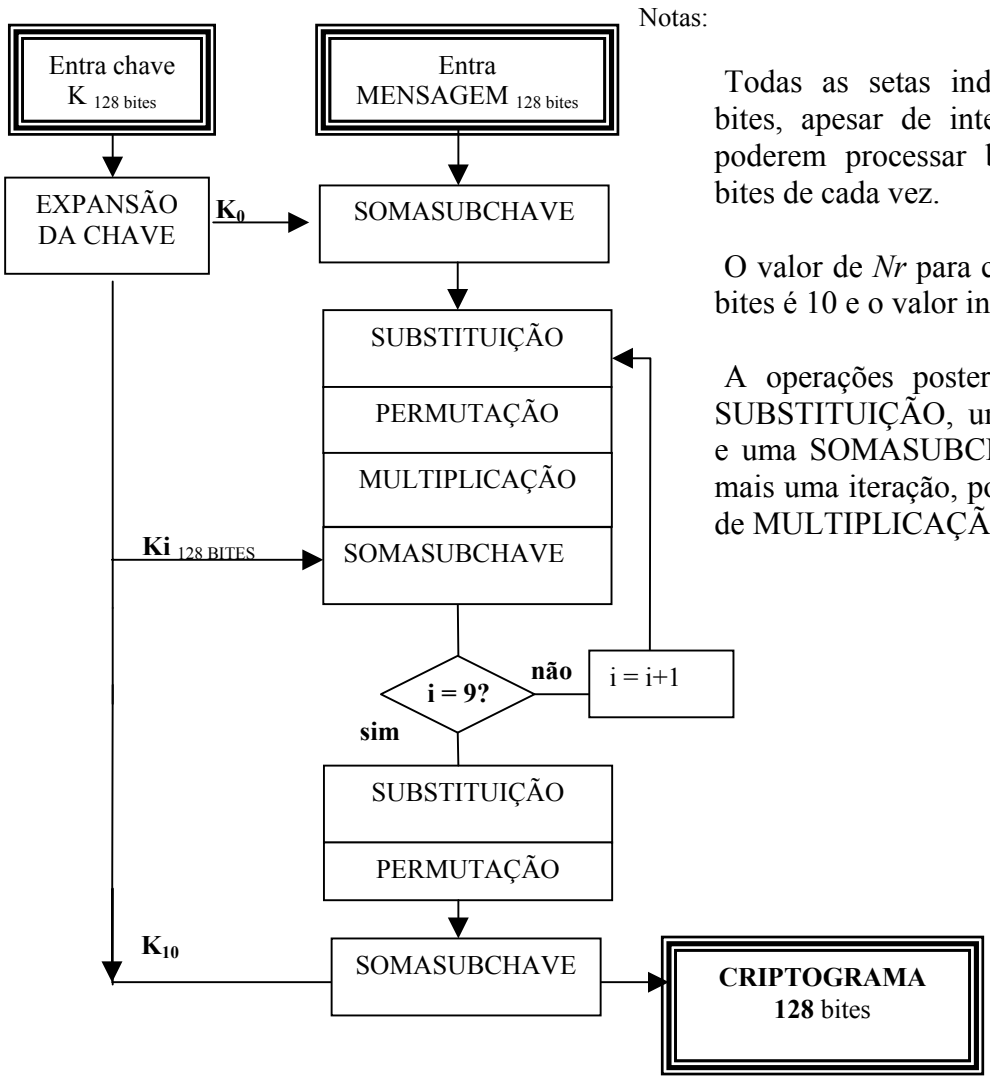
TAB. 5.1 – Terminologia alternativa para as transformações do AES

5.2.2 Terminologia sugerida	5.2.3 Designação no FIPS 197
SOMASUBCHAVE	***AddRoundKey***
SUBSTITUIÇÃO	***SubBytes***
PERMUTAÇÃO	***ShiftRows***
MULTIPLICAÇÃO	***MixColumns***
EXPANSÃO	***Key Expansion***
ROTAÇÃO	***RotWord***

5.2.4 ESTRUTURA DO AES

A transformação do Rijndael com *Nr = 10* iterações cifrando um bloco de mensagem está ilustrada no fluxograma da FIG. 4.1. Em seguida são apresentadas as descrições de cada transformação. Como auxílio à compreensão de cada transformação, poderá ser consultado o tópico 12.1.4 do apêndice 1, que apresenta todos os estágios intermediários de uma cifragem do Rijndael.

[126] Característica de difusão.



Notas:

Todas as setas indicam fluxo de 128 bites, apesar de internamente as etapas poderem processar blocos de 1, 8, ou 32 bites de cada vez.

O valor de *Nr* para chave e bloco de 128 bites é 10 e o valor inicial de ***i*** é 1.

A operações posteriores ao laço (uma SUBSTITUIÇÃO, uma PERMUTAÇÃO e uma SOMASUBCHAVE) equivalem a mais uma iteração, porém sem a operação de MULTIPLICAÇÃO.

FIG. 5.1 – Fluxograma do Rijndael (AES)

5.2.5 EXPANSÃO DA CHAVE

Inicialmente, a partir da chave *K* de 128 bites fornecida, são geradas *Nr+1* sub-chaves (uma para cada operação **SOMASUBCHAVE**), também de 128 *bites* cada. No caso de 128 bites (chave e bloco), o algoritmo realiza 10 iterações e são necessárias de 11 sub-chaves. A expansão da chave encontra-se ilustrada no detalhamento da função[127] de expansão das

[127]Cada sub-chave é função apenas da sub-chave anterior e de uma constante, de forma que, alternativamente, cada sub-chave pode ser gerada dentro da respectiva iteração, se o implementador julgar conveniente. Nesse exemplo, todas as sub-chaves foram geradas antes de o algoritmo entrar no laço. A primeira sub-chave (K_0) é a própria chave fornecida pelo usuário.

chaves no apêndice 1. Esta descrição é semelhante à encontrada na bibliografia (CASTAÑO, 1998).

5.2.6 OPERAÇÃO SOMASUBCHAVE

Trata-se de uma operação lógica OU-EXCLUSIVO (sempre realizada bite a bite) entre dois blocos de mesmo tamanho. A inversão é trivial.

5.2.7 SUBSTITUIÇÃO

É uma transformação não-linear. A operação é realizada baite a baite, em todo o bloco argumento *E=(E0,E1,E2,...E15)*. O resultado é o bloco *S=SUB(E)=(S0,S1,S2,...S15)*, onde *Si=SUB(Ei)*, $i \in \{0, 1, 2,..., 15\}$, de forma que cada baite *Si =SUB(Ei), i={0,1,2,...,15}*. O resultado da operação de substituição sobre o bloco *E* de 128 bites é portanto a concatenação dos dezesseis baites de retorno da substituição de cada um dos baites do bloco argumento. A operação de substituição utiliza a caixa de substituição apresentada na *tabela 2* (em formato hexadecimal, por economia de espaço). A caixa tem 16 linhas, numeradas de ***0*** a ***f*** e 16 colunas igualmente numeradas de ***0*** a ***f*** (hexadecimal). A operação é feita sobre cada baite do argumento *E,* da seguinte maneira: seja o baite hexadecimal $E0=x_h|y_h$. O resultado *SUB(E0)* da substituição é o baite *S0,* encontrado na intersecção da linha x_h com a coluna y_h da caixa. A operação é invertida utilizando-se a tabela de substituição inversa mostrada no tópico 11.2.2 do apêndice 1. **Exemplo**: Para o baite $E0=01010011_2 = 53_h$, temos $x = 5_h$ e $y = 3_h$. Entrando na linha *5* e coluna *3* da tabela de substituição temos $S0=ed_h = SUB\ (E0)$. A representação binária deste *baite* é 11101101_2.

5.2.8 PERMUTAÇÃO

É uma permutação dos baites do bloco argumento *E* segundo a seguinte definição:

P (E) = E0| E5| E10| E15| E4| E9| E14| E3| E8| E13| E2| E7| E12| E1| E6| E11

A operação inversa é definida no tópico 12.1.2.2.

TAB. 5.2 : Tabela para operação de substituição para cifragem do baite xy_h

Número da coluna (y) hexadecimal

Número da linha (x)

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

5.2.9 MULTIPLICAÇÃO

Esta transformação linear consiste em multiplicar cada *palavra* da entrada pela matriz $[M]_{32x32}$ de valores binários. O resultado de cada multiplicação é também uma palavra de trinta e dois bites. Assim, a operação é realizada em cada palavra do bloco de 128 bites (quatro vezes em cada bloco, sempre utilizando a mesma matriz de transformação). A operação inversa é a mesma operação de multiplicação, porém utilizando a matriz $[M]^{-1}$. As matrizes $[M]$ e $[M]^{-1}$ são constantes. $[M]$ é apresentada na FIG. 5.2 e $[M]^{-1}$ pode ser vista no apêndice 1, na FIG. 12.1.2. Assim, por exemplo, para uma palavra $S = [M] \times E$, o bite $s0$ é dado pela seguinte expressão booleana: $s0 = (0.e0)+(1.e1)+(0.e2)+....+(0.e31)= e1+ e8+e9 +e16+ e24.$ *(vide* FIG. 5.2*)*

5.3 DES – ESTRUTURA FEISTEL

O DES é uma estrutura Feistel[128] baseada e permutações e substituições (SPN). Sejam n o tamanho do bloco operado pelo cifrador, e k uma quantidade de bites obtidos a partir da chave. A função f de um sistema Feistel convencional é definida da seguinte maneira:

[128] Este tipo de estrutura foi criado por Horst Feistel para ser utilizado no Lucifer em 1973 (FEISTEL, 1973). Posteriormente, foram apresentadas estruturas Feistel desequilibradas, nas quais o bloco é dividido em duas

$$f: \{0,1\}^{n/2} \times \{0,1\}^{m} \rightarrow \{0,1\}^{n/2}$$

Sejam $Bi=Li|Ri$ e Ki dois blocos de tamanhos n e m que representam o bloco que está sendo *cifrado* e a sub-chave de passo utilizada na *i-ésima* iteração de um cifrador que utiliza função *f*, respectivamente. Note que Li e Ri são blocos de mesmo tamanho $n/2$. Um passo de uma estrutura Feistel balanceada é representado pelas expressões 5-I e 5-II, conforme ilustra a FIG. 5.3.

s0		.	1	.	.	.	.	.	.	1	1	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.		e0
s1		.	.	1	.	.	.	.	.	.	1	1	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	.	.	.	.	.	.		e1
s2		.	.	.	1	.	.	.	.	.	.	1	1	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	.	.	.	.	.		e2
s3		1	.	.	.	1	.	.	.	1	.	.	1	1	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	.	.	.	.		e3
s4		1	.	.	.	.	1	.	.	1	.	.	.	1	1	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	.	.	.		e4
s5		.	.	.	.	.	.	1	.	.	.	.	.	.	1	1	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	.	.		e5
s6		1	.	.	.	.	.	.	1	1	.	.	.	.	.	1	1	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	.		e6
s7		1	.	.	.	.	.	.	.	1	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1		e7
s8		1	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	1	1	.	.	.	.	.	.	1	.	.	.	.	.	.	.		e8
s9		.	1	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	1	1	.	.	.	.	.	.	1	.	.	.	.	.	.		e9
s10		.	.	1	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	1	1	.	.	.	.	.	.	1	.	.	.	.	.		e10
s11		.	.	.	1	.	.	.	.	1	.	.	.	1	.	.	.	1	.	.	1	1	.	.	.	.	.	.	1	.	.	.	.		e11
s12		.	.	.	.	1	.	.	.	1	.	.	.	.	1	.	.	1	.	.	.	1	1	.	.	.	.	.	.	1	.	.	.		e12
s13	=	.	.	.	.	.	1	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	1	1	.	.	.	.	.	.	1	.	.	x	e13
s14		.	.	.	.	.	.	1	.	1	.	.	.	.	.	.	1	1	.	.	.	.	.	1	1	.	.	.	.	.	.	1	.		e14
s15		.	.	.	.	.	.	.	1	1	.	.	.	.	.	.	.	1	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1		e15
s16		1	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	1	1	.	.	.	.	.	.		e16
s17		.	1	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	1	1	.	.	.	.	.		e17
s18		.	.	1	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	1	1	.	.	.	.		e18
s19		.	.	.	1	.	.	.	.	.	.	.	1	.	.	.	.	1	.	.	.	1	.	.	.	1	.	.	1	1	.	.	.		e19
s20		.	.	.	.	1	.	.	.	.	.	.	.	1	.	.	.	1	.	.	.	.	1	.	.	1	.	.	.	1	1	.	.		e20
s21		.	.	.	.	.	1	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.	1	1	.		e21
s22		.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	.	1	.	.	.	.	.	.	1	1	.	.	.	.	.	1	1		e22
s23		.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	1	.	.	.	.	.	.	.	1	.	.	.	.	.	.	1		e23
s24		1	1	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	1	.	.	.	.	.	.		e24
s25		.	1	1	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	1	.	.	.	.	.		e25
s26		.	.	1	1	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	1	.	.	.	.		e26
s27		1	.	.	1	1	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	.	.	.	.	1	.	.	.	1	.	.	.		e27
s28		1	.	.	.	1	1	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	.	.	.	1	.	.	.	.	1	.	.		e28
s29		.	.	.	.	.	1	1	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	.	1	.		e29
s30		1	.	.	.	.	.	1	1	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	.	1	.	.	.	.	.	.	1		e30
s31		1	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	.	.	.	.	.	.	.	1	1	.	.	.	.	.	.	.		e31

FIG. 5.2 – Ilustração da multiplicação pela matriz M (Obs: "." = 0)

Analogamente, os demais 31 bites de cada palavra resultam de 31 expressões booleanas obtidas da matriz [M].

---

partes de tamanhos diferentes (KELSEY, 1996). O DES é então, mais precisamente, uma estrutura Feistel *convencional* ou *equilibrada*, uma vez que o bloco a ser cifrado é dividido em duas partes de mesmo tamanho

$$\boldsymbol{Ri = Li\text{-}_1 \oplus f(R_{i-1}\,,\, Ki)} \quad (\text{5-I})$$

$$\boldsymbol{Li = R_{i-1}} \quad (\text{5-II})$$

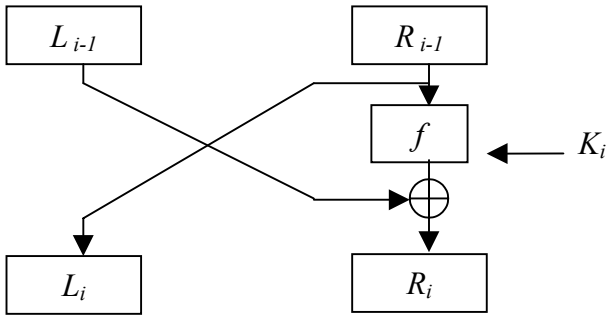


FIG. 5.3 – Estrutura Feistel para cifrar (*i = 0, 1, 2...r*)

Note-se que B_0 é o texto em claro e *Br* é o texto cifrado, onde *r* é o número de iterações.

O aspecto mais importante deste tipo de estrutura decorre do processo de decifração. Para entender a decifração, convém lembrar a seguinte propriedade da soma módulo dois (operação *ou exclusivo*): $(X \oplus Y) \oplus Y = X$, onde *X e Y* são blocos do mesmo tamanho.

Em conseqüência desta propriedade, para decifrar, é utilizada a mesma função *f*. Cada passo da decifração é representado pelas expressões 5-III e 5-IV e ilustrado na FIG. 5.4, que decorrem trivialmente das equações I e II.

$$\boldsymbol{R_{i-1} = L_i} \quad (\text{5- III})$$

$$\boldsymbol{L_{i-1} = R\,i \oplus f\,(R_{i-1}\,,\, K_i)} \quad (\text{5-IV})$$

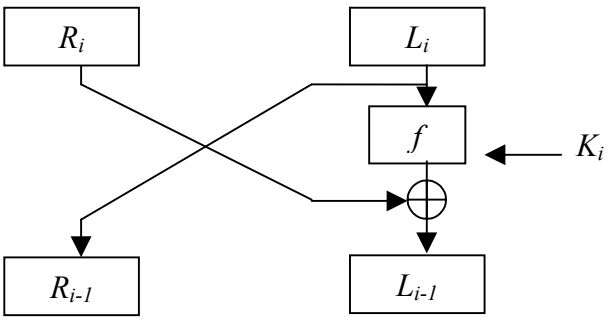


FIG. 5.4 – Estrutura Feistel para decifrar (*i = r, r-1, r-2...0*)

(32 bites).

Deve ficar claro que as estruturas nas FIG. 5.3 e FIG. 5.4 diferem apenas pela posição dos blocos *L* e *R* e pela seqüência de utilização das sub-chaves.

Em conseqüência, as estruturas Feistel convencionais apresentam as seguintes vantagens:

- a função *f* pode ser projetada tão complexa quanto se queira, e não precisa ser invertível;
- não é necessário projetar nem implementar um algoritmo especificamente para decifrar, o que economiza tempo e memória;
- a adição das duas metades do bloco, a cada iteração, contribui para aumentar a *difusão(efeito avalanche)*;

5.4 DES – A FUNÇÃO DE EXPANSÃO DA CHAVE

Estudar as funções de expansão de chaves não é o principal objetivo deste trabalho. Entretanto, no caso do DES, a natureza da função de chaves é responsável pela existência de quatro *chaves fracas*, doze *chaves semi-fracas* e quarenta e oito *chaves possivelmente fracas*. A existência destas chaves é indesejável, embora não comprometa a segurança de um algoritmo pois estas chaves são facilmente identificáveis e constituem um universo muito pequeno quando comparado ao tamanho do espaço de chaves.

Definição 5.1: Uma *chave fraca* é uma chave *K* tal que $E_K(E_K(X))=X$ para todo *X*.

Definição 5.2: Uma *chave semi-fraca* é uma chave pertencente a um par (K_1, K_2) tal que $E_{K1}(E_{K2}(X))=X$ para todo *X*. A operação de cifrar com K_1 equivale a decifrar com K_2 e *vice-versa*.

Definição 5.3: Uma *chave possivelmente fraca* é uma chave a partir da qual são geradas apenas 4 sub-chaves distintas, cada uma utilizada quatro vezes.

As 64 chaves mencionadas estão relacionadas em (CARVALHO, 2001, p. 113), (SCHNEIER, 1996, p. 280) e (MENEZES, 1996, p. 257), por exemplo.

Seja o bloco $C_0|D_0$ a chave *K* de 56 bites (64 menos os bites de paridade, após permutação inicial chamada PC1 do algoritmo de escalonamento de chaves (MENEZES, 1996, p. 256)), onde C_0 e D_0 são dois blocos de 28 bites. A função de escalonamento de chaves do DES faz a metade esquerda de cada sub-chave K_i ser um seqüência de 24 bites escolhidos de C_0 concatenado com 24 bites escolhidos de D_0. As sub-chaves de passo são $K_i=C_i|D_i$. (i = 1,2,3...16). No apêndice 5 estão apresentadas as 16 sub-chaves do DES em

função dos bites de C_0 e D_0. Além disso, no mesmo apêndice, encontram-se matrizes ilustrando as *dependências* de algumas[129] sub-chaves em relação à chave *K*.

Para tornar improvável a existência de chaves que gerem efeitos inconvenientes, é desejável que cada bite de K_i ($i=1,2..,Nr$, onde Nr = *número de iterações*) seja uma função não-linear de vários bites de *K* e que a partir de um determinado valor de *i*, K_i seja *completa* em relação *K*. Isso é conseguido usando no algoritmo de expansão de chaves uma combinação de TL e TNL, como ocorre nas funções de escalonamento de chaves do Rijndael e do Serpent.

5.5 DES – A FUNÇÃO DE PASSO *f*

A função *f* do DES é definida por

$f(X) = P(S(E(R_{i-1}) \oplus K_i))$ **,** onde

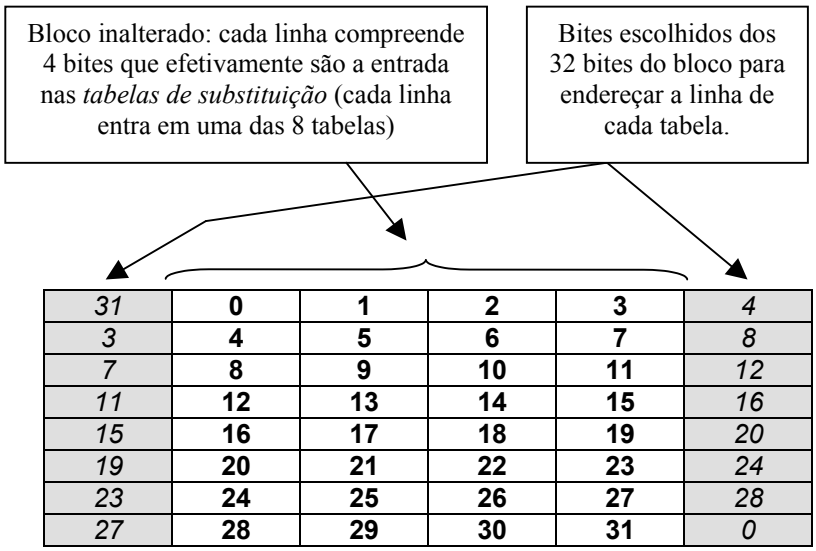
- *E(X)* representa uma operação de *expansão*, definida na estrutura do DES, cujo argumento é um vetor de 32 bites e que retorna um vetor de 48 bites;
- *S(X)* representa uma função de Substituição, definida na estrutura do DES por meio um conjunto de 8 tabelas de substituição *(S-boxes)* padronizadas, cujo argumento é um vetor de 48 bites e que retorna um vetor de 32 bites[130];
- *P(X)* representa uma função de transposição (uma Permutação), definida no corpo do DES, cujo argumento é um vetor de 32 bites e que retorna um outro vetor de 32 bites;

5.5.1 ANÁLISE DO PODER DE DIFUSÃO DA TL DO DES COM AS MATRIZES DE AVALANCHE

Analisando a função *f* do DES, pode-se concluir que transformação de *expansão* nela presente não modifica a posição dos bites na entrada das caixas de substituição. Ela apenas divide o bloco de 32 bites em 8 quartetos, cada um dos quais será substituído em uma das 8 tabelas de substituição por outro quarteto. Desta forma, o valor do *i-ésimo* quarteto define qual das 16 colunas da *i-ésima* tabela de substituição contêm o conjunto de quatro bites de substituto. A "expansão" define que a linha da *i-ésima* caixa de substituição que contém o *i-*

[129] A bem da concisão, foram colocadas apenas 3, incluindo a primeira e a última.
[130] Vide tópico 5.5.2, características das s-boxes.

ésimo quarteto substituto é definida pelo valor do par de bites formado pelo último bite *do (i-1) ésimo* quarteto e pelo primeiro bite do *(i+1)-ésimo* quarteto[131].



31	**0**	**1**	**2**	**3**	*4*
3	**4**	**5**	**6**	**7**	*8*
7	**8**	**9**	**10**	**11**	*12*
11	**12**	**13**	**14**	**15**	*16*
15	**16**	**17**	**18**	**19**	*20*
19	**20**	**21**	**22**	**23**	*24*
23	**24**	**25**	**26**	**27**	*28*
27	**28**	**29**	**30**	**31**	*0*

FIG. 5.5 *Expansão* da função de passo *f* do DES

Pelo acima exposto, a transformação de expansão não foi considerada parte da transformação linear do DES, e sim parte da operação de *substituição*. Sendo assim, a representação de um passo do DES para a construção da matriz de dependências e análise da transformação linear é como está representada na FIG. 5.6.

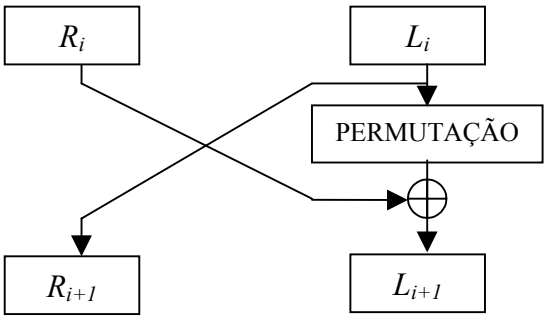


FIG. 5.6 Um passo da TL do DES para construção das matrizes de avalanche.

[131] O último quarteto utiliza o primeiro bite do primeiro quarteto e *vice-versa*, como pode ser visto na FIG 5.5.

A matriz de dependências $[D_1]$ para a TL do DES incorpora, portanto, os efeitos da permutação associados ao efeito da estrutura Feistel. A matriz $[D_1]$ e as MDR $[D_2]$ e $[D_{16}]$ podem ser vistas no apêndice 5, no item 12.5.2. A TL do DES, *ignorando o efeito das tabelas de substituição*, não é suficiente para garantir a *completude* da função criptográfica[132] com 16 passos. Em (FERREIRA, 2003) pode ser visto que esta afirmação é extensível até os 50 passos, pelo menos[133] (vide conclusão do item 5.5.2).

5.5.2 DES – A ESTRUTURA DAS TABELAS DE SUBSTITUIÇÃO

Durante este trabalho foram feitas várias tentativas de inversão da operação de substituição do DES. Se esta transformação for interpretada como uma função de $Z_2^6 \rightarrow Z_2^4$, é natural que a inversão não seja possível, pois para cada elemento de Z_2^4 existem quatro possíveis entradas de Z_2^6. Entretanto, se a expansão que ocorre antes da substituição for interpretada como parte do mecanismo de endereçamento para a busca nas tabelas de substituição, pode-se dizer que a substituição é uma operação de $Z_2^{32} \rightarrow Z_2^{32}$. Neste caso, *a priori*, não há razão para afirmar que a inversão não é possível. Em (XEXEO, 1983) mostrou-se que para cada *caixa-s* existem valores de saída que determinam alguns bites da entrada (entradas invariantes). Este resultado encontra-se também em (LAMBERT, 2003d) onde esta propriedade é explorada (vide apêndice 6). A conseqüência importante é o seguinte aprendizado: em uma transformação de substituição de $Z_2^n \rightarrow Z_2^m$ projetada para ser não-invertível, deve-se tomar cuidado para que a análise dos 2^{n-m} valores de entrada possíveis não permita a dedução de qualquer bite da entrada.

Em relação à CL e à CD a TNL do DES é a mais vulnerável entre os três algoritmos estudados neste trabalho e, por isso, foi dada maior importância às TNL do AES e a TNL baseada nos quadrados latinos (QL – capítulo 7) neste trabalho.

[132] Note que para esta afirmação não é necessária a HNC.
[133] No capítulo 9 (item 9.3.1) são feitas considerações sobre a avalanche nas estruturas Feistel.

5.6 AES – RIJNDAEL –ANÁLISE DA TRANSFORMAÇÃO LINEAR UTILIZANDO AS MATRIZES DE AVALANCHE[134]

5.6.1 A TRANSFORMAÇÃO LINEAR DO AES

Seja *g* a função de passo do Rijndael, definida como a seqüência de aplicações das transformações SUBSTITUIÇÃO, PERMUTAÇÃO, MULTIPLICAÇÃO, SOMASUBCHAVE, conforme apresentadas no início deste capítulo.

As componentes da função *g* que interessam[135] nesta análise são a *PERMUTAÇAO*[136] (*Shift Rows*) e a *MULTIPLICAÇÃO* (*Mix Columns*). O objeto de estudo é o poder de avalanche da transformação linear *g'***,** aplicação em seqüência das transformações *PERMUTAÇÃO* e *MULTIPLICAÇÃO*. A função *g'* está ilustrada na FIG. 5.7.

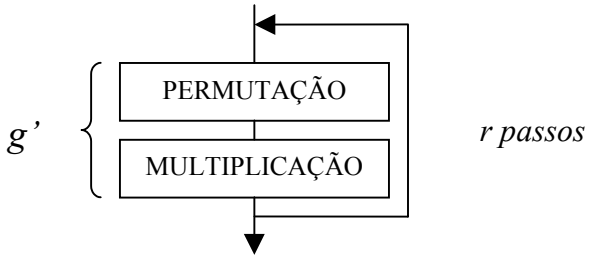


FIG. 5.7 Transformação linear do AES

5.6.2 O PODER DE AVALANCHE DA *PERMUTAÇÃO*

A *PERMUTAÇÃO* eqüivale a multiplicar o bloco de entrada *E*, baite a baite, pela matriz binária [***P***], conforme ilustrado na FIG. 5.8. A matriz que representa uma permutação de *n* símbolos é obtida aplicando-se a permutação em questão às linhas da matriz identidade $[I]_{nxn}$.

A mesma permutação pode ser representada no nível dos bites utilizando uma matriz $[P']_{128x128}$.

---

[134] Obs. A operação não-linear no Rijndael satisfaz a HNC.

[135] A operação de *SUBSTITUIÇÃO* não é levada em consideração por ser não linear e a operação *SOMASUBCHAVE* também não é levada em consideração por ser a chave de passo K_i produzida por um processo de expansão de chaves que independe das transformações lineares que se pretende analisar.

[136] Note-se que uma permutação também é essencialmente uma multiplicação por matriz binária em que cada linha e cada coluna possuem exatamente um termo igual a 1.

O **aspecto importante da permutação** *[P]* é que esta transformação modifica apenas a ordenação dos baites. Ela modifica apenas a composição das palavras, respeitando a integridade dos baites. Cada palavra do vetor de saída é composta por quatro baites distintos, sendo que cada um destes baites tem origem em uma das quatro palavras do vetor de entrada. Esta transformação não pode, isoladamente, produzir efeito avalanche, pois a modificação de qualquer bite (baite) da entrada afetará apenas um bite (baite) na saída. É um exemplo de transformação que produz *difusão* sem apresentar efeito avalanche. Esta transformação é útil para aumentar a avalanche quando convenientemente associada a outra transformação. No caso do Rijndael, esta outra transformação é a operação de *MULTIPLICAÇÃO* (*MixColumns*).

S0		**1**	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.		E0
S1		.	.	.	.	.	**1**	.	.	.	.	.	.	.	.	.	.		E1
S2		.	.	.	.	.	.	.	.	.	.	**1**	.	.	.	.	.		E2
S3		.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	**1**		E3
S4		.	.	.	.	**1**	.	.	.	.	.	.	.	.	.	.	.		E4
S5		.	.	.	.	.	.	.	.	.	**1**	.	.	.	.	.	.		E5
S6		.	.	.	.	.	.	.	.	.	.	.	.	.	.	**1**	.		E6
S7		.	.	.	**1**	.	.	.	.	.	.	.	.	.	.	.	.		E7
S8	=	.	.	.	.	.	.	.	.	**1**	.	.	.	.	.	.	.	x	E8
S9		.	.	.	.	.	.	.	.	.	.	.	.	.	**1**	.	.		E9
S10		.	.	**1**	.	.	.	.	.	.	.	.	.	.	.	.	.		E10
S11		.	.	.	.	.	.	.	**1**	.	.	.	.	.	.	.	.		E11
S12		.	.	.	.	.	.	.	.	.	.	.	.	**1**	.	.	.		E12
S13		.	**1**	.	.	.	.	.	.	.	.	.	.	.	.	.	.		E13
S14		.	.	.	.	.	.	**1**	.	.	.	.	.	.	.	.	.		E14
S15		.	.	.	.	.	.	.	.	.	.	.	**1**	.	.	.	.		E15

FIG. 5.8 Forma matricial[137] da PERMUTAÇÃO do AES

Uma conclusão de caráter geral da análise da permutação acima descrita é que sempre que uma transformação pode ser representada por uma matriz, na qual cada linha apresenta apenas um valor não nulo, esta transformação não tem poder de produzir o efeito avalanche[138].

[137] Os pontos representam o valor *0*.

[138] Entretanto, este tipo de transformação é utilizado como amplificador do efeito avalanche de outras transformações, lineares ou não, como ocorre no DES e no AES. No AES ela é fundamental para a argumentação da resistência contra a CD e a CL (CASTANÕ, 1998, cap. 5)(DAEMEN, 1999, p. 30).

5.6.3 OPERAÇÃO DE MULTIPLICAÇÃO

A multiplicação pela matriz $[M]_{32x32}$ (FIG. 5.2) é uma TL de Z_2^{32} sobre Z_2^{32}. O bloco *S* resultante da *MULTIPLICAÇÃO* aplicada a um bloco *E* ($S \in Z_2^{128}$, $E \in Z_2^{128}$) é a concatenação das quatro *palavras* W_0, W_1, W_2, W_3, *onde*

$W_0 = S0|S1|S2|S3 = [M]_{32x32}\ x\ (E0|E1|E2|E3)^T$,

$W_1 = S4|S5|S6|S7 = [M]_{32x32}\ x\ (E4|E5|E6|E7)^T$,

$W_2 = S8|S9|S10|S11 = [M]_{32x32}\ x\ (E8|E9|E10|E11)^T$

$W_3 = S12|S13|S14|S15 = [M]_{32x32}\ x\ (E12|E13|E14|E15)^T$,

Observe[139], por exemplo, o baite *S0 = (s0|s1|s2|s3|s4|s5|s6|s7)* do bloco de saída *S*. Cada um dos bites de *S0* é função de mais de um bite da entrada *E* (*avalanche*) e estes bites são oriundos de quatro baites distintos (*difusão*). Estes quatro baites distintos, entretanto, são oriundos da mesma *palavra* da entrada. Nota-se, portanto, que o efeito avalanche é observado de cada *palavra* de entrada em relação à *palavra* de mesma posição na saída. Ou seja, o efeito avalanche introduzido por esta transformação restringe-se aos grupos de 32 bites que estão sendo operados de cada vez. Cada baite (bite) de saída é afetado apenas por baites (bites) da respectiva palavra do bloco de entrada. Portanto, existe *avalanche* e *difusão*, mas a *MULTIPLICAÇAO* não é capaz de produzir a *completude* da TL. É fundamental, então[140], o uso da permutação vista 5.2.8 e 5.6.2 para possibilitar a *difusão* por todo o bloco de 128 bites. Para avaliar o poder de avalanche de *g'* utilizam-se as matrizes de dependências recursivas.

5.6.4 A CONSTRUÇÃO DA MATRIZ DE DEPENDÊNCIAS DA TL *g'* DO RIJNDAEL

A transformação linear *g'* tem como argumento um bloco de 128 bites e como saída outro bloco de 128 bites. Convém interpretar a *MULTIPLICAÇÃO* como uma TL de Z_2^{128} sobre Z_2^{128}. Para tanto, , observando que cada palavra da saída é o resultado da aplicação da matriz $[M]_{32x32}$ sobre a respectiva palavra da entrada, é suficiente que a matriz $[M]_{32x32}$ seja substituída pela matriz $[M']_{128x128}$, construída a partir de $[M]_{32x32}$ da seguinte forma:

[139] Pode ser visto com mais detalhe na TAB. 12.1.3, item 12.1.3.2.2.

[140] Observação: uma vez que a operação de substituição atua em cada baite separadamente, sem a *permutação* a função criptográfica do Rijndael (AES) jamais seria *completa*, independentemente do número de iterações.

$$[M']_{128x128} = \begin{vmatrix} [M]_{32x32} & [N]_{32x32} & [N]_{32x32} & [N]_{32x32} \\ [N]_{32x32} & [M]_{32x32} & [N]_{32x32} & [N]_{32x32} \\ [N]_{32x32} & [N]_{32x32} & [M]_{32x32} & [N]_{32x32} \\ [N]_{32x32} & [N]_{32x32} & [N]_{32x32} & [M]_{32x32} \end{vmatrix}$$

Onde $[N]_{32x32}$ é uma matriz com todos os elementos nulos. A matriz $[M']_{128x128}$ pode ser vista no item 12.3.1 do apêndice 3.

Para completar a matriz que representa a função de transformação linear *g'*, permutam-se as colunas da matriz *[M']* segundo a permutação *P*. A matriz resultante é apresentada no item 12.3.2 e é a *matriz de dependências* $[D_1]_{128x128}$ de *g'*. Para ilustrar a equivalência[141] entre $[D_1]$ e a aplicação da PERMUTAÇÃO seguida da MULTIPLICAÇÃO, foi feita uma implementação do Rijndael em que estas operações foram substituídas por uma única multiplicação pela matriz $[D_1]$. Para testar a implementação foram usados os mesmos blocos de texto em claro e chave apresentados para teste no (FIPS 197, 2001) e no apêndice 2. O criptograma resultante foi o mesmo encontrado pelos autores da referência, o que, juntamente com a observação do processo de construção de $[D_1]$ deve ser suficiente para mostrar a equivalência das transformações. O resultado e o código fonte utilizados podem ser vistos nos itens 12.3.3 e 12.3.4 do apêndice 3.

5.6.5 MATRIZES DE AVALANCHE $[D_r]$ PARA O AES *(r>1)*

As *r-ésimas MDR* $[D_r]_{128x128}$ foram construídas recursivamente, conforme é descrito no capítulo 4.

As *MDR* da transformação linear do Rijndael e o código fonte utilizado para construí-las estão apresentados no apêndice 4.

É importante observar nas MDR (e nas MDE) do Rijndael que cada baite afeta exatamente quatro baites em uma iteração e afeta 16 baites em duas iterações. Esta observação, juntamente com as TNL, é a base para a demonstração da resistência contra a CD e a CL realizada em (CASTAÑO, 1998), (DAEMEN, 1995) e (DAEMEN, 1999).

[141] $M(P(E)) = [D_1].E$, onde *M* é a operação *MULTIPLICAÇÃO (Mix Columns)* e *P* é a *PERMUTAÇÃO (Shift Rows)*.

5.6.6 LIMITES INFERIOR E SUPERIOR DO PODER DE AVALANCHE – *LIPA e LSPA*

No caso do Rijndael, observou-se que a *completude* da TL é atingida na **sétima** iteração. A TAB. 5.3 apresenta os valores dos limites de poder de avalanche nas MDR da TL do Rijndael (FERREIRA, 2003).

TAB. 5.3: Valores LIPA e LSPA da TL do Rijndael para cada iteração

	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter	7ª iter
LIPA	5	24	55	78	96	112	128
LSPA	7	40	95	126	128	128	128

5.6.7 ANÁLISE DAS MDE DO RIJNDAEL

No apêndice 4, a apreciação das MDE do Rijndael(AES) permite observar que a cifra é completa para duas iterações[142]. Além disso, também pode ser observado que em uma iteração exatamente quatro baites da saída são afetados por cada baite da entrada em uma iteração.

5.6.8 RESISTÊNCIA DA TNL CONTRA CL E CD

Seja uma TNL com entrada de tamanho *m* bites e saída de tamanho *n* bites (LIMA, 1999). Existem 2^{m+n} equações lineares ε_i envolvendo bites da entrada e bites da saída$(i=0,1,2,3... 2^{m+n}-1)$. Seja p_i a probabilidade de uma equação linear envolvendo bites da entrada e da saída da TNL valer. A vulnerabilidade da TNL em relação a CL depende dos valores $|p_i-1/2|$. Se $|p_i-1/2| = 0$ para todo *i,* a CL não pode ser realizada. Para a TNL do Rijndael (com 2^{16} equações e saída de 8 bites), temos o valor de $|p_i-1/2|$ máximo igual 16/256 = 1/16 (obtido com o mesmo método utilizado para os QL e apresentado no código fonte, apêndice 11).

O maior desvio $|p_i-1/2|$ para a TNL do Serpent é ¼ (e 1/8 para equações que só envolvem um bite em cada membro). Para o DES este valor é 5/16 (CASTAÑO, 1998, p. 17, 109).

[142] A matriz completa foi omitida, mas pode ser observado na matriz de uma iteração, pelas relações de dependências entre os baites.

Para a tabela de substituições do AES (tabela 5.2), a probabilidade média de ocorrer uma diferença *S'* dada uma diferença *E'* na entrada é 1/256. A diferença *S'* mais provável de ocorrer, dada uma diferença *E'*, ocorre[143] com probabilidade 4/256. A *RS/R*[144] da tabela é portanto ***(4/256)/(1/256)* = 4**. Esse é o valor limite sugerido para considerar aceitável uma TNL (que opera por baites) no que diz respeito à resistência contra a CD. Entretanto, para que a cifra seja no mínimo tão forte quanto o AES, é necessário que a TL tenha um poder de difusão no mínimo tão grande quanto o da TL do AES, considerando o número de iterações a ser adotado.

Pelo acima exposto, a TNL do Rijndael é adotada para ser utilizada com uma pequena adaptação no cifrador *Alpha*, capítulo 8.

5.7 SERPENT – ANÁLISE DA TL UTILIZANDO AS MATRIZES DE AVALANCHE

5.7.1 DESCRIÇÃO DA TRANSFORMAÇÃO LINEAR DO SERPENT

Na função de passo do Serpent a operação de substituição (TNL) ocorre antes da TL.

A substituição foi projetada e é descrita em um modo de separação de bites (*bitslice*), que se destina a melhorar a performance em *hardware*. Para implementação em *software* do Serpent, a operação de substituição deve ser antecedida de uma permutação inicial (IP) e sucedida por uma permutação final (FP). Uma vez que estas permutações são uma adaptação da operação de substituição para a implementação em *software*, elas não foram consideradas parte da TL na construção das matrizes de avalanche, pois não interferem na complexidade da criptoanálise.

Sejam *X0,X1,X2* e *X3* quatro *palavras* que compõem o bloco de 128 *bites* *X= X0|X1|X2|X3*. O resultado *Y=Y1|Y2|Y3|Y4=TL(X)* é obtido por meio da seguinte seqüência de transformações:

$$X0 = X0 \lll 13$$
$$X2 = X2 \lll 3$$
$$X1 = X1 \oplus X0 \oplus X2$$
$$X3 = X3 \oplus X2 \oplus (X0 \ll 3)$$
$$X1 = X1 \lll 1$$
$$X3 = X3 \lll 7$$

[143] As tabelas foram geradas utilizando o mesmo código fonte que foi utilizado na construção das matrizes de distribuição diferencial e linear dos QL(apêndice 11), mudando-se os parâmetros referentes à saída de 4 para 8 bites nos metodos *teste linear* e *teste diferencial*.
[144] Definido no capítulo 6, item 6.5.1.

$$X0 = X0 \oplus X1 \oplus X3$$
$$X2 = X2 \oplus X3 \oplus (X1 << 7)$$
$$X0 = X0 <<< 5$$
$$X2 = X2 <<< 22$$
$$Yi=Xi, (i=0,1,2,3)$$

Onde "<<" representa deslocamento e "<<<" rotação (CASTAÑO, 1998, p. 98). O conjunto de expressões booleanas equivalente a esta seqüência de operações, e que portanto representa a TL do Serpent, é apresentado em (KNUDSEN, 1999, Apd. A) e foi utilizado para a construção da matriz de dependências $[D_1]$. A matriz $[D_1]$ e as MDR da TL do Serpent encontram-se no apêndice 7. A TL do Serpent é completa em seis iterações. A TAB. 5.4 apresenta os valores dos limites de poder de avalanche nas MDR da TL do Serpent (FERREIRA, 2003).

TAB. 5.4: Valores LIPA e LSPA da TL do Serpent para até 6 iterações

	1ª iter	2ª iter	3ª iter	4ª iter	5ª iter	6ª iter
LIPA	2	9	33	92	126	128
LSPA	7	34	98	128	128	128

5.7.2 RESISTÊNCIA DA TNL CONTRA CL E CD

Vide item 5.6.8. O maior desvio $|p_i-1/2|$ para a TNL do Serpent é ¼ (e 1/8 para equações que só envolvem um bite em cada membro).

5.8 CONCLUSÃO DO CAPÍTULO

O sistema Feistel apresenta a vantagem de utilizar a mesma estrutura para cifrar e decifrar, além de possibilitar o uso de uma função de passo não invertível.

A função de expansão de chaves, quando for usada, deve ser constituída de TL e TNL e apresentar avalanche e completude, a fim de evitar simetrias que facilitem a existência de chaves fracas.

As TL do Rijndael e do Serpent apresentam boa difusão. A TL do Rijndael, combinada com a TNL, produz a completude da função em 2 iterações, o que é um ótimo resultado.

Analisando isoladamente as TL, a TL do Rijndael apresentou os melhores resultados, uma vez que utiliza menor número de bites por linha da matriz de dependências D_1 e produz a completude da função em apenas 2 iterações. Entretanto, na estrutura utilizada para decifrar, o AES apresenta de 11 a 19 coeficientes não nulos por equação (figura 12.1.2, apendice 1). Isso indica um aumento do número de portas lógicas empregadas no circuito para decifrar pode representar algum inconveniente, se a implementação for feita em *hardware*. É importante levar em consideração o desempenho em *hardware* e *software*.

Em relação às características diferenciais e lineares, a TNL do AES foi tomada como referência. No conjunto de princípios de projeto proposto no capítulo 6 serão buscados valores de |p-1/2| limitados superiormente por 1/16 e RS/R limitadas superiormente por 4.

6 FILOSOFIA DE PROJETO RECOMENDADA PARA CIFRADORES SIMÉTRICOS DE BLOCOS

Este capítulo baseia-se principalmente no estudo dos processos de seleção de algoritmos realizados pelo NIST (BIHAM, 1999)(NECHVATAL, 1999), no estudo dos algoritmos DES, Rijndael e Serpent e nos conceitos presentes em (MENEZES, 1996), (SCHNEIER, 1996) e (DAEMEN, 2003).

6.1 REFLEXÃO INICIAL

Eis algumas das muitas características necessárias em um cifrador de blocos*: não deve haver nenhuma relação estatística evidente entre texto em claro e texto cifrado; alterando-se qualquer bite no texto em claro todos os bites do texto cifrado devem se alterar com probabilidade ½* e *alterando-se qualquer bite do texto cifrado a alteração no texto em claro recuperado deve ser imprevisível* (MENEZES, 1996, p. 256).

Antes de iniciar a concepção do projeto de um algoritmo, as seguintes perguntas precisam ser respondidas:

- O algoritmo será baseado em projetos existentes ou será totalmente novo?
- O algoritmo será do tipo Feistel/SPN?
- Como será construída a função de passo?
- Qual a margem de segurança desejada e quantas iterações serão utilizadas ?
- Que operações básicas (componentes) serão empregadas?
- Que técnicas serão empregadas?
- Em que plataforma será implementado o algoritmo?

Todas as respostas às perguntas acima refletem decisões que afetam fundamentalmente o projeto.

Com base na análise feita nos capítulos precedentes e na bibliografia estudada, este capítulo apresenta uma *filosofia de projeto para cifradores simétricos de blocos*. Esta filosofia se traduz no respeito aos *cinco princípios de projeto* apresentados a seguir: *segurança, eficiência, flexibilidade, simplicidade, credibilidade.* Dentro do respeito a cada princípio, é sugerido um conjunto de requisitos para o projeto e para transformações criptográficas que o algoritmo incorpora. A satisfação destes requisitos parece ser condição

suficiente para projetar um cifrador simétrico de blocos dentro dos atuais padrões internacionais de aceitação.

6.2 PRIMEIRO PRINCÍPIO: SEGURANÇA

A segurança de um algoritmo simétrico é função de dois parâmetros essenciais: *a força do algoritmo* e o *tamanho do espaço de chaves*[145]. Ambas são igualmente importantes, mas a primeira torna-se mais crítica, por ser muito mais difícil de avaliar. O algoritmo é considerado *forte* se o método mais rápido para quebrar o algoritmo é a busca exaustiva da chave. Provar isso é um problema mais difícil que a própria criptoanálise. Apesar disso, é possível estabelecer limites inferiores para o fator trabalho da criptoanálise que são aceitos pela comunidade científica. Para isso, não deve haver nenhum *alçapão* ou *atalho* aparente, o algoritmo deve ser resistente contra as formas de criptoanálise conhecidas e deve ser considerado improvável o surgimento de uma nova forma de criptoanálise que o ameace. Isso deve ser verdadeiro para algum número de iterações[146] menor do que o especificado para a cifra. Portanto, o algoritmo deve ser baseado em alguma estrutura criptográfica bem conhecida e consagrada. Além disso, o espaço de chaves deve ser grande o suficiente para tornar a busca exaustiva um *problema intratável*[147].

Primeiro requisito:

A especificação completa do algoritmo deve ser pública ou *suficientemente*[148] semelhante a alguma estrutura de conhecimento público que esteja submetida à comunidade científica há muitos anos, sem ser comprometida.

Segundo requisito:

A *chave*, os blocos de *texto em claro* e os blocos de *texto cifrado* devem ter, no mínimo, 128 bites. Deve ser possível utilizar chaves maiores até o limite em que o usuário, se desejar, gera independentemente todas as sub-chaves de fase.

---

[145] Rigorosamente, é importante o *número de classes de equivalência* no espaço de chaves. Como este é um parâmetro usualmente muito difícil de ser aferido, é aceito o tamanho do espaço de chaves (2^n, n = tamanho da chave)(DAEMEN, 2003).

[146] Aqui admite-se que será utilizada uma cifra iterativa com estrutura S-TL

[147] Na prática, é suficiente que a busca exaustiva não seja compensadora. Isso depende do valor da informação. Se o algoritmo destina-se a proteger informações capazes de comprometer grandes empresas ou interesses de estado, a solução do problema deve estar além de toda a capacidade de processamento disponível. Vide (SCHNEIER, 1996, cap 7) e (BLAZE, 1995).

[148] As semelhanças devem ser suficientes para que as críticas e ataques dirigidos a um algoritmo possam ser estendidos ao outro.

Terceiro requisito:

O algoritmo deve ser baseado em uma estrutura tipo Feistel e deve ser utilizada uma função de passo não-linear e não-invertível. Tal estrutura já foi exaustivamente estudada pela comunidade científica por mais de 27 anos, o que torna menos esperado o surgimento de uma nova forma de ataque.

Quarto requisito:

As características diferenciais e lineares da transformação não-linear devem ser tão boas quanto as da transformação linear do AES, buscando valores de |p-1/2| limitados superiormente por 1/16 e RS/R limitadas superiormente por 4. Além disso, as caracterírticas de difusão na TL devem ser tão boas quanto as do AES, considerando o número de iterações adotado. Isso porque para o AES já está demonstrada a resistência contra a CD e a CL e a demonstração baseia-se nestes parâmetros (DAEMEN, 2003).

Quinto requisito:

A transformação não-linear deve possibilitar reduzir as características diferenciais e lineares pela substituição ou aumento da quantidade de das tabelas de substituição e tal opção deve ser simples de implementar (vide técnica dos quadrados latinos no capítulo 7).

Sexto requisito:

A função criptográfica deve ser *completa* em no máximo[149] 4 iterações. Isso porque a análise do AES mostrou ser possível atingir a completude em apenas duas iterações, com uma TL que tem até sete coeficientes não nulos por equação e quatro é o número mínimo de iterações necessário para a completude em um Feistel (vide capítulo 9, item 9.3.1). O número de iterações mínimo[150] deve ser 12, por razões de segurança e credibilidade. A transformação linear da função de passo deve ser poderosa o suficiente para produzir a *completude* da função desde que a TNL satisfaça à HNC.

Sétimo requisito:

Quando for usada uma função de expansão de chaves, esta função deve ser não linear, não invertível, deve apresentar o *efeito avalanche*[151], e não deve ser fácil obter o conjunto de

[149] Para blocos de 128 bites.

[150] Isso porque que estão sendo adotados como parâmetros de resistência contra a CD e a CL os valores de difusão da TL, distribuição linear e diferencial do Rijndael (com 128 bites de chave e bloco). O valor doze equivale a uma difusão completa além das oito fases para as quais se admitiu possível encontrar uma trilha de propagação explorável criptoanaliticamente. Vide 7.2.4.2, capítulo 7.

[151] Estas condições tornam muito improvável a existência de chaves fracas e ou semi-fracas.

chaves de fase a partir da sub-chave de fase K_r (, $r \geq i$, para algum $i<Nr$, *Nr = número de iterações*).[152]

6.3 SEGUNDO PRINCÍPIO: EFICIÊNCIA

O algoritmo deve basear-se em operações que sejam executadas rapidamente, tais como deslocamentos, somas módulo dois e busca em tabelas de substituição. O algoritmo deve ser uma *estrutura S-TL,* pois este tipo de estrutura é bem conhecido e permite produzir boa complexidade a partir de operações simples, iterativamente. A função de passo deve ser composta de uma transformação linear capaz de produzir a *completude* da função em poucas iterações e a margem de segurança não deve ser "excessiva". Deve ser respeitado um limite superior de quantidade de bites suportada pelos componentes de *hardware* e um limite inferior que possibilite os efeitos desejados (avalanche, completude, não-linearidade, por exemplo) com um número não muito grande de iterações;

As transformações não lineares devem ser projetadas e empregadas para aumentar o poder de avalanche das transformações lineares. Deve ser levada em consideração também a velocidade da decifração, sobretudo em sistemas que utilizam transformações significativamente diferentes[153] para cifrar e decifrar.

Oitavo requisito:

As transformações lineares devem estabelecer *matrizes dependências*[154] que apresentem em todas as linhas um mesmo peso *hamming* maior ou igual a quatro e menor ou igual a oito.[155]

---

[152] Esta condição facilita a defesa contra o ataque por DPA, apresentado em (KOCHER, 199X)(SHAMIR, 1999). Esse tipo de ataque atua principalmente descobrindo a sub-chave que é utilizada no *clareamento* (*whitenning* - soma de um bloco de sub-chave antes de cifragem de cada bloco, normalmente K_0), a partir da qual as demais sub-chaves são obtidas. Se a expansão de chaves for feita e não for possível obter o conjunto de sub-chaves a partir de K_4, por exemplo, pode-se utilizar K_4 para fazer o *clareamento* e K_0 na fase 4. Essa é apenas uma sugestão para defesa contra DPA quando se pretende utilizar *clareamento*. No caso do cifrador apresentado no capítulo 7 esta técnica não é empregada.

[153] No caso do Rijndael, a transformação *Inv Mix Columns*, utilizada na decifração utiliza uma matriz com constantes de maior magnitude, o que aumenta o fator trabalho da operação em relação à *Mix Columns*. Quando traduzida em uma matriz de dependências, esta operação inversa apresenta de 11 a 19 coeficientes não nulos por equação (vide apendice 1).

[154] Vide capítulo 6.

[155] Nos testes com matrizes de avalanche em diversas transformações lineares foi constatado que pesos *hamming* menores que quatro têm como conseqüência uma avalanche mais lenta que a do Rijndael, sendo necessárias mais iterações para a *completude* da função. Pesos *hamming* maiores que 8 bites aumentam desnecessariamente a complexidade do algoritmo e o custo do hardware, uma vez que é possível construir transformações lineares

Nono requisito:

As transformações não-lineares devem ser feitas com tabelas de substituição e cada busca na tabela de substituição deve ter como argumento de entrada um bloco de *8.n* bites e como saída um bloco de *4.m* bites (*n, m* inteiros positivos, $n \geq m$). A substituição deve ser *completa* a cada busca na tabela. Ou seja, cada um dos *4.m* bites da saída deve ser *afetado* por todos[156] os *8.n* bites da entrada. Isso porque é interessante que a entrada nas tabelas de substituição opere por baites, o que facilita as demonstrações em relação ao AES e o processamento em processadores de oito bites, além de resultar em tabelas relativamente pequenas. O valor múltiplo de 4 na saída foi sugerido tendo em vista as operações não-invertíveis como aquelas baseadas nos QL (capítulo 7).

Décimo requisito:

O tamanho dos blocos deve ser múltiplo de 64 bites, para melhorar a eficiência em processadores de 32 e 64 bites.

Décimo primeiro requisito:

O número de iterações[157] não deve ser muito maior que 16. Uma vez que as TL e TNL estão sendo projetadas com maior resistência contra a CD e a CL (em relação ao DES) e, no mínimo tão resistentes quanto às do AES esse número deve ser suficiente para tornar o fator trabalho esperado para a CD e a CL maior que o da busca exaustiva. Números maiores de iterações devem ser evitados a bem da eficiência.

Décimo segundo requisito:

Quando for utilizada uma função de expansão de chaves *não deve ser necessário* armazenar todas as sub-chaves. Deve ser possível gerar cada sub-chave de passo a partir da(s) sub-chave(s) do(s) passos anterior(es). Esse requisito só é importante quando o algoritmo for utilizado em dispositivos com restrições de memória.

---

mais eficientes (em relação ao número de iterações para a completude) que a do Rijndael e a do Serpent com pesos *hamming* iguais a 7, como pode ser visto no capítulo 7.

[156] Note-se que esta condição garante que a transformação não linear satisfaz a hipótese do não cancelamento das dependências.

[157] Considerando chave e blocos de 128 bites.

6.4 TERCEIRO PRINCÍPIO: FLEXIBILIDADE

O tamanho fixo da chave praticamente condenou o DES à substituição pelo AES, o que sugere que a *flexibilidade em relação ao tamanho da chave* é um aspecto desejável em um projeto de algoritmo criptográfico. Essa caracterírtica é importante para a *segurança* e a *credibilidade* da cifra, além de aumentar a sua expectativa de vida útil.

Flexibilidade em relação à plataforma de implementação: um algoritmo criptográfico deve ser implementável em diferentes plataformas com eficiência satisfatória. Esta característica é importante, pois proporciona a boa compatibilidade entre os diferentes equipamentos em que o algoritmo será executado, contribuindo para sua popularização.

Flexibilidade em relação à estrutura[158]*:* deve ser possível substituir as transformações linear não-linear por outras (igualmente seguras).

Flexibilidade em relação ao tamanho dos blocos: deve ser possível implementar o algoritmo para cifrar blocos de tamanhos diferentes.

Praticamente todos os requisitos já relacionados contribuem para a flexibilidade do projeto. Entretanto, merecem destaque os seguintes: *segundo, terceiro, quinto e o décimo primeiro requisitos.*

6.5 QUARTO PRINCÍPIO: SIMPLICIDADE

Este princípio está associado à facilidade de compreensão do algoritmo. O algoritmo deve ser suficientemente simples para compreensão e implementação em *hardware* e em *software.* É desejável que o conceito de simplicidade se estenda aos processos de construção das transformações, sobretudo das não lineares. A simplicidade também é importante quando da validação do algoritmo, quando é desejável que se possam utilizar as técnicas já consagradas. Do estudo feito ao longo deste trabalho, considerando o grau de conhecimento do DES pelo público, é razoável dizer que um algoritmo estruturalmente parecido com o DES é de simples compreensão e implementação.

A preocupação com a *eficiência*, a *flexibilidade* e a *credibilidade* devem naturalmente conduzir a um cifrador *simples*. Entretanto, merecem destaque sob a ótica da simplicidade o *primeiro*, o *terceiro*, *oitavo*, o *nono* e o *décimo* requisitos.

[158] Modificações nas transformações lineares e não lineares, mesmo que pequenas, podem comprometer a segurança da cifra. Só são permitidas modificações deste tipo se as transformações substitutas forem adequadas, não comprometendo os demais requisitos.

6.6 QUINTO PRINCÍPIO: CREDIBILIDADE

Um algoritmo criptográfico será tão mais útil quanto mais for utilizado pela comunidade, pois para que um remetente utilize uma cifra ele precisa saber que o destinatário também a utiliza. Para que um algoritmo seja bem aceito pela comunidade, a credibilidade é mais importante que a própria *segurança* do algoritmo. É extremamente improvável que um algoritmo inseguro (com algum alçapão, por exemplo) tenha credibilidade na comunidade científica. Por outro lado, é relativamente fácil a credibilidade de um algoritmo ser comprometida. Isso pode acontecer mesmo que não seja encontrada uma fraqueza no algoritmo que comprometa de fato a sua segurança. É suficiente que não fiquem claras as razões de projeto e que a estrutura do algoritmo seja difícil de analisar com as técnicas tradicionais. Isso ficou claro quando a IBM não publicou na década de 70 as regras para a distribuição das constantes nas tabelas de substituição e a razão para o número de iterações adotado no DES (SCHNEIER, 1996, p. 284, 293). Portanto, as regras seguidas para a construção das transformações utilizadas pelo cifrador devem ser claras, simples e divulgadas, para que não haja suspeitas da existência de *alçapões* ou *atalhos*. Utilizando técnicas de construção conhecidas, a análise teórica da vulnerabilidade do algoritmo é mais fácil, pois já existem técnicas desenvolvidas para estabelecer limites inferiores para o fator trabalho nas criptoanálises linear e diferencial (vide capítulo 6). É prudente esperar que as novas técnicas criptográficas tenham um tempo maior de maturação antes de utilizá-las ((BIHAM, 1999), (SCHNEIER, 2002), (SCHNEIER, 2002b), (SCHNEIER, 1996, p. 346)).

Todos os requisitos supracitados são importantes para a credibilidade do algoritmo. O mais importante deles é o *primeiro requisito*, pois o fator mais importante para a *credibilidade* do algoritmo é o tempo que este é utilizado publicamente sem que seja encontrada qualquer vulnerabilidade. Como, infelizmente, o projetista de um novo algoritmo não tem este fator a seu favor, seguir o primeiro requisito pode ajudar.

Também é de especial importância a demonstração da resistência contra as formas conhecidas de criptoanálise (CD e CL).

Décimo terceiro requisito:

Quando forem utilizadas constantes escolhidas pseudo-aleatoriamente, o algoritmo gerador de pseudo-aleatórios deve ser "confiável"[159].

6.7 CONCLUSÃO DO CAPÍTULO

O conjunto de requisitos apresentado neste capítulo conduz ao projeto de uma cifra de bloco de acordo com o que foi observado como características positivas nos capítulos 3 e 4. É possível construir cifras de boa qualidade sem necessariamente atender a todos os requisitos mencionados. Entretanto, atender a estes requisitos deve facilitar o desenvolvimento e a validação do algoritmo. Nos capítulos seguintes estes requisitos são empregados de maneira mais prática.

[159] Uma sugestão para este fim é utilizar uma função que gere uma distribuição uniforme de números a partir de um texto conhecido e antigo. Desta forma, não deverá ser esperado nenhum *alçapão* escondido nas constantes da cifra.

7 TÉCNICAS DE PROJETO SUGERIDAS

7.1 INTRODUÇÃO

Neste capítulo são apresentadas duas novas técnicas para construção de transformações criptográficas: uma para a construção de transformações lineares e outra para a construção de transformações e não-lineares.

7.2 CONSTRUÇÃO DE TRANSFORMAÇÕES LINEARES – MÉTODO DOS NÓS

O objetivo do método a seguir é construir transformações lineares com boa difusão (padrão de propagação) e que garantam[160] a *completude* da função criptográfica após um determinado número de iterações. As tabelas de substituição (usadas nas TNL) também são projetadas para produzir *efeito avalanche,* de modo que mais de um de bite da saída da transformação de substituição seja *afetado* pela alteração de qualquer dos bites da entrada (SCHNEIER, 1996, p. 349, 350). O *método dos nós* leva essa característica em consideração para melhorar as propriedades da função criptográfica.

7.2.1 A IMPORTÂNCIA DO NÚMERO DE COEFICIENTES NÃO NULOS EM CADA EQUAÇÃO DA TRANSFORMAÇÃO LINEAR

Em geral, quanto maiores os valores de *hmin* e *hmax* na matriz de dependências $[D_1]$, maior o poder de difusão uma TL e existe uma tendência de que a completude da função seja atingida com um número menor de iterações. Isso sugere que o número de fases a ser especificado para a cifra com maiores pesos *Hamming* em $[D_1]$ pode ser menor. Por outro lado, quanto maiores os valores de *hmin* e *hmax*, maior tende[161] a ser o fator trabalho da

[160] Supondo que as demais transformações no algoritmo satisfazem a HNC.

[161] Para implementação em hardware, se a TL for implementada com portas lógicas, será necessária uma maior quantidade de portas. Em *software* isso também parece ser verdade. No caso do AES, por exemplo, a multiplicação inversa tem maiores pesos Hamming que a multiplicação. Isso está associado aos valores das constantes na matriz original para a *InvMixColumns,* que são maiores que as constantes da operação *MixColumns* (FIPS 197, 2001) . Em software, multiplicação por constantes maiores também costuma ser mais lenta que a multiplicação por constantes menores. Em 12.1.3.2.1 é possível ver como os pesos *Hamming* de $[D_1]$ crescem com a magnitude das constantes multiplicativas.

execução da TL. Parece existir então uma relação de compromisso que precisa ser considerada quando do projeto de uma transformação linear.

Seja *c* um número inteiro que representa a quantidade de ciclos necessária para computar uma operação *ou-exclusivo* em uma dada plataforma. Seja *q* a quantidade de bites que pode ser operada em *c* ciclos. Um aspecto importante a ser considerado é que normalmente o tempo gasto para operar uma quantidade de bites menor ou igual a *q* é *c*. Da mesma forma, o número de ciclos (tempo) necessário para computar uma expressão com *b* bites *(a.q < b ≤ (a+1).q, a* inteiro*) é c.(a+1).* Exemplo: Para um *hardware* que utiliza registradores de 8 bites, armazenar e/ou transferir um bite tem o mesmo custo que armazenar ou transferir 2, 3, 4 ..., ou 8 bites. Para armazenar 9 bites, este *hardware* ocupa a mesma quantidade de registradores que ocuparia para armazenar até 16 bites, e assim por diante. Uma boa compreensão do relacionamento entre as instruções e as interfaces *hardware/software* pode ser obtida estudando o capítulo 3 de (PATTERSON, 2000) e realizando uma leitura de (SCHNEIER, 1999).

Observando o que foi exposto, são importantes as considerações a seguir.

Consideração número 1: Os múltiplos de *q* são limites superiores para *hmax* e que *hmin* deve preferencialmente ter valores próximos de *hmax* a fim de aumentar o poder de avalanche de todas as linhas com o mínimo custo (os ciclos de processamento devem ser bem aproveitados).

Consideração número 2*: A minimização da diferença Δh=hmax-hmin deve ser, portanto, um dos objetivos que devem ser buscados no projeto de uma transformação linear.*

Nos algoritmos Rijndael e Serpent os valores de *hmin* são *5* e *2*, respectivamente, e o valor de *hmax* é 7 (para ambos) (LAMBERT, 2003), (LAMBERT, 2003), (KNUDSEN, 1999). Apesar de não ter sido estabelecida na bibliografia pesquisada uma ligação rigorosa entre Δ*h* e o desempenho do algoritmo, o Serpent, que apresenta a maior diferença Δ*h*, mostrou-se sugestivamente o mais lento em todas as implementações (HINZ, 2000) (SCHNEIER, 1999) (BAUDRON, 199X) (DRAY, 1999) (NIST, 1999). A principal razão da maior lentidão do *Serpent* é o grande número de iterações especificado. Convém ressaltar que transformações capazes de produzir a *completude* da função mais rapidamente tendem a exigir um menor número de iterações na especificação do algoritmo.

7.2.2 MATRIZES DE AVALANCHE (MDR)[162] E O PROJETO DAS TL

Considere o desenvolvimento apresentado em 4.2.2 e 4.2.3. $S^r = s^r_0|s^r_1|...|s^r_{n-1}$ representa o vetor de saída da função $(f')^r$, (FIG. 4.1. Na primeira execução de *f',* $r = 1$). $E^r = e^r_0|e^r_1|...|e^r_{n-1}$ é o vetor entrada (argumento) de $(f')^r$ na repetição número *r*.

Como pode ser visto no tópico 4.2.3, as MDR, denotadas por $[D_r]$ $(r>1)$, da transformação linear *L* são construídas com base na seguinte idéia recursiva: se o bite e^r_j afeta o bite s^r_i, todos os bites que afetaram s^{r-1}_j afetam s^r_i $(i,j \in \{0,1,...n-1\})$, $r \in \{1, 2,...m\}$, onde *n* é o tamanho do bloco (no exemplo $n=6$), *m* é o menor inteiro que satisfaz a condição *"$(f')^m$ é completa"*.

O processo de construção da *r-ésima* MDR pode ser traduzido no seguinte algoritmo. Seja σ_i o conjunto dos valores de *j* para os quais $D_{0,i,j}$ é igual a 1 $(i,j \in Z_n)$. A *i*-ésima linha da *r*-ésima MDR é o resultado da operação lógica "OU" bite a bite entre todas as linhas da *(r-1)*-ésima[163] MDR $(r>1)$ cujos índices pertencem ao conjunto σ_i. [164]

As MDR do cifrador *Beta* estão apresentadas no tópico 4.2.3. À direita de cada linha estão os respectivos conjuntos σ_i referidos acima, obtidos a partir de $[D_1]$.

As observações i) e ii) são fundamentais para entender o método para construção de transformações lineares que é apresentado no item 7.2.4.

i) A matriz $[D_1]$ determina que bites de $E^r=S^{r-1}$ afetam cada bite s^r_i de $S^r=E^{r+1}$.

ii) A matriz $[D_p]$ mostra quais os bites de E^0 que afetam s^p_i da saída S^p (onde $p=1,2,3..$ e $i=0,1,2..n-1$)

Na terceira linha (linha 2) da matriz $[D_1]$ podem-se observar os bites da entrada E^0 que, por ação de *L*, *afetam* s^1_2. Estes bites *são* e^0_0, e^0_2 e e^0_4. Pela forma como é construída a matriz de avalanche $[D_2]$, a sua linha *2* é o resultado de uma operação *OU* entre as linhas 1 e 3 da matriz $[D_1]$, pois $s^2_2 = e^2_1 + e^2_3$. Observe que na matriz $[D_1]$ o poder de avalanche da linha *1* é $h^1_1=2$ e da linha 3 é $h^1_3=3$. Sejam Q_1 e Q_3 dois conjuntos. Considere a seguinte propriedade:

[162] Matrizes de dependências recursivas (LAMBERT, 2003b).

[164] Se for definido o produto de matrizes do forma que a soma dos elementos seja substituída por uma operação lógica *OU,* temos que $[D_r]$ é dada pela r-ésima potência $[D_1]^r$ da matriz $[D_1]$. Esta construção tem paralelos

$|Q_1 \cup Q_3| = |Q_1|+|Q_3| - |Q_1 \cap Q_3|$. Graças a esta propriedade, o peso *Hamming* (h^2_2) do resultado da operação lógica *OU* entre as linhas 1 e 3 da matriz $[D_1]$ e tem necessariamente que estar no intervalo fechado [3, 2+3=5]. Como os bites ativos da linha *1* também estão ativos na linha *3* de $[D_1]$, o número de bites da entrada que afetam s^2_2 é $h^2_2 = 3$. Se a transformação *L* tivesse sido projetada de maneira que os bites ativos nas linhas *1* e *3* de $[D_1]$ fossem distintos, h^2_2 seria *5*, seu valor máximo. Aumentar os pesos *Hamming* das matrizes $[D_p]$ para $p>1$ sem aumentar os pesos *Hamming* das linhas de $[D_1]$ é o principal objetivo de uma transformação linear em um algoritmo criptográfico.

Consideração número 3:

Seja L: $Z_2^n \rightarrow Z_2^n$ uma TL que se pretende projetar e que será definida por uma matriz de dependências $[D_1]$. Seja σ_i o conjunto dos valores de *j* para os quais $D_{1,i,j}$ é igual a *1 ($i, j \in Z_n$). Se j_1 e j_2 pertencem a um mesmo conjunto σ_i é desejável que as linhas j_1 e j_2 de D_1 apresentem uma distância de Hamming*[165] *tão grande quanto possível. Se isso se verificar para todo i e para todo j, a transformação produzirá boa difusão sem que seja necessário aumentar a sua complexidade (pesos Hamming das linhas de $[D_1]$).*

7.2.3 O TAMANHO DOS ARGUMENTOS NAS TABELAS DE SUBSTITUIÇÃO COMO PARÂMETRO DE PROJETO

Em geral, os cifradores do tipo redes de transformação_linear/substituição (SLTN) têm tabelas de substituição que operam sub-blocos de tamanho 4, 6 ou 8 bites de cada vez. O DES, por exemplo, opera com sub-blocos de 4 bites (estendido para 6 antes da operação e reduzido para 4 ao final da mesma). O Rijndael opera com sub-blocos de 8 bites e o Serpent com blocos de 4 bites. Operando desta forma cada bite da entrada de uma *substituição* pode afetar, no máximo, todos os bites do respectivo sub-bloco de saída da mesma. Uma transformação deste tipo não tornará a função criptográfica *completa* mesmo que seja realizado um número ilimitado de iterações, pois cada bite de qualquer sub-bloco da saída será afetado apenas por bites do sub-bloco de mesma posição no bloco de entrada. Neste

---

interessantes com a teoria de grafos que pode ser encontrada em (KEMENY, 1958) e (WEISS, 1978), por exemplo.

momento, fica clara a importância das transformações lineares, cuja principal finalidade é *difundir* o efeito das transformações não-lineares por todo o bloco.

O tamanho do argumento nas tabelas de substituição é um aspecto a ser considerado quando do projeto das transformações lineares. Isto porque se considera que o projeto das tabelas de substituição deve ser feito levando em consideração as características da transformação linear que será empregada no algoritmo criptográfico e vice-versa (DAEMEN, 2000) (DAEMEN, 2002) (DAEMEN, 2003)(LANDAU, 2000)(SCHNEIER, 1996).

Tendo em mente o tamanho dos sub-blocos operados individualmente pelas operações de substituição, a transformação linear deve ser projetada de tal forma que o poder de avalanche da transformação não-linear seja aproveitado o máximo possível. Para isso, os bites da entrada da TL que afetam cada bite da saída da TL devem ser oriundos de sub-blocos distintos (tanto quanto possível)[166].

Consideração número 4: *No projeto de uma transformação linear L:* $Z_2^n \rightarrow Z_z^n$ *que integra um cifrador de blocos cuja TNL opera sub-blocos de tamanho m (m é divisor de n), deve-se considerar o seguinte: seja o bloco* $E=E_0|E_1|...|E_{z-1}$ *(z =n/m), onde* $E_0,E_1,...,E_{z-1}$ *são os z sub-blocos de tamanho m do bloco E que serão operados, um de cada vez pelas tabelas de substituição; seja* $h_i=h^1{}_i$ *o poder de avalanche da i-ésima linha da matriz de dependências de L, denotada por* $[D_1]$ *. Seja* $S = s_0|s_1|...|s_{n-1}=L(E)$*. Os* h_i *bites distintos, cuja soma módulo dois resulta em* s_i*, devem ser preferencialmente oriundos de sub-blocos de E distintos.*

7.2.4 MÉTODO DOS NÓS

O método aqui apresentado foi inspirado nas estruturas dos algoritmos destinados a bancos de dados (MARKEZON, 1994, p. 70-85, Cap. 7, Cap 8). O método tem por objetivo construir, observando as considerações *1 a 4,* matrizes de transformação linear que resultem em rápida[167] completude da função.

---

[165] Distância de *Hamming* entre dois vetores *A* e *B* = peso *hamming* de (*A XOR B*); número de bites a_i que diferem de b_i ($i=0,1...,n-1$; *n=tamanho de A e B*);

[166] Esse foi um dos aspectos observados no projeto do DES, por exemplo (SCHNEIER, 1996).

[167] Neste contexto é considerada "rápida" uma avalanche que conduza à completude da função (transformação linear) em um número de iterações menor do que 6 para blocos de tamanho 128. Este valor foi adotado por ter sido mostrado em (LAMBERT, 2003b) que a transformação linear do Rijndael é completa em 7 iterações. A função criptográfica do DES, com blocos de 64 bites e incluindo o efeito das *s-boxes,* é completa em 8 iterações.

7.2.4.1 CONDIÇÕES DE CONTORNO DO PROJETO

O poder de avalanche de uma transformação linear L: $Z_2^n \rightarrow Z_z^n$ que representa a camada linear da função de passo $f:Z_2^n \rightarrow Z_z^n$ de um dado cifrador de blocos iterativo tem um importante relacionamento com o número de iterações. Note que, mesmo se $\Delta h=0$ *(hmin=hmax)*, a *completude* da *transformação linear* não pode ocorrer para $r < log_{hmax}\ n$. Na verdade, mesmo que seja seguida a recomendação feita na *consideração número 3*, para valores usualmente pequenos[168] de Δh, a transformação linear precisará de mais iterações do que $log_{hmin}\ n$. Isso acontece porque, depois da primeira iteração, é praticamente impossível ter um crescimento geométrico dos pesos *Hamming* de cada linha das *r*-ésimas MDR em função de *r*.

De acordo com o que foi observado nos algoritmos estudados, 8 parece ser um limite superior conveniente para o número de coeficientes não nulos por equação de uma TL para argumentos de até 128 bites. Esse valor também é suficientemente grande[169] para produzir a completude da transformação linear em menos de 6 iterações para $n=128$. De fato, como pode ser visto adiante (12.10.3), é possível produzir o resultado desejado com apenas 7 elementos[170] não nulos em cada linha da matriz de dependências $[D_1]$.

Uma vez que a transformação linear do Rijndael (bloco e chave de 128 bites) atinge a sua completude em 7 iterações, consideramos que este é o número máximo que deve ser gasto por uma transformação linear construída pelo *Método dos Nós* para a *completude* da transformação.

Para entender o funcionamento desta técnica, considere a transformação linear $L:Z_2^n \rightarrow Z_2^n$ e suponha que *m* é uma quantidade de bites conveniente para ser utilizada na plataforma em que se pretende implementar o algoritmo.

Um *nó* é uma estrutura de armazenamento e organização de valores que possui $m + 4$ campos. Para construir uma transformação linear $L:Z_2^n \rightarrow Z_z^n$ serão utilizados *n* nós, dispostos em uma árvore, conforme ilustra a FIG. 7.1 , para o caso de $n=16$. Os nós encontram-se

[168] Como por exemplo aqueles encontrados no Rijndael ($\Delta h=2$) e Serpent ($\Delta h=5$).

[169] Em (BAUDRON, 199X), (PATTERSON, 1999) e (SCHNEIER, 1999) podem ser encontradas discussões sobre números de bites utilizados pelas diferentes plataformas. O principal aspecto considerado aqui é que 8 bites ainda é o tamanho dos registradores utilizados em muitos processadores e é suficiente para produzir a avalanche almejada.

[170] Se estes 7 bites forem armazenados em um elemento de 8 bites (um baite ou um registrador), o oitavo bite pode ser utilizado para o resultado da equação (a paridade: *ou-exclusivo* dos 7 bites). Essa propriedade pode ser útil em alguma aplicação.

hachurados e numerados de *0* a *n-1*. A estrutura da árvore é montada de cima para baixo de forma que de cada *nó* N_i saem *m-1* linhas retas descendentes. No final de cada linha está a posição para um novo *nó*.

A quantidade de nós em cada *nível* cresce em progressão geométrica de razão *m-1* até que sejam posicionados todos os *n* nós. Os *níveis* são numerados de *0* (superior, onde se encontra N_0) a *u* (inferior, onde se encontra o nó N_{n-1}).

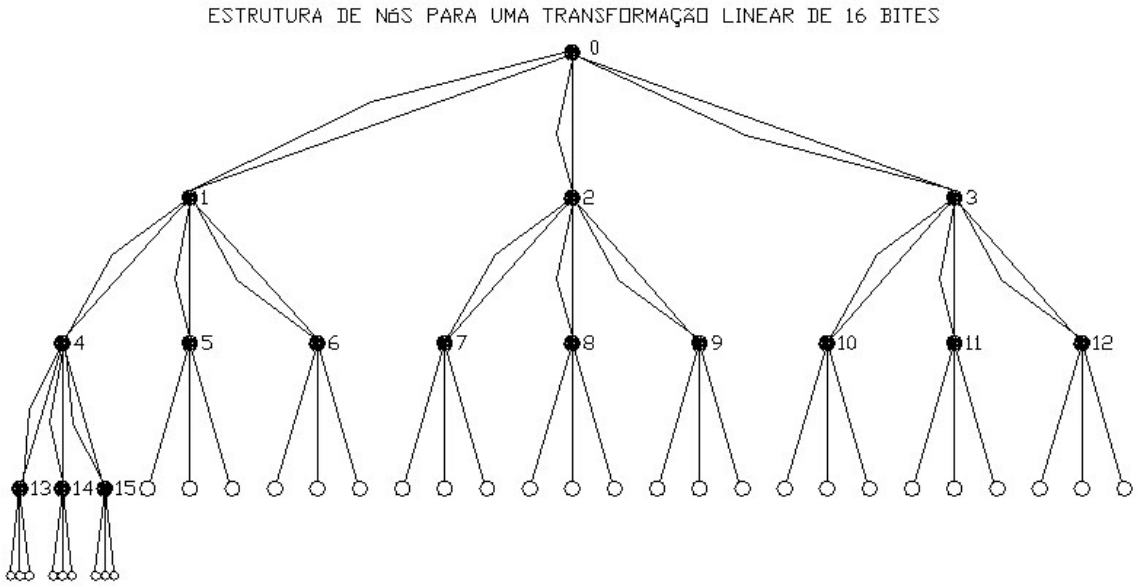


FIG. 7.1 Árvore de 16 nós

A quantidade *Q* de nós nos *x* primeiros níveis é dada pela expressão[171]

$$Q(x) = \begin{cases} [(m-1)^x - 1]/(m-2), \text{ para } x \leq u \\ n, \text{ para } x = u+1. \end{cases}$$

Consequentemente, no nível *y* estão os $(m-1)^y$ nós N_p ,

$[((m-1)^{y+1}-1)/(m-2)] - 1 \leq p < [(m-1)^{y+1}-1]/(m-2) + (m-1)^y$, $y \in Z_u$

e no último nível estão os

$n - [(m-1)^u - 1] / (m-2)$

últimos nós**.**

Cada nível *y* está subdividido em $(m-1)^{y-1}$ *grupos* numerados de *0* a $(m-1)^{y-1}-1$ que, por sua vez, têm *m-1 posições* numeradas de *0* a *m-2*.

[171] As expressões são obtidas a partir da expressões de termo geral e da soma finita das progressões geométricas.

Cada nó N_i dá origem a uma expressão booleana (uma linha da matriz $[D_1]$ da transformação linear) e é um formado por $m+4$ campos. As informações armazenadas em cada campo encontram-se relacionadas na TAB 7.1.

As linhas retas indicam que os campos *filho* do nó ascendente recebem os valores armazenados nos campos *bite* do nó descendente (o "descendente" *afeta* o "ascendente"). As linhas segmentadas indicam que o campo *pai* do nó descendente recebe o valor do campo *bite* do nó ascendente (o "ascendente" afeta o "descendente").

Cada nó N_i dará origem a uma equação da forma:

$$s_{N_{i,3}} = e_{N_{i,4}} \oplus e_{N_{i,5}} \oplus e_{N_{i,6}} \oplus \ldots\ldots \oplus e_{N_{i,m+3}}$$

Em conseqüência, árvore dará origem ao conjunto de n equações que define a matriz de dependências $[D_1]$ da TL.

O algoritmo para construir uma TL utilizando este método consiste dos seguintes passos:

(i) A partir de n e m, calcular o número de níveis, a quantidade de nós em cada nível, a quantidade de grupos em cada nível, o grupo e a posição de cada nó.

(ii) Preencher os campos *nível*, *grupo* e *posição* de cada nó.

(iii) Realizar n escolhas pseudo-aleatórias de valores para preencher os campos $N_{i,3}$. Cada escolha pseudo-aleatória deve selecionar um dos números (bites) menos utilizados[172] dentro do sub-grupo menos utilizado.

(iv) Para $i = 0,1,2...n-1$: - preenche o campo "*pai*" do *nó* N_i, vai ao *nó "pai"* do *nó* N_i e preenche o primeiro campo "*filho*" (ainda não preenchido) com o valor em $N_{i,3}$ *(*lembrar que o "*pai*" do *nó 0* é o nó *n-1*).

Neste ponto estão estabelecidas todas as ligações simbolizadas pelas duplas de linhas acima de cada nó numerado ilustrado na FIG. 7.1. Ou seja, todos os campos "*pai*" estão preenchidos e todos os campos "*filho*" dos nós N_0 a N_4, além do primeiro campo filho do último nó. Resta preencher os campos *"filho"* ainda não preenchidos. A partir deste ponto a escolha do valor de cada filho deverá ser feita com o cuidado adicional[173] de não colocar valores repetidos dentro do mesmo grupo nem em grupos vizinhos. Caso a escolha se torne difícil e não seja possível completar a matriz respeitando o número máximo de m utilizações

[172] Cada vez que um bite for escolhido deve ser computada mais uma utilização para o bite em questão e para o sub-grupo ao qual este bite pertence. Esta técnica de escolha observa as considerações 3 e 4.
[173] Até então, este cuidado era desnecessário.

para cada bite, a restrição pode ser relaxada para escolher valores que ainda não tenham sido utilizados *m* vezes e não se repitam dentro do mesmo grupo.

TAB 7.1: Descrição dos campos dos nós.

Campo	**Informação armazenada**	**Observação**
$N_{i,0}$	*Nível* a que pertence o nó. Assume valores de *0* a *u*.	*Campos auxiliares no algoritmo de construção da TL.*
$N_{i,1}$	*Grupo* a que pertence o nó. Assume valores de $(m\text{-}1)\text{^}(N_{i,0}\text{-}1)\text{-}1$	
$N_{i,2}$	*Posição* do nó dentro do *grupo*. Assume valores de *0 a m-1*.	
$N_{i,3}$	*Bite - Identifica o bite* de saída. Recebe o número da expressão booleana armazenada no nó N_i. Os demais campos $(N_{i,a}\ ,\ a>4)$ armazenam os números das colunas dos elementos iguais a *1* na $(N_{i,3})$-ésima linha da matriz $[D_1]$ de *L*.	*Campos que representam os lados esquerdo e direito da $(N_{i,3})$-ésima expressão booleana. Estes campos assumem valores de 0 a n-1*
$N_{i,4}$	*Pai*. Recebe o campo $N_{k,3}$ do nó ascendente N_k (*k* é o índice do nó ascendente na árvore). Uma exceção é feita para o nó N_0, que recebe em seu campo $N_{0,4}$ o valor do campo $N_{n-1,3}$. Os nós de um mesmo *grupo* têm o mesmo "pai".	
$N_{i,5}$	*Primeiro filho*. Recebe o campo $N_{v,3}$ do primeiro nó descendente N_v (*v* é o índice do nó descendente observado na árvore)	
....	....	
$N_{i,m+3}$	*Último dos (m-1) filhos*. Recebe o campo $N_{(v+m-2),3}$ do primeiro nó descendente N_v (*v* é o índice do nó descendente observado na árvore)	

(v) Preencher os campos "*filho"* ainda não preenchidos.

As dependências se propagam no sentido ascendente (linhas retas na FIG. 7.1 e no sentido descendente da árvore (linhas segmentadas). Em cada iteração cada *"pai*" é afetado por todos os seus *m-1 "filhos*" e *vice-versa*. Em conseqüência, na iteração seguinte, os "*avós"* são afetados pelos "*netos*" e os "*primos*" e "*irmãos*" afetam-se mutuamente. Uma vez que este efeito é cumulativo espera-se que toda a árvore seja afetada por todos os nós em uma quantidade de iterações relativamente menor do que a necessária para as transformações

lineares usualmente encontradas. Ressalta-se também que, apesar de a estrutura apresentada na FIG. 7.1 ser utilizada para construir sistematicamente a TL, essa é apenas uma das árvores presentes, pois cada *nó* é origem de uma árvore e todas elas estão entrelaçadas. Os nós não hachurados foram desenhados na FIG. 7.1 apenas para dar idéia de continuidade.

O algoritmo que executa o *Método dos Nós* foi implementado em Java e encontra-se no apêndice 10. Complementarmente, o programa em Java também mostra as matrizes de avalanche das transformações lineares construídas.

No item 12.10.1 do apêndice 10 pode ser visto o resultado da execução do algoritmo para $n = 16$ e $m = 4$ (exemplo da FIG. 7.1) com um resultado intermediário para o algoritmo executado até o passo iv) e em seguida o resultado após a execução do passo (v). Os valores -1 apresentados no resultado intermediário indicam que o campo ainda não foi preenchido.

No item 12.10.2 é apresentado o resultado da execução do algoritmo para $n = 64$ e $m = 7$ e no item 12.10.3 é apresentado o resultado da execução para $n=128$ e $m=7$.

Em todos os exemplos apresentados a transformação linear é completa para $r > 3$.

7.3 TRANSFORMAÇÕES NÃO-LINEARES – MÉTODO DOS QUADRADOS LATINOS

A seguir é apresentada uma técnica para construção de tabelas de substituição (*s-boxes*) para cifradores de blocos tipo Feistel. A técnica apresentada baseia-se nos *quadrados latinos de lado* 2^n (MASSEY, 1987)[174]. É fornecida também uma demonstração de que, se utilizadas apropriadamente, tais estruturas podem fornecer boa proteção contra as duas formas mais importantes de criptoanálise conhecidas: a criptoanálise diferencial e a criptoanálise linear (DICHTL, 1999)(CASTAÑO, 1998). Como exemplo, é apresentada uma *tabela de substituição tipo quadrado latino* $2^n x 2^n$ $(n = 4)$.

Definição 7.1: *Um* ***quadrado latino de tamanho m x m*** *é uma matriz quadrada de ordem m cujos elementos pertencem a um alfabeto de m símbolos, com a propriedade de que cada símbolo aparece uma e somente uma vez em cada linha e em cada coluna.*

No contexto desta seção, em geral, é utilizada a notação hexadecimal. Quando é utilizada notação decimal, m é uma potência de 2 e o alfabeto considerado é Z_2^m. Quando forem operadas cadeias de n bites, o alfabeto será o *corpo finito CG* (2^n) $(n = \log_2 m)$. Note-se que é apenas uma diferença de notação.

[174] Alguns trabalhos interessantes relacionados aos QL podem ser encontrados em http://www.ciphersbyritter.com.

A FIG. 7.2 mostra um quadrado latino com o alfabeto Z_{16} em notação hexadecimal[175].

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	**6**	**a**	**b**	**8**	**f**	**4**	**d**	**2**	**e**	**3**	**5**	**7**	**1**	**0**	**9**	**c**
1	**4**	**d**	**2**	**e**	**3**	**5**	**7**	**1**	**0**	**9**	**c**	**6**	**a**	**b**	**8**	**f**
2	**5**	**7**	**1**	**0**	**9**	**c**	**6**	**a**	**b**	**8**	**f**	**4**	**d**	**2**	**e**	**3**
3	**C**	**6**	**a**	**b**	**8**	**f**	**4**	**d**	**2**	**e**	**3**	**5**	**7**	**1**	**0**	**9**
4	**F**	**4**	**d**	**2**	**e**	**3**	**5**	**7**	**1**	**0**	**9**	**c**	**6**	**a**	**b**	**8**
5	**3**	**5**	**7**	**1**	**0**	**9**	**c**	**6**	**a**	**b**	**8**	**f**	**4**	**d**	**2**	**e**
6	**9**	**c**	**6**	**a**	**b**	**8**	**f**	**4**	**d**	**2**	**e**	**3**	**5**	**7**	**1**	**0**
7	**8**	**f**	**4**	**d**	**2**	**e**	**3**	**5**	**7**	**1**	**0**	**9**	**c**	**6**	**a**	**b**
8	**e**	**3**	**5**	**7**	**1**	**0**	**9**	**c**	**6**	**a**	**b**	**8**	**f**	**4**	**d**	**2**
9	**0**	**9**	**c**	**6**	**a**	**b**	**8**	**f**	**4**	**d**	**2**	**e**	**3**	**5**	**7**	**1**
a	**b**	**8**	**f**	**4**	**d**	**2**	**e**	**3**	**5**	**7**	**1**	**0**	**9**	**c**	**6**	**a**
b	**2**	**e**	**3**	**5**	**7**	**1**	**0**	**9**	**c**	**6**	**a**	**b**	**8**	**f**	**4**	**d**
c	**1**	**0**	**9**	**c**	**6**	**a**	**b**	**8**	**f**	**4**	**d**	**2**	**e**	**3**	**5**	**7**
d	**a**	**b**	**8**	**f**	**4**	**d**	**2**	**e**	**3**	**5**	**7**	**1**	**0**	**9**	**c**	**6**
e	**d**	**2**	**e**	**3**	**5**	**7**	**1**	**0**	**9**	**c**	**6**	**a**	**b**	**8**	**f**	**4**
f	**7**	**1**	**0**	**9**	**c**	**6**	**a**	**b**	**8**	**f**	**4**	**d**	**2**	**e**	**3**	**5**

FIG. 7.2 Exemplo de quadrado latino

Teoremas

Sejam *A, B, C* três polinômios tais que $A \in CG(2^{\alpha})$, $B \in CG(2^{\beta})$ e $C \in CG(2^{\varphi})$, α,β,φ inteiros positivos. O número de valores distintos que *A* pode assumir é $V_A=|CG(2^{\alpha})|= 2^{\alpha}$. Analogamente, $V_B = |CG(2^{\beta})|=2^{\beta}$ e $V_C = |CG(2^{\varphi})| = 2^{\varphi}$.

Seja $\Delta=\alpha+\beta+\varphi$ e seja *D* um polinômio de $CG(2^{\Delta})$ tal que $D =2^{\beta+\varphi}.A+2^{\varphi}.B+C$. O número de valores possíveis de *D* é o produto $V_A.V_B.V_C = 2^{\alpha+\beta+\varphi} = 2^{\Delta} = |CG(2^{\Delta})|$. Assim se $A=a_0a_1a_2a_3...a_{\alpha-1}$ $B= b_0b_1b_2b_3...b_{\beta-1}$ e $C=c_0c_1c_2c_3...c_{\varphi-1}$, temos que $D = a_0a_1a_2a_3...a_{\alpha-1}\ b_0b_1b_2b_3...b_{\beta-1}\ c_0c_1c_2c_3...c_{\varphi-1}= A|B|C$.

Teorema 7.1: *Fixados* β *e um polinômio* B_1 *e escolhido aleatoriamente um polinômio D,* $D \in CG(2^{\Delta})$, *a probabilidade de B ser igual a* B_1 *é* $p(B=B_1)=1/2^{\beta}$ *para todo* B_1, $B_1 \in CG(2^{\beta})$.

Prova: Para calcular a probabilidade de que trata o teorema 1, escolhido ao acaso um polinômio D de $CG(2^{\Delta})$, *será chamado de "sucesso" o evento em que* $B=B_1$. *Para cada valor de B existem* $V_A.V_C = 2^{\alpha+\varphi}$ *polinômios distintos em* $CG(2^{\Delta})$. *Desta forma, o número de eventos que representam um sucesso é* $2^{\alpha+\varphi}$. *O número total de escolhas possíveis é a cardinalidade*

[175] Cada elemento de Z_{16} tem exatamente um polinômios correspondente em $CG(2^4)$.

de $CG(2^{\Delta})=2^{\Delta}$. A probabilidade é dada pelo quociente entre o número de sucessos pelo número total de eventos possíveis. Logo,

$$p(B=B_1) = 2^{\alpha+\varphi}/2^{\Delta} = 2^{\alpha+\varphi}/2^{\alpha+\beta+\varphi} = 1/2^{\beta}\text{ , C.Q.D.}$$

Teorema 7.2: Sejam as funções $f(x)$: $CG(2^m) \rightarrow CG(2)$ e $g(y)$: $CG(2^n) \rightarrow CG(2)$. Seja a função $h(x,y)$: $\{CG(2^m)\ x\ CG(2^n)\} \rightarrow CG(2)$, $h(x,y) = f(x) \oplus g(y)$, uma função de $CG(2^{m+n}) \rightarrow CG(2)$. *Se g é uniformemente distribuída*[176] *sobre CG(2) e x não tem correlação com y, h é uniformemente distribuída sobre CG(2).*

Prova: Seja x_i um polinômio em $CG(2^m)$, $i \in \{0,1,2,...2^m-1\}$. Por definição de função (injetora), para cada x_i existe apenas um elemento w em $CG(2)$ tal que $f(x_i) = w$. Em conseqüência, para toda função $f: CG(2^m) \rightarrow CG(2)$ existem z_f valores distintos de i tais que $f(x_i)=0$ e u_f valores distintos de i tais que $f(x_i)=1$, $u_f+z_f=2^m$. A probabilidade de obter-se $f(x)=1$ escolhendo-se x ao acaso é dada por $p(f(x)=1)= u_f/(u_f+z_f)$. Se $u_f = z_f$, temos que $p(f(x)=1)=p(f(x)=0)=1/2$ e f é *uniformemente distribuída* sobre $CG(2)$. As possíveis entradas distintas de $h(x, y)$ são os pares (x_i, y_j), $i \in \{0,1,2,...2^m-1\}$, $j \in \{0,1,2,...2^n-1\}$. Existem portanto 2^{m+n} entradas possíveis. Ou seja, para cada valor de i, existem 2^n pares (x_i, y_j) distintos. Por hipótese, $g(y)=1$ para 2^{n-1} valores de y e $g(y)=0$ para 2^{n-1} valores de y. Se $f(x_i)=1$, temos que $h(x_i, y_j)=1+g(y_j)$. Uma vez que $g(y_j)=1$ para metade (2^{n-1} valores) dos possíveis valores de j e $g(y_j)=0$ para a outra metade, pode-se afirmar que para cada valor de i tal que $f(x_i)=1$, $h(x_i, y_j)$ assumirá o valor 0 para metade dos possíveis valores de y_j e $h(x_i, y_j)$ assumirá o valor 1 para a outra metade dos valores. Esta afirmação permanece válida para $f(x_i)=0$ e para todos os valores de i.

O número de pares (x,y) que resultam em $h(x,y) = 1$ é portanto $u_h = 2^m \cdot 2^{n-1}$. Analogamente, número de pares (x,y) que resultam em $h(x,y) = 0$ é $z_h = 2^m \cdot 2^{n-1}$. Logo,

$p(h(x,y)=1)= p(h(x,y)=0) = 2^m \cdot 2^{n-1}/(2^m \cdot 2^{n-1}+2^m \cdot 2^{n-1}) = ½$ e *h é uniformemente distribuída sobre CG(2), CQD.*

[176] Todos os valores da imagem são equiprováveis, pois têm a mesma frequência de ocorrência.

7.3.1 SUGESTÃO DE TRANSFORMAÇÃO NÃO-LINEAR UTILIZANDO QUADRADOS LATINOS

Seja a TNL $N:Z_2^8 \rightarrow Z_2^4$ uma *substituição* utilizando um QL como o da FIG. 7.2. A transformação é feita por baites, da seguinte maneira: seja o baite $A=x_h|y_h$. O resultado $N(A)$ da *substituição* é encontrado na linha x_h e coluna y_h da caixa. A operação não é invertível.

Exemplo: Para o baite $A=01011011_2 = 5b_h$, temos $x = 5$ e $y = b$. Logo, $N(A) = f_h = 1111_2$.

7.3.1.1 PROPRIEDADES DAS TNL COM QL

As TNL usando QL têm as seguintes propriedades:

1ª Propriedade:

Fixado x_h , y_h se relaciona biunivocamente com $N(A)$. Logo, se todos os valores de y_h são equiprováveis, todos os valores de $N(A)$ são equiprováveis. Analogamente, fixado y_h, se todos os valores de x_h são equiprováveis, todos os valores de $N(A)$ são equiprováveis (decorre da definição 7.1 e do teorema 7.1). Em conseqüência, para que $N(A)$ seja uniformemente distribuída, é suficiente que x_h ou y_h seja uniformemente distribuído.

2ª Propriedade:

Fixado o valor de $N(A)$, todos os valores de A são equiprováveis (decorre da primeira propriedade).

3ª Propriedade:

Seja B' um bloco de bites escolhidos de $N(A)$. Fixado B', todos os valores de A são equiprováveis[177](decorre da segunda propriedade e do teorema 7.1).

4ª Propriedade:

Sejam A' e B' dois blocos de bites escolhidos de A e $N(A)$, respectivamente. Fixado B', todos os valores de A' são equiprováveis[178] (decorre da terceira propriedade e do teorema 7.1).

[177] A recíproca não é verdadeira.
[178] A recíproca não é verdadeira.

5ª Propriedade:

A transformação *N* é *completa*[179] (decorre da terceira propriedade).

A quarta propriedade garante que não é possível determinar o valor de qualquer bite ou conjunto de bites da entrada a partir da saída[180].

A *quinta propriedade* garante que *N* pode contribuir para o *efeito avalanche* em um algoritmo criptográfico.

Uma vez que as recíprocas da terceira e da quarta propriedades não são verdadeiras, nem todas as *correlações* em uma TNL tipo QL são nulas. Em geral as TNL tipo *QL não garantem imunidade contra a criptoanálise linear.*

As propriedades apresentadas não são extensíveis às relações entre diferenças na entrada e diferenças na saída e em geral as TNL tipo *QL não garantem imunidade contra a criptoanálise diferencial.*

Toda a apresentação neste tópico em relação à TNL $N:Z_2^8 \rightarrow Z_2^4$ vale para a TNL $N:Z_2^m \rightarrow Z_2^{m/2}$, sem perda de generalidade (com *m* par).

7.3.1.2 TNL COM BOAS CARACTERÍSTICAS DIFERENCIAIS E LINEARES

Os QL apresentam a propriedade de que trocando-se duas linhas quaisquer ou duas colunas quaisquer entre si, a estrutura resultante também é um QL. Se o procedimento for repetido aleatória e indefinidamente será gerada uma grande quantidade de QL distintos (para $m>7$ e $N:Z_2^m \rightarrow Z_2^{m/2}$ existem $3 \times 2^{m/2}!$ QL distintos, no mínimo)[181].

Sejam $N_1, N_2,...N_z$ *z* TNL de $Z_2^m \rightarrow Z_2^{m/2}$ e sejam $D_1, D_2,...D_z$ as respectvas matrizes de distribuição de diferenças das TNL. Seja *j* o valor do baite que representa a diferença entre dois argumentos *X* e *X'*. Para cada j $(j \in Z^m)$ existem 2^{m-1} pares (X, X') tais que $X \oplus X' = j$. Cada elemento $d_{p,i,j}$ representa o número de vezes que a diferença $N_p(X) \oplus N_p(X') = i$. A probabilidade de ocorrer a diferença *i* na saída dada uma diferença *j* na entrada é dada pelo quociente $\pi_{pij}=d_{pij}/2^{m-1}$. Seja $\pi_{p,máx}$ o maior valor π_{pij} encontrado na matriz D_p. Reduzir o valor

[179] Logo, a transformação *N* satisfaz a HNC.
[180] Isso não se verifica no DES, apesar da *substituição* não ser bijetora (vide apêndice 6).
[181] A primeira coluna do QL pode ser qualquer das $2^{m/2}!$ permutações distintas do alfabeto. As colunas seguintes podem ser geradas recursivamente por *d* rotações ciclicas da anterior. Existem pelo menos três valores de d que geram QL distintos: *d=1, d=2, d=3.*

de $\pi_{p\text{-}max}$ é o objetivo quando se pretende aumentar a resistência da TNL N_p contra a criptoanálise diferencial.

Seja N_3 a TNL definida pelo uso da TNL N_1 ou da TNL N_2, com escolha aleatória a cada iteração. A probabilidade de ocorrer uma diferença i na saída para uma diferença na entrada j em N_3 é dada pela expressão $\pi_{3ij} = \pi_{1ij} \,.\, ½ + \pi_{2ij} \,.\, ½ = ½(\pi_{1ij} + \pi_{2ij})$. Uma vez que, em geral $\pi_{1máx}$ e $\pi_{2máx}$ ocorrem para pares (i,j) distintos[182] $\pi_{3máx}$ será menor que o maior entre $\pi_{1máx}$ e $\pi_{2máx}$. Ou seja, é possível encontrar associações de QL que sejam mais resistentes contra a CD do que cada um dos QL individualmente.

O resultado também vale para a matriz de distribuição linear.

Foram construídas as matrizes de distribuição linear e diferencial para mais de 9700 QL com alfabeto hexadecimal (Z_2^4) gerados aleatoriamente. Em seguida foram selecionados os 10 QL com melhores[183] matrizes de distribuição diferencial e linear. Aparentemente, existe um limite inferior de 1/8 para do desvio máximo | p - ½ | nas distribuições lineares de TNL com entrada de 8 ou 4 bites e saída de 4 bites. Este foi o menor valor de desvio máximo encontrado em todos os mais de 9700 QL analisados. Para o Serpent este valor é ¼ (e 1/8 para equações que só envolvem um bite em cada membro). Para o DES este valor é 5/16 (CASTAÑO, 1998, p. 17, 109). Para o Rijndael (com saída de 8 bites), temos os menor valor de |p-1/2| máximo que é 16/256=1/16 (vide capítulo 4, item 4.5.7).

Os dez melhores QL encontrados, bem como os resultados de distribuição linear e diferencial encontram-se no apêndice 11.

Os QL com melhor resultado em relação à CD apresentaram RS/R=(1/4)/(1/16) = 4 (mesma do Rijndael).

Foram testadas[184] associações de 4 QL. Um dos resultados conseguidos foi, por exemplo, RS/R= 0.15625/(1/16)= 2,5 e |p-0.5|= 0.0625. Os QL utilizados nesta associação estão no apêndice 11 (item 12.11.3). Esses valores são melhores que os encontrados para a tabela de substituição do AES. Esta TNL utilizando QL não foi utilizada no cifrador *Alpha* porque o código ficou muito lento (em Java).

[182] Pode ser visto no apêndice 11, item 12.11.1.2, que mostra resumo da análise de 10 tabelas individualmente.
[183] Foram consideradas melhores as matrizes com menor valor máximo de |p-1/2| e RS/R.
[184] Considerando um dos 4 QL escolhido aleatoriamente (pela chave) em cada fase. Os primeiros 4 bites da entrada definem a linha e os últimos 4 bites definem a coluna. Saída com 4 bites.

7.4 CONCLUSÃO DO CAPÍTULO

O *método dos nós* pode ser utilizado para construir sistematicamente TL que apresentem boa difusão, produzindo a completude da função criptográfica em um número de iterações suficientemente pequeno (menor que os das TL do AES, do Serpent e do DES). A TL (128 bites com 7 coeficientes por equação) construída por este método é completa em apenas 4 iterações (a TL do AES é completa em 7 passos, a do Serpent em 6).

Os QL apresentam propriedades que podem ser úteis para a construção de TNL para sistemas tipo Feistel. A associação de QL pode ser útil para melhorar as características diferenciais e lineares da TNL.

No capítulo 8 é apresentado um cifrador cujo projeto baseia-se em tudo que já foi apresentado até este ponto do trabalho.

8 PROPOSTA DE CIFRADOR SIMÉTRICO DE BLOCOS

8.1 INTRODUÇÃO

Neste capítulo é apresentada a descrição do cifrador *Alpha*. O cifrador *Alfa* é o resultado do estudo realizado nos capítulos precedentes. O *Alpha* é uma estrutura Feistel equilibrada com a característica adicional de receber em cada iteração uma sub-chave do mesmo tamanho do bloco de texto a ser transformado[185]. No apêndice 8 é apresentada uma implementação[186] em Java, incluindo uma execução passo a passo do algoritmo, que é útil para ilustrar o efeito de cada transformação criptográfica e testar implementações.

8.2 ESPECIFICAÇÕES

Os parâmetros do *Alfa* são os seguintes:

Tamanho do bloco: $n = 128$ *bites*

Número de iterações: Nr ($Nr \geq 16$)

Tamanho da chave: t (t é múltiplo de 32, $128 \leq t \leq 128.Nr$)

Função de passo: $f:Z_2^{64} \rightarrow Z_2^{64}$, tipo *SLTN*.

Expansão da chave: $g:Z_2^{t} \rightarrow Z_2^{128.Nr}$, tipo *SLTN*.

O cifrador pode ser implementado com os parâmetros t e Nr fixos ou fornecidos pelo usuário. Se $t = Nr.128$, todas as sub-chaves são independentes.

A estrutura do algoritmo e as transformações criptográficas envolvidas são descritas nos tópicos seguintes.

A FIG. 8.1 mostra o fluxograma do cifrador *Alpha*. Cada sub-chave K_i de *128* bites é dividida em dois blocos de 64 bites Ke_i e Kd_i tais que $K_i = Ke_i|Kd_i$.

[185] Essa característica foi uma decorrência da idéia de somar bloco de chave ao bloco R_i quando este é transportado para L_{i+1}. Essa idéia foi apresentada pelo professor Paz de Lima e não foi encontrada na bibliografia consultada.

[186] Com chave e bloco de 128 bites

8.2.1 ESTRUTURA DO CIFRADOR ALPHA

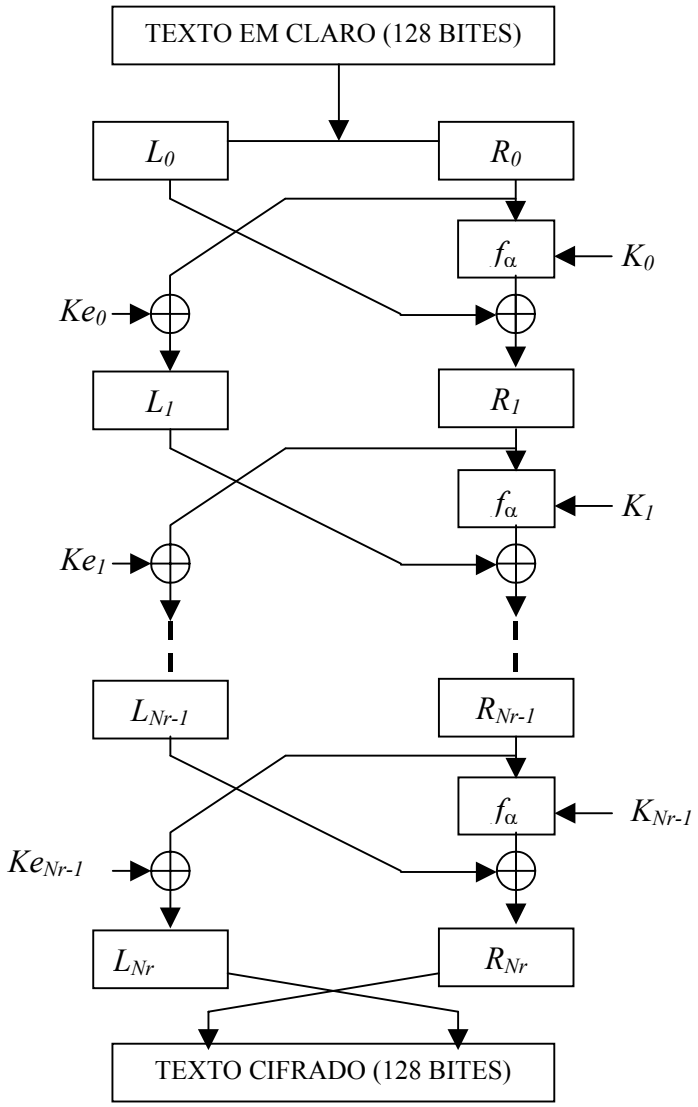


FIG. 8.1– Fluxograma do Cifrador *Alpha*

Para cifrar (ou decifrar) um bloco de 128 bites, o *Alpha* realiza as 3 seguintes etapas:

1- Divide o bloco de texto em claro *M* em duas metades, L_0 e R_0.

2- Realiza *Nr* iterações calculando L_i e R_i, definidos pelas expressões recursivas *8-I* e *8-II* ($i = 1,2,3...Nr$).

3- Obtém o texto cifrado[187] $C=R_{Nr}|\ L_{Nr}$

[187] A troca dos lados L e R é necessária para a decifração (Feistel).

$$Ri = Li\text{-}_1 \oplus f_\alpha(R_{i-1}\ ,\ K_{i-1}) \quad (8\text{-I})$$

$$Li = R_{i-1} \quad \oplus Ke_i \quad (8\text{-II})$$

8.2.2 A FUNÇÃO DE PASSO f_α

A função de passo f_α: $Z_2^{64} x\ Z_2^{128} \rightarrow Z_2^{64}$ do cifrador *Alpha* é composta por *uma ou duas*[188] *somas módulo dois de 64 bites, uma TL denominada* $\underline{L}_\alpha$ *e uma TNL* N_α. A função f_α é definida pelas expressões 8-III e 8-IV.

$$f_{\alpha c}\ (X\ ,\ K) = N_\alpha\ (\underline{L}_\alpha(X) \oplus Kd)) \qquad (8\text{- III})$$

$$f_{\alpha d}\ (X\ ,\ K) = N_\alpha\ (\underline{L}_\alpha(X \oplus Ke) \oplus Kd) \qquad (8\text{- IV})$$

Onde $X \in Z_2^{64}$ e $K \in Z_2^{128}$ tal que $K = Ke|Kd$, $Ke \in Z_2^{64}$ e $Kd \in Z_2^{64}$. As transformações $\underline{L}_\alpha$ *e* N_α são descritas nos sub-itens 8.2.2.2 e 8.2.2.3. Se for definida uma variável lógica ρ , tal que $\rho = 0$ para cifrar e $\rho = 1$ para decifrar, f_α pode ser definida por uma única expressão para cifrar e decifrar, a expressão 8-V.

$$f_\alpha\ (X\ ,\ K) = N_\alpha\ (\underline{L}_\alpha(X \oplus (Ke\ \hat{}\ \rho^{64})) \oplus Kd) \qquad (8\text{- V})$$

A FIG. 8.2 ilustra a função de passo para cifrar $\boldsymbol{f}_{\alpha c}$ e para decifrar $\boldsymbol{f}_{\alpha d}$ aplicadas na *i-ésima* iteração do cifrador *Alpha*.

8.2.2.1 AS SOMAS MÓDULO DOIS

Representam a operação lógica "*ou exclusivo*" bite a bite dos argumentos.

8.2.2.2 A TRANSFORMAÇÃO LINEAR L_α

A transformação linear é definida pela matriz $[\underline{L}_\alpha]$ apresentada no apêndice 8, no item 12.8.1. A matriz $[\underline{L}_\alpha]$ define o conjunto de equações utilizadas para a implementação no

código fonte apresentado em 12.8.3. Considerações sobre o *poder de difusão* desta TL, que foi construída pelo *método dos nós*, são feitas no capítulo 9.

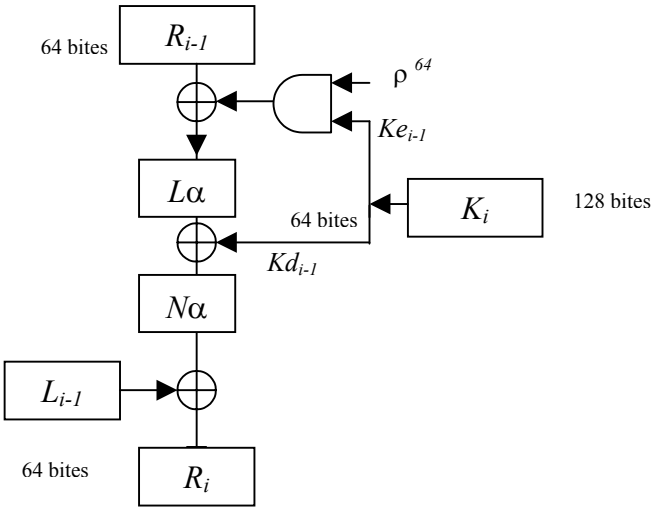


FIG. 8.2 Função f_α para cifrar e para decifrar

8.2.2.3 A TRANSFORMAÇÃO NÃO-LINEAR N_α

A transformação N_α: $Z_2^{64} \rightarrow Z_2^{64}$ é realizada em oito etapas, uma para cada baite. Cada etapa é uma transformação de Z_2^{12} *sobre* Z_2^8. Sejam $X=X0|X1|...|X7$ $(X \in Z_2^{64})$ e $Y=Y0|Y1|...|Y7$, $(Y \in Z_2^{64})$ onde Xi é o *i-ésimo* baite de X e Yi é o *i-ésimo* baite de Y e $Y = N_\alpha(X)$. Sejam os baites de X da forma $Xi = x_i|x'_i$, (x_i e x'_i *pertencem a* Z_2^4). Cada baite Yi (i=1,2,3..6,7) de Y é encontrado na linha $(x_i + x_{i-1}) \bmod 16$ e na coluna x'_i da tabela[189] na FIG. 8.3. O baite $Y0$ é encontrado na linha $(x_0+x_7) \bmod 16$ e na coluna x'_0 da mesma tabela. Por exemplo, N_α(`5a4f953f fa3f4fb7` $_h$) = `67db038a e515d268`$_h$. O baite 67_h é encontrado na linha $0 = (5+b_h) \bmod 16$ e coluna a_h. O baite db_h é encontrado na linha $9 = (4+5) \bmod 16$ e coluna f_h, e assim por diante.

---

[188] Uma para cifrar e duas para decifrar.

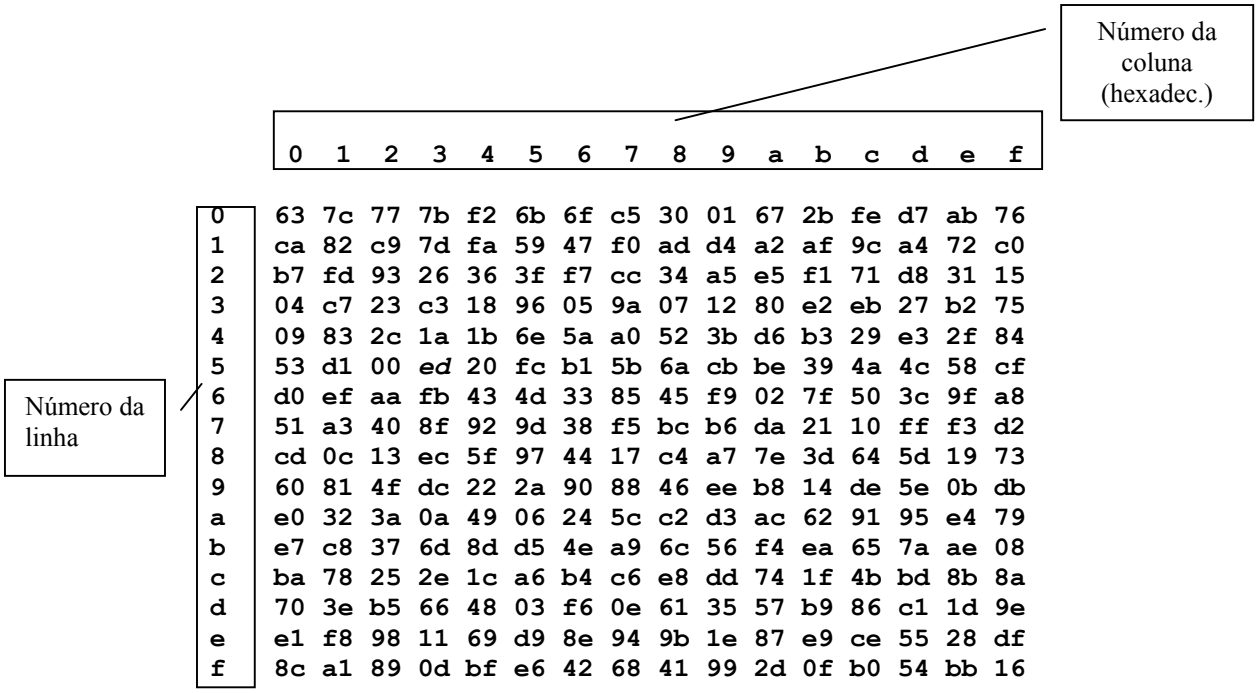


FIG 8.3 Tabela de substituição do cifrador *Alpha*

8.2.3 A FUNÇÃO DE EXPANSÃO DE CHAVES

O algoritmo *Alpha* foi concebido para utilizar *Nr* sub-chaves de fase independentes, o que ocorre quando o usuário fornece os $t = 128 \times Nr$ bites. Se for julgado conveniente possibilitar o uso de chaves menores, pode ser utilizada a mesma função de expansão de chaves do Rijndael (apêndice 1) para gerar as $((128 \times Nr)-t)/32$ palavras que não forem fornecidas pelo usuário[190]. A implementação que foi feita como exemplo, encontrada no apêndice 8, adota chave de 128 bites. Mais detalhes sobre a função de expansão de chaves podem ser vistos em (DAEMEN, 1999).

[189] A tabela na FIG. 8.3 é a mesma utilizada no AES (FIPS 197, 2001). Entretanto, para cada baite de saída existem 16 possíveis baites de entrada.
[190] Caso se deseje implementar o algoritmo com mais de 16 iterações devem ser inseridos os valores das demais constantes de fase na função de expansão de chaves.

8.2.4 SEGURANÇA

8.2.4.1 BUSCA EXAUSTIVA DA CHAVE

A complexidade média esperada da busca exaustiva da chave no *Alpha* é no mínimo 2^{127} (para chaves com128 bites)[191]. O limite inferior considerado atualmente para a segurança de informações não classificadas é 90 bites (BLAZE, 1995).

8.2.4.2 RESISTÊNCIA CONTRA AS CRIPTOANÁLISES DIFERENCIAL E LINEAR

As matrizes de distribuição diferencial e linear da tabela de substituição do *Alpha* são as mesmas do AES (a tabela é a mesma).

A difusão produzida pela TL é tal que em uma fase *r* cada *bite*[192] afeta ***pelo menos*** quatro baites na fase *r+1*. Na fase *r+2* esses quatro baites afetam os oito baites da metade direita do bloco e quatro baites do lado esquerdo. Ainda, cada baite afeta pelo menos 4 baites do lado direito. Isso pode ser visto na MDR e MDE nos apêndices 8 (item 12.8.1) e 9. Por basear-se em um sistema Feistel (vide 9.3.1), a função criptográfica do *Alpha* necessita de quatro iterações para ser completa[193].

Para o Rijndael, o limite inferior de 10 fases foi definido com base no melhor atalho encontrado, que explora trilhas de propagação de diferenças de 4 fases para atacar implementações reduzidas com até 6 fases. Os autores adotaram então mais 4 fases como margem de segurança, o que equivale a adicionar uma *difusão completa* antes e outra depois das 6 fases atacadas (DAEMEN, 1999, p. 29). A transformação criptográfica do AES-128 é completa em 2 fases, como pode ser visto no apêndice 4, na segunda MDE. Uma vez que a transformação criptográfica do *Alpha* gasta o dobro do número de fases que a do Rijndael para se tornar completa, foi suposto que pode ser encontrada uma trilha de propagação para

[191] Se forem processadas 1.000.000.000 de mensagens por segundo, serão necessários 5,4 . 10^{21} anos em média para encontrar uma chave. Esse tempo é mais de 1000 vezes maior do que o quadrado do tempo necessário ao para o Sol tornar-se uma *nova* (10^9 anos (SCHNEIER, 1996, p. 18)).

[192] No AES esta afirmação é feita em relação a *cada baite* (DAEMEN, 1995, cap. 5)(DAEMEN, 1999)(CASTAÑO, 1998). Aqui, entretanto, a afirmação, refere-se ao lado direito do bloco apenas. Esse fato não enfraquece o argumento, uma vez que esse lado tem necessariamente que ser usado nas equações para a criptoanálise diferencial e linear em um sistema Feistel, mesmo que de três fases apenas.

[193] A TL (em 12.8.1), juntamente com a TNL, produz a completude do semibloco direito em 2 iterações, como pode ser visto na MDE em 12.9.2.2, observando o quadrante superior direito da matriz (nas MDE os lados são trocados na última iteração). Por ser um sistema Feistel, a completude do bloco de 128 bites se dá em 4 iterações (item 12.9.2.4).

até 8 fases[194]. Em seguida, foram adicionadas mais 8 fases, o que equivale a duas difusões completas, uma no início e outra no final de uma transformação de 8 fases[195].

8.2.4.3 CRIPTOANÁLISE DIFERENCIAL DE POTÊNCIA E DE TEMPO (DPA E DTA)

Estes tipos de ataque não foram analisados especificamente.

As operações de busca em tabela, deslocamentos fixos e operações booleanas são as menos vulneráveis a estes tipos de ataque. Todas as transformações no *Alpha* utilizam apenas estes três tipos de operação. Os exemplos práticos encontrados na bibliografia referem-se apenas às vulnerabilidades teóricas da função de expansão de chaves em *hardware*(cartões inteligentes, por exemplo), especificamente quando é utilizada a técnica de *clareamento*[196]. Aparentemente, o ataque não é viável para implementações em outras plataformas ou se forem utilizadas sub-chaves de fase independentes (KOCHER, 1996), (KOCHER, 199X), (NECHVATAL, 1999, p. 20, 22).

8.2.4.4 CHAVES FRACAS E SEMIFRACAS

A não-linearidade e a avalanche da função de expansão de chaves praticamente elimina a possibilidade de existência de chaves fracas ou chaves semifracas que possam ser exploradas por um criptoanalista.

8.2.5 EFICIÊNCIA

Em relação ao espaço de memória requerido para o código, o *Alpha* exige apenas o armazenamento de uma estrutura, uma vez que apenas uma estrutura é suficiente para cifrar e decifrar. Se forem empregadas até duas tabelas do AES ou até quatro QL, o espaço exigido para o armazenamento de tais tabelas não é maior do que o exigido no caso do AES.

---

[194] Essa consideração é conservadora, pois apesar de a função Feistel ser completa em 4 iterações, pode ser visto no item 9.3.1 que as dependências nos quadrantes indicam equações de 2 fases (1 quadrante), 3 fases (dois quadrantes) e 4 fases (1 quadrante). No AES a completude é atingida em 2 fases, mas cada bite é resultado de uma equação que envolve apenas duas fases.

[195] Poderia ser considerada a possibilidade de utilizar doze iterações, o que eqüivaleria ao acréscimo de uma difusão completa a uma transformação de 8 fases. Essa opção foi preterida a fim de aumentar a margem de segurança, de manter forma similar ao que foi feito no AES(DAEMEN, 1999).

[196] *Whitenning*: técnica que consiste em adicionar uma sub-chave no bloco a ser cifrado antes da primeira e depois da última fase. Como esta operação é fácil de identificar no tempo de processamento, cria vulnerabilidade à DPA. Por esta razão optou-se por adicionar mais 64 bites ao lado esquerdo do bloco em cada iteração, o que tem um efeito similar ao *clareamento* em relação à iteração seguinte.

Em relação ao AES, na operação de decifrar, convém observar que a matriz que define a TL apresenta de 11 a 19 coeficientes não nulos para cada uma das 128 equações contra os 7 coeficientes não nulos para cada uma das 64 equações do *Alpha*.

As operações no *Alpha* são todas operações rápidas (que consomem poucos ciclos) e a quantidade de operações realizadas por iteração é menor do que no caso do AES e do *Serpent*, uma vez que a função f_α opera apenas metade do bloco de 128 bites. *A priori*, não há razão para esperar que o algoritmo seja significativamente mais lento que o AES[197]. Mesmo realizando um número maior de iterações, em cada iteração apenas meio bloco é operado pela função de passo.

Espera-se que a TL seja realmente muito rápida em *hardware*, uma vez que pode ser realizada por um circuito combinacional.

8.2.6 FLEXIBILIDADE

Versões do cifrador *Alpha* podem ser construídas para cifrar blocos de 256 (ou qualquer múltiplo de 16 bites). Para isso, basta utilizar o *Método dos Nós* (cap. 7) para construção de uma TL de tamanho adequado (metade do bloco). Para o caso de 256 bites pode ser utilizada a TL apresentada no item 12.10.3 do apêndice 10 ou a própria TL do AES. Versões com blocos maiores que 128 bites não foram testadas, embora não seja conhecida razão para não utilizá-las.

Não foi encontrada qualquer[198] tabela de substituição com características diferenciais e lineares melhores do que as da tabela de substituição do AES (adotada no *Alpha*). Entretanto, o *Alpha* pode ser implementado com qualquer TNL que mapeie *n/2* bites em *n/2* bites ou *n* bites em *n/2* bites (neste caso, também são utilizados os bites de Ke_i em cada iteração).

Como foi visto no capítulo 7, podem ser utilizadas associações de tabelas ao invés de uma só tabela, o que praticamente inviabiliza as criptoanálises diferencial e linear, se forem escolhidas tabelas de um conjunto suficientemente grande. Se a TNL for tal que utiliza dois QL (não necessariamente distintos, vide FIG. 8.4) em cada iteração e estes QL forem escolhidos aleatoriamente do conjunto de 10 QL apresentado no apêndice 11, haverá 10^{2Nr} combinações possíveis (neste caso N_α: $Z_2^{128} \rightarrow Z_2^{64}$).

[197] O AES é mais rápido que o Serpent.
[198] Foram pesquisadas apenas tabelas com entradas de 8 bites e saídas de 4 ou 8 bites.

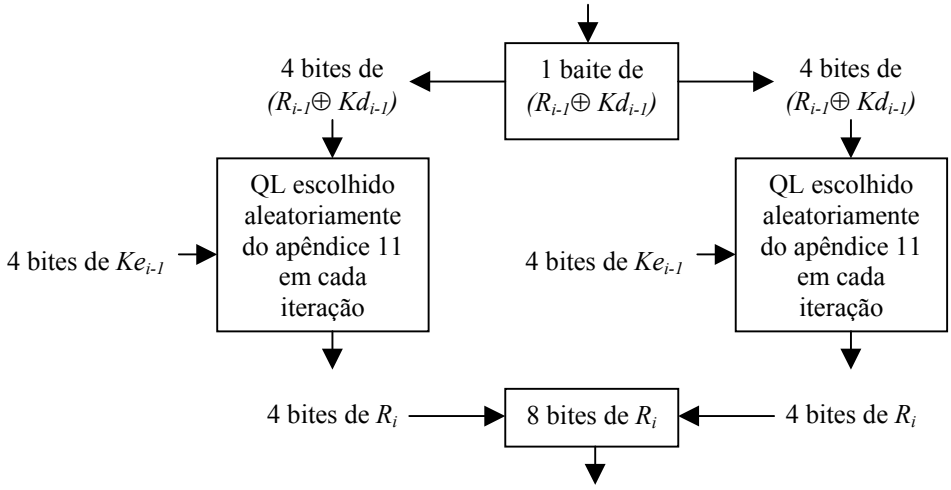


FIG. 8.4 Uma TNL usando associação de QL

8.2.7 SIMPLICIDADE

O *Alpha* apresenta uma descrição bastante semelhante à do DES, baseada em uma estrutura exaustivamente estudada há mais de 30 anos. Todas as operações envolvidas usam poucos ciclos de processamento.

8.2.8 CREDIBILIDADE

O cifrador *Alpha* utiliza uma consagrada estrutura Feistel, com a propriedade adicional de utilizar sub-chaves do tamanho do bloco e chave de qualquer tamanho (até *128.Nr*). É pouco esperado, portanto, o surgimento de alguma forma de ataque aplicável ao *Alpha* mais poderosa que a CD e a CL.

Em relação à CL e à CD, a tabela de substituição utilizada apresenta características diferenciais e lineares superiores às do DES e tão boas quanto as do AES, com a propriedade adicional de não ser invertível. Além disso, é menos esperado o surgimento de um possível ataque algébrico[199] que se baseie nas estruturas dos corpos finitos do que no caso do AES. A transformação linear do *Alpha* (considerando as diferenças impostas pela estrutura Feistel) tem uma taxa de propagação de diferenças (difusão) no mínimo tão boa quanto a do AES, sobretudo considerando o maior número de iterações.

[199] Como os apresentados em (OSWALD, 2002) e (COURTOIS, 2002).

Todas as escolhas realizadas no projeto foram justificadas ao longo deste trabalho e a estrutura do *Alpha* é suficientemente simples para que seja extremamente difícil ocultar um alçapão ou atalho.

8.3 VALIDAÇÃO

Alguns dos principais testes de aleatoriedade recomendados pelo NIST e pelo NESSIE estão apresentados no capítulo 9.

8.4 CONCLUSÃO DO CAPÍTULO

Neste capítulo foi apresentada uma cifra de blocos simétrica com uma estrutura Feistel aprimorada. A cifra foi projetada segundo os princípios de projeto preconizados no capítulo 6.

As transformações criptográficas presentes na cifra foram projetadas para ser, no mínimo, tão resistentes contra as criptoanálises linear e diferencial quanto as transformações do AES. O número de 16 iterações foi especificado para garantir uma transformação criptográfica no mínimo tão poderosa quanto a do AES.

A cifra utiliza apenas operações que envolvem poucos ciclos de processamento. É portanto esperado que o desempenho do algoritmo seja bom nas implementações em *software* e principalmente em *hardware*.

O algoritmo pode ser modificado se for considerado conveniente. Tanto a TL quanto a TNL presentes no algoritmo podem ser substituídas por outras similares, desde que as transformações substitutas apresentem boa difusão e boas características diferenciais e lineares. Uma TNL alternativa, baseada nas propriedades dos quadrados latinos foi apresentada.

9 ALGUNS TESTES PARA ALGORITMOS CRIPTOGRÁFICOS

9.1 INTRODUÇÃO

Um aspecto consensualmente aceito como um dos mais importantes é a análise das medidas relativas à *difusão*, que avaliam o efeito *avalanche*, o *critério de avalanche estrito* e a *completude* da função criptográfica para versões com número de passos reduzido e com os números de passos especificados para as cifras (PRENEEL, 2000, 4, 24)(OSWALD,2002).

Outra parte importante da análise de qualquer cifra de blocos é a resistência às principais formas de criptoanálise conhecidas, sobretudo a CD e a CL (OSWALD, 2000).

Além disso, o NIST, o NESSIE e o RACE utilizam ferramentas para *rejeitar* ou *não rejeitar* estatisticamente as cifras de blocos. O resultado positivo destes testes é da forma "*os resultados dos testes estatísticos aplicados no algoritmo X não indicam qualquer desvio em relação a um comportamento aleatório*", e não permite fazer qualquer outra afirmação sobre a segurança da cifra analisada (PYKA, 2001). Um resultado negativo, entretanto, compromete seriamente a segurança da cifra analisada.

Este capitulo é baseado em (DICHTL, 1999), (PYKA, 2001), (PRENNEL, 2000) e (MENEZES, 1996), onde podem ser vistos diversos testes. Alguns deles em desenvolvimento (na data da referência).

Testes de desempenho não fizeram parte deste trabalho. Entretanto, devido às semelhanças estruturais existentes entre o cifrador *Alpha* , o DES e o AES, não é de se esperar que o *Alpha* seja (significativamente) mais lento que o AES ou o (triplo) DES (FIPS 46, 1977) (FIPS 46-3, 1999).

9.2 RESISTÊNCIA CONTRA AS CRIPTOANÁLISES DIFERENCIAL E LINEAR

O cifrador Alpha foi construído para ser, no mínimo tão resistente contra a CL e a CD quanto o AES (vide capítulo 4, item 4.3 e capítulo 8, item 8.2.4.2). Cálculos detalhados sobre a resistência do Rijndael a estes dois tipos de para características de propagação de diferenças de 4 e 8 fases podem ser encontrados em (DAEMEN, 1995, cap. 5), (DAEMEN, 1999), (DAEMEN, 2002) e (CASTAÑO, 1998, cap 5).

9.3 MEDIDAS DE DIFUSÃO

9.3.1 AVALIAÇÃO DO EFEITO AVALANCHE E DA COMPLETUDE DA TRANSFORMAÇÃO CRIPTOGRÁFICA

Foi adotado como critério de avaliação do efeito avalanche o número de iterações mínimo para o qual transformação criptográfica é *completa* e o número de baites que são afetados por cada bite da entrada para uma ou mais fases.

As transformações criptográficas não são projetadas para produzir a *completude* em uma única iteração. Isso não é feito porque tal característica aumentaria o fator trabalho de cada iteração e o número de iterações não poderia ser reduzido significativamente, uma vez que o fator trabalho de algumas formas de criptoanálise (CD e CL, por exemplo) depende criticamente[200] do número de iterações(OSWAD, 02). São necessárias, portanto, pelo menos duas iterações para a *completude* de uma cifra iterativa usual (como DES, AES e Serpent, por exemplo).

Quando se desejar comparar um cifrador Feistel com uma estrutura *não-Feistel*, é preciso observar que, no caso das estruturas Feistel, a função de passo atua apenas de um dos lados do bloco (usualmente o direito) e, por melhor que seja a difusão nesta função de passo, são necessárias pelo menos 4 iterações para a *completude*. Justificativa: analisando unicamente as dependências de R_i em relação a $R_{i-1}(i=1,2,3...Nr)$, suponha que a função de passo é completa em duas iterações. Neste caso, que é a melhor hipótese, seguem as configurações esperadas para as MDR (apresentadas divididas em quatro quadrantes de lado *n/2*). O símbolo "I" indica apenas a diagonal preenchida, "X^r" indica *r* iterações da transformação X ($X:Z_2^{n/2}\rightarrow Z_2^{n/2}$, é a função de passo, que atua do lado direito[201] do bloco ou a MDR $[D_r]$ da TL).

$$L_1 = R_0,$$
$$R_1 = L_0+f(R_0)$$

Padrão de dependências após uma iteração

Matriz

0	I
I	X

$$L_2 = L_0+f(R_0),$$
$$R_2 = f(L_0+f(R_0))+ R_0$$

Padrão de dependências após duas iterações

Matriz

I	X
X	X^2

[200] exponencialmente (OSWALD, 2002)

[201] quando for comparar esta demonstração com as MDE, é preciso levar em consideração que nas MDE as metades direita e esquerda do bloco cifrado são trocadas após a ultima fase.

$$L_3 = f(L_0+f(R_0))+ R_0,$$
$$R_3 = f(f(L_0+f(R_0))+ R_0) + L_0+f(R_0)$$

Matriz	
X	X^2
X^2	X^3

Padrão de dependências após três iterações

$$L_4 = f(f(L_0+f(R_0))+ R_0) + L_0+f(R_0),$$
$$R_4 = f(f(f(L_0+f(R_0))+ R_0) + L_0+f(R_0))+ f(L_0+f(R_0))+ R_0$$

Matriz	
X^2	X^3
X^3	X^4

Padrão de dependências após quatro iterações

$$L_5 = f(f(f(L_0+f(R_0))+ R_0) + L_0+f(R_0))+ f(L_0+f(R_0))+ R_0,$$
$$R_5 = f(f(f(f(L_0+f(R_0))+ R_0) + L_0+f(R_0))+ f(L_0+f(R_0))+ R_0)+ f(f(L_0+f(R_0))+ R_0) + L_0+f(R_0)$$

Matriz	
X^3	X^4
X^4	X^5

Padrão de dependências após cinco iterações

Esse padrão prossegue indefinidamente. Se a transformação *X* é completa em 2 iterações, a transformação criptográfica é completa em 4 iterações. Se *X* é completa em 4 iterações, a transformação criptográfica é completa em 6 iterações. Uma ilustração disso é o que acontece com o *Alpha* (vide item 9.5.1 e apêndice 9).

9.3.2 CRITÉRIO DE AVALANCHE ESTRITO

Para avaliação do critério de avalanche estrito, é sugerida a utilização das MDE.

Sejam *i* e *j* dois inteiros pertencentes a $\{0,1,2...n-1\}$. Na construção das MDE para uma transformação criptográfica $f:Z_2^n \rightarrow Z_2^n$ são realizados *w testes de dependência* para cada par *(i,j)* e o elemento d_{ij} armazena o número de vezes que o bite *i* da saída mudou de valor mudando-se apenas o bite *j* na entrada. Se a função satisfaz o critério de avalanche estrito, a probabilidade d_{ij}/w de o *i-ésimo* bite da saída mudar de valor devido à complementação *do j-ésimo* bite da entrada deve tender para 0,5 à medida que *w* cresce. Isso deve valer para todo par *(i,j)*. Este teste ainda carece de um procedimento para definir o tamanho de *w* e o nível de confiança estabelecido pelos desvios $|d_{ij}/w - 0,5|$ encontrados nos resultados. Uma possibilidade é utilizar $w = 20000$ e *deixar de rejeitar* $9654<d_{ij}<10346$, o que é aparentemente equivalente aos limites definidos pelo NIST para o teste monobite (item 9.4.4).

9.4 TESTES DE ALEATORIEDADE

No *FIPS 140-1 Testes estatísticos de aleatoriedade*[202] (disponível na Internet e em (MENEZES, 1996, cap. 5, p. 183)) são especificados quatro testes de aleatoriedade de forma bastante objetiva. Os testes são descritos a seguir, nos itens 9.4.1 a 9.4.4. Em vez de deixar que o usuário escolha usar níveis de significância estatística para os testes, o NIST explicitou limites para as freqüências analisadas na estatística dos testes. Estes limites são apresentados em 9.4.4. Estes testes não são específicos para cifradores de blocos.

9.4.1 TESTE DE FREQÜÊNCIA – DESCRIÇÃO

Seja $s = s_0, s_1,s_2,s_3, ..., s_{n-1}$ uma seqüência binária de tamanho n. O objetivo deste teste é verificar se o número de bites iguais a *0* é aproximadamente igual ao número de bites iguais a *1*, como seria esperado em uma seqüência aleatória. Sejam n_0 o número de bites iguais a *0* e n_1 número de bites iguais a *1*. A estatística utilizada é a seguinte:

$$X_1=(n_0-n_1)^2/n \qquad \text{(9-I)}$$

Esta estatística deve seguir aproximadamente uma distribuição χ^2 com 1 grau de liberdade se $n>10$. Na prática, recomenda-se $n>20000$.

9.4.2 TESTE POKER – DESCRIÇÃO

Seja m um inteiro positivo, tal que $n/m \geq 5 \, . \, (2^m)$, e seja k o maior inteiro menor ou igual a n/m. Divide-se a seqüência s em k blocos de tamanho m (sem superposição) e representa-se por n_i o número de ocorrências do *i-ésimo* tipo de seqüência de tamanho m ($1 \leq i \leq 2^m$). O teste poker determina se cada sequência de tamanho m aparece o mesmo número de vezes, como seria esperado em uma distribuição aleatória. A estatística utilizada é a seguinte:

$$X_3 = (2^m/k) \left(\sum_{i=1}^{i=2^m} n_i^{\,2} \right) - k \qquad \text{(9-II)}$$

Esta estatística deve seguir aproximadamente uma distribuição χ^2 com 2^m-1 graus de liberdade. Esse teste é uma generalização de teste de freqüência apresentado em 9.4.1.

[202] *Statistical tests for Randomness*

9.4.3 TESTE DE SEQÜÊNCIAS CORRIDAS[203]

O objetivo deste teste é verificar se o número de ocorrências de seqüências compostas somente de bites iguais a 0 (delimitadas à esquerda e à direita por 1) ou somente bites iguais a 1(delimitadas à esquerda e à direita por 0) é compatível com uma distribuição aleatória.

O número esperado de tais seqüências de comprimento i em uma seqüência de tamanho n é $e_i=(n-i+3)/2^{i+2}$. Seja k o maior inteiro i para o qual $e_i \geq 5$. Sejam U_i e Z_i os números de seqüências de *1's* e *0's* de tamanho i presentes em n, respectivamente, para cada i ($1 \leq i \leq k$). A estatística usada é

$$X_4 = \sum_{i=1}^{i=k} \frac{(U_i - e_i)^2}{e_i} + \sum_{i=1}^{i=k} \frac{(Z_i - e_i)^2}{e_i} \qquad \text{(9-III)}$$

Esta estatística deve seguir aproximadamente uma distribuição χ^2 com $2k-2$ graus de liberdade.

9.4.4 LIMITES DEFINIDOS PELO NIST

Uma seqüência s de bites de tamanho 20000, saída do gerador de aleatórios a ser analisado, está sujeita a cada um dos seguintes testes. Se algum dos testes der resultado negativo, o gerador é rejeitado.

1. *Teste de freqüência de um bite (monobit test)* – O número n_1 de bites iguais a *'1'* em s deve ser tal que $9654<n_1<10346$.

2. *Teste Poker* - A estatística X_3 definida na expressão 9-I é computada para $m=4$. O gerador "passa" no teste se $1.03 < X_3 < 57.4$.

3. *Teste de seqüências corridas* – Os números U_i e Z_i de tamanho i são contados para cada i, ($1 \leq i \leq 6$. Para os fins deste teste, as seqüências de tamanho maior que 6 são

[203] *runs test* (MENEZES, 196, p. 182)

consideradas tendo tamanho 6). O gerador "passa" neste teste se as 12 contagens U_i e Z_i encontram-se nos intervalos especificados na TAB. 9.1.

TAB. 9.1: Valores permitidos para as contagens do teste de seqüências.

Tamanho da Seqüência	Valores permitidos Para U_i e Z_i
1	2267 a 2733
2	1079 a 1421
3	502 a 748
4	223 a 402
5	90 a 223
6	90 a 223

4. *Teste de sequência longa (long run test)* – O gerador de aleatórios "passa" neste teste se nenhuma seqüência de bites iguais de tamanho maior que 33 é encontrada.

O NIST admite a substituição destes testes por testes de comprovada equivalência estatística.

9.5 RESULTADOS OBTIDOS COM O CIFRADOR ALPHA

9.5.1 TESTES DE DEPENDÊNCIAS

No item 12.9.1.6, no apêndice 9, pode ser visto que D_6 é completa. Isso ocorre por que a transformação linear L_α é completa para *4* iterações (pela forma como foi construída pelo método dos nós).

A função f_α (considerando o lado direito do bloco) é completa para duas iterações. Em conseqüência a quarta MDE é completa, como pode ser visto em 12.9.2.4 (apêndice 9) e na TAB. 9.2.

9.5.2 CRITÉRIO DE AVALANCHE ESTRITO

Foram construídas as 10 primeiras MDE para o cifrador *Alpha* com $w = 2000$ e *com* $w = 4000$. A TAB. 9.2 mostra o resumo dos resultados para $w = 4000$.

TAB. 9.2: Resumo dos testes com as MDE (*Alpha* 2, *w=4000*)

MDE	d_{ij}_min	i	j	dp_max(lin):	LIPA:	LSPA:
1:	900	15	115	371,871	1	56
2:	853	34	36	376,553	49	119
3:	871	74	57	163,007	112	128
4:	1859	4	72	35,448	**128**	128
5:	1864	92	99	36,724	128	128
6:	1866	14	69	36,043	128	128
7:	1869	108	8	36,074	128	128
8:	1868	12	122	35,652	128	128
9:	1881	56	81	36,372	128	128
10:	1869	127	95	37,231	128	128

Com *w = 4000*, considerando a média em cada linha da MDE para 10 iterações, todos os valores de média encontrados estavam no intervalo 2000 ± 5,56. Para *w = 2000,* considerando a média em cada linha da MDE para 10 iterações, todos os valores de média encontrados estavam no intervalo 1000 ± 4,52. Uma vez que 5,56/2000=0,00278 e 4,52/1000=0,00452, temos que d_{ij} aproxima-se de *w/2* à proporção que *w* cresce. Esse comportamento é compatível com o CAE, uma vez que a média da amostra tende para a média verdadeira[204] à proporção que a amostra cresce. Entretanto, este teste precisa de um melhor tratamento estatístico.

Outro resultado que é compatível com a satisfação do CAE é o resultado dos testes de *perturbação*[205]. O resultado dos testes de perturbação encontra-se no item 12.9.5 (apêndice 9). As perturbações médias para cada linha são muito próximas de 64 para todos os testes a partir de 4 iterações. Esse é outro teste que pode ser mais bem fundamentado teoricamente.

Os resultados apresentados em 9.5.4 também são compatíveis com a satisfação do CAE. Foram construídas MDE com *w = 20000* e todos os valores de dependências encontrados estavam dentro do intervalo preconizado no teste monobite do NIST, o que também é compatível com a satisfação do CAE. Esse resultado foi observado para 4, 5 e 6 iterações. O teste não foi realizado com mais iterações.

[204] Para uma seqüência aleatória a média verdadeira é *w/2*.
[205] Definido no item 2.3, xxxiv.

9.5.3 TESTES DE ALEATORIEDADE

Foram aplicados ao *Alpha* os testes descritos em 9.4. Para gerar o vetor de 20000 bites foram cifrados 157 blocos B_i *(i=0,1,2...156)* nos quais apenas o bite b_j (*j=i mod 128*) foi feito igual a *1*. Para *i<128* foi utilizada uma chave com todos os bites iguais a zero e nos demais casos apenas o primeiro bite da chave foi feito igual a *1*. Estas entradas foram escolhidas por serem consideradas as mais desfavoráveis para a cifra, uma vez que *0* é o elemento neutro da soma e esta é a operação que mais ocorre durante o processo de cifragem[206].

Os resultados do teste são apresentados a seguir. O algoritmo foi testado para até *16* iterações e foi aprovado para todos os valores de *Nr* maiores que três. Os resultados abaixo foram obtidos para *Nr = 4*. É interessante notar que, mesmo com apenas quatro iterações, os valores encontrados estão relativamente próximos dos valores centrais dos intervalos estabelecidos pelo NIST. O código fonte utilizado encontra-se no apêndice I.

Teste Monobit (9654 < aprovado < 10346)

RESULTADO: n1=9924 APROVADO!

_____

Teste Poker (1,03 < aprovado < 57,4)

RESULTADO:X3=15,859199999999873 APROVADO!

_____

Teste de Seqüências de 0's

Tamanho1: 2507 ocorrências. Valores permitidos:2267 a 2733. RESULTADO:APROVADO!
Tamanho2: 1272 ocorrências. Valores permitidos:1079 a 1421. RESULTADO: APROVADO!
Tamanho 3: 650 ocorrências. Valores permitidos:502 a 748. RESULTADO: APROVADO!
Tamanho 4: 314 ocorrências. Valores permitidos:223 a 402. RESULTADO: APROVADO!
Tamanho 5: 154 ocorrências. Valores permitidos:90 a 223. RESULTADO: APROVADO!
Tamanho 6: 148 ocorrências. Valores permitidos:90 a 223. RESULTADO: APROVADO!

_____

Teste de Seqüências de 1's

_____

Tamanho 1: 2507 ocorrências. Valores permitidos:2267 a 2733. RESULTADO: APROVADO!
Tamanho 2: 1272 ocorrências. Valores permitidos:1079 a 1421. RESULTADO: APROVADO!
Tamanho 3: 650 ocorrências. Valores permitidos:502 a 748. RESULTADO: APROVADO!
Tamanho 4: 314 ocorrências. Valores permitidos:223 a 402. RESULTADO: APROVADO!
Tamanho 5: 154 ocorrências. Valores permitidos:90 a 223. RESULTADO: APROVADO!
Tamanho 6: 148 ocorrências. Valores permitidos:90 a 223. RESULTADO: APROVADO!

_____

Maior Seqüência de 0's

Valores permitidos: até 33. RESULTADO: 13 APROVADO!

Maior Seqüência de 1's

[206] Para outros vetores testados os resultados também foram positivos, mas foi feita a opção de registrar o teste com maior número possível de zeros nos argumentos (chave e texto em claro).

Valores permitidos: até 33. RESULTADO: 12 APROVADO!

9.6 TESTES RECOMENDADOS PELO NIST, NESSIE E RACE

Em (DICHTL, 1999) é apresentado um pacote de ferramentas e testes para avaliação dos algoritmos submetidos ao NESSIE. Alguns testes ainda estavam em desenvolvimento na época da publicação do documento e os autores comentam sobre a dificuldade de se implementar testes efetivos de aplicação geral. Os testes que tratam da resistência contra a CD e a CL são considerados os mais importantes. A avaliação das tabelas de substituição é feita analisando as propriedades diferenciais e lineares das mesmas. Entretanto, o documento relata que é difícil avaliar as propriedades de propagação de diferenças com as ferramentas (*software*) disponíveis atualmente.

No mesmo documento é apresentado um conjunto de testes desenvolvido no projeto RIPE (*RACE Integrity Primitives Evaluation*) para avaliação dos algoritmos submetidos ao NESSIE para análise. Os testes especificados para cifradores de blocos são os seguintes:

- *teste de dependências*: computa a matriz de dependências[207] de um cifrador de blocos e avalia o grau de avalanche, CAE e completude.
- *teste de equações lineares, aproximações lineares e teste de imunidade às correlações*: Este teste é baseado na *transformada rápida de Walsh* e não pode ser aplicado em cifras de blocos 128 bites, apenas para as tabelas de substituição.
- *teste de fatores lineares*: determina equações lineares entre bites de texto em claro, chave e texto cifrado obtidas quando um bite da chave ou do texto em claro é modificado.
- *teste de ciclos:* usado para determinar ciclos quando uma cifra de blocos é aplicada repetidamente.

Os testes de aleatoriedade apresentados no item 9.4 também estão relacionados pelo NESSIE, embora não estejam pré-definidos os limites de aceitação/rejeição.

[207] Esse teste não foi analisado com detalhes, mas as definições parecem ser bastante semelhantes aos conceitos desenvolvidos independentemente durante este trabalho no IME.

9.7 CONCLUSÃO DO CAPÍTULO

Os resultados dos testes de aleatoriedade realizados com o cifrador *Alpha* com 4 ou mais fases não apresentaram nenhum desvio estatístico comprometedor, não havendo portanto razão aparente para rejeitá-lo.

É recomendável verificar se existem novas ferramentas de análise disponíveis e utilizá-las para testes com o *Alpha*.

Não foram realizados os testes de correlação nem testes de ciclos. Além disso, aparentemente, existem medidas de grau de avalanche, CAE e completude mais bem fundamentadas estatisticamente em (PRENEEL, 2000).

10 CONCLUSÃO

O resultado deste trabalho confirmou que o estudo dos processos de seleção de padrões e dos aspectos positivos e negativos de algoritmos criptográficos conhecidos pode constituir uma base de conhecimento útil para novos desenvolvimentos de projetos.

10.1 SOBRE ESTE TRABALHO

Os cifradores simétricos de blocos DES, AES e Serpent podem ser descritos e implementados de maneiras diferentes. É útil definir um formato padrão para descrever tais algoritmos antes de proceder a um estudo comparativo entre eles.

O formato de descrição dos algoritmos apresentado neste trabalho, ou seja uma TL representada por uma matriz e uma TNL representada por tabelas de substituições e principalmente, por suas matrizes de distribuição linear e diferencial, mostrou-se conveniente para auxiliar na comparação de cifras de blocos, sobretudo no aspecto segurança.

Apesar de ter sido adotado o AES como padrão mais recente de cifrador de blocos, o único ponto de comprometimento da segurança do DES demonstrado na prática é o tamanho da chave.

As formas mais importantes de criptoanálise são a CD e a CL, contra as quais pode ser aumentada a resistência de uma cifra observando-se as matrizes de distribuição de diferenças e de distribuição linear das TNL, a difusão produzida pelas TL e o número de iterações.

O AES, que difere do Rijndael apenas na faixa de valores aceitos para tamanho de bloco e chave[208], é uma versão particular do Rijndael, pode ser descrito de maneira até mais simples que o DES, o que facilita a compreensão da transformação criptográfica para aqueles que estão acostumados com o DES. As implementações em s*oftware* da descrição alternativa do Rijndael, em princípio, destinam-se apenas a testes, e não a aplicativos destinem a sistemas criptográficos de uso prático (a menos que testes sejam realizados e indiquem tal aplicação). Quanto à implementação em *hardware* dedicado, a descrição alternativa do AES permite realizar as duas operações lineares e a soma da sub-chave de passo em uma única operação de soma módulo dois envolvendo 6 a 8 bites (5 a 7 da TL mais um da chave) da entrada para

[208] O Rijndael aceita qualquer multiplo de 32 desde 128 até 256 para o tamanho do bloco e da chave, independentemente. O AES aceita blocos de 128 bites e chaves de 128, 192 e 256 bites.

cada bite da saída. Isso deve reduzir sensivelmente o número de ciclos necessário para cada iteração do Rijndael, tornando-o significativamente mais rápido.

A técnica das *matrizes de avalanche* é útil para comparar o efeito final de transformações lineares, mesmo que estas operações tenham sido descritas inicialmente de maneiras distintas e de difícil visualização dos resultados. A comparação entretanto não é definitiva, sendo necessário considerar o efeito da transformação não-linear presente em cada cifra e o número de iterações previsto. Para auxiliar nesta avaliação mais completa, podem ser utilizadas as matrizes de dependências estatísticas (MDE), que computam os efeitos de todas as transformações realizadas.

A análise das perturbações e das MDE decimais e binárias dá informações sobre o *efeito avalanche*, sobre *critério de avalanche estrito* e sobre a *completude* da função criptográfica. A análise destes aspectos é fundamental na avaliação de qualquer cifra de blocos. A forma de analisar estes dados, entretanto, precisa ser melhor estudada para ganhar mais consistência estatística.

Em (PRENEEL, 2000) e (DICHTL, 1999) são feitas referências a matrizes de dependências para avaliação de efeito avalanche, completude e CAE. Não foi possível estudar estas matrizes dentro do tempo destinado a este trabalho, mas elas apresentam um grau de semelhança de nomenclatura e definições muito grande em relação às MDR e MDE. Isso reforça a idéia de que vale a pena envidar esforços no sentido de desenvolver mais conhecimento no Brasil, pois os resultados produzidos aqui aparentemente são de tão boa qualidade quanto os que são produzidos nos EUA e na Europa.

O *método dos nós* é útil para construir transformações lineares com bom efeito avalanche e que produzem a completude da função em poucas iterações. Esse aspecto é muito positivo, uma vez que tende a aumentar a complexidade das equações que venham a ser montadas para a tentativa de criptoanálise dos algoritmos. Quanto à eficiência, é possível que as TL construídas pelo método dos nós sejam menos rápidas em *software* e é preciso verificar se o resultado final é compensador. Em *hardware* entretanto, é mais provável que tais transformações sejam mais rápidas.

O método dos QL para construção de TNL pode ser usado para evitar a possibilidade de CEI em cifras tipo Feistel e a associação de vários QL de maneira aleatória pode aumentar a resistência contra as CL e CD.

Os testes realizados em algoritmos criptográficos pelo NIST e pelo NESSIE são testes de *rejeição* ou *não-rejeição* e não são suficientes para garantir a segurança de uma cifra, apesar de um insucesso em qualquer destes testes indicar um comprometimento sério da segurança.

A comparação de cifradores Feistel com cifradores não-Feistel deve ser feita com cautela, uma vez que os sistemas Feistel têm peculiaridades que o distinguem, como por exemplo o fato de que a *completude* só é possível na prática após 4 iterações, por melhor que seja a *difusão* da TL. Deve ser lembrado que os cifradores Feistel tem a seu favor a semelhança com o DES (que sugere ser pouco esperado o surgimento de uma nova forma de criptoanálise que o ameace) e que em cada iteração apenas metade do bloco é transformado pela função de passo *f* (o que sugere que cada passo de um sistema Feistel tende a ser mais rápido que cada passo de um sistema não-Feistel).

Em conseqüência das reflexões feita sobre os processos de seleção e análise de algoritmos do NIST e do NESSIE, foi sugerida uma filosofia de projeto traduzida em cinco princípios: *segurança, eficiência, flexibilidade, simplicidade e credibilidade*. De acordo com esta filosofia, foi apresentado o cifrador simétrico de blocos *Alpha*, que, até onde foi possível avaliar, está de acordo com os padrões de aceitação estabelecidos pela comunidade científica atualmente. O cifrador *Alpha* cifra blocos de 128 bites, em 16 iterações, com uma chave cujo tamanho pode ser qualquer multiplo de 32 bites (mínimo de 128). Os testes realizados com esta versão do cifrador encontram-se apresentados no capítulo 9.

O *Alpha* foi aprovado nos testes a que foi submetido para qualquer numero de iterações a partir de 4.

10.2 CONTINUIDADE DA PESQUISA EM CRIPTOGRAFIA NO IME

Este trabalho deu continuidade aos seguintes trabalhos realizados no IME/DE-9: (CASTAÑO, 1998), (LIMA,1998), (XAVIER JR, 1999), (XEXEO, 1983).

A vastidão, natureza e a profundidade dos assuntos relacionados à criptologia dão ao tema um caráter singular na pesquisa científica, uma vez que grande esforço é empreendido continuamente para aumentar a complexidade de muitos dos problemas relacionados ao assunto.

O cifrador *Alpha* foi implementado ao longo deste trabalho em 4 versões. Na primeira versão o cifrador utiliza a mesma TNL do AES, que apresenta as melhores características

diferenciais e lineares encontradas em todas as TNL analisadas durante este trabalho. Na segunda versão, o *Alpha* utiliza a tabela de substituição do AES mais quatro bites na entrada, o que torna esta operação "não invertível". Na terceira versão foi utilizado um par de QL, conforme sugerido no capítulo 7. Na quarta versão foram utilizados 2 QL sorteados de um grupo de 4 em cada iteração. A Segunda dois foi escolhida por ser a mais rápida dentre as que usam TNL não invertíveis. A TL foi construída utilizando o método dos nós. Sugere-se que novos testes sejam realizados em implementações otimizadas das outras versões.

Como sugestão para trabalhos futuros, podem ser pesquisados os seguintes temas:

- Aprimoramento do método dos nós, para possivelmente construir transformações lineares mais eficientes e que levem em consideração mais aspectos das TNL;
- Estudo mais aprofundado dos QL e suas aplicações criptográficas;
- Refinar as análises das MDR e MDE para levantar taxas de propagação de diferenças e de correlações lineares presentes nas cifras de maneira mais precisa, determinando os limites inferiores do fator trabalho esperado para as formas mais importantes de criptoanálise;
- Verificar as possibilidades de aplicação das MDE e MDR em criptoanálise, utilizando-as inicialmente para construção das equações algébricas das cifras com um número de passos reduzido;
- Estudar as aplicações de sistemas Feistel desequilibrados;
- Realizar mais testes com o cifrador Alpha, inclusive com vistas a avaliar o desempenho em diferentes ambientes (PRENEEL, 2000) e (DICHTL, 1999),;
- Estudar versões do cifrador *Alpha* com outras TL e TNL;
- Estudar o emprego de cifras de blocos na geração de números aleatórios;
- Estudar a possibilidade de associar as matrizes de avalanche com a teoria presente em (DAEMEN, 1995, cap. 5), (DAEMEN, 1999, anexo), (DAEMEN, 2002) e (CASTAÑO, 1998, cap 5) para calcular de maneira mais precisa limites inferiores para o fator trabalho da CD e da CL envolvendo diferentes numeros de fases.

10.3 CONTRIBUIÇÕES DESTE TRABALHO

- Foi apresentada uma descrição simplificada e didática do AES, que ajudará bastante na compreensão do funcionamento da cifra;
- Foi apresentado um conjunto de princípios que podem nortear um projeto de uma cifra de blocos de boa qualidade de maneira objetiva;
- Foi apresentada uma técnica para avaliação e comparação de transformações lineares destinadas a cifras de blocos baseadas em substituições e transformações lineares, tornando possível comparar transformações lineares que originalmente são descritas de maneiras distintas;
- Foi desenvolvida e apresentada uma técnica para construção sistemática de transformações lineares com boas caracterírticas de difusão;
- Foi desenvolvida e apresentada uma técnica para auxiliar avaliação da completude e do critério de avalanche estrito em cifras de blocos;
- Foi desenvolvida e apresentada uma técnica para construção sistemática de transformações não-lineares com boas propriedades de resistência contra as formas mais importantes de criptoanálise;
- Foi apresentada uma nova estrutura para algoritmos criptográficos que aparentemente constitui um aprimoramento das cifras Feistel;
- Foi desenvolvido, apresentado e (parcialmente) testado um cifrador de blocos compatível com os padrões de exigência internacionais atuais.

11 REFRÊNCIAS BIBLIOGRÁFICAS

BAUDRON, Olivier, GILBERT, Henri, GRANBOULAN, Louis, HANDSCHUH, Helena, JOUX, Antoine, NGUYEN, Phong, NOILHAN, Fabrice, POINTCHEVAL, David, PORNIN, Thomas, POUPARD, Guillaume, STERN, Jacques, and VAUDENAY, S. **Report on the AES Candidates** disponível em http://csrc.nist.gov/CryptoToolkit/aes/round1/conf2/papers/baudron1.pdf [capturado em 8 fev 2003]

BIHAM, Eli, SHAMIR, Adir **Differential Cryptanalysis of the Data Encryption Standard** Springer-Verlag – New York – 1993

BIHAM, Eli, **A Note on Comparing the AES Candidates (1999)** http://secinf.net/cryptography/A_Note_on_Comparing_the_AES_Candidates_Revised_Version.html [capturado em 8 fev 2003]

BLAZE, M., DIFFIE, W.,RIVEST, R., SCHNEIER, B., SHIMOMURA, T.,THOMPSON, E., WIENER, M. **Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security. A report by an Ad Hoc group of cryptographers and computer scientists...** *http://www.bsa.org/policy/encryption/cryptographers.html* [capturado em 15 fev 2003]

BOWMAN, Tom, SHANE, Scott **Rigging the game: Spy Sting**, Baltimore Sun, 10 December 1995. http://www.iptvreports.mcmail.com/ic2kreport.htm [capturado em 14 fev 2003]

BUCHMANN, J. A. **Introdução à Criptografia** – Ed. Berkeley, São Paulo, 2002

BURWICK, C., COPPERSMITH, D., D'AVIGNON, E., GENNARO, G., HALEVI, S., JUTLA, C., O'CONNOR, L., PEYRAVIAN, M., SAFFORD, D., ZUNIC, N. **MARS – A candidate cipher for AES** – IBM Corporation – Revised – Agosto de 1999 disponível na Internet [capturado em 15 dez 2002]

CARROLL, John M., ROBBINS, Lynda E**. Using Binary Derivatives to test an Enhancement of DES** – Cryptologya Vol XII Number 4, pags. 193-207 – October 1988

CARVALHO, D. B. **Criptografia: Métodos e Algoritmos**, 2ª Ed., Book Express, Rio de Janeiro, 2001

CASTAÑO, F. C. **Análise das Cifras Rijndael e Serpent contra as Criptoanálises Linear e Diferencial** - Tese de Mestrado, IME, Rio de Janeiro, Brasil, Dez 1998.

CHARI, Suresh, JUTLA, Charanjit, RAO, Josyula, ROHATGI, Pankaj **A Cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards** Second Advanced

Encryption Standard (AES) Candidate Conference - 1999 , disponível em http://citeseer.nj.nec.com/chari99cautionary.html [capturado em 15 mar 2003]

CHEON, J. H., KIM, MunJu, LEE, Yung Yeun, KANG, Sung Woo **Improved Impossible Differencial Cryptanalysis of Rijndael and Crypton** em K. Kim, editor, *Information Security and Cryptology – ICISC 2001*, number 2288 emin *Lecture Notes in Computer Science*, pages 39-49. Springer 2001

COPPERSMITH[A], D., GENNARO, R., HALEVI, S., JUTLA, C., MATYAS Jr., S., PEYRAVIAN, M., SAFFORD, D., ZUNIC, N. **IBM Comments – Third AES Conference** - April 13 – 2000 , disponível em http://www.research.ibm.com/security

COPPERSMITH[B], D., GENNARO, R., HALEVI, S., JUTLA, C., MATYAS JR., S., PEYRAVIAN, M., SAFFORD, D., ZUNIC, N. **MARS and the AES Selection Criteria** – 15 de maio de 2000 , disponível em http://www.research.ibm.com/security/final-comments.pdf [capturado em 16 mar 2003]

COPPERSMITH[C] D., GENNARO, R., HALEVI, S., JUTLA, C., MATYAS JR., S., PEYRAVIAN, M., SAFFORD, D., ZUNIC, N. **Third AES Comments – IBM** – April 13 2000 – disponível em http://csrc.nist.gov/CryptoToolkit/aes/round2/conf3/papers/mars-statement.pdf [capturado em 15 jan 2003]

COURTOIS, Nicolas T., PIEPRZYK, Josef **Cryptanalysis of Block Ciphers with Overdefined Systems of Equations** . In Yuliang Zheng, editor, *Proceedings of Asiacrypt'02, Lecture Notes in Computer Science.* Springer-Verlag, 2002.

DAEMEN, Joan, RIJMEN, Vincent. **The Rijndael Block Cipher - AES Proposal: Rijndael***,* versão 2 – Brussel, Bégica - Setembro 1999

DAEMEN, Joan, RIJMEN, Vincent. **Answer to "New Observations on Rijndael"** –Agosto de 2000 – disponível em http://www.esat.kuleuven.ac.be/~rijmen/rijndael/answer.pdf [capturado em 16 mar 2003]

DAEMEN, Joan, RIJMEN, Vincent. **The Design of Rijndael – AES – The Advanced Encryption Standard** : Springer-Verlag Berlin Heidelberg – 2002 – http://www.springer.de [capturado em 16 mar 2003]

DAVIDA, G., KAM, J. **Structured Design of Substitution-Permutation Encryption Networks**, IEEE Transactions on Computers, 28(1979), 747-753

DECRETO Nº 2.910**,** de 29 de dezembro de 1998, Presidência da República**,** disponível em https://www.presidenciada república.gov.br/gsi/cgsi [capturado em 12 Jan 2004]

DECRETO Nº 3.505, de 13 de junho de 2000, Presidência da República**,** disponível em https://www.presidenciada república.gov.br/gsi/cgsi [capturado em 12 Jan 2004]

DENNING, Dorothy E. **Cryptography and Data Security** – Addison Wessley Publishing Company – Reading, Massachussets, 1983.

DER SPIEGEL: **Wer ist der Befugte Vierte?** Der Spiegel, 36, 1996, pp. 206-7. http://www.iptvreports.mcmail.com/ic2kreport.htm [capturado em 10 out 2003]

DICHTL, Markus, **List of General NESSIE Test Tools-Vs.1**(Document Reference: NES/DOC/SAG/WP2/D03/1), 1999 – Disponível na Internet [capturado em 16 set 2003]

DIFFIE, W., HELLMAN, M. **New Directions in Cryptography** – IEE International Symposium on Information Theory, Sweden, 24 Jun 1976 http://www.cs.rutgers.edu/~tdnguyen/classes/cs671/presentations/Arvind-NEWDIRS.pdf [capturado em 23 abr 2002]

DIFFIE, W., HELLMAN, M. **Exhaustive Cryptanalysis of the NBS Data Encryption Standard**, Computer Magazine, Long Beach, Califórnia - June 1977

DIFFIE, W., LANDAU, S. **The Export of Cryptography in the 20th Century (SMLI TR -2001-102)** – Sun Microsystems, Palo Alto, CA – Outubro, 2001.

DRAY, Jim. **Report on the NIST Java™ AES Candidate Algorithm Analysis** Computer Security Division - National Institute of Standards and Technology - November 8, 1999, disponível em http://csrc.nist.gov/CryptoToolkit/aes/round1/r1-java.pdf [capturado em 17 mar 2003]

ELBIRT, A. J., PAAR, C. **An FPGA Implementation and Performance Evaluation of the Serpent Block Cipher**, preprint a ser a presentado na FPGA2000 – disponível na Internet [capturado em 16 set 2003]

FERGUSON, N., KELSEY, J., LUCKS, S., SCHNEIER, B., STAY, M., WAGNER, D., WHITING, D. **Improved Cryptanalysis of Rijndael**. In Bruce Schneier, editor, Proceedings of Fast Software Encryption – FSE'00, number 1978 in Lecture Notes in Computer Science, pags. 213-230. Springer-Verlag 2000

FEISTEL, Horst. **Cryptography and Computer Privacy**. In Scientific American. 228(5)-1973

FERREIRA, A. R., NAPOLITANO, F. M. **Estudo das Transformações Lineares dos Algoritmos Criptográficos Rijndael, DES, Serpent e *Alpha* Utilizando as Matrizes de Avalanche -** Trabalho de Iniciação à Pesquisa – Departamento de Engenharia de Sistemas – IME – Rio de Janeiro, Dez 2003

FIPS 197 - Announcing the Advanced Encryption Standard – AES: National Institute of Standards and Technology. November 26 –2001

FIPS 46 - Data Encryption Standard, Federal Information Processing Standard, National Institute of Standards and Technology, Publication 46, National Bureau of Standards,US Department of Commerce, Washington D.C., 1977

FIPS 46-3 - DATA ENCRYPTION STANDARD (DES), National Institute of Standards and Technology. October 25 – 1999

FOTI, J. **Status of The Advanced Encryption Standard (AES) Development Effort** 1998 National Information Systems Security Conference – disponível em http://csrc.nist.gov/nissc/1998/proceedings/paperG1.pdf [capturado em 12 jan 2003]

FRALEIGH, John B. **A First Course in Abstract Algebra**, 6th Ed, Addison-Wesley, Boston, MA 1998

GILBERT, H., MINIER, M. **A Collision Attack on 7 Rounds of Rijndael** In Proceedings of the Third Advanced Encryption Standard Conference. Pags. 230-241 NIST, Abril 2000.

GUSTAVSON, Helen, DAWSON, Ed, CAELLI, Bill **Applying Randomness tests to Commercial Level Block Ciphers** - Lecture Notes in Computer Science (1996) - University of Utah Department of Mathematics, Salt Lake City, USA

HOFFMAN, Lance J., BALENSON, David M., METIVIER-CARREIRO, Karen, KIM, A., MUNDY, Mathew G. **Growing Development of Foreign Encryption Products in the Face of U.S. Export Regulations – Report GWU-CPI-1999-02 –** Cyberspace Policy Institute of The George Washington University – School of Engineering and Applied Science – Washington DC. Website: http://www.seas.gwu.edu/seas/institutes/cpi [capturado em 12 ago 2003]

KAHN, David**. The Codebreakers – The Story of Secret Writing** – Eighth Printing – Macmillan Publisshing Co. Inc. – New York – 1976

KELSEY, J., SCHNEIER, B. **Unbalanced Feistel Networks and Block-Cipher Design (1996)** – disponível na Internet em http://citeseer.nj.nec.com/cache/papers [capturado em 12 jul 2003]

KEMENY, John G., SNELL, J. Laurie, GERALD, L. Thompson**. Introduction to Finite Mathmatics**, 3ª Ed., Prentice Hall, Englewood Cliffs – NJ, 1958

KOHNO, T., KELSEY, J., SCHNEIER, B. **Preliminary Cryptanalysis of Reduced Round Serpent (2000)** – disponível na Internet - [capturado em 15 dez 2002]

KNUDSEN, L., ANDERSON, R., BIHAM, Eli. **Serpent: A Proposal for the Advanced Encryption Standard (1999) -** http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/serpent.pdf

KNUDSEN, L., HAVARD, R. **Recommendation to NIST for The AES** Dept. of Informatics, University of Bergen, Norway, May 15 2000

KOCHER, P. **Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems** Advances in Cryptology, Proceedings of Crypto 96, LNCS, N. Kobltz, editor, Springer 1996 http://www.cryptography.com/timingattack.old/timingattack.html http://www.cryptography.com/resources/whitepapers/TimingAttacks.pdf [capturado em 20 jan 2003]

KOCHER, P., JAFFE, Joshua, JUN, Benjamin. **Differencial Power Analysis**, disponível em http://cryptography.com/resources/whitepapers/DPA.pdf [capturado em 20 jan 2003]

LAMBERT[A], J., XEXÉO, José A. M., PAZ DE LIMA, A. **Rijndael – Um Algoritmo no Estilo DES** Relatório Técnico Nr 084/DE9/FEV03 do Departamento de Engenharia de Sistemas do Instituto Militar de Engenharia - Rio de Janeiro, Fev 2003 - disponível em www.ime.eb.br

LAMBERT[B], J., FERREIRA, Anderson R., NAPOLITANO, Fillipe M. P. **Matrizes de Dependências e Matrizes de Avalanche** Relatório Técnico Nr 095/DE9/DEZ03 do Departamento de Engenharia de Sistemas do Instituto Militar de Engenharia - Rio de Janeiro, Dez 2003 - disponível em www.ime.eb.br

LAMBERT[C], J. **Construção de Transformações Lineares para Cifradores de Blocos - Método dos Nós** Relatório Técnico Nr 096/DE9/DEZ03 do Departamento de Engenharia de Sistemas do Instituto Militar de Engenharia - Rio de Janeiro, Dez 2003 - disponível em www.ime.eb.br

LAMBERT[D], J. **Criptoanálise por Entradas Invariantes** Relatório Técnico Nr 092/DE9/SET03 do Departamento de Engenharia de Sistemas do Instituto Militar de Engenharia - Rio de Janeiro, Set 2003 - disponível em www.ime.eb.br

LANDAU, Susan. **Designing Cryptography for The New Century**, Communications of the ACM, Volume 43, Issue 5, pags. 115-120 -Mai-2000

LAURIN, Fredrik, FROSTE, Calle. **Secret Swedish E-Mail Can Be Read by the U.S.A**, Svenska Dagbladet, 18 November 1997. http://www.iptvreports.mcmail.com/ic2kreport.htm [capturado em 25 out 2003]

LEI nº 8.159, de 8 de janeiro de 1991, disponível em https://www.presidenciada república.gov.br/gsi/cgsi [capturado em 12 Jan 2004]

LE MONDE: **Espionage's Holy Alliance**, 30 Mar 2000, p. 16 http://www.lemonde.fr/article/0,2320,48510,00.html [capturado em 25 out 2003]

LEECH, David P. **Planning Report 01-2 The Economic Impacts of NIST's Data Encryption Standard (DES) Program** Prepared by: TASC, Inc. for National Institute of Standards & Technology Program Office Strategic Planning and Economic Analysis Group - October 2001 http://www.nist.gov/director/prog-ofc/report01-2.pdf

LIMA, Almir P., CASTAÑO, F**. Criptoanálise Linear - RT034/DE9FEV99** – Instituto Militar de Engenharia, Rio de Janeiro, Brasil, Fevereiro de 1999.

LOUDON, Kyle. **Dominando Altgoritmos com C** – Editora Ciência Moderna – Rio de Janeiro, 2000

LUCKS, Stefan. **Attacking 7 Rounds of Rijndael Under 192 bit and 256 bit keys. In Proceedings of the Third Advanced Encryption Standard Conference.** NIST, Abril 2000. http://csrc.nist.gov/CryptoToolkit/aes/round2/conf3/papers/04-slucks.pdf [capturado em 25 abr 2003]

MARKEZON, Lilian, SZWARCFITER, Jayme. **Estruturas de Dados e Seus Algoritmos, 2ª Edição** – LTC Editora, Rio de Janeiro, 1994

MASSEY, James L., MAURER, Ueli, WANG, Muzhong, **NON-EXPANDING, KEY-MINIMAL, ROBUSTLY-PERFECT, LINEAR AND BILINEAR CIPHERS** – (Preprint of a paper to appear in the Proceedings of EUROCRYPT **87**, in Advances in Cryptology, Lecture Notes in Computer Science, published by Springer-Verlag) – Institute for Signal and Iinformationn Processing - Swiss Federal Institute of Technology – CH-8092 Zürich, Switzerland – Disponível na Internet [capturado em 23 abr 2003]

MASSEY, James L. **On the Optimality of SAFER+ Diffusion** Cylink Corporation, Sunnyvale, CA, USA ano 199X (Corresponding address: Trondhjemsgade 3, 2t.h. DK-2100 Copenhagen East Denmark e-mail: 101767.233@compuserve.com) http://csrc.nist.gov/CryptoToolkit/aes/round1/conf2/papers/massey.pdf [capturado em 23 abr 2003]

MATSUI, M. **Linear Cryptanalysis Methood for DES Cipher** Advances in Cryptology: Proceedings of EUROCRYPT '93, Springer-Verlag, Berlin, Pages 386-397, 1994.

MENEZES, A. **Handbook of Applied Cryptography** - CRC Press, 1996 http://www.cacr.math.uwaterloo.ca/hac/ [capturado em 23 jun 2002]

MURPHY, Sean, ROBSHAW, Matt **New Observations on Rijndael** – preliminary draft of the Information Security Group, Royal Holloway, University of London, Egham, Surrey, TW20 0EX, U.K. – 7 August - 2000

NECHVATAL, J., BARKER, E., DODSON, D., DWORKIN, M., FOTI, J., ROBACK, E. **Status Report on The 1st Round of the Development of the Advanced Encryption Standard** - NIST, August 1999. - Journal of Research of the National Institute of Standards and Technology, Volume 104, Number 5, September-October 1999 - Disponível em http://csrc.nist.gov/cryptotoolkit/aes/round1/r1report.htm e http://nvl.nist.gov/pub/nistpubs/jres/104/5/cnt104-5.htm [capturado em 23 set 2003]

NEUHAUS, Stephan. **Statistical Properties of IDEA Session Keys in PGP**, 13 Jun 1993 Stephan Neuhaus, Hilgardring 32 W67657 Kaiserslautern – correio eletrônico: neuhaus@informatik.uni-kl.de

NIST's Information Tchnology Laboratory Bulletim – The AES: A Status Report – US Department of Commerce – Agosto 1999

OSWALD, Elisabeth, DAEMEN, Joan, RIJMEN, Vincent. **AES – The State of the Art of Rijndael's Security** – October 30, 2002 disponível em http://www.a-sit.at/technologieb/evaluation/aes_report_e.pdf [capturado em 23 abr 2003]

PATTERSON, Kenneth G. **Imprimitive Permutation Groups and Trapdoors in Iterated Block Ciphers** – Relatório Técnico HPL-1999-12R1 do Hewlet Packard Laboratories

Bristol – Janeiro - 1999 disponível em http://www.hpl.hp.com/techreports/1999/HPL-1999-12R1.html [capturado em 23 abr 2003]

PATTERSON, David A., HENESSY, John L. **Organização e Projeto de Computadores – A interface hardware/software – Segunda Edição** – Editora LTC – Rio de Janeiro, 2000

PRENEEL, B., BOSSELAERS, A., RIJMEN, V., VAN RONPAY, B., GRANBOULAN, L., STERN, J., MURPHY, S., DICHTL, M., SERF, P., BIHAM, E., DUNKELMAN, O., FURMAN, V., KOEUNE, F., PIRET, G.,QUISQUATER, J-J., KNUDSEN, L., RADDUM, H., **Comments by the NESSIE Project on the AES Finalists** – May 24, 2000 disponível em Bart.Preneel@esat.kuleuven.ac.be , http://www.cryptonessie.org [capturado em 23 nov 2002]

PRENEEL, Bart **NESSIE Announces Final Selection of Cripto Algorithms** – Feb 27, 2003 disponível em Bart.Preneel@esat.kuleuven.ac.be , http://www.cryptonessie.org. [capturado em 23 nov 2002]

PYKA, Stephan**. The Statistical Evaluation of the NESSIE Submission BMGL** – NESSIE Document *(Document Reference NES/DOC/SAG/WP3/039/1),* Agosto 2001 – Disponível na Internet [capturado em 23 nov 2002]

RITTER, Terry**. Ritters' Crypto Glossary and Dictionary of Technical Cryptography,** atualizado em 14 de maio de 2003 (Copyright 1995) - disponível em http://www.ciphersbyritter.com

RSAS - Regulamento Para Salvaguarda de Assuntos Sigilosos do Ministério do Exército, *1ª Edição*, EGGCF, Outubro 1989

SHAMIR, A., BIHAM, E. **Power analysis of the keys scheduling of the AES Candidates** – The Second AES Conference, March 22-23, 1999, p. 115-121.

SHANNON, C. E. **"A Mathematical Theory of Communication"** – Reprinted with corrections from the Bell System Technical Journal, vol. 27, pag. 379-423, 623-656, Oct., 1948

SHANNON, C. E. **"Communication Theory of Secrecy Systems"** – Bell System Technical Journal , vol. 28, pag. 656-715, Oct., 1949

SCHNEIER, Bruce. **Applied Cryptography**, Addison-Wesley Publishing Company – Reading, Massachussets, 1996

SCHNEIER, Bruce, KELSEY, John, WHITING, Doug, WAGNER, David, HALL, Chris, FERGUSON, Niels. **Performance Comparison of the AES Submissions Version 2.0 - February 1, 1999** disponível em http://www.counterpane.com/aes-performance.pdf [capturado em 23 fev 2002]

SCHNEIER, Bruce **Self-Study Course in Block Cipher Cryptanalysis -** Cryptologia, v.24, n.1, Jan 2000, pp. 18-34.

SCHNEIER[A], Bruce. **AES News – Cripto-Gram Newsletter - Counterpane Internet Security, September 15, 2002 –** http://www.counterpane.com [capturado em 10 fev 2003]

SCHNEIER[B], Bruce. **AES News –** ***Cripto-Gram Newsletter*** **- Counterpane Internet Security, October 15, 2002** – http://www.counterpane.com [capturado em 10 fev 2003]

SINGH, Simon. **O Livro dos Códigos**, Ed. Record - Rio de Janeiro – 2001

STINSON, Douglas. **Cryptography Theory and Practice** CRC Press, Inc., 2000, Corporate Blvd., Boca Raton, FL, 33431-9868, ISBN 0-8493-8521-0, 1995

STOA (SCIENTIFIC and Technological Options Assessment) – Development of Surveillance Technology and Risk of Abuse of Economic Information – The State of the Art in Communications Intelligence (COMINT*)* – Report of the European Parliament – Luxemburg, April 1999 – disponível em http://www.iptvreports.mcmail.com /ic2kreport.htm, http://www.europarl.eu.int/dg4/stoa/en [capturado em 15 mar 2003]

TAVARES, Stafford E., HEYS, Howard M**. Substitution-Permutation Networks Resistant to Differential and Linear Cryptanalysis –** Journal of Cryptology, 9 (1996) – International Association for Cryptologic Research - www1.cs.columbia.edu/~dcook/candexam/ Y_10_sub-perm_resist.pdf [capturado em 10 fev 2003]

VAUDENAY, S. **An Experiment on DES Statistical Cryptanalysis***,* publicado no Proceedings of the 3rd ACM Conference on Computer and Communications Security - *CCS' 1996 , New Delhi, India, pp. 139-147* – ACM Press – New York – 1996.

WEISS, Yosseloff. **Matemática Finita** Ed. Guanabara 2, Rio de Janeiro, 1978

WERNSDORF, R**. The Round Functions of SERPENT Generate the Alternating Group***,* disponível em http://csrc.nist.gov/CryptoToolkit/aes/round2/comments/20000512-rwernsdorf.pdf [capturado em 10 fev 2003]

WOOLSEY, R. James. **Why We Spy on Our Allies -** The Wall Street Journal , March 17, 2000. Disponível em www.geocities.com/cpa_blacktown/20000324woolswsjus.htm [capturado em 10 fev 2003]

XAVIER JUNIOR, Judson B., GOMES, C. H. Franco. **Segurança de Dados em Redes de Computadores de Aplicação Militar** – Projeto de Final de Curso do Departamento de Engenharia de Sistemas – IME – Rio de Janeiro, 1999

XEXÉO, J. A. M. **Criptoanálise de um Modelo Reduzido de DES – tese de Mestrado** - Instituto Militar de Engenharia, Rio de Janeiro, Brasil, 1983

12 APÊNDICES

12.1 APÊNDICE 1: COMPLEMENTO DA DESCRIÇÃO ALTERNATIVA DO AES

Neste apêndice são apresentados detalhes complementares da descrição alternativa do AES que é apresentada no capítulo 4.

12.1.1 DETALHAMENTO DA FUNÇÃO DE EXPANSÃO DE CHAVES

A partir da chave de *K* (um bloco de 128 bites) fornecida pelo usuário, o AES realiza uma operação de expansão que tem por objetivo produzir *Nr+1* blocos distintos de 128 bites (que chamamos *sub-chaves*). A operação XOR é realizada uma vez antes do laço, *Nr-1* vezes dentro do laço e uma vez ao final da cifragem. Temos, portanto, *Nr+1* sub-chaves distintas geradas a partir da chave fornecida. No caso específico que está sendo tratado (bloco e chave de 128 bites), são geradas 11 sub-chaves de 128 bites, a saber: K_0, K_1, K_2...K_{10}. K_0 é a própria chave fornecida pelo usuário. Para $0 < i \leq Nr$, K_i é gerada a partir de $K_{(i-1)}$ e de um baite retirado de uma tabela. Esses baites são constantes do padrão (FIPS 197, 2001) e estão apresentados na TAB. 12.1.1.

Para explicar o processo de expansão da chave, será útil a seguinte notação: cada sub-chave *Ki* é formada pela concatenação de quatro palavras (blocos de 32 bites). Designaremos cada um destes blocos por *W[i,j]*, de tal forma que *W[i,j]* representa a *j-ésima palavra* da *i-ésima* sub-chave.

TAB 12.1.1: Relação de constantes de fase do AES.

Designação	*Baite*	
R Con 1	0,0,0,0,0,0,0,1	
R Con 2	0,0,0,0,0,0,1,0	Nota: O termo *R Con* vem de *Round Constant. R Con i* é a constante a ser somada módulo dois na *i*-ésima iteração.
R Con 3	0,0,0,0,0,1,0,0	
R Con 4	0,0,0,0,1,0,0,0	
R Con 5	0,0,0,1,0,0,0,0	
R Con 6	0,0,1,0,0,0,0,0	Obs: $R\ Con\ i = x^{i-1}\ mod\ m(x)$
R Con 7	0,1,0,0,0,0,0,0	
R Con 8	1,0,0,0,0,0,0,0	
R Con 9	0,0,0,1,1,0,1,1	
R Con 10	0,0,1,1,0,1,1,0	

No processo de geração das sub-chaves, para *i>0* e *j=0*, o primeiro bloco (*W[i,0]*) de 32 bites de cada sub-chave é gerado da seguinte maneira (vide FIG. 12.1.1):

- toma-se a última palavra (*W[i-1,3]*) da sub-chave anterior ;
- realiza-se a transformação SUBSTITUIÇÃO em cada um dos seus quatro baites;
- realiza-se a operação de ROTAÇÃO[209] na palavra resultante;
- realiza-se uma operação XOR do primeiro baite desta palavra com *R Con i*;
- realiza-se uma operação XOR do resultado com a primeira palavra (*W[i-1,0]*) da sub-chave anterior.

O bloco de 32 bites encontrado é a primeira palavra de *Ki* (*W[i,0]*).

As demais palavras da sub-chave *Ki* são obtidas de maneira bem mais simples: $W[i,j] = W[i-1,j] \oplus W[i,j-1]$, *com* $i > 0$, $j > 0$. O fluxograma na FIG. 12.1.1 ilustra a expansão da chave no AES. Para complementação é apresentado um exemplo de expansão de chave no tópico 12.1.4.

12.1.2 TRANSFORMAÇÕES INVERSAS

A seguir são apresentadas as descrições das transformações inversas da substituição, da permutação e da multiplicação.

12.1.2.1 SUBSTITUIÇÃO INVERSA

Seja S^{-1} a operação inversa da SUBSTITUIÇÃO *S*. S^{-1} é idêntica à *S*, diferindo apenas pela tabela de substituição utilizada.

[209] A operação de ROTAÇÃO consiste em um deslocamento cíclico dos baites à esquerda, caracterizado da seguinte maneira: Seja a palavra formada pelos baites *B0,B1,B2,B3*. O resultado da ROTAÇÃO desta palavra é a palavra *B1,B2,B3,B0*.

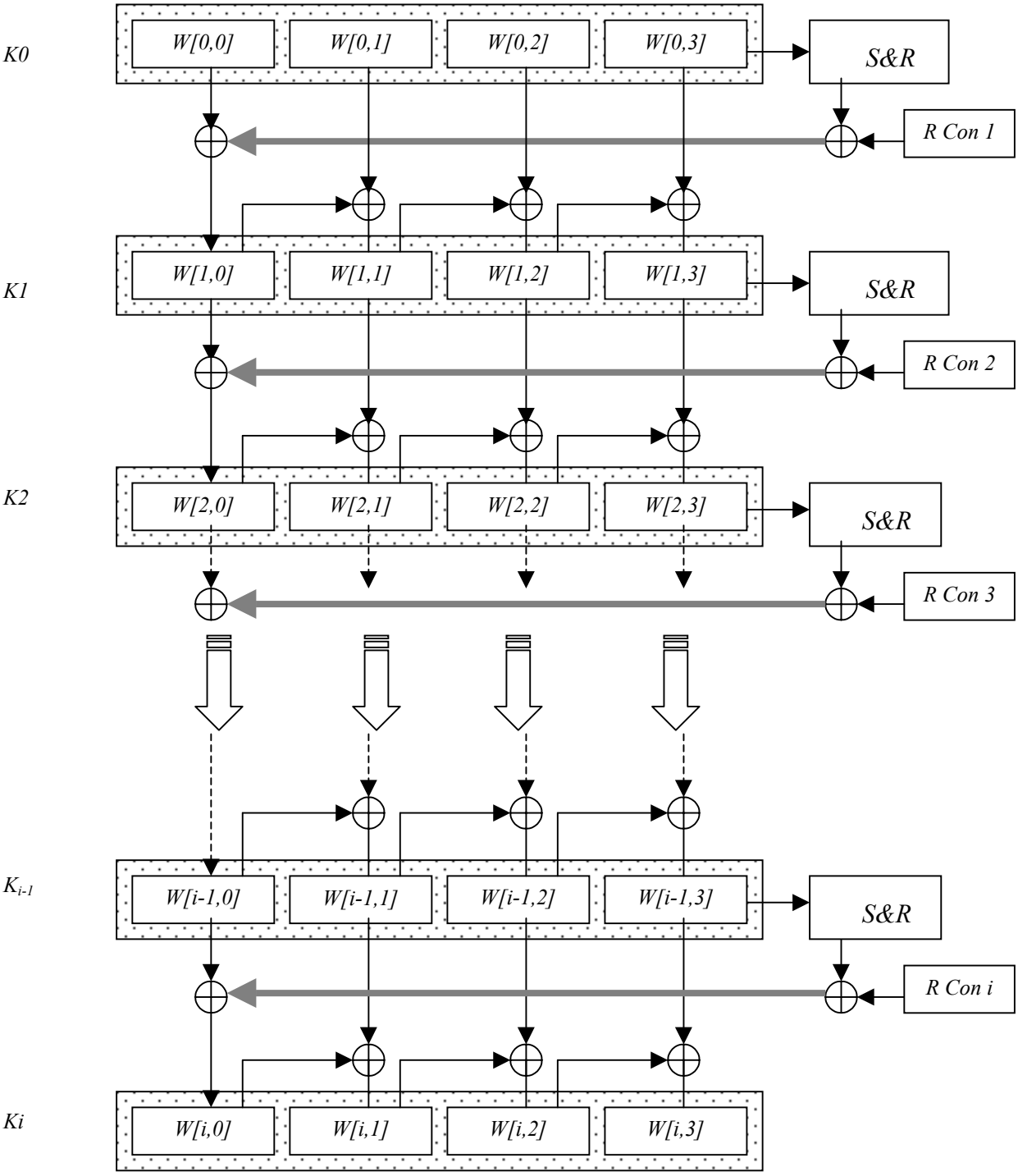


FIG 12.1.1 - Fluxograma[210] da expansão da chave do AES

[210] S&R= substituição seguida de rotação.

TAB 12.1.2: Tabela para operação de substituição (inversa).

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

12.1.2.2 PERMUTAÇÃO INVERSA

$$P^{-1}(E) = E0|\ E13|\ E10|\ E7|\ E4|\ E1|\ E14|\ E11|\ E8|\ E5|\ E2|\ E15|\ E12|\ E9|\ E6|\ E3$$

As operações P e P^{-1} podem também ser definidas por bites. Neste caso, são representadas por vetores de 128 posições.

12.1.2.3 MULTIPLICAÇÃO INVERSA

Usa-se a notação $[M]^{-1}$ para representar a operação inversa da MULTIPLICAÇÃO *M*. Seja *W* uma palavra. Temos então que $[M]^{-1}\ x\ (MxW) = W$. Ou seja, $M^{-1}(M(W))=W$. A multiplicação inversa encontra-se ilustrada na FIG. 12.1.2.

s0		.	1	1	1	.	.	.	.	1	1	.	1	.	.	.	.	1	.	1	1	.	.	.	.	1	.	.	1	.	.	.	.		e0
s1		1	.	1	1	1	.	.	.	1	1	1	.	1	.	.	.	1	1	.	1	1	.	.	.	1	1	.	.	1	.	.	.		e1
s2		.	1	.	1	1	1	.	.	1	1	1	1	.	1	.	.	.	1	1	.	1	1	.	.	1	1	1	.	.	1	.	.		e2
s3		.	.	1	.	1	1	1	.	1	1	1	1	1	.	1	.	1	.	1	1	.	1	1	.	.	1	1	1	.	.	1	.		e3
s4		.	1	1	.	.	1	1	1	.	.	1	.	1	1	.	1	1	1	1	.	1	.	1	1	1	.	1	.	1	.	.	1		e4
s5		.	1	.	.	.	.	1	1	1	1	.	.	.	1	1	.	.	1	.	.	.	1	.	1	1	1	.	.	.	1	.	.		e5
s6		.	.	1	.	.	.	.	1	1	1	1	.	.	.	1	1	1	.	1	.	.	.	1	.	.	1	1	.	.	.	1	.		e6
s7		1	1	1	.	.	.	.	.	1	.	1	.	.	.	.	1	.	1	1	.	.	.	.	1	.	.	1	.	.	.	.	1		e7
s8		1	.	.	1	.	.	.	.	.	1	1	1	.	.	.	.	1	1	.	1	.	.	.	.	.	1	.	1	1	.	.	.		e8
s9		1	1	.	.	1	.	.	.	1	.	1	1	1	.	.	.	1	1	1	.	1	.	.	.	1	1	.	1	1	.	.	.		e9
s10		1	1	1	.	.	1	.	.	.	1	.	1	1	1	.	.	1	1	1	1	.	1	.	.	.	1	1	.	1	1	.	.		e10
s11		.	1	1	1	.	.	1	.	.	.	1	.	1	1	1	.	1	1	1	1	1	.	1	.	1	.	1	1	.	1	1	.		e11
s12		1	.	1	.	1	.	.	1	.	1	1	.	.	1	1	1	.	.	1	.	1	1	.	1	1	1	1	.	1	.	1	1		e12
s13		1	1	.	.	.	1	.	.	.	1	.	.	.	.	1	1	1	1	.	.	.	1	1	.	.	1	.	.	.	1	.	1		e13
s14		.	1	1	.	.	.	1	.	.	.	1	.	.	.	.	1	1	1	1	.	.	.	1	1	1	.	1	.	.	.	1	.		e14
s15		.	.	1	.	.	.	.	1	1	1	1	.	.	.	.	.	1	.	1	.	.	.	.	1	.	1	1	.	.	.	.	1		e15
s16		1	.	1	1	.	.	.	.	1	.	.	1	.	.	.	.	.	1	1	1	.	.	.	.	1	1	.	1	.	.	.	.		e16
s17	=	1	1	.	1	1	.	.	.	1	1	.	.	1	.	.	.	1	.	1	1	1	.	.	.	1	1	1	.	1	.	.	.	x	e17
s18		.	1	1	.	1	1	.	.	1	1	1	.	.	1	.	.	.	1	.	1	1	1	.	.	1	1	1	1	.	1	.	.		e18
s19		1	.	1	1	.	1	1	.	.	1	1	1	.	.	1	.	.	.	1	.	1	1	1	.	1	1	1	1	1	.	1	.		e19
s20		1	1	1	.	1	.	1	1	1	.	1	.	1	.	.	1	.	1	1	.	.	1	1	1	.	.	1	.	1	1	.	1		e20
s21		.	1	.	.	.	1	.	1	1	1	.	.	.	1	.	.	.	1	.	.	.	.	1	1	1	1	.	.	.	1	1	.		e21
s22		1	.	1	.	.	.	1	.	.	1	1	.	.	.	1	.	.	.	1	.	.	.	.	1	1	1	1	.	.	.	1	1		e22
s23		.	1	1	.	.	.	.	1	.	.	1	.	.	.	.	1	1	1	1	.	.	.	.	.	1	.	1	.	.	.	.	1		e23
s24		1	1	.	1	.	.	.	.	1	.	1	1	.	.	.	.	1	.	.	1	.	.	.	.	.	1	1	1	.	.	.	.		e24
s25		1	1	1	.	1	.	.	.	1	1	.	1	1	.	.	.	1	1	.	.	1	.	.	.	1	.	1	1	1	.	.	.		e25
s26		1	1	1	1	.	1	.	.	.	1	1	.	1	1	.	.	1	1	1	.	.	1	.	.	.	1	.	1	1	1	.	.		e26
s27		1	1	1	1	1	.	1	.	1	.	1	1	.	1	1	.	.	1	1	1	.	.	1	.	.	.	1	.	1	1	1	.		e27
s28		.	.	1	.	1	1	.	1	1	1	1	.	1	.	1	1	1	.	1	.	1	.	.	1	.	1	1	.	.	1	1	1		e28
s29		1	1	.	.	.	1	1	.	.	1	.	.	.	1	.	1	1	1	.	.	.	1	.	.	.	1	.	.	.	.	1	1		e29
s30		1	1	1	.	.	.	1	1	1	.	1	.	.	.	1	.	.	1	1	.	.	.	1	.	.	.	1	.	.	.	.	1		e30
s31		1	.	1	.	.	.	.	1	.	1	1	.	.	.	.	1	.	.	1	.	.	.	.	1	1	1	1	.	.	.	.	.		e31

FIG. 12.1.2 Ilustração da multiplicação pela matriz M^{-1}.

12.1.3 DEMONSTRAÇÕES DAS SIMPLIFICAÇÕES DE *SHIFT ROWS* E *MIX COLUMNS*

A seguir são apresentadas as demonstrações das equivalências entre as formas originais e as formas simplificadas das transformações *Shift Rows* e *Mix Columns*.

12.1.3.1 A TRANSFORMAÇÃO *SHIFTROWS* E A PERMUTAÇÃO *P*

Na descrição apresentada no (FIPS 197, 2001), esta transformação consiste em dispor as palavras do bloco argumento *E* por colunas de maneira que cada bloco fica representado como uma matriz de baites com 4 linhas e 4 colunas, como ilustrado abaixo. A transformação consiste em deslocar, ciclicamente, *i* posições à esquerda a linha *i* ($i \in \{0,1,2,3\}$) do argumento para obter o resultado da transformação *S=P(E)*.

E0	*E4*	*E8*	*E12*	Linha 0	*E0*	*E4*	*E8*	*E12*		*S0*	*S4*	*S8*	*S12*
E1	*E5*	*E9*	*E13*	Linha 1	*E5*	*E9*	*E13*	*E1*	=	*S1*	*S5*	*S9*	*S13*
E2	*E6*	*E10*	*E14*	Linha 2	*E10*	*E14*	*E2*	*E6*		*S2*	*S6*	*S10*	*S14*
E3	*E7*	*E11*	*E15*	Linha 3	*E15*	*E3*	*E7*	*E11*		*S3*	*S7*	*S11*	*S15*
Bloco *E*					*P(E)*				=	Bloco *S*			

O bloco *S* é então obtido retirando-se os baites de cima para baixo, dentro de cada coluna, começando pela coluna da esquerda e seguindo à direita.

A transformação é, portanto, uma permutação realizada nos baites do bloco argumento *A,* de acordo com a seqüência abaixo:

S=P(E)=[*E0, E5, E10, E15, E4, E9, E14, E3, E8, E13, E2, E7, E12, E1, E6, E11*]

12.1.3.2 A TRANSFORMAÇÃO *MIX COLUMNS* E A MULTIPLICAÇÃO PELA MATRIZ *M*

A seguir é demonstrada a equivalência entre a transformação *Mix Columns* e a MULTIPLICAÇÃO pela matriz *M.*

12.1.3.2.1 A DESCRIÇÃO DA TRANSFORMAÇÃO MIX COLUMNS NO FIPS 197 (FIPS 197, 2001)

Esta operação é realizada em cada palavra do argumento e descrita no FIPS 197 como um produto matricial no corpo finito CG(2^8) com módulo $m(x)= x^8 + x^4 + x^3 + x^1 + x^0$. A matriz de transformação é fixa e a operação se dá como ilustrado a seguir.

$$\begin{vmatrix} S0 \\ S1 \\ S2 \\ S3 \end{vmatrix} = \begin{vmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{vmatrix} \begin{vmatrix} E0 \\ E1 \\ E2 \\ E3 \end{vmatrix}$$

$$\begin{vmatrix} S4 \\ S5 \\ S6 \\ S7 \end{vmatrix} = \begin{vmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{vmatrix} \begin{vmatrix} E4 \\ E5 \\ E6 \\ E7 \end{vmatrix}$$

$$\begin{vmatrix} S8 \\ S9 \\ S10 \\ S11 \end{vmatrix} = \begin{vmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{vmatrix} \begin{vmatrix} E8 \\ E9 \\ E10 \\ E11 \end{vmatrix}$$

$$\begin{vmatrix} S12 \\ S13 \\ S14 \\ S15 \end{vmatrix} = \begin{vmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{vmatrix} \begin{vmatrix} E12 \\ E13 \\ E14 \\ E15 \end{vmatrix}$$

A matriz de transformação é a mesma para as quatro *palavras* do bloco. Pode-se ver também que cada *baite* do bloco *S* é afetado apenas pelos 4 *baites* da respectiva *palavra* no bloco argumento *E*.

S0 = 2.E0 + 3.E1 + 1.E2 + 1.E3
S1 = 1.E0 + 2.E1 + 3.E2 + 1.E3
S2 = 1.E0 + 1.E1 + 2.E2 + 3.E3
S3 = 3.E0 + 1.E1 + 1.E2 + 2.E3
S4 = 2.E4 + 3.E5 + 1.E6 + 1.E7
S5 = 1.E4 + 2.E5 + 3.E6 + 1.E7
S6 = 1.E4 + 1.E5 + 2.E6 + 3.E7
S7 = 3.E4 + 1.E5 + 1.E6 + 2.E7
S8 = 2.E8 + 3.E9 + 1.E10 + 1.E11
S9 = 1.E8 + 2.E9 + 3.E10 + 1.E11
S10 = 1.E8 + 1.E9 + 2.E10 + 3.E11
S11=3.E8 + 1.E9 + 1.E10 + 2.E11
S12 = 2.E12 + 3.E13 + 1.E14 + 1.E15
S13 = 1.E12 + 2.E13 + 3.E14 + 1.E15
S14 = 1.E12 + 1.E13 + 2.E14 + 3.E15
S15 = 3.E12 + 1.E13 + 1.E14 + 2.E15

Cada baite é tratado como o vetor dos coeficientes de um polinômio do $CG(2^8)$. Assim, o baite[211] $E0 = \{e0,e1,e2,e3,e4,e5,e6,e7\}$, por exemplo, é representado pelo polinômio binário $e0.x^7 + e1.x^6 + e2.x^5 + e3.x^4 + e4.x^3 + e5.x^2 + e6.x^1 + e7.x^0$.

A multiplicação do baite *E0* por *2* no $CG(2^8)$ é equivalente a multiplicar o polinômio acima por *x* (o que resulta em $e0.x^8 + e1.x^7 + e2.x^6 + e3.x^5 + e4.x^4 + e5.x^3 + e6.x^2 + e7.x^1 + 0.\ x^0$) e reduzir este polinômio módulo *m(x)*. A *redução módulo m(x)* consiste realizar um XOR dos coeficientes do polinômio a ser reduzido com os coeficientes de mesmo índice do polinômio *m(x),* caso o coeficiente do termo x^8 seja *1* (caso em que não seria possível representar o polinômio por um baite de 8 bites). Caso o coeficiente do termo x^8 seja *0*, a redução já está realizada.

Exemplos:

b) 2 . 01100111 = 011001110 = 11001110 (simples deslocamento à esquerda)

c) 2 . 11001110 = 110011100 + 100011011 = 010000111 = 10000111 (neste caso, além do deslocamento, foi necessário reduzir módulo $x^8 + x^4 + x^3 + x^1 + x^0$, o que equivale[212] a realizar um XOR com o baite *100011011*)

[211] Não confundir com a constante *A0* hexadecimal que é 10100000.
[212] A operação XOR que aparece neste exemplo é equivalente a anular o bite mais significativo de 110011100 e fazer um XOR com 00011011.

Observação I - Uma vez que a multiplicação por 2 equivale ao deslocamento à esquerda seguido de uma operação XOR com *100011011* condicionada ao valor do bite mais significativo ser 1, pode-se concluir que

$$2.\{e0|e1|e2|e3|e4|e5|e6|e7\}=\{ e1|\ e2|\ e3|\ (e4+e0)|\ (e5+e0)|\ e6|\ (e7+e0)|\ e0\},$$

pois o lado direito da equação será $\{e1|e2|e3|e4|e5|e6|e7|0\}$, ***se e0=0***, e

$$\{e1|e2|e3|e4+1|e5+1|e6|e7+1|1\},\ \textbf{se e0=1}.$$

Observação II – Pelo mesmo raciocínio seguido na *observação I*, pode-se afirmar que

$$4.\{e0|e1|e2|e3|e4|e5|e6|e7\} = 2.(2.\{e0|e1|e2|e3|e4|e5|e6|e7\})=$$

$$2.\{e1|\ e2|\ e3|\ (e4+e0)|\ (e5+e0)|\ e6|\ (e7+e0)|\ e0\}=$$

$$\{e2|\ e3|\ (e4+e0)|\ ((e5+e0)+e1)|\ (e6+e1)|\ (e7+e0)|\ (e0+e1)|\ e1\}$$

e

$$8.\{e0|e1|e2|e3|e4|e5|e6|e7\} = 2.(4.\{e0|e1|e2|e3|e4|e5|e6|e7\}) =$$

$$\{ e3|\ (e4+e0)|\ ((e5+e0)+e1)|\ ((e6+e1)+e2)|\ (e7+e0)+e2)|\ (e0+e1)|\ (e1+e2)|\ e2\}$$

Fim da observação II.

Para multiplicar um baite por uma constante, inicialmente decompõe-se esta constante em parcelas que sejam potências de 2 e utilizam-se os resultados apresentados nas observações precedentes. Todas as constantes utilizadas nas transformações *Mix Columns* e *Inv Mix Columns* são menores que 16, e, consequentemente, podem ser representadas como a soma das potências de 2: *1, 2, 4 e 8*. Assim, qualquer produto de constante por baite pode ser reduzido a uma seqüência de XOR's entre os bites do baite, aproveitando os resultados apresentados nas observações I e II. Exemplos:

$$\textbf{3.E0} = 2.E0+E0 = \{ e1|\ e2|\ e3|\ (e4+e0)|\ (e5+e0)|\ e6|\ (e7+e0)|\ e0\}$$
$$+\ \{e0|e1|e2|e3|e4|e5|e6|e7\}$$

$$\textbf{9.E0} = 8.E0 + E0 =$$
$$\{ e3|\ e4+e0|\ e5+e0+e1|\ e6+e1+e2|\ e7+e0+e2|\ e0+e1|\ e1+e2|\ e2\}$$
$$+ \{e0|e1|e2|e3|e4|e5|e6|e7\}$$

$$\boldsymbol{e_h.E0} = 8.E0 + 4.E0 + 2.E0 =$$

$$\{ e3|\ e4+e0|\ e5+e0+e1|\ e6+e1+e2|\ e7+e0+e2|\ e0+e1|\ e1+e2|\ e2\}$$

$$+$$

$$\{e2|\ e3|\ (e4+e0)|\ ((e5+e0)+e1)|\ (e6+e1)|\ (e7+e0)|\ (e0+e1)|\ e1\}$$

$$+$$

$$\{ e1|\ e2|\ e3|\ e4+e0|\ e5+e0|\ e6|\ e7+e0|\ e0\}$$

12.1.3.2.2 DESCRIÇÃO LÓGICA DA TRANSFORMAÇÃO *MIXCOLUMNS*

Seja a multiplicação no CG(2^8)

$$\begin{vmatrix} S0 \\ S1 \\ S2 \\ S3 \end{vmatrix} = \begin{vmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{vmatrix} \begin{vmatrix} E0 \\ E1 \\ E2 \\ E3 \end{vmatrix}$$

Cada um dos 4 baites de resposta da transformação, pelo exposto em 12.1.3.2.1, pode ser equivalentemente calculado determinando-se cada um dos seus bites por meio de uma soma módulo dois de bites do argumento. A seguir, como ilustração, mostra-se a determinação do baite *S0. Cada bite de S0 é dado por uma expressão booleana dos bites do argumento (32 bites).*

									Parcela da soma
e1	*e2*	*e3*	*e4+e0*	*e5+e0*	*e6*	*e7+e0*	*e0*	=	*2 x E0*
e9	*e10*	*e11*	*e12+e8*	*e13+e8*	*e14*	*e15+e8*	*e8*	=	*2 x E1*
e8	*e9*	*e10*	*e11*	*e12*	*e13*	*e14*	*e15*	=	*E1*
e16	*e17*	*e18*	*e19*	*e20*	*e21*	*e22*	*e23*	=	*E2*
e24	*e25*	*e26*	*e27*	*e28*	*e29*	*e30*	*e31*	=	*E3*
s0	*s1*	*s2*	*s3*	*s4*	*s5*	*s6*	*s7*	=	*Byte S0*

TAB. 12.1.3: Expressões booleanas da transformação S = *MixColumns* (E).

S0	s0	=	e1	⊕	e9	⊕	e8	⊕	e16	⊕	e24	
	s1	=	e2	⊕	e10	⊕	e9	⊕	e17	⊕	e25	
	s2	=	e3	⊕	e11	⊕	e10	⊕	e18	⊕	e26	
	s3	=	e4	⊕	e12	⊕	e11	⊕	e19	⊕	e27	⊕ (e0⊕e8)
	s4	=	e5	⊕	e13	⊕	e12	⊕	e20	⊕	e28	⊕ (e0⊕e8)
	s5	=	e6	⊕	e14	⊕	e13	⊕	e21	⊕	e29	
	s6	=	e7	⊕	e15	⊕	e14	⊕	e22	⊕	e30	⊕ (e0⊕e8)
	s7	=	0	⊕	0	⊕	e15	⊕	e23	⊕	e31	⊕ (e0⊕e8)
S1	s8	=	e0	⊕	e9	⊕	e17	⊕	e16	⊕	e24	
	s9	=	e1	⊕	e10	⊕	e18	⊕	e17	⊕	e25	
	s10	=	e2	⊕	e11	⊕	e19	⊕	e18	⊕	e26	
	s11	=	e3	⊕	e12	⊕	e20	⊕	e19	⊕	e27	⊕ (e8⊕e16)
	s12	=	e4	⊕	e13	⊕	e21	⊕	e20	⊕	e28	⊕ (e8⊕e16)
	s13	=	e5	⊕	e14	⊕	e22	⊕	e21	⊕	e29	
	s14	=	e6	⊕	e15	⊕	e23	⊕	e22	⊕	e30	⊕ (e8⊕e16)
	s15	=	e7	⊕	0	⊕	0	⊕	e23	⊕	e31	⊕ (e8⊕e16)
S2	s16	=	e0	⊕	e8	⊕	e17	⊕	e25	⊕	e24	
	s17	=	e1	⊕	e9	⊕	e18	⊕	e26	⊕	e25	
	s18	=	e2	⊕	e10	⊕	e19	⊕	e27	⊕	e26	
	s19	=	e3	⊕	e11	⊕	e20	⊕	e28	⊕	e27	⊕ (e16⊕e24)
	s20	=	e4	⊕	e12	⊕	e21	⊕	e29	⊕	e28	⊕ (e16⊕e24)
	s21	=	e5	⊕	e13	⊕	e22	⊕	e30	⊕	e29	
	s22	=	e6	⊕	e14	⊕	e23	⊕	e31	⊕	e30	⊕ (e16⊕e24)
	s23	=	e7	⊕	e15	⊕	0	⊕	0	⊕	e31	⊕ (e16⊕e24)
S3	s24	=	e1	⊕	e0	⊕	e8	⊕	e16	⊕	e25	
	s25	=	e2	⊕	e1	⊕	e9	⊕	e17	⊕	e26	
	s26	=	e3	⊕	e2	⊕	e10	⊕	e18	⊕	e27	
	s27	=	e4	⊕	e3	⊕	e11	⊕	e19	⊕	e28	⊕ (e0⊕e24)
	s28	=	e5	⊕	e4	⊕	e12	⊕	e20	⊕	e29	⊕ (e0⊕e24)
	s29	=	e6	⊕	e5	⊕	e13	⊕	e21	⊕	e30	
	s30	=	e7	⊕	e6	⊕	e14	⊕	e22	⊕	e31	⊕ (e0⊕e24)
	s31	=	0	⊕	e7	⊕	e15	⊕	e23	⊕	0	⊕ (e0⊕e24)

As duas parcelas designadas por *2 x E1* e *E1* compõem o resultado *3 x E1*. Observe que se *e0=1*, *2xE0* será maior que um *byte* e devemos fazer um *ou-exclusivo* com 00011011 para obter *2.E0* (o coeficiente de x^8 foi ignorado, pelo exposto na observação II). O mesmo se passa em relação ao bite *e8* do baite *E1*.

Utilizando este procedimento, são obtidas as 32 expressões booleanas da transformação linear definida por *[M]*. Vide a TAB. 12.3.

Esta transformação pode ser representada por uma equação matricial onde o produto de dois termos eqüivale a um "E" bite a bite e a soma é um "XOR" (soma módulo 2). Logo,

$[S]_{32x1} = [M]_{32x32}. [E]_{32x1}$. A matriz *M* está apresentada na FIG. 4.2 e a matriz para a operação inversa, obtida de maneira análoga, é dada na FIG 12.1.2.

A equivalência das transformações também foi verificada por meio do exemplo de execução apresentado no tópico 12.1.4, que é o mesmo exemplo apresentado no (FIPS 197, 2001).

12.1.4 EXECUÇÃO – EXEMPLO DO FIPS 197 (FIPS 197, 2001)

Neste tópico é apresentada a execução de uma cifragem de um bloco de 128 bites com uma chave de 128 bites pelo AES. A implementação foi feita seguindo a descrição apresentada neste trabalho. O bloco a ser cifrado (**Entrada**) e a chave (***K***) são os mesmos apresentados no exemplo do FIPS 197, o que colabora para ilustrar validade da descrição aqui apresentada.

Expansão da Chave

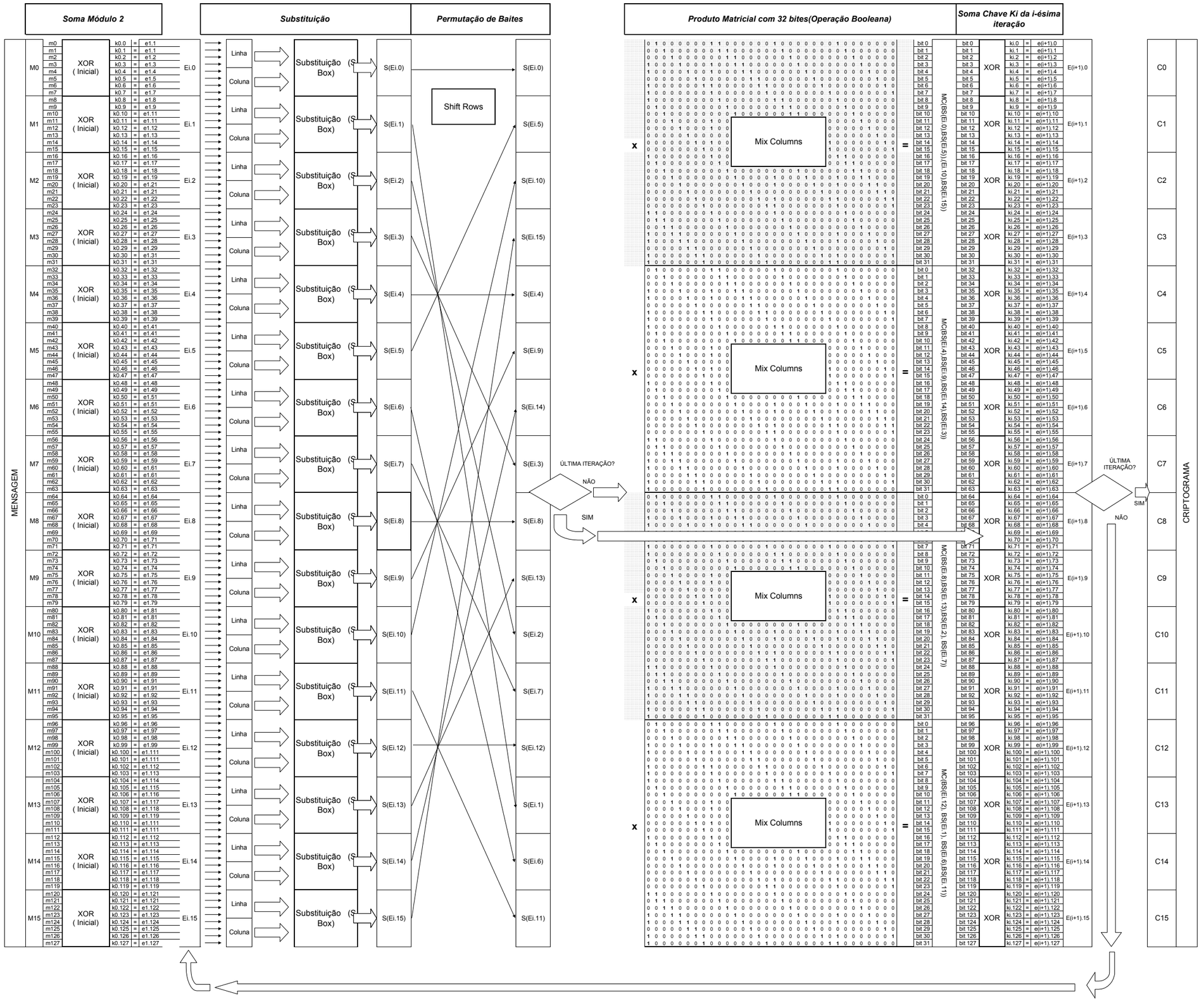
```
subchave k0=K:  2b7e1516 28aed2a6 abf71588 09cf4f3c
subchave k1:    a0fafe17 88542cb1 23a33939 2a6c7605
subchave k2:    f2c295f2 7a96b943 5935807a 7359f67f
subchave k3:    3d80477d 4716fe3e 1e237e44 6d7a883b
subchave k4:    ef44a541 a8525b7f b671253b db0bad00
subchave k5:    d4d1c6f8 7c839d87 caf2b8bc 11f915bc
subchave k6:    6d88a37a 110b3efd dbf98641 ca0093fd
subchave k7:    4e54f70e 5f5fc9f3 84a64fb2 4ea6dc4f
subchave k8:    ead27321 b58dbad2 312bf560 7f8d292f
subchave k9:    ac7766f3 19fadc21 28d12941 575c006e
subchave k10:   d014f9a8 c9ee2589 e13f0cc8 b6630ca6
```

Cifragem

```
Entrada:                3243f6a8 885a308d 313198a2 e0370734
APOS SOMASUBCHAVE   :   193de3be a0f4e22b 9ac68d2a e9f84808
APOS SUBSTITUIÇÃO  1:   d42711ae e0bf98f1 b8b45de5 1e415230
APOS PERMUTAÇÃO  1 :    d4bf5d30 e0b452ae b84111f1 1e2798e5
APOS MULTIPLICAÇÃO 1 :  046681e5 e0cb199a 48f8d37a 2806264c
APOS SOMASUBCHAVE 1:    a49c7ff2 689f352b 6b5bea43 026a5049
APOS SUBSTITUIÇÃO  2:   49ded289 45db96f1 7f39871a 7702533b
APOS PERMUTAÇÃO  2:     49db873b 45395389 7f02d2f1 77de961a
APOS MULTIPLICAÇÃO  2:  584dcaf1 1b4b5aac dbe7caa8 1b6bb0e5
APOS SOMASUBCHAVE 2:    aa8f5f03 61dde3ef 82d24ad2 6832469a
APOS SUBSTITUIÇÃO  3:   ac73cf7b efc111df 13b5d6b5 45235ab8
APOS PERMUTAÇÃO  3:     acc1d6b8 efb55a7b 1323cfdf 457311b5
APOS MULTIPLICAÇÃO  3:  75ec0993 200b6333 53c0cf7c bb25d0dc
APOS SOMASUBCHAVE 3:    486c4eee 671d9d0d 4de3b138 d65f58e7
APOS SUBSTITUIÇÃO  4:   52502f28 85a45ed7 e311c807 f6cf6a94
APOS PERMUTAÇÃO  4:     52a4c894 85116a28 e3cf2fd7 f6505e07
```

```
APOS MULTIPLICAÇÃO 4:   0fd6daa9 603138bf 6fc0106b 5eb31301
APOS SOMASUBCHAVE 4:    e0927fe8 c86363c0 d9b13550 85b8be01
APOS SUBSTITUIÇÃO  5:   e14fd29b e8fbfbba 35c89653 976cae7c
APOS PERMUTAÇÃO  5:     e1fb967c e8c8ae9b 356cd2ba 974ffb53
APOS MULTIPLICAÇÃO 5:   25d1a9ad bd11d168 b63a338e 4c4cc0b0
APOS SOMASUBCHAVE 5:    f1006f55 c1924cef 7cc88b32 5db5d50c
APOS SUBSTITUIÇÃO  6:   a163a8fc 784f29df 10e83d23 4cd503fe
APOS PERMUTAÇÃO  6:     a14f3dfe 78e803fc 10d5a8df 4c632923
APOS MULTIPLICAÇÃO 6:   4b868d6d 2c4a8980 339df4e8 37d218d8
APOS SOMASUBCHAVE 6:    260e2e17 3d41b77d e86472a9 fdd28b25
APOS SUBSTITUIÇÃO  7:   f7ab31f0 2783a9ff 9b4340d3 54b53d3f
APOS PERMUTAÇÃO  7:     f783403f 27433df0 9bb531ff 54aba9d3
APOS MULTIPLICAÇÃO 7:   1415b5bf 461615ec 274656d7 342ad843
APOS SOMASUBCHAVE 7:    5a4142b1 1949dc1f a3e01965 7a8c040c
APOS SUBSTITUIÇÃO  8:   be832cc8 d43b86c0 0ae1d44d da64f2fe
APOS PERMUTAÇÃO  8:      be3bd4fe d4e1f2c8 0a642cc0 da83864d
APOS MULTIPLICAÇÃO 8:    00512fd1 b1c889ff 54766dcd fa1b99ea
APOS SOMASUBCHAVE 8:     ea835cf0 0445332d 655d98ad 8596b0c5
APOS SUBSTITUIÇÃO 9:     87ec4a8c f26ec3d8 4d4c4695 9790e7a6
APOS PERMUTAÇÃO  9:      876e46a6 f24ce78c 4d904ad8 97ecc395
APOS MULTIPLICAÇÃO 9:    473794ed 40d4e4a5 a3703aa6 4c9f42bc
APOS SOMASUBCHAVE 9:     eb40f21e 592e3884 8ba113e7 1bc342d2
APOS SUBSTITUIÇÃO final: e9098972 cb31075f 3d327d94 af2e2cb5
APOS PERMUTAÇÃO  final:  e9317db5 cb322c72 3d2e895f af090794
APOS SOMASUBCHAVE FINAL:  3925841d 02dc09fb dc118597 196a0b32 = Criptograma
```

12.1.5 FLUXOGRAMA DETALHADO DE UMA CIFRAGEM COM O AES



12.2 APÊNDICE 2: IMPLEMENTAÇÃO DA DESCRIÇÃO ALTERNATIVA DO RIJNDAEL (CÓDIGO FONTE EM JAVA)

O código a seguir foi utilizado para verificar a correção da descrição alternativa do Rijndael. A saída gerada pela execução deste código encontra-se no 12.1.4.

Apesar de não serem utilizadas nesse teste, as transformações inversas também foram conferidas.

Esta implementação é inadequada para cifrar arquivos.

```
package rijndael;

public class Rijndael {

public static void main(String[] args) {

int  chave[]=  {0,0,1,0,  1,0,1,1,  0,1,1,1,  1,1,1,0,  0,0,0,1,  0,1,0,1,
0,0,0,1,
              0,1,1,0,   0,0,1,0,   1,0,0,0,   1,0,1,0,   1,1,1,0,   1,1,0,1,
0,0,1,0,
              1,0,1,0,   0,1,1,0,   1,0,1,0,   1,0,1,1,   1,1,1,1,   0,1,1,1,
0,0,0,1,
              0,1,0,1,   1,0,0,0,   1,0,0,0,   0,0,0,0,   1,0,0,1,   1,1,0,0,
1,1,1,1,
              0,1,0,0,  1,1,1,1,  0,0,1,1,  1,1,0,0};//exemplo apendice A e B
do FIPS

int  entrada[]={0,0,1,1,  0,0,1,0,  0,1,0,0,  0,0,1,1,  1,1,1,1,  0,1,1,0,
1,0,1,0,
               1,0,0,0,   1,0,0,0,   1,0,0,0,   0,1,0,1,   1,0,1,0,   0,0,1,1,
0,0,0,0,
               1,0,0,0,   1,1,0,1,   0,0,1,1,   0,0,0,1,   0,0,1,1,   0,0,0,1,
1,0,0,1,
               1,0,0,0,   1,0,1,0,   0,0,1,0,   1,1,1,0,   0,0,0,0,   0,0,1,1,
0,1,1,1,
               0,0,0,0,  0,1,1,1,  0,0,1,1,  0,1,0,0};//exemplo apendice B do
FIPS 197


 int cripto[]=new int [128];
 int temp []=new int[4];
 int temp1 []=new int[8];
 int temp2 []=new int[32];
 int bit,byt,w,i,j;
/*****************************************************************
 *                    expande a chave                            *
 *****************************************************************/
int K[][]= E(chave);                          // (EXPAND KEY)

// chave ja expandida
for (i=0;i<11;i++){//para gerar saida de ilustração das sub-chaves, apenas
    System.out.print("subchave k"+i+":  ");
    for (j=0;j<128;j++){// exibe sub-chaves
        if ((j%4)==0){
```

```
                temp[0]=K[i][j];
                temp[1]=K[i][j+1];
                temp[2]=K[i][j+2];
                temp[3]=K[i][j+3];
                System.out.print(hexa(temp));
                }
          if (j>0){

                if (((j+1)%32)==0) System.out.print(" ");
                }
         //System.out.print(K[i][j]);
     }
     System.out.println("");
 }// fim do loop de exibicao das sub-chaves

System.out.print("Entrada:  ");
w=exibe(entrada);

/*****************************************************************
 *                   CIFRA ENTRADA                               *
 *****************************************************************/

for (bit=0;bit<128;bit++)
          cripto[bit]=(entrada[bit]+K[0][bit])%2;//Sub-chave (ARK Inicial)


System.out.println("APOS ADD ROUND KEY INICIAL ");
w=exibe(cripto);


for (int r=1;r<10;r++){// Inicia as 9 iteracoes
      for (byt=0;byt<16;byt++){
           for (bit=0;bit<8;bit++)
                temp1[bit]=cripto[byt*8+bit];
           temp1=SBS(temp1);                      // Substituicao (BYTE SUB)
           for (bit=0;bit<8;bit++)
                cripto[byt*8+bit]=temp1[bit];
           }// fim do for byt

      System.out.println("APOS BYTE SUB  "+r);
      w=exibe(cripto);


      cripto=PSR(cripto);                    // Permutacao   (SHIFT ROWS)



      System.out.println("APOS SHIFT ROWS  "+r);
      w=exibe(cripto);


      for (w=0;w<4;w++){
                for (bit=0;bit<32;bit++)
                     temp2[bit]=cripto[w*32+bit];
                temp2=MC(temp2);                         // Multip por M (MIX
COLUMNS)
                for (bit=0;bit<32;bit++)
                     cripto[w*32+bit]=temp2[bit];
            }// fim do for w
```

```
            System.out.println("APOS MIX COLUMNS   "+r);
            w=exibe(cripto);

      for (bit=0;bit<128;bit++)
          cripto[bit]=(cripto[bit]+K[r][bit])%2;//Soma  Sub-chaveADD  ROUND
KEY )
          System.out.println("APOS ADD ROUND KEY "+r);
          w=exibe(cripto);
       }// fim das iteracoes ************************************

for ( byt=0;byt<16;byt++){
           for (bit=0;bit<8;bit++)
                temp1[bit]=cripto[byt*8+bit];
           temp1=SBS(temp1); // Substituicao (BYTE SUB exterior)
           for (bit=0;bit<8;bit++)
                cripto[byt*8+bit]=temp1[bit];
           }// fim do for byt

System.out.println("APOS BYTE SUB  final");
w=exibe(cripto);


cripto=PSR(cripto); // Permutacao   (SHIFT ROWS exterior)

System.out.println("APOS SHIFT ROWS  final");
w=exibe(cripto);


for (bit=0;bit<128;bit++)
          cripto[bit]=(cripto[bit]+K[10][bit])%2;  //  Soma  Sub-chave  (ADD
ROUND KEY FINAL)
System.out.println("APOS ARK FINAL");
w=exibe(cripto);



 /*****************************************************************
 *                     EXIBE CRIPTOGRAMA                          *
 *****************************************************************/
System.out.print("Criptograma:  ");

w=exibe(cripto);//exibicao do criptograma hexa. Obs: a variavel w é dont
care
for (bit=0;bit<128;bit++){// exibe CRIPTOGRAMA binario
        if ((bit%4)==0)
              System.out.print(" ");
        System.out.print(cripto[bit]);
    }// fim do loop de exibicao do criptograma
}// fim de main


/*****************************************************************
 *****            metodo expansão da chave - (EXPAND KEY)         ******
 **********            argumento = int[128] binario               ******
 **********            retorna = int [11][128] binario            ******
 *****************************************************************/
public static int[][] E(int KEY[]){
int XK[][] = new int [11][128];
int temp1[]= new int[8];
int temp2[]= new int[8];
```

```
int temp3[]= new int[8];
int temp4[]= new int[8];
int temptemp[]= new int[8];
String temp;
int i,j,c,c2;
int RCON [][]={{0,0,0,0,0,0,0,0},// este byte nao é usado (apenas para
coerencia dos indices)
               {0,0,0,0,0,0,0,1},// este byte e o RCon[1] pela notação do
FIPS 197 (Nk=4)
               {0,0,0,0,0,0,1,0},// este byte e o RCon[2]
               {0,0,0,0,0,1,0,0},// este byte e o RCon[3]
               {0,0,0,0,1,0,0,0},// este byte e o RCon[4]
               {0,0,0,1,0,0,0,0},// este byte e o RCon[5]
               {0,0,1,0,0,0,0,0},// este byte e o RCon[6]
               {0,1,0,0,0,0,0,0},// este byte e o RCon[7]
               {1,0,0,0,0,0,0,0},// este byte e o RCon[8]
               {0,0,0,1,1,0,1,1},// este byte e o RCon[9]
               {0,0,1,1,0,1,1,0}};//este byte e o RCon[10]

for (j=0;j<128;j++)// atribui a chave aa primeira sub-chave
    XK[0][j]=KEY[j];
for (i=1;i<11;i++){// i do FIPS para chave do round i. no ARK inicial i=0.
    for (c=0;c<8;c++){//atribui a word W[i-1,3] a temp
        temp1[c]=XK[i-1][c+96];
        temp2[c]=XK[i-1][c+104];
        temp3[c]=XK[i-1][c+112];
        temp4[c]=XK[i-1][c+120];
        }//fim do for c


temptemp=temp1;temp1=temp2;temp2=temp3;temp3=temp4;temp4=temptemp;//ROTWORD
    //BYTESUB WORD
    temp1=SBS(temp1);
    temp2=SBS(temp2);
    temp3=SBS(temp3);
    temp4=SBS(temp4);
    //fim de BYTESUB WORD
     for (c=0;c<8;c++){//soma RCon e faz XOR para gerar W[i,0],
     // que é a primeira WORD(de 32 bits) da sub-chave i
        temp1[c]=(temp1[c]+RCON[i][c])%2;
        XK[i][c]=(temp1[c]+XK[i-1][c])%2;
        XK[i][c+8]=(temp2[c]+XK[i-1][c+8])%2;
        XK[i][c+16]=(temp3[c]+XK[i-1][c+16])%2;
        XK[i][c+24]=(temp4[c]+XK[i-1][c+24])%2;
        }
     for (j=1;j<4;j++){//para gerar as WORDS W[i,1],W[i,2],W[i,3]
        for (c=0;c<32;c++){//
            c2=j*32+c;
            XK[i][c2]=(XK[i-1][c2]+XK[i][c2-32])%2;
            }//fim do for c
        }//fim do for j
     }//fim do for i

return XK;
}// ***************** fim do metodo expand key  **********************

/*********************************************************************
 *****   metodo substituicao - (BYTESUB) = S Box                  ******
 **********   argumento = int[8] binario                          ******
 ***********  restorna = int[8] binario                           ******
```

```
 ***********************************************************************/
 public static int[] SBS(int xy[]){//byte xy e o que sera sub. por zw
 int lin,col,z;
 int[] zw=new int[8];
 int BS[][] = {
{   0,1,1,0,0,0,1,1,   0,1,1,1,1,1,0,0,   0,1,1,1,0,1,1,1,   0,1,1,1,1,0,1,1,
1,1,1,1,0,0,1,0,    0,1,1,0,1,0,1,1,    0,1,1,0,1,1,1,1,    1,1,0,0,0,1,0,1,
0,0,1,1,0,0,0,0,    0,0,0,0,0,0,0,1,    0,1,1,0,0,1,1,1,    0,0,1,0,1,0,1,1,
1,1,1,1,1,1,1,0, 1,1,0,1,0,1,1,1, 1,0,1,0,1,0,1,1, 0,1,1,1,0,1,1,0},
{   1,1,0,0,1,0,1,0,   1,0,0,0,0,0,1,0,   1,1,0,0,1,0,0,1,   0,1,1,1,1,1,0,1,
1,1,1,1,1,0,1,0,    0,1,0,1,1,0,0,1,    0,1,0,0,0,1,1,1,    1,1,1,1,0,0,0,0,
1,0,1,0,1,1,0,1,    1,1,0,1,0,1,0,0,    1,0,1,0,0,0,1,0,    1,0,1,0,1,1,1,1,
1,0,0,1,1,1,0,0, 1,0,1,0,0,1,0,0, 0,1,1,1,0,0,1,0, 1,1,0,0,0,0,0,0},
{   1,0,1,1,0,1,1,1,   1,1,1,1,1,1,0,1,   1,0,0,1,0,0,1,1,   0,0,1,0,0,1,1,0,
0,0,1,1,0,1,1,0,    0,0,1,1,1,1,1,1,    1,1,1,1,0,1,1,1,    1,1,0,0,1,1,0,0,
0,0,1,1,0,1,0,0,    1,0,1,0,0,1,0,1,    1,1,1,0,0,1,0,1,    1,1,1,1,0,0,0,1,
0,1,1,1,0,0,0,1, 1,1,0,1,1,0,0,0, 0,0,1,1,0,0,0,1, 0,0,0,1,0,1,0,1},
{   0,0,0,0,0,1,0,0,   1,1,0,0,0,1,1,1,   0,0,1,0,0,0,1,1,   1,1,0,0,0,0,1,1,
0,0,0,1,1,0,0,0,    1,0,0,1,0,1,1,0,    0,0,0,0,0,1,0,1,    1,0,0,1,1,0,1,0,
0,0,0,0,0,1,1,1,    0,0,0,1,0,0,1,0,    1,0,0,0,0,0,0,0,    1,1,1,0,0,0,1,0,
1,1,1,0,1,0,1,1, 0,0,1,0,0,1,1,1, 1,0,1,1,0,0,1,0, 0,1,1,1,0,1,0,1},
{   0,0,0,0,1,0,0,1,   1,0,0,0,0,0,1,1,   0,0,1,0,1,1,0,0,   0,0,0,1,1,0,1,0,
0,0,0,1,1,0,1,1,    0,1,1,0,1,1,1,0,    0,1,0,1,1,0,1,0,    1,0,1,0,0,0,0,0,
0,1,0,1,0,0,1,0,    0,0,1,1,1,0,1,1,    1,1,0,1,0,1,1,0,    1,0,1,1,0,0,1,1,
0,0,1,0,1,0,0,1, 1,1,1,0,0,0,1,1, 0,0,1,0,1,1,1,1, 1,0,0,0,0,1,0,0},
{   0,1,0,1,0,0,1,1,   1,1,0,1,0,0,0,1,   0,0,0,0,0,0,0,0,   1,1,1,0,1,1,0,1,
0,0,1,0,0,0,0,0,    1,1,1,1,1,1,0,0,    1,0,1,1,0,0,0,1,    0,1,0,1,1,0,1,1,
0,1,1,0,1,0,1,0,    1,1,0,0,1,0,1,1,    1,0,1,1,1,1,1,0,    0,0,1,1,1,0,0,1,
0,1,0,0,1,0,1,0, 0,1,0,0,1,1,0,0, 0,1,0,1,1,0,0,0, 1,1,0,0,1,1,1,1},
{   1,1,0,1,0,0,0,0,   1,1,1,0,1,1,1,1,   1,0,1,0,1,0,1,0,   1,1,1,1,1,0,1,1,
0,1,0,0,0,0,1,1,    0,1,0,0,1,1,0,1,    0,0,1,1,0,0,1,1,    1,0,0,0,0,1,0,1,
0,1,0,0,0,1,0,1,    1,1,1,1,1,1,0,1,    0,0,0,0,0,0,1,0,    0,1,1,1,1,1,1,1,
0,1,0,1,0,0,0,0, 0,0,1,1,1,1,0,0, 1,0,0,1,1,1,1,1, 1,0,1,0,1,0,0,0},
{   0,1,0,1,0,0,0,1,   1,0,1,0,0,0,1,1,   0,1,0,0,0,0,0,0,   1,0,0,0,1,1,1,1,
1,0,0,1,0,0,1,0,    1,0,0,1,1,1,0,1,    0,0,1,1,1,0,0,0,    1,1,1,1,0,1,0,1,
1,0,1,1,1,1,0,0,    1,0,1,1,0,1,1,0,    1,1,0,1,1,0,1,0,    0,0,1,0,0,0,0,1,
0,0,0,1,0,0,0,0, 1,1,1,1,1,1,1,1, 1,1,1,1,0,0,1,1, 1,1,0,1,0,0,1,0},
{   1,1,0,0,1,1,0,1,   0,0,0,0,1,1,0,0,   0,0,0,1,0,0,1,1,   1,1,1,0,1,1,0,0,
0,1,0,1,1,1,1,1,    1,0,0,1,0,1,1,1,    0,1,0,0,0,1,0,0,    0,0,0,1,0,1,1,1,
1,1,0,0,0,1,0,0,    1,0,1,0,0,1,1,1,    0,1,1,1,1,1,1,0,    0,0,1,1,1,1,0,1,
0,1,1,0,0,1,0,0, 0,1,0,1,1,1,0,1, 0,0,0,1,1,0,0,1, 0,1,1,1,0,0,1,1},
{   0,1,1,0,0,0,0,0,   1,0,0,0,0,0,0,1,   0,1,0,0,1,1,1,1,   1,1,0,1,1,1,0,0,
0,0,1,0,0,0,1,0,    0,0,1,0,1,0,1,0,    1,0,0,1,0,0,0,0,    1,0,0,0,1,0,0,0,
0,1,0,0,0,1,1,0,    1,1,1,0,1,1,1,0,    1,0,1,1,1,0,0,0,    0,0,0,1,0,1,0,0,
1,1,0,1,1,1,1,0, 0,1,0,1,1,1,1,0, 0,0,0,0,1,0,1,1, 1,1,0,1,1,0,1,1},
{   1,1,1,0,0,0,0,0,   0,0,1,1,0,0,1,0,   0,0,1,1,1,0,1,0,   0,0,0,0,1,0,1,0,
0,1,0,0,1,0,0,1,    0,0,0,0,0,1,1,0,    0,0,1,0,0,1,0,0,    0,1,0,1,1,1,0,0,
1,1,0,0,0,0,1,0,    1,1,0,1,0,0,1,1,    1,0,1,0,1,1,0,0,    0,1,1,0,0,0,1,0,
1,0,0,1,0,0,0,1, 1,0,0,1,0,1,0,1, 1,1,1,0,0,1,0,0, 0,1,1,1,1,0,0,1},
{   1,1,1,0,0,1,1,1,   1,1,0,0,1,0,0,0,   0,0,1,1,0,1,1,1,   0,1,1,0,1,1,0,1,
1,0,0,0,1,1,0,1,    1,1,0,1,0,1,0,1,    0,1,0,0,1,1,1,0,    1,0,1,0,1,0,0,1,
0,1,1,0,1,1,0,0,    0,1,0,1,0,1,1,0,    1,1,1,1,0,1,0,0,    1,1,1,0,1,0,1,0,
0,1,1,0,0,1,0,1, 0,1,1,1,1,0,1,0, 1,0,1,0,1,1,1,0, 0,0,0,0,1,0,0,0},
{   1,0,1,1,1,0,1,0,   0,1,1,1,1,0,0,0,   0,0,1,0,0,1,0,1,   0,0,1,0,1,1,1,0,
0,0,0,1,1,1,0,0,    1,0,1,0,0,1,1,0,    1,0,1,1,0,1,0,0,    1,1,0,0,0,1,1,0,
1,1,1,0,1,0,0,0,    1,1,0,1,1,1,0,1,    0,1,1,1,0,1,0,0,    0,0,0,1,1,1,1,1,
0,1,0,0,1,0,1,1, 1,0,1,1,1,1,0,1, 1,0,0,0,1,0,1,1, 1,0,0,0,1,0,1,0},
{   0,1,1,1,0,0,0,0,   0,0,1,1,1,1,1,0,   1,0,1,1,0,1,0,1,   0,1,1,0,0,1,1,0,
0,1,0,0,1,0,0,0,    0,0,0,0,0,0,1,1,    1,1,1,1,0,1,1,0,    0,0,0,0,1,1,1,0,
```

```
0,1,1,0,0,0,0,1,    0,0,1,1,0,1,0,1,    0,1,0,1,0,1,1,1,    1,0,1,1,1,0,0,1,
1,0,0,0,0,1,1,0, 1,1,0,0,0,0,0,1, 0,0,0,1,1,1,0,1, 1,0,0,1,1,1,1,0},
{   1,1,1,0,0,0,0,1,   1,1,1,1,1,0,0,0,   1,0,0,1,1,0,0,0,   0,0,0,1,0,0,0,1,
0,1,1,0,1,0,0,1,    1,1,0,1,1,0,0,1,    1,0,0,0,1,1,1,0,    1,0,0,1,0,1,0,0,
1,0,0,1,1,0,1,1,    0,0,0,1,1,1,1,0,    1,0,0,0,0,1,1,1,    1,1,1,0,1,0,0,1,
1,1,0,0,1,1,1,0, 0,1,0,1,0,1,0,1, 0,0,1,0,1,0,0,0, 1,1,0,1,1,1,1,1},
{   1,0,0,0,1,1,0,0,   1,0,1,0,0,0,0,1,   1,0,0,0,1,0,0,1,   0,0,0,0,1,1,0,1,
1,0,1,1,1,1,1,1,    1,1,1,0,0,1,1,0,    0,1,0,0,0,0,1,0,    0,1,1,0,1,0,0,0,
0,1,0,0,0,0,0,1,    1,0,0,1,1,0,0,1,    0,0,1,0,1,1,0,1,    0,0,0,0,1,1,1,1,
1,0,1,1,0,0,0,0, 0,1,0,1,0,1,0,0, 1,0,1,1,1,0,1,1, 0,0,0,1,0,1,1,0}
};

/*******************************************
 * converte byte - transformacao BS           *
 *******************************************/
  lin=xy[0]*8+xy[1]*4+xy[2]*2+xy[3];//clacula linha de entrada na caixa
(caso hexa)
  col=xy[4]*8+xy[5]*4+xy[6]*2+xy[7];//calcula coluna de entrada na caixa
(caso hexa)
  col=col*8;// converte o valor de col para o caso da nossa s-box expandida
em 8 bits
  for (z=0;z<8;z++)// substitui byte bit a bit (operacao de ida)
  zw[z]=BS[lin][col+z];
return zw;
}// ********************fim do metodo bytesub ****************************

/**********************************************************************
 *****   metodo substituicao inversa - (INV BYTESUB) = S Box      ******
 **********   argumento = int[8] binario                          ******
 ***********  restorna = int[8] binario                           ******
 **********************************************************************/
 public static int[] INVBS(int zw[]){//byte zw e o que sera subs.por zw
 int lin,col,z;
 int[] xy=new int[8];
 int Inv_BS[][] = {
{   0,1,0,1,0,0,1,0,   0,0,0,0,1,0,0,1,   0,1,1,0,1,0,1,0,   1,1,0,1,0,1,0,1,
0,0,1,1,0,0,0,0,    0,0,1,1,0,1,1,0,    1,0,1,0,0,1,0,1,    0,0,1,1,1,0,0,0,
1,0,1,1,1,1,1,1,    0,1,0,0,0,0,0,0,    1,0,1,0,0,0,1,1,    1,0,0,1,1,1,1,0,
1,0,0,0,0,0,0,1, 1,1,1,1,0,0,1,1, 1,1,0,1,0,1,1,1, 1,1,1,1,1,0,1,1},
{   0,1,1,1,1,1,0,0,   1,1,1,0,0,0,1,1,   0,0,1,1,1,0,0,1,   1,0,0,0,0,0,1,0,
1,0,0,1,1,0,1,1,    0,0,1,0,1,1,1,1,    1,1,1,1,1,1,1,1,    1,0,0,0,0,1,1,1,
0,0,1,1,0,1,0,0,    1,0,0,0,1,1,1,0,    0,1,0,0,0,0,1,1,    0,1,0,0,0,1,0,0,
1,1,0,0,0,1,0,0, 1,1,0,1,1,1,1,0, 1,1,1,0,1,0,0,1, 1,1,0,0,1,0,1,1},
{   0,1,0,1,0,1,0,0,   0,1,1,1,1,0,1,1,   1,0,0,1,0,1,0,0,   0,0,1,1,0,0,1,0,
1,0,1,0,0,1,1,0,    1,1,0,0,0,0,1,0,    0,0,1,0,0,0,1,1,    0,0,1,1,1,1,0,1,
1,1,1,0,1,1,1,0,    0,1,0,0,1,1,0,0,    1,0,0,1,0,1,0,1,    0,0,0,0,1,0,1,1,
0,1,0,0,0,0,1,0, 1,1,1,1,1,0,1,0, 1,1,0,0,0,0,1,1, 0,1,0,0,1,1,1,0},
{   0,0,0,0,1,0,0,0,   0,0,1,0,1,1,1,0,   1,0,1,0,0,0,0,1,   0,1,1,0,0,1,1,0,
0,0,1,0,1,0,0,0,    1,1,0,1,1,0,0,1,    0,0,1,0,0,1,0,0,    1,0,1,1,0,0,1,0,
0,1,1,1,0,1,1,0,    0,1,0,1,1,0,1,1,    1,0,1,0,0,0,1,0,    0,1,0,0,1,0,0,1,
0,1,1,0,1,1,0,1, 1,0,0,0,1,0,1,1, 1,1,0,1,0,0,0,1, 0,0,1,0,0,1,0,1},
{   0,1,1,1,0,0,1,0,   1,1,1,1,1,0,0,0,   1,1,1,1,0,1,1,0,   0,1,1,0,0,1,0,0,
1,0,0,0,0,1,1,0,    0,1,1,0,1,0,0,0,    1,0,0,1,1,0,0,0,    0,0,0,1,0,1,1,0,
1,1,0,1,0,1,0,0,    1,0,1,0,0,1,0,0,    0,1,0,1,1,1,0,0,    1,1,0,0,1,1,0,0,
0,1,0,1,1,1,0,1, 0,1,1,0,0,1,0,1, 1,0,1,1,0,1,1,0, 1,0,0,1,0,0,1,0},
{   0,1,1,0,1,1,0,0,   0,1,1,1,0,0,0,0,   0,1,0,0,1,0,0,0,   0,1,0,1,0,0,0,0,
1,1,1,1,1,1,0,1,    1,1,1,0,1,1,0,1,    1,0,1,1,1,0,0,1,    1,1,0,1,1,0,1,0,
0,1,0,1,1,1,1,0,    0,0,0,1,0,1,0,1,    0,1,0,0,0,1,1,0,    0,1,0,1,0,1,1,1,
1,0,1,0,0,1,1,1, 1,0,0,0,1,1,0,1, 1,0,0,1,1,1,0,1, 1,0,0,0,0,1,0,0},
```

```
{   1,0,0,1,0,0,0,0,   1,1,0,1,1,0,0,0,   1,0,1,0,1,0,1,1,   0,0,0,0,0,0,0,0,
1,0,0,0,1,1,0,0,    1,0,1,1,1,1,0,0,    1,1,0,1,0,0,1,1,    0,0,0,0,1,0,1,0,
1,1,1,1,0,1,1,1,    1,1,1,0,0,1,0,0,    0,1,0,1,1,0,0,0,    0,0,0,0,0,1,0,1,
1,0,1,1,1,0,0,0, 1,0,1,1,0,0,1,1, 0,1,0,0,0,1,0,1, 0,0,0,0,0,1,1,0},
{   1,1,0,1,0,0,0,0,   0,0,1,0,1,1,0,0,   0,0,0,1,1,1,1,0,   1,0,0,0,1,1,1,1,
1,1,0,0,1,0,1,0,    0,0,1,1,1,1,1,1,    0,0,0,0,1,1,1,1,    0,0,0,0,0,0,1,0,
1,1,0,0,0,0,0,1,    1,0,1,0,1,1,1,1,    1,0,1,1,1,1,0,1,    0,0,0,0,0,0,1,1,
0,0,0,0,0,0,0,1, 0,0,0,1,0,0,1,1, 1,0,0,0,1,0,1,0, 0,1,1,0,1,0,1,1},
{   0,0,1,1,1,0,1,0,   1,0,0,1,0,0,0,1,   0,0,0,1,0,0,0,1,   0,1,0,0,0,0,0,1,
0,1,0,0,1,1,1,1,    0,1,1,0,0,1,1,1,    1,1,0,1,1,1,0,0,    1,1,1,0,1,0,1,0,
1,0,0,1,0,1,1,1,    1,1,1,1,0,0,1,0,    1,1,0,0,1,1,1,1,    1,1,0,0,1,1,1,0,
1,1,1,1,0,0,0,0, 1,0,1,1,0,1,0,0, 1,1,1,0,0,1,1,0, 0,1,1,1,0,0,1,1},
{   1,0,0,1,0,1,1,0,   1,0,1,0,1,1,0,0,   0,1,1,1,0,1,0,0,   0,0,1,0,0,0,1,0,
1,1,1,0,0,1,1,1,    1,0,1,0,1,1,0,1,    0,0,1,1,0,1,0,1,    1,0,0,0,0,1,0,1,
1,1,1,0,0,0,1,0,    1,1,1,1,1,0,0,1,    0,0,1,1,0,1,1,1,    1,1,1,0,1,0,0,0,
0,0,0,1,1,1,0,0, 0,1,1,1,0,1,0,1, 1,1,0,1,1,1,1,1, 0,1,1,0,1,1,1,0},
{   0,1,0,0,0,1,1,1,   1,1,1,1,0,0,0,1,   0,0,0,1,1,0,1,0,   0,1,1,1,0,0,0,1,
0,0,0,1,1,1,0,1,    0,0,1,0,1,0,0,1,    1,1,0,0,0,1,0,1,    1,0,0,0,1,0,0,1,
0,1,1,0,1,1,1,1,    1,0,1,1,0,1,1,1,    0,1,1,0,0,0,1,0,    0,0,0,0,1,1,1,0,
1,0,1,0,1,0,1,0, 0,0,0,1,1,0,0,0, 1,0,1,1,1,1,1,0, 0,0,0,1,1,0,1,1},
{   1,1,1,1,1,1,0,0,   0,1,0,1,0,1,1,0,   0,0,1,1,1,1,1,0,   0,1,0,0,1,0,1,1,
1,1,0,0,0,1,1,0,    1,1,0,1,0,0,1,0,    0,1,1,1,1,0,0,1,    0,0,1,0,0,0,0,0,
1,0,0,1,1,0,1,0,    1,1,0,1,1,0,1,1,    1,1,0,0,0,0,0,0,    1,1,1,1,1,1,1,0,
0,1,1,1,1,0,0,0, 1,1,0,0,1,1,0,1, 0,1,0,1,1,0,1,0, 1,1,1,1,0,1,0,0},
{   0,0,0,1,1,1,1,1,   1,1,0,1,1,1,0,1,   1,0,1,0,1,0,0,0,   0,0,1,1,0,0,1,1,
1,0,0,0,1,0,0,0,    0,0,0,0,0,1,1,1,    1,1,0,0,0,1,1,1,    0,0,1,1,0,0,0,1,
1,0,1,1,0,0,0,1,    0,0,0,1,0,0,1,0,    0,0,0,1,0,0,0,0,    0,1,0,1,1,0,0,1,
0,0,1,0,0,1,1,1, 1,0,0,0,0,0,0,0, 1,1,1,0,1,1,0,0, 0,1,0,1,1,1,1,1},
{   0,1,1,0,0,0,0,0,   0,1,0,1,0,0,0,1,   0,1,1,1,1,1,1,1,   1,0,1,0,1,0,0,1,
0,0,0,1,1,0,0,1,    1,0,1,1,0,1,0,1,    0,1,0,0,1,0,1,0,    0,0,0,0,1,1,0,1,
0,0,1,0,1,1,0,1,    1,1,1,0,0,1,0,1,    0,1,1,1,1,0,1,0,    1,0,0,1,1,1,1,1,
1,0,0,1,0,0,1,1, 1,1,0,0,1,0,0,1, 1,0,0,1,1,1,0,0, 1,1,1,0,1,1,1,1},
{   1,0,1,0,0,0,0,0,   1,1,1,0,0,0,0,0,   0,0,1,1,1,0,1,1,   0,1,0,0,1,1,0,1,
1,0,1,0,1,1,1,0,    0,0,1,0,1,0,1,0,    1,1,1,1,0,1,0,1,    1,0,1,1,0,0,0,0,
1,1,0,0,1,0,0,0,    1,1,1,0,1,0,1,1,    1,0,1,1,1,0,1,1,    0,0,1,1,1,1,0,0,
1,0,0,0,0,0,1,1, 0,1,0,1,0,0,1,1, 1,0,0,1,1,0,0,1, 0,1,1,0,0,0,0,1},
{   0,0,0,1,0,1,1,1,   0,0,1,0,1,0,1,1,   0,0,0,0,0,1,0,0,   0,1,1,1,1,1,1,0,
1,0,1,1,1,0,1,0,    0,1,1,1,0,1,1,1,    1,1,0,1,0,1,1,0,    0,0,1,0,0,1,1,0,
1,1,1,0,0,0,0,1,    0,1,1,0,1,0,0,1,    0,0,0,1,0,1,0,0,    0,1,1,0,0,0,1,1,
0,1,0,1,0,1,0,1, 0,0,1,0,0,0,0,1, 0,0,0,0,1,1,0,0, 0,1,1,1,1,1,0,1},
};

/*****************************************************
 * converte byte - transformacao inversa INVBS       *
 *****************************************************/
  lin=zw[0]*8+zw[1]*4+zw[2]*2+zw[3];//clacula linha de entrada na caixa
(caso hexa)
  col=zw[4]*8+zw[5]*4+zw[6]*2+zw[7];//calcula coluna de entrada na caixa
(caso hexa)
  col=col*8;// converte o valor de col para o caso da nossa s-box expandida
em 8 bits
  for (z=0;z<8;z++)// substitui byte bit a bit (operacao de ida)
  xy[z]=Inv_BS[lin][col+z];
return xy;
}// ********************fim do metodo inv bytesub ***********************

/*********************************************************************
 **********       metodo PSR     - permutacao de bytes (SHIFTROW)    ******
 **********  obs: foi implementado bit a bit                          ******
 **********   argumento = int[128] binario                            ******
```

```
 **********   retorna = int[128] binario                              ******
 ************************************************************************/
 public static int[] PSR(int arg[]){//
 int[] gar=new int[128];
 int SR[] = {
0, 1, 2, 3, 4, 5, 6, 7,
40,  41,  42,  43,  44,  45,  46,  47,
80,  81,  82,  83,  84,  85,  86,  87,
120, 121, 122, 123, 124, 125, 126, 127,
32,  33,  34,  35,  36,  37,  38,  39,
72,  73,  74,  75,  76,  77,  78,  79,
112, 113, 114, 115, 116, 117, 118, 119,
24,  25,  26,  27,  28,  29,  30,  31,
64,  65,  66,  67,  68,  69,  70,  71,
104, 105, 106, 107, 108, 109, 110, 111,
16,  17,  18,  19,  20,  21,  22,  23,
56,  57,  58,  59,  60,  61,  62,  63,
96,  97,  98,  99,  100, 101, 102, 103,
8, 9, 10,  11,  12,  13,  14,  15,
48,  49,  50,  51,  52,  53,  54,  55,
88,  89,  90,  91,  92,  93,  94,  95};
for (int z=0;z<128;z++)// permutacao propriamente dita
   gar[z]=arg[SR[z]];
return gar;

}//*************** fim do metodo permutacao SR ***********************

/************************************************************************
 **********      metodo PSR Inversa - (INV SHIFT ROW)                ******
 **********       obs: foi implementado bit a bit                    ******
 **********       argumento = int[128] binario                       ******
 **********        retorna = int[128] binario                        ******
 ************************************************************************/
 public static int[] INVPSR(int gar[]){//
 int[] arg=new int[128];
int Inv_SR[] = {
0,    1,    2,    3,    4,    5,    6,    7,
104,  105,  106,  107,  108,  109,  110,  111,
80,   81,   82,   83,   84,   85,   86,   87,
56,   57,   58,   59,   60,   61,   62,   63,
32,   33,   34,   35,   36,   37,   38,   39,
8,    9,    10,   11,   12,   13,   14,   15,
112,  113,  114,  115,  116,  117,  118,  119,
88,   89,   90,   91,   92,   93,   94,   95,
64,   65,   66,   67,   68,   69,   70,   71,
40,   41,   42,   43,   44,   45,   46,   47,
16,   17,   18,   19,   20,   21,   22,   23,
120,  121,  122,  123,  124,  125,  126,  127,
96,   97,   98,   99,   100,  101,  102,  103,
72,   73,   74,   75,   76,   77,   78,   79,
48,   49,   50,   51,   52,   53,   54,   55,
24,   25,   26,   27,   28,   29,   30,   31};
for (int z=0;z<8;z++)// permutacao inversa propriamente dita
   arg[z]=gar[Inv_SR[z]];
return arg;
}//********* fim do metodo permutacao SR inversa ***********************

/************************************************************************
 **********       metodo MC   -  multiplicacao (MIX COLUMNS)        ******
 **********  obs: foi implementado bit a bit                        ******
```

```
 **********    argumento = int[32] binario                              ******
 **********     retorna = int[32] binario                               ******
 ************************************************************************/
 public static int[] MC(int arg[]){//
 int M[][] = {
{0, 1, 0, 0, 0, 0, 0, 0,  1, 1, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0},{
0, 0, 1, 0, 0, 0, 0, 0,  0, 1, 1, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0},{
0, 0, 0, 1, 0, 0, 0, 0,  0, 0, 1, 1, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0},{
1, 0, 0, 0, 1, 0, 0, 0,  1, 0, 0, 1, 1, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0},{
1, 0, 0, 0, 0, 1, 0, 0,  1, 0, 0, 0, 1, 1, 0, 0,  0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0},{
0, 0, 0, 0, 0, 0, 1, 0,  0, 0, 0, 0, 0, 1, 1, 0,  0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0},{
1, 0, 0, 0, 0, 0, 0, 1,  1, 0, 0, 0, 0, 0, 1, 1,  0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 1, 0},{
1, 0, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 1,  0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 1},{

1, 0, 0, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0,  1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0},{
0, 1, 0, 0, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0,  0, 1, 1, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0},{
0, 0, 1, 0, 0, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0,  0, 0, 1, 1, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0},{
0, 0, 0, 1, 0, 0, 0, 0,  1, 0, 0, 0, 1, 0, 0, 0,  1, 0, 0, 1, 1, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0},{
0, 0, 0, 0, 1, 0, 0, 0,  1, 0, 0, 0, 0, 1, 0, 0,  1, 0, 0, 0, 1, 1, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0},{
0, 0, 0, 0, 0, 1, 0, 0,  0, 0, 0, 0, 0, 0, 1, 0,  0, 0, 0, 0, 0, 1, 1, 0,
0, 0, 0, 0, 0, 1, 0, 0},{
0, 0, 0, 0, 0, 0, 1, 0,  1, 0, 0, 0, 0, 0, 0, 1,  1, 0, 0, 0, 0, 0, 1, 1,
0, 0, 0, 0, 0, 0, 1, 0},{
0, 0, 0, 0, 0, 0, 0, 1,  1, 0, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 1},{

1, 0, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0,
1, 1, 0, 0, 0, 0, 0, 0},{
0, 1, 0, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0,
0, 1, 1, 0, 0, 0, 0, 0},{
0, 0, 1, 0, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 1, 1, 0, 0, 0, 0},{
0, 0, 0, 1, 0, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0,  1, 0, 0, 0, 1, 0, 0, 0,
1, 0, 0, 1, 1, 0, 0, 0},{
0, 0, 0, 0, 1, 0, 0, 0,  0, 0, 0, 0, 1, 0, 0, 0,  1, 0, 0, 0, 0, 1, 0, 0,
1, 0, 0, 0, 1, 1, 0, 0},{
0, 0, 0, 0, 0, 1, 0, 0,  0, 0, 0, 0, 0, 1, 0, 0,  0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 1, 1, 0},{
0, 0, 0, 0, 0, 0, 1, 0,  0, 0, 0, 0, 0, 0, 1, 0,  1, 0, 0, 0, 0, 0, 0, 1,
1, 0, 0, 0, 0, 0, 1, 1},{
0, 0, 0, 0, 0, 0, 0, 1,  0, 0, 0, 0, 0, 0, 0, 1,  1, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 1},{

1, 1, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0},{
0, 1, 1, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0},{
```

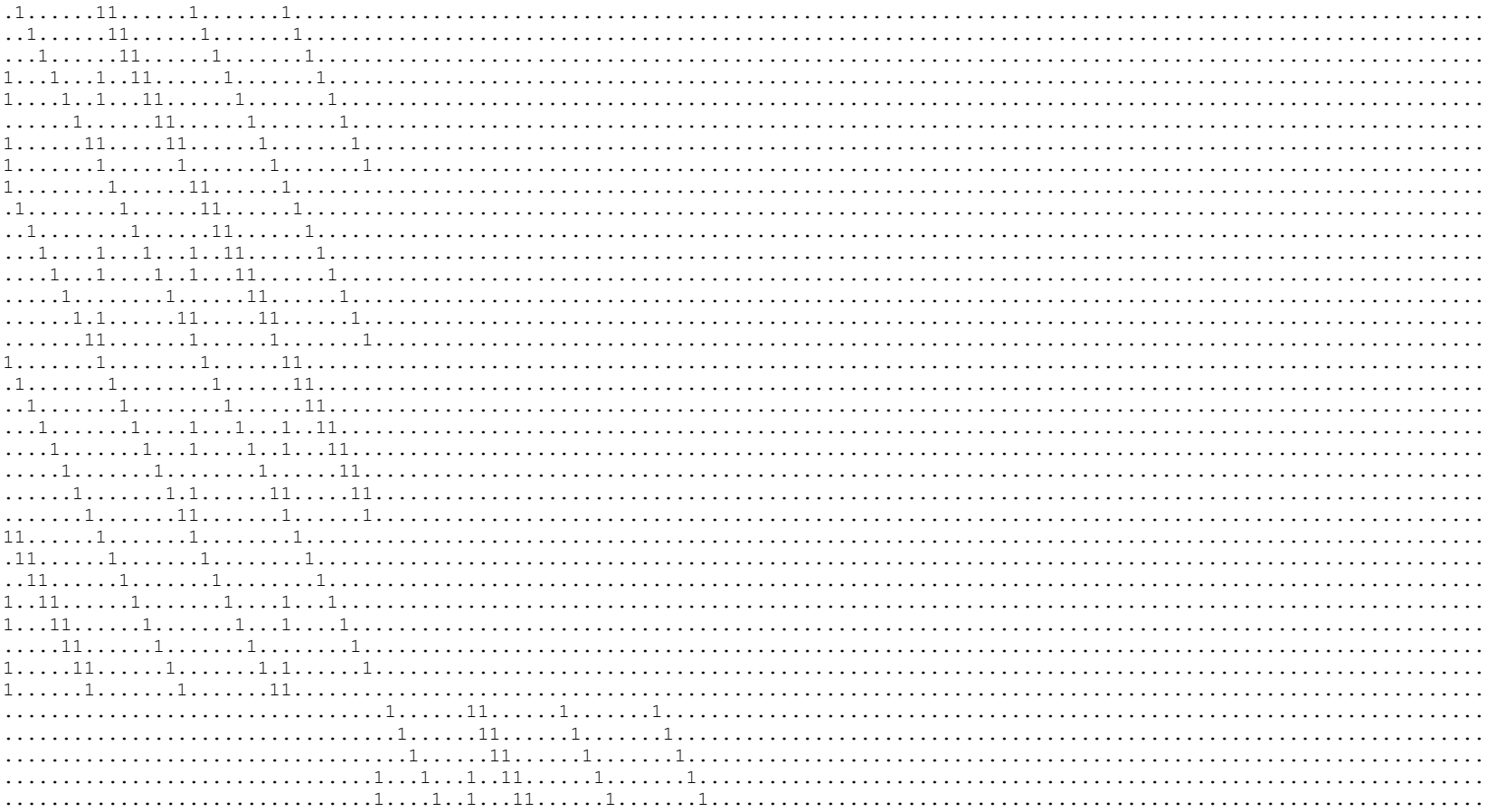
```
0, 0, 1, 1, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0},{
1, 0, 0, 1, 1, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0},{
1, 0, 0, 0, 1, 1, 0, 0,  0, 0, 0, 0, 1, 0, 0, 0,  0, 0, 0, 0, 1, 0, 0, 0,
1, 0, 0, 0, 0, 1, 0, 0},{
0, 0, 0, 0, 0, 1, 1, 0,  0, 0, 0, 0, 0, 1, 0, 0,  0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0},{
1, 0, 0, 0, 0, 0, 1, 1,  0, 0, 0, 0, 0, 0, 1, 0,  0, 0, 0, 0, 0, 0, 1, 0,
1, 0, 0, 0, 0, 0, 0, 1},{
1, 0, 0, 0, 0, 0, 0, 1,  0, 0, 0, 0, 0, 0, 0, 1,  0, 0, 0, 0, 0, 0, 0, 1,
1, 0, 0, 0, 0, 0, 0, 0}};

int ret[] = new int [32];
for (int i=0;i<32;i++){// Transformação Mix_Columns Propriamente dita *
  for (int j=0;j<32;j++){
   ret[i]= (M[i][j]*arg[j]+ret[i])%2;
        }
  }// fim da transformação
return ret;
}//**************    fim    do    metodo    multiplicacao    (MIX    COLUMNS)
***************

/*************************************************************************
 *     metodo INVMC   -  multiplicacao inversa (MIX COLUMNS)              *
 * obs: foi implementado bit a bit                                         *
argumento = int[32] binario                          ******
 **********                                    retorna  =  int[32]  binario
******

*************************************************************************
*********/
 public static int[] INVMC(int arg[]){//
 int Inv_M[][] = {
{0, 1, 1, 1, 0, 0, 0, 0,  1, 1, 0, 1, 0, 0, 0, 0,  1, 0, 1, 1, 0, 0, 0, 0,
1, 0, 0, 1, 0, 0, 0, 0},
{1, 0, 1, 1, 1, 0, 0, 0,  1, 1, 1, 0, 1, 0, 0, 0,  1, 1, 0, 1, 1, 0, 0, 0,
1, 1, 0, 0, 1, 0, 0, 0},
{0, 1, 0, 1, 1, 1, 0, 0,  1, 1, 1, 1, 0, 1, 0, 0,  0, 1, 1, 0, 1, 1, 0, 0,
1, 1, 1, 0, 0, 1, 0, 0},
{0, 0, 1, 0, 1, 1, 1, 0,  1, 1, 1, 1, 1, 0, 1, 0,  1, 0, 1, 1, 0, 1, 1, 0,
0, 1, 1, 1, 0, 0, 1, 0},
{0, 1, 1, 0, 0, 1, 1, 1,  0, 0, 1, 0, 1, 1, 0, 1,  1, 1, 1, 0, 1, 0, 1, 1,
1, 0, 1, 0, 1, 0, 0, 1},
{0, 1, 0, 0, 0, 0, 1, 1,  1, 1, 0, 0, 0, 1, 1, 0,  0, 1, 0, 0, 0, 1, 0, 1,
1, 1, 0, 0, 0, 1, 0, 0},
{0, 0, 1, 0, 0, 0, 0, 1,  1, 1, 1, 0, 0, 0, 1, 1,  1, 0, 1, 0, 0, 0, 1, 0,
0, 1, 1, 0, 0, 0, 1, 0},
{1, 1, 1, 0, 0, 0, 0, 0,  1, 0, 1, 0, 0, 0, 0, 1,  0, 1, 1, 0, 0, 0, 0, 1,
0, 0, 1, 0, 0, 0, 0, 1},

{1, 0, 0, 1, 0, 0, 0, 0,  0, 1, 1, 1, 0, 0, 0, 0,  1, 1, 0, 1, 0, 0, 0, 0,
1, 0, 1, 1, 0, 0, 0, 0},
{1, 1, 0, 0, 1, 0, 0, 0,  1, 0, 1, 1, 1, 0, 0, 0,  1, 1, 1, 0, 1, 0, 0, 0,
1, 1, 0, 1, 1, 0, 0, 0},
{1, 1, 1, 0, 0, 1, 0, 0,  0, 1, 0, 1, 1, 1, 0, 0,  1, 1, 1, 1, 0, 1, 0, 0,
0, 1, 1, 0, 1, 1, 0, 0},
{0, 1, 1, 1, 0, 0, 1, 0,  0, 0, 1, 0, 1, 1, 1, 0,  1, 1, 1, 1, 1, 0, 1, 0,
1, 0, 1, 1, 0, 1, 1, 0},
```

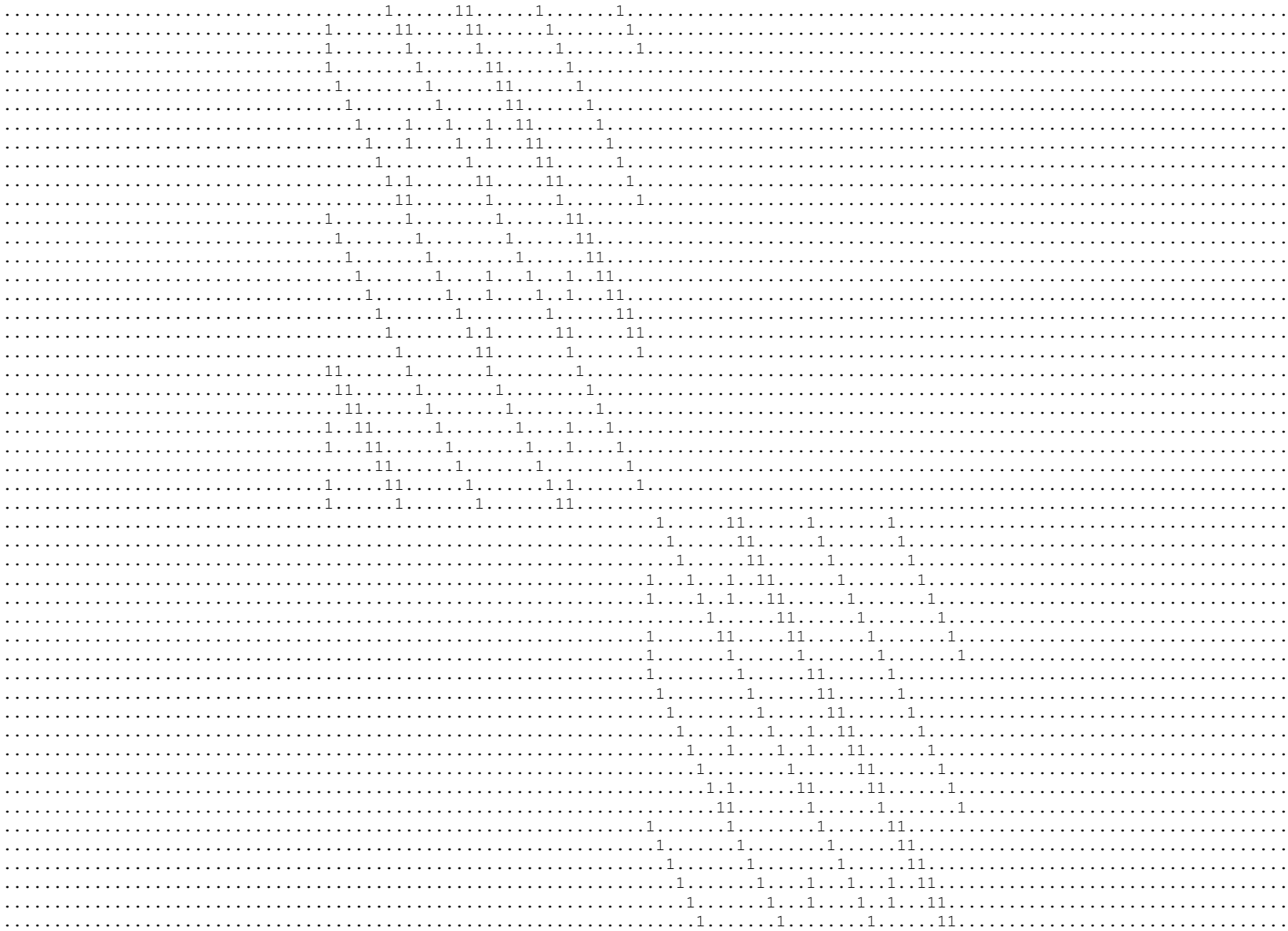
```
{1, 0, 1, 0, 1, 0, 0, 1,  0, 1, 1, 0, 0, 1, 1, 1,  0, 0, 1, 0, 1, 1, 0, 1,
1, 1, 1, 0, 1, 0, 1, 1},
{1, 1, 0, 0, 0, 1, 0, 0,  0, 1, 0, 0, 0, 0, 1, 1,  1, 1, 0, 0, 0, 1, 1, 0,
0, 1, 0, 0, 0, 1, 0, 1},
{0, 1, 1, 0, 0, 0, 1, 0,  0, 0, 1, 0, 0, 0, 0, 1,  1, 1, 1, 0, 0, 0, 1, 1,
1, 0, 1, 0, 0, 0, 1, 0},
{0, 0, 1, 0, 0, 0, 0, 1,  1, 1, 1, 0, 0, 0, 0, 0,  1, 0, 1, 0, 0, 0, 0, 1,
0, 1, 1, 0, 0, 0, 0, 1},

{1, 0, 1, 1, 0, 0, 0, 0,  1, 0, 0, 1, 0, 0, 0, 0,  0, 1, 1, 1, 0, 0, 0, 0,
1, 1, 0, 1, 0, 0, 0, 0},
{1, 1, 0, 1, 1, 0, 0, 0,  1, 1, 0, 0, 1, 0, 0, 0,  1, 0, 1, 1, 1, 0, 0, 0,
1, 1, 1, 0, 1, 0, 0, 0},
{0, 1, 1, 0, 1, 1, 0, 0,  1, 1, 1, 0, 0, 1, 0, 0,  0, 1, 0, 1, 1, 1, 0, 0,
1, 1, 1, 1, 0, 1, 0, 0},
{1, 0, 1, 1, 0, 1, 1, 0,  0, 1, 1, 1, 0, 0, 1, 0,  0, 0, 1, 0, 1, 1, 1, 0,
1, 1, 1, 1, 1, 0, 1, 0},
{1, 1, 1, 0, 1, 0, 1, 1,  1, 0, 1, 0, 1, 0, 0, 1,  0, 1, 1, 0, 0, 1, 1, 1,
0, 0, 1, 0, 1, 1, 0, 1},
{0, 1, 0, 0, 0, 1, 0, 1,  1, 1, 0, 0, 0, 1, 0, 0,  0, 1, 0, 0, 0, 0, 1, 1,
1, 1, 0, 0, 0, 1, 1, 0},
{1, 0, 1, 0, 0, 0, 1, 0,  0, 1, 1, 0, 0, 0, 1, 0,  0, 0, 1, 0, 0, 0, 0, 1,
1, 1, 1, 0, 0, 0, 1, 1},
{0, 1, 1, 0, 0, 0, 0, 1,  0, 0, 1, 0, 0, 0, 0, 1,  1, 1, 1, 0, 0, 0, 0, 0,
1, 0, 1, 0, 0, 0, 0, 1},

{1, 1, 0, 1, 0, 0, 0, 0,  1, 0, 1, 1, 0, 0, 0, 0,  1, 0, 0, 1, 0, 0, 0, 0,
0, 1, 1, 1, 0, 0, 0, 0},
{1, 1, 1, 0, 1, 0, 0, 0,  1, 1, 0, 1, 1, 0, 0, 0,  1, 1, 0, 0, 1, 0, 0, 0,
1, 0, 1, 1, 1, 0, 0, 0},
{1, 1, 1, 1, 0, 1, 0, 0,  0, 1, 1, 0, 1, 1, 0, 0,  1, 1, 1, 0, 0, 1, 0, 0,
0, 1, 0, 1, 1, 1, 0, 0},
{1, 1, 1, 1, 1, 0, 1, 0,  1, 0, 1, 1, 0, 1, 1, 0,  0, 1, 1, 1, 0, 0, 1, 0,
0, 0, 1, 0, 1, 1, 1, 0},
{0, 0, 1, 0, 1, 1, 0, 1,  1, 1, 1, 0, 1, 0, 1, 1,  1, 0, 1, 0, 1, 0, 0, 1,
0, 1, 1, 0, 0, 1, 1, 1},
{1, 1, 0, 0, 0, 1, 1, 0,  0, 1, 0, 0, 0, 1, 0, 1,  1, 1, 0, 0, 0, 1, 0, 0,
0, 1, 0, 0, 0, 0, 1, 1},
{1, 1, 1, 0, 0, 0, 1, 1,  1, 0, 1, 0, 0, 0, 1, 0,  0, 1, 1, 0, 0, 0, 1, 0,
0, 0, 1, 0, 0, 0, 0, 1},
{1, 0, 1, 0, 0, 0, 0, 1,  0, 1, 1, 0, 0, 0, 0, 1,  0, 0, 1, 0, 0, 0, 0, 1,
1, 1, 1, 0, 0, 0, 0, 0}};

int ret[] = new int [32];
for (int i=0;i<32;i++){// Multiplicacao pela inversa
  for (int j=0;j<32;j++){
   ret[i]= (Inv_M[i][j]*arg[j]+ret[i])%2;
        }
  }// fim da transformação
return ret;
}// *************** fim do metodo mult. p/ Matriz inversa  **************

/*********************************************************************
 *****            metodo hexa - converte bin-hexa                 ******
 **********            argumento = int[4] binario                  ******
 ***********           retorna = um caracter hexa                  ******
 *********************************************************************/
public static String hexa(int bin[]){
```

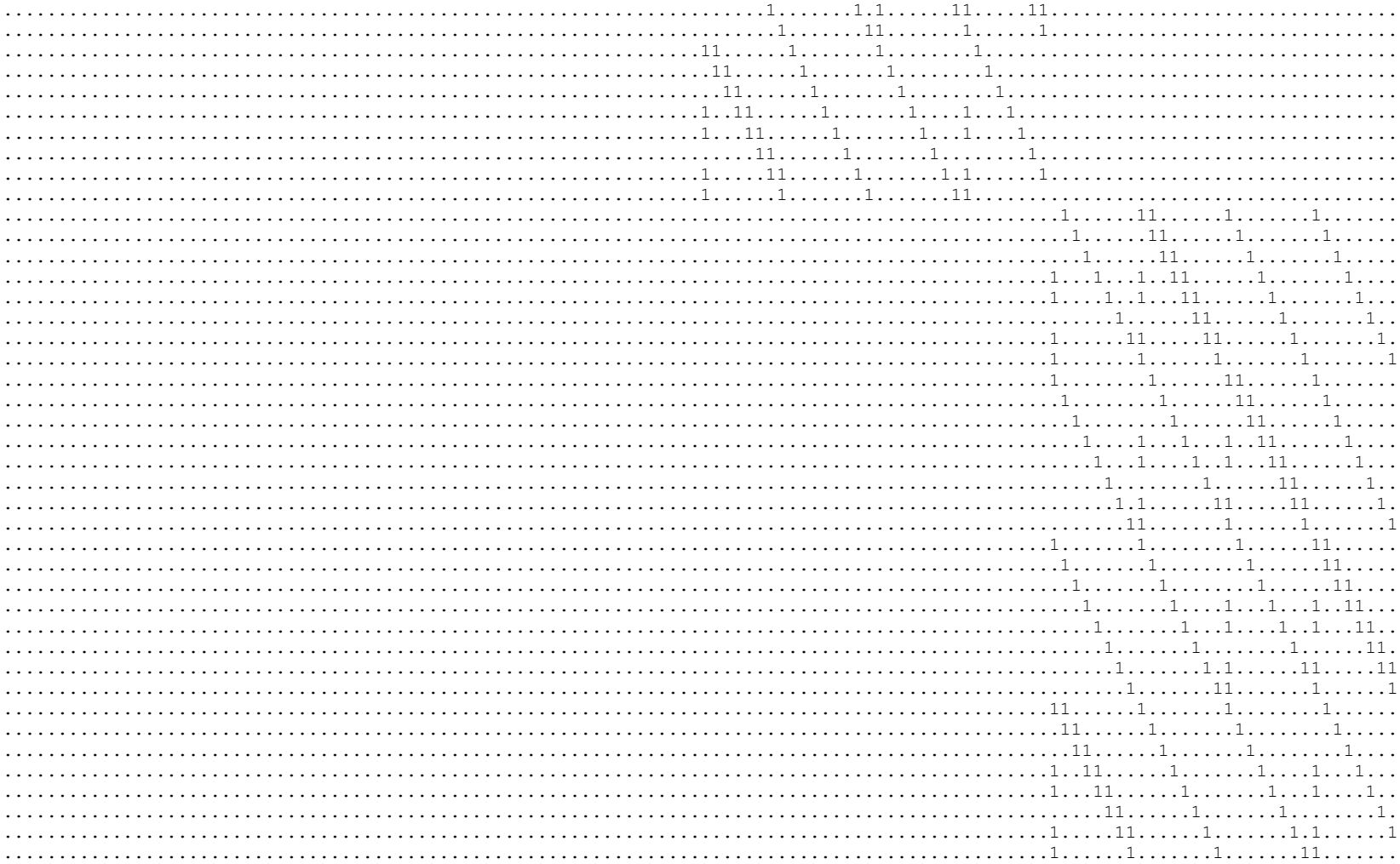
```
String                                                                  []
Simbolos={"0","1","2","3","4","5","6","7","8","9","a","b","c","d","e","f"};

String h= Simbolos[(8*bin[0]+4*bin[1]+2*bin[2]+bin[3])];
return h;
}// ******************** fim do metodo hexa  ***********************

/**********************************************************************
 *****            metodo exibe 128 - exibe  array                  ******
 **********************************************************************/
public static int exibe(int vetor[]){
int temp[]=new int [4];
int h=0;
for (int j=0;j<128;j++){// exibe criptograma hexa
        if ((j%4)==0){
               temp[0]=vetor[j];
               temp[1]=vetor[j+1];
               temp[2]=vetor[j+2];
               temp[3]=vetor[j+3];
               System.out.print(hexa(temp));
               }
         if (j>0){
               if (((j+1)%32)==0) System.out.print(" ");
               }
    }// fim do for j de exibicao do criptograma hexa
System.out.println(" ");

return h;
}// ******************** fim do metodo exibe 128  ******************/

}// fim da classe rijndael
```

12.3 APÊNDICE 3: CONSTRUÇÃO DA MATRIZ DE DEPENDÊNCIAS DA TL DO RIJNDAEL

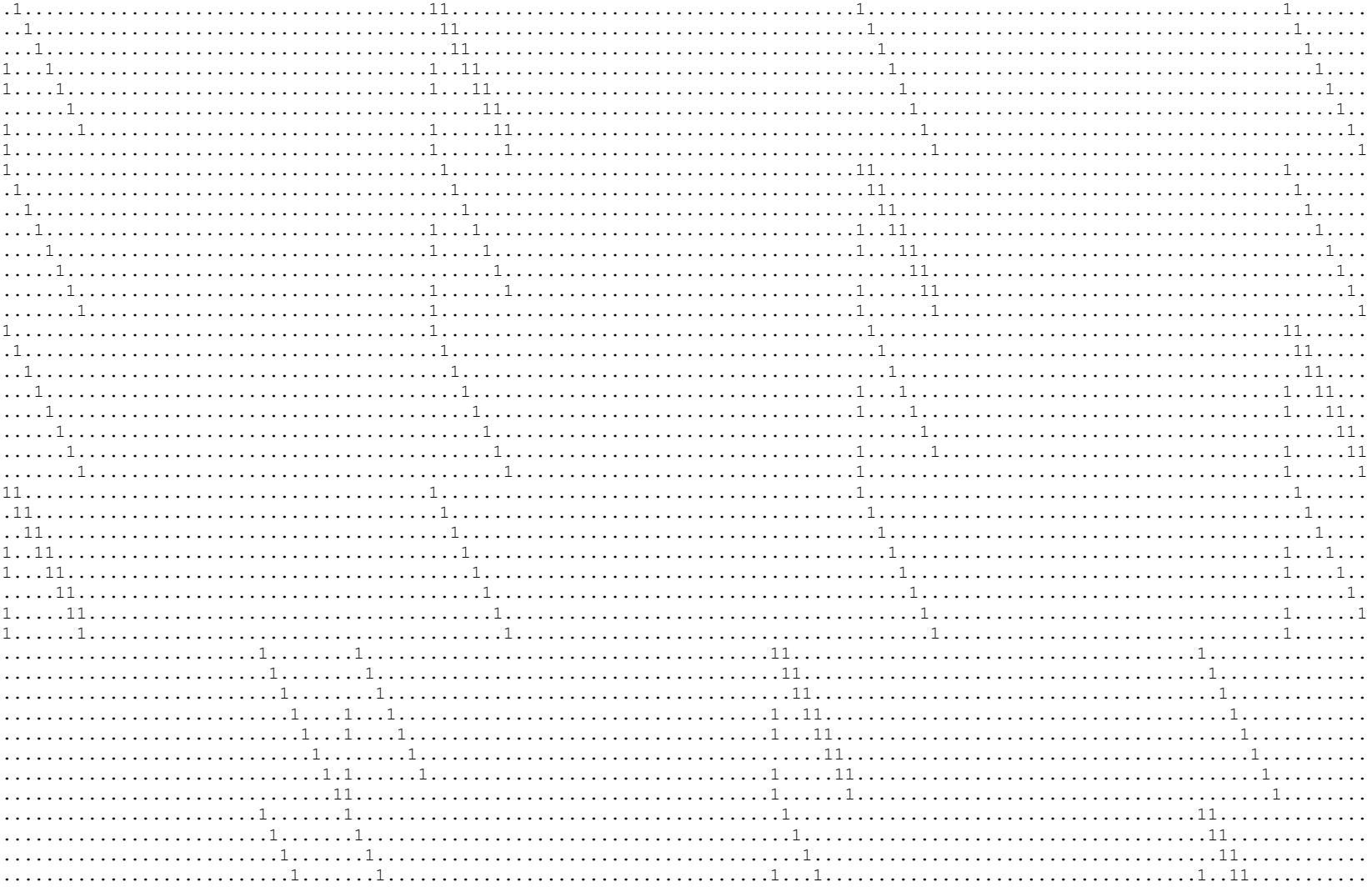
12.3.1 MATRIZ [M']128X128

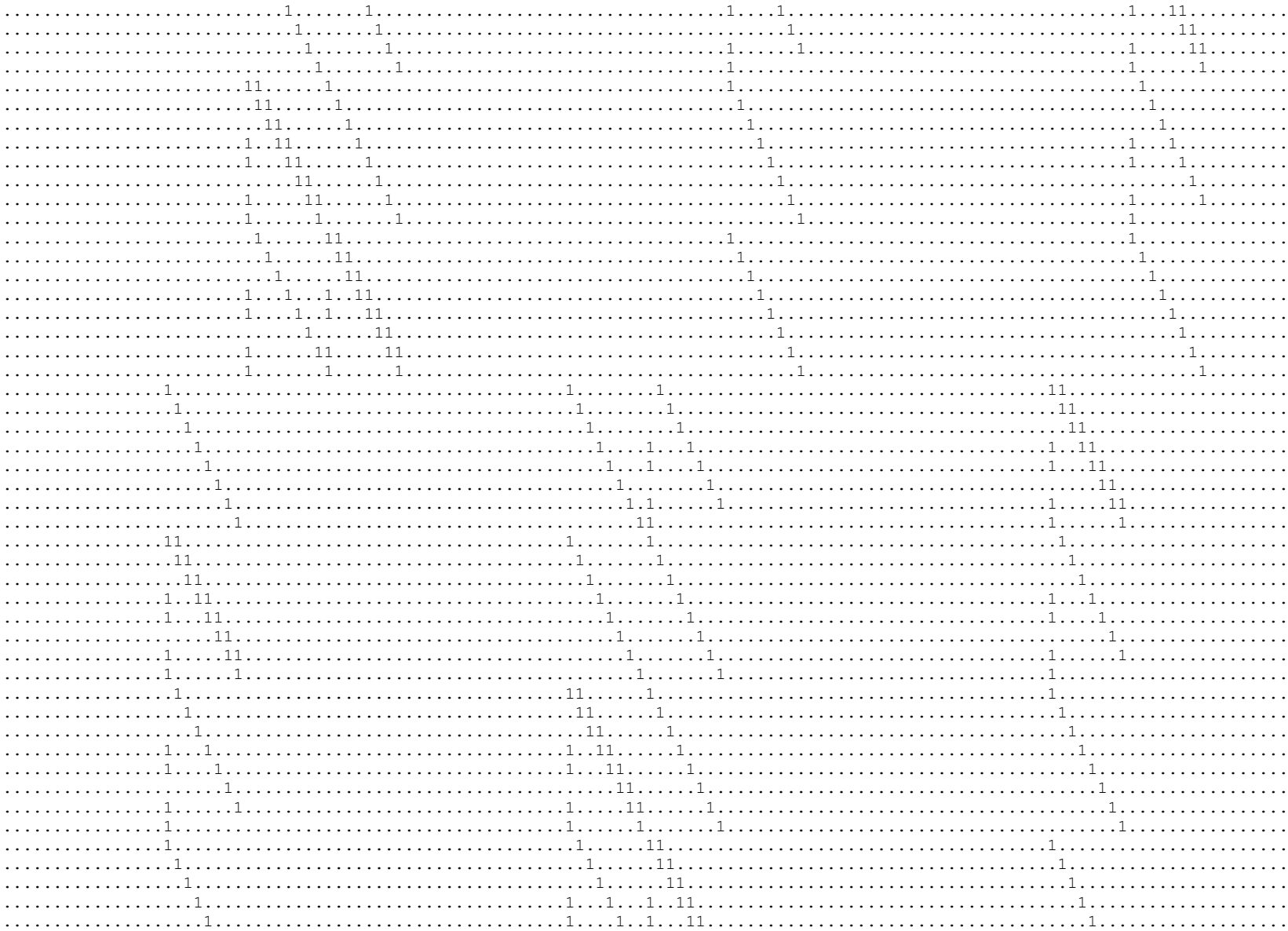


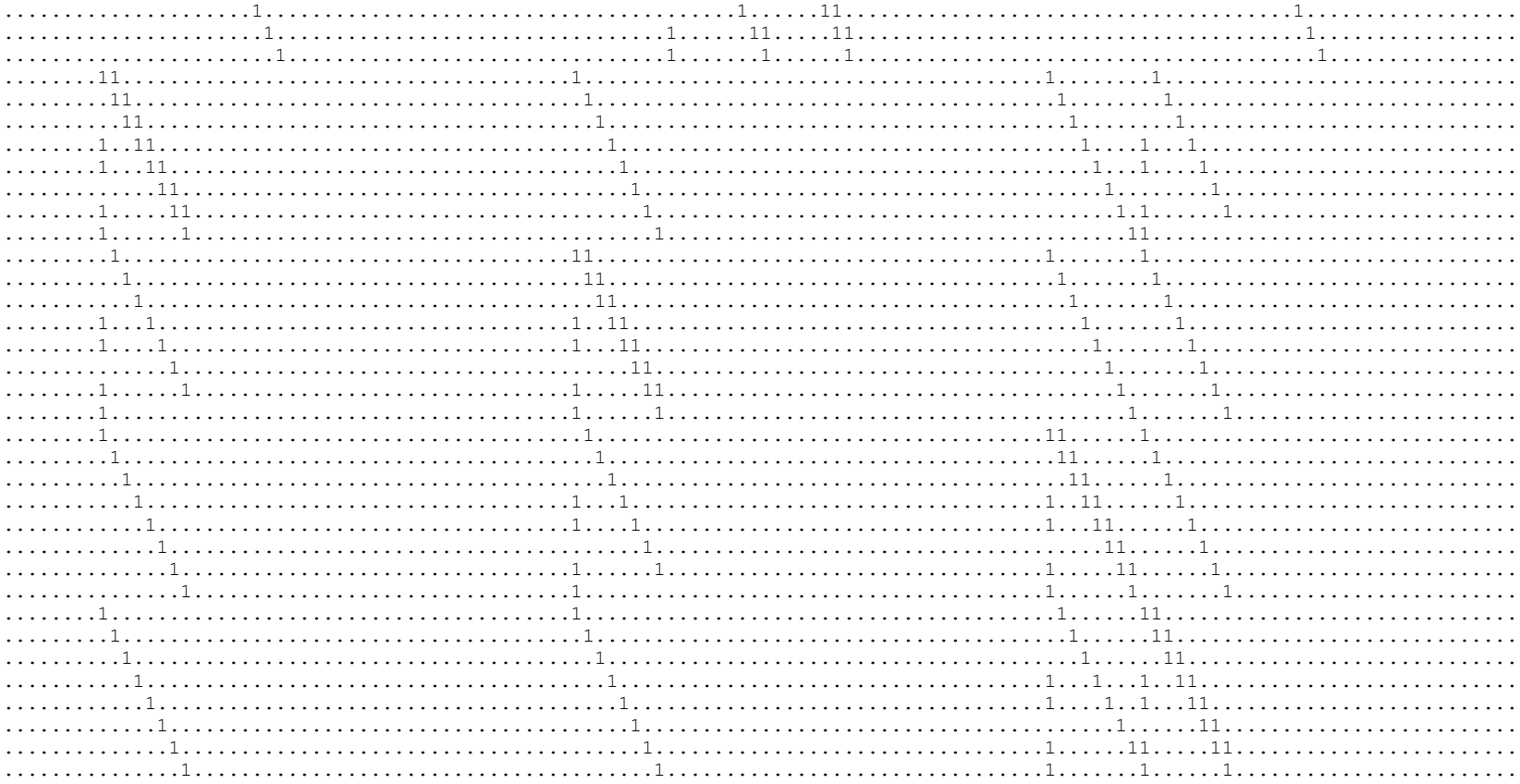




12.3.2 MATRIZ DE DEPENDÊNCIAS [D_1] DA TRANSFORMAÇÃO LINEAR DO RIJNDAEL = [M']$_{128X128}$, COM COLUNAS PERMUTADAS (CONDENSA A *PERMUTAÇÃO* E A *MULTIPLICAÇÃO*)







12.3.3 PROCESSAMENTO DO BLOCO DE TESTE APRESENTADO EM (FIPS 197, 2001) UTILIZANDO O CÓDIGO DO RIJNDAEL EM QUE AS OPERAÇÕES SHIFT ROWS(PERMUTAÇÃO) E MIX COLUMNS(MULTIPLICAÇÃO) FORAM CONDENSADAS NA MATRIZ APRESENTADA EM 12.3.2. O CÓDIGO FONTE EONTRA-SE NO ITEM 12.3.4.

```
Texto em claro: 3243f6a8 885a308d 313198a2 e0370734      Chave: 2b7e1516 28aed2a6 abf71588 09cf4f3c

O bloco transformado após a 1a iteração (hexadecimal) é:        a49c7ff2 689f352b 6b5bea43 026a5049
O bloco transformado após a 2a iteração (hexadecimal) é:        aa8f5f03 61dde3ef 82d24ad2 6832469a
O bloco transformado após a 3a iteração (hexadecimal) é:        486c4eee 671d9d0d 4de3b138 d65f58e7
O bloco transformado após a 4a iteração (hexadecimal) é:        e0927fe8 c86363c0 d9b13550 85b8be01
O bloco transformado após a 5a iteração (hexadecimal) é:        f1006f55 c1924cef 7cc88b32 5db5d50c
O bloco transformado após a 6a iteração (hexadecimal) é:        260e2e17 3d41b77d e86472a9 fdd28b25
O bloco transformado após a 7a iteração (hexadecimal) é:        5a4142b1 1949dc1f a3e01965 7a8c040c
O bloco transformado após a 8a iteração (hexadecimal) é:        ea835cf0 0445332d 655d98ad 8596b0c5
O bloco transformado após a 9a iteração (hexadecimal) é:        eb40f21e 592e3884 8ba113e7 1bc342d2
O criptograma apos Sub Byte exterior ao laço (hexadecimal) é:   e9098972 cb31075f 3d327d94 af2e2cb5
O criptograma apos Shift Rows exterior ao laço (hexadecimal) é: e9317db5 cb322c72 3d2e895f af090794
O criptograma (hexadecimal) é:                                  3925841d 02dc09fb dc118597 196a0b32
```

12.3.4 CÓDIGO FONTE (EM JAVA)

```
/*********************************************************************
*   CÓDIGO UTILIZADO PARA GERAR AS MATRIZES APRESENTADAS EM C.1, C.2 E O *
* TESTE DE C.4                                                        *
* AUTORES: JORGE DE A. LAMBERT, ANDERSON R. FERREIRA, FILLIPE M.P.    *
* NAPOLITANO, IME, 15 OUT 2003                                        *
*********************************************************************/
public class rt_avalanche{

public static int SR[] = {
0, 1, 2, 3, 4, 5, 6, 7,
40,  41,  42,  43,  44,  45,  46,  47,
80,  81,  82,  83,  84,  85,  86,  87,
120, 121, 122, 123, 124, 125, 126, 127,
32,  33,  34,  35,  36,  37,  38,  39,
72,  73,  74,  75,  76,  77,  78,  79,
112, 113, 114, 115, 116, 117, 118, 119,
24,  25,  26,  27,  28,  29,  30,  31,
64,  65,  66,  67,  68,  69,  70,  71,
104, 105, 106, 107, 108, 109, 110, 111,
16,  17,  18,  19,  20,  21,  22,  23,
56,  57,  58,  59,  60,  61,  62,  63,
96,  97,  98,  99,  100, 101, 102, 103,
8, 9, 10,  11,  12,  13,  14,  15,
48,  49,  50,  51,  52,  53,  54,  55,
88,  89,  90,  91,  92,  93,  94,  95};

public static int M[][] = {
{0, 1, 0, 0, 0, 0, 0, 0,  1, 1, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0},{
0, 0, 1, 0, 0, 0, 0, 0,  0, 1, 1, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0},{
0, 0, 0, 1, 0, 0, 0, 0,  0, 0, 1, 1, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0},{
1, 0, 0, 0, 1, 0, 0, 0,  1, 0, 0, 1, 1, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0},{
1, 0, 0, 0, 0, 1, 0, 0,  1, 0, 0, 0, 1, 1, 0, 0,  0, 0, 0, 0, 1, 0, 0, 0,
```

```
0, 0, 0, 0, 1, 0, 0, 0},{
0, 0, 0, 0, 0, 0, 1, 0,  0, 0, 0, 0, 0, 1, 1, 0,  0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0},{
1, 0, 0, 0, 0, 0, 0, 1,  1, 0, 0, 0, 0, 0, 1, 1,  0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 1, 0},{
1, 0, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 1,  0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 1},{

1, 0, 0, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0,  1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0},{
0, 1, 0, 0, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0,  0, 1, 1, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0},{
0, 0, 1, 0, 0, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0,  0, 0, 1, 1, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0},{
0, 0, 0, 1, 0, 0, 0, 0,  1, 0, 0, 0, 1, 0, 0, 0,  1, 0, 0, 1, 1, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0},{
0, 0, 0, 0, 1, 0, 0, 0,  1, 0, 0, 0, 0, 1, 0, 0,  1, 0, 0, 0, 1, 1, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0},{
0, 0, 0, 0, 0, 1, 0, 0,  0, 0, 0, 0, 0, 0, 1, 0,  0, 0, 0, 0, 0, 1, 1, 0,
0, 0, 0, 0, 0, 1, 0, 0},{
0, 0, 0, 0, 0, 0, 1, 0,  1, 0, 0, 0, 0, 0, 0, 1,  1, 0, 0, 0, 0, 0, 1, 1,
0, 0, 0, 0, 0, 0, 1, 0},{
0, 0, 0, 0, 0, 0, 0, 1,  1, 0, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 1},{

1, 0, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0,
1, 1, 0, 0, 0, 0, 0, 0},{
0, 1, 0, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0,
0, 1, 1, 0, 0, 0, 0, 0},{
0, 0, 1, 0, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 1, 1, 0, 0, 0, 0},{
0, 0, 0, 1, 0, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0,  1, 0, 0, 0, 1, 0, 0, 0,
1, 0, 0, 1, 1, 0, 0, 0},{
0, 0, 0, 0, 1, 0, 0, 0,  0, 0, 0, 0, 1, 0, 0, 0,  1, 0, 0, 0, 0, 1, 0, 0,
1, 0, 0, 0, 1, 1, 0, 0},{
0, 0, 0, 0, 0, 1, 0, 0,  0, 0, 0, 0, 0, 1, 0, 0,  0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 1, 1, 0},{
0, 0, 0, 0, 0, 0, 1, 0,  0, 0, 0, 0, 0, 0, 1, 0,  1, 0, 0, 0, 0, 0, 0, 1,
1, 0, 0, 0, 0, 0, 1, 1},{
```

```
0, 0, 0, 0, 0, 0, 0, 1,  0, 0, 0, 0, 0, 0, 0, 1,  1, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 1},{

1, 1, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0},{
0, 1, 1, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0},{
0, 0, 1, 1, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0},{
1, 0, 0, 1, 1, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 0, 0},{
1, 0, 0, 0, 1, 1, 0, 0,  0, 0, 0, 0, 1, 0, 0, 0,  0, 0, 0, 0, 1, 0, 0, 0,
1, 0, 0, 0, 0, 1, 0, 0},{
0, 0, 0, 0, 0, 1, 1, 0,  0, 0, 0, 0, 0, 1, 0, 0,  0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0},{
1, 0, 0, 0, 0, 0, 1, 1,  0, 0, 0, 0, 0, 0, 1, 0,  0, 0, 0, 0, 0, 0, 1, 0,
1, 0, 0, 0, 0, 0, 0, 1},{
1, 0, 0, 0, 0, 0, 0, 1,  0, 0, 0, 0, 0, 0, 0, 1,  0, 0, 0, 0, 0, 0, 0, 1,
1, 0, 0, 0, 0, 0, 0, 0}};

public static int[][] E(int KEY[]){
int XK[][] = new int [11][128];
int temp1[]= new int[8];
int temp2[]= new int[8];
int temp3[]= new int[8];
int temp4[]= new int[8];
int temptemp[]= new int[8];
int i,j,c,c2;
int RCON [][]={{0,0,0,0,0,0,0,0},
               {0,0,0,0,0,0,0,1},// este byte e o RCon[1]
               {0,0,0,0,0,0,1,0},// este byte e o RCon[2]
               {0,0,0,0,0,1,0,0},// este byte e o RCon[3]
               {0,0,0,0,1,0,0,0},// este byte e o RCon[4]
               {0,0,0,1,0,0,0,0},// este byte e o RCon[5]
               {0,0,1,0,0,0,0,0},// este byte e o RCon[6]
               {0,1,0,0,0,0,0,0},// este byte e o RCon[7]
               {1,0,0,0,0,0,0,0},// este byte e o RCon[8]
               {0,0,0,1,1,0,1,1},// este byte e o RCon[9]
               {0,0,1,1,0,1,1,0}};//este byte e o RCon[10]
```

```
for (j=0;j<128;j++)// atribui a chave a primeira sub-chave
    XK[0][j]=KEY[j];
for (i=1;i<11;i++){// i do FIPS para chave do round i. no ARK inicial i=0.
    for (c=0;c<8;c++){//atribui a word W[i-1,3] a temp
        temp1[c]=XK[i-1][c+96];
        temp2[c]=XK[i-1][c+104];
        temp3[c]=XK[i-1][c+112];
        temp4[c]=XK[i-1][c+120];
        }//fim do for c
    //ROTWORD
    temptemp=temp1;
    temp1=temp2;
    temp2=temp3;
    temp3=temp4;
    temp4=temptemp;
    //fim do ROTWORD

    //BYTESUB WORD
    temp1=SBS(temp1);
    temp2=SBS(temp2);
    temp3=SBS(temp3);
    temp4=SBS(temp4);
    //fim de BYTESUB WORD

     for (c=0;c<8;c++){//soma RCon e faz XOR para gerar W[i,0],
     // que é a primeira WORD(de 32 bits) da sub-chave i
        temp1[c]=(temp1[c]+RCON[i][c])%2;
        XK[i][c]=(temp1[c]+XK[i-1][c])%2;
        XK[i][c+8]=(temp2[c]+XK[i-1][c+8])%2;
        XK[i][c+16]=(temp3[c]+XK[i-1][c+16])%2;
        XK[i][c+24]=(temp4[c]+XK[i-1][c+24])%2;
        }
     for (j=1;j<4;j++){//para gerar as WORDS W[i,1],W[i,2],W[i,3]
        for (c=0;c<32;c++){//
            c2=j*32+c;
            XK[i][c2]=(XK[i-1][c2]+XK[i][c2-32])%2;
            }//fim do for c
        }//fim do for j
```

```
     }//fim do for i
return XK;
}

public static void exibe(int vetor[]){
int temp[]=new int [4];
for (int j=0;j<128;j++){// exibe criptograma hexa
        if ((j%4)==0){
               temp[0]=vetor[j];
               temp[1]=vetor[j+1];
               temp[2]=vetor[j+2];
               temp[3]=vetor[j+3];
               System.out.print(hexa(temp));
               }
         if (j>0){
               if (((j+1)%32)==0) System.out.print(" ");
               }
    }// fim do for j de exibicao do criptograma hexa
System.out.println(" ");
}

public static String hexa(int bin[]){
String []
Simbolos={"0","1","2","3","4","5","6","7","8","9","a","b","c","d","e","f"};
String h= Simbolos[(8*bin[0]+4*bin[1]+2*bin[2]+bin[3])];
return h;
}

public static int[] SBS(int xy[]){//byte xy e o que sera substituido por zw
int lin,col,z;
int[] zw=new int[8];
int BS[][] = {
{ 0,1,1,0,0,0,1,1, 0,1,1,1,1,1,0,0, 0,1,1,1,0,1,1,1, 0,1,1,1,1,0,1,1,
1,1,1,1,0,0,1,0, 0,1,1,0,1,0,1,1, 0,1,1,0,1,1,1,1, 1,1,0,0,0,1,0,1,
0,0,1,1,0,0,0,0, 0,0,0,0,0,0,0,1, 0,1,1,0,0,1,1,1, 0,0,1,0,1,0,1,1,
1,1,1,1,1,1,1,0, 1,1,0,1,0,1,1,1, 1,0,1,0,1,0,1,1, 0,1,1,1,0,1,1,0},
{ 1,1,0,0,1,0,1,0, 1,0,0,0,0,0,1,0, 1,1,0,0,1,0,0,1, 0,1,1,1,1,1,0,1,
1,1,1,1,1,0,1,0, 0,1,0,1,1,0,0,1, 0,1,0,0,0,1,1,1, 1,1,1,1,0,0,0,0,
1,0,1,0,1,1,0,1, 1,1,0,1,0,1,0,0, 1,0,1,0,0,0,1,0, 1,0,1,0,1,1,1,1,
```

```
1,0,0,1,1,1,0,0, 1,0,1,0,0,1,0,0, 0,1,1,1,0,0,1,0, 1,1,0,0,0,0,0,0},
{ 1,0,1,1,0,1,1,1, 1,1,1,1,1,1,0,1, 1,0,0,1,0,0,1,1, 0,0,1,0,0,1,1,0,
0,0,1,1,0,1,1,0, 0,0,1,1,1,1,1,1, 1,1,1,1,0,1,1,1, 1,1,0,0,1,1,0,0,
0,0,1,1,0,1,0,0, 1,0,1,0,0,1,0,1, 1,1,1,0,0,1,0,1, 1,1,1,1,0,0,0,1,
0,1,1,1,0,0,0,1, 1,1,0,1,1,0,0,0, 0,0,1,1,0,0,0,1, 0,0,0,1,0,1,0,1},
{ 0,0,0,0,0,1,0,0, 1,1,0,0,0,1,1,1, 0,0,1,0,0,0,1,1, 1,1,0,0,0,0,1,1,
0,0,0,1,1,0,0,0, 1,0,0,1,0,1,1,0, 0,0,0,0,0,1,0,1, 1,0,0,1,1,0,1,0,
0,0,0,0,0,1,1,1, 0,0,0,1,0,0,1,0, 1,0,0,0,0,0,0,0, 1,1,1,0,0,0,1,0,
1,1,1,0,1,0,1,1, 0,0,1,0,0,1,1,1, 1,0,1,1,0,0,1,0, 0,1,1,1,0,1,0,1},
{ 0,0,0,0,1,0,0,1, 1,0,0,0,0,0,1,1, 0,0,1,0,1,1,0,0, 0,0,0,1,1,0,1,0,
0,0,0,1,1,0,1,1, 0,1,1,0,1,1,1,0, 0,1,0,1,1,0,1,0, 1,0,1,0,0,0,0,0,
0,1,0,1,0,0,1,0, 0,0,1,1,1,0,1,1, 1,1,0,1,0,1,1,0, 1,0,1,1,0,0,1,1,
0,0,1,0,1,0,0,1, 1,1,1,0,0,0,1,1, 0,0,1,0,1,1,1,1, 1,0,0,0,0,1,0,0},
{ 0,1,0,1,0,0,1,1, 1,1,0,1,0,0,0,1, 0,0,0,0,0,0,0,0, 1,1,1,0,1,1,0,1,
0,0,1,0,0,0,0,0, 1,1,1,1,1,1,0,0, 1,0,1,1,0,0,0,1, 0,1,0,1,1,0,1,1,
0,1,1,0,1,0,1,0, 1,1,0,0,1,0,1,1, 1,0,1,1,1,1,1,0, 0,0,1,1,1,0,0,1,
0,1,0,0,1,0,1,0, 0,1,0,0,1,1,0,0, 0,1,0,1,1,0,0,0, 1,1,0,0,1,1,1,1},
{ 1,1,0,1,0,0,0,0, 1,1,1,0,1,1,1,1, 1,0,1,0,1,0,1,0, 1,1,1,1,1,0,1,1,
0,1,0,0,0,0,1,1, 0,1,0,0,1,1,0,1, 0,0,1,1,0,0,1,1, 1,0,0,0,0,1,0,1,
0,1,0,0,0,1,0,1, 1,1,1,1,1,1,0,0,1, 0,0,0,0,0,0,1,0, 0,1,1,1,1,1,1,1,
0,1,0,1,0,0,0,0, 0,0,1,1,1,1,1,0,0, 1,0,0,1,1,1,1,1, 1,0,1,0,1,0,0,0},
{ 0,1,0,1,0,0,0,1, 1,0,1,0,0,0,1,1, 0,1,0,0,0,0,0,0, 1,0,0,0,1,1,1,1,
1,0,0,1,0,0,1,0, 1,0,0,1,1,1,0,1, 0,0,1,1,1,0,0,0, 1,1,1,1,0,1,0,1,
1,0,1,1,1,1,0,0, 1,0,1,1,0,1,1,0, 1,1,0,1,1,0,1,0, 0,0,1,0,0,0,0,1,
0,0,0,1,0,0,0,0, 1,1,1,1,1,1,1,1,1, 1,1,1,1,0,0,1,1, 1,1,0,1,0,0,1,0},
{ 1,1,0,0,1,1,0,1, 0,0,0,0,1,1,0,0, 0,0,0,1,0,0,1,1, 1,1,1,0,1,1,0,0,
0,1,0,1,1,1,1,1, 1,0,0,1,0,1,1,1, 0,1,0,0,0,1,0,0, 0,0,0,1,0,1,1,1,
1,1,0,0,0,1,0,0, 1,0,1,0,0,1,1,1, 0,1,1,1,1,1,1,0, 0,0,1,1,1,1,0,1,
0,1,1,0,0,1,0,0, 0,1,0,1,1,1,0,1, 0,0,0,1,1,0,0,1, 0,1,1,1,0,0,1,1},
{ 0,1,1,0,0,0,0,0, 1,0,0,0,0,0,0,1, 0,1,0,0,1,1,1,1, 1,1,0,1,1,1,0,0,
0,0,1,0,0,0,1,0, 0,0,1,0,1,0,1,0, 1,0,0,1,0,0,0,0, 1,0,0,0,1,0,0,0,
0,1,0,0,0,1,1,0, 1,1,1,0,1,1,1,0, 1,0,1,1,1,0,0,0, 0,0,0,1,0,1,0,0,
1,1,0,1,1,1,1,0, 0,1,0,1,1,1,1,0, 0,0,0,0,1,0,1,1, 1,1,0,1,1,0,1,1},
{ 1,1,1,0,0,0,0,0, 0,0,1,1,0,0,1,0, 0,0,1,1,1,0,1,0, 0,0,0,0,1,0,1,0,
0,1,0,0,1,0,0,1, 0,0,0,0,0,0,1,1,0, 0,0,1,0,0,1,0,0, 0,1,0,1,1,1,0,0,
1,1,0,0,0,0,1,0, 1,1,0,1,0,0,1,1, 1,0,1,0,1,1,0,0, 0,1,1,0,0,0,1,0,
1,0,0,1,0,0,0,1, 1,0,0,1,0,1,0,1, 1,1,1,0,0,1,0,0, 0,1,1,1,1,0,0,1},
{ 1,1,1,0,0,1,1,1, 1,1,0,0,1,0,0,0, 0,0,1,1,0,1,1,1, 0,1,1,0,1,1,0,1,
1,0,0,0,1,1,0,1, 1,1,0,1,0,1,0,1, 0,1,0,0,1,1,1,0, 1,0,1,0,1,0,0,1,
```

```
0,1,1,0,1,1,0,0, 0,1,0,1,0,1,1,0, 1,1,1,1,0,1,0,0, 1,1,1,0,1,0,1,0,
0,1,1,0,0,1,0,1, 0,1,1,1,1,0,1,0, 1,0,1,0,1,1,1,0, 0,0,0,0,1,0,0,0},
{ 1,0,1,1,1,0,1,0, 0,1,1,1,1,0,0,0, 0,0,1,0,0,1,0,1, 0,0,1,0,1,1,1,0,
0,0,0,1,1,1,0,0, 1,0,1,0,0,1,1,0, 1,0,1,1,0,1,0,0, 1,1,0,0,0,1,1,0,
1,1,1,0,1,0,0,0, 1,1,0,1,1,1,0,1, 0,1,1,1,0,1,0,0, 0,0,0,1,1,1,1,1,
0,1,0,0,1,0,1,1, 1,0,1,1,1,1,0,1, 1,0,0,0,1,0,1,1, 1,0,0,0,1,0,1,0},
{ 0,1,1,1,0,0,0,0, 0,0,1,1,1,1,1,0, 1,0,1,1,0,1,0,1, 0,1,1,0,0,1,1,0,
0,1,0,0,1,0,0,0, 0,0,0,0,0,0,1,1, 1,1,1,1,0,1,1,0, 0,0,0,0,1,1,1,0,
0,1,1,0,0,0,0,1, 0,0,1,1,0,1,0,1, 0,1,0,1,0,1,1,1, 1,0,1,1,1,0,0,1,
1,0,0,0,0,1,1,0, 1,1,0,0,0,0,0,1, 0,0,0,1,1,1,0,1, 1,0,0,1,1,1,1,0},
{ 1,1,1,0,0,0,0,1, 1,1,1,1,1,0,0,0, 1,0,0,1,1,0,0,0, 0,0,0,1,0,0,0,1,
0,1,1,0,1,0,0,1, 1,1,0,1,1,0,0,1, 1,0,0,0,1,1,1,0, 1,0,0,1,0,1,0,0,
1,0,0,1,1,0,1,1, 0,0,0,1,1,1,1,0, 1,0,0,0,0,1,1,1, 1,1,1,0,1,0,0,1,
1,1,0,0,1,1,1,0, 0,1,0,1,0,1,0,1, 0,0,1,0,1,0,0,0, 1,1,0,1,1,1,1,1},
{ 1,0,0,0,1,1,0,0, 1,0,1,0,0,0,0,1, 1,0,0,0,1,0,0,1, 0,0,0,0,1,1,0,1,
1,0,1,1,1,1,1,1, 1,1,1,0,0,1,1,0, 0,1,0,0,0,0,1,0, 0,1,1,0,1,0,0,0,
0,1,0,0,0,0,0,1, 1,0,0,1,1,0,0,1, 0,0,1,0,1,1,0,1, 0,0,0,0,1,1,1,1,
1,0,1,1,0,0,0,0, 0,1,0,1,0,1,0,0, 1,0,1,1,1,0,1,1, 0,0,0,1,0,1,1,0}
};

/*********************************************
* converte byte - transformacao BS           *
*********************************************/
  lin=xy[0]*8+xy[1]*4+xy[2]*2+xy[3];//calcula linha de entrada na caixa

  col=xy[4]*8+xy[5]*4+xy[6]*2+xy[7];//calcula coluna de entrada na caixa

  col=col*8;// converte o valor de col para o caso da s-box expandida em 8 bits
  for (z=0;z<8;z++)// substitui byte bit a bit (operacao de ida)
  zw[z]=BS[lin][col+z];
return zw;
}// ********************fim do metodo bytesub ***************************

public static int[] PSR(int arg[]){//
int[] gar=new int[128];

for (int z=0;z<128;z++)// permutacao propriamente dita
   gar[z]=arg[SR[z]];
return gar;
```

```
}//fim do metodo PSR
/*************************************************************************
*  Constrói a matriz de avalanche de primeira ordem das traansformação    *
* linear do Rijndael que agrega as operações Shift Rows e Mix Columns     *
*************************************************************************/
public static int[] MC2(int arg[],int nr){//Faz o MIX COLUMNS com SHIFT ROWS

int ret[] =new int[128];
int M1[][]=new int[128][128];
int Avalanche1[][]=new int[128][128];

for (int w=0;w<4;w++){//gera matriz Mix columns para 128 bits
   for(int i=0;i<32;i++){
      for(int j=0;j<32;j++){
         M1[w*32+i][w*32+j]=M[i][j];
         }
     }
}


if (nr==1){

    System.out.println("A.1 MATRIZ [M]128X128, (Equivalente a MULTIPLICAÇÃO");
    for(int i=0;i<128;i++){//imprime matriz [M]128x128 (Mix Columns que opera 128 bites)
        for(int j=0;j<128;j++){
        System.out.print(M1[i][j]);
        }
        System.out.println();
    }
}
for(int i=0;i<128;i++){  //gera matriz avalanche de 1a ordem permutando a MixColumns 128x128
     for(int j=0;j<128;j++){
        Avalanche1[i][SR[j]]=M1[i][j];
     }
}

if (nr==1){
```

```
    System.out.println("A.2 MATRIZ DE AVALANCHE DE PRIMEIRA ORDEM DA TRANSFORMAÇÃO LINEAR DO RIJNDAEL = [M]128X128,
permutada(AGREGA PERMUTAÇÃO E MULTIPLICAÇÃO");

    for(int i=0;i<128;i++){//imprime matriz avalanche 1a ordem
     for(int j=0;j<128;j++){
        System.out.print(Avalanche1[i][j]);
        }
        System.out.println();
    }
}

for (int i=0;i<128;i++){// Transformação Mix_Columns Propriamente dita
  for (int j=0;j<128;j++){
   ret[i]= (Avalanche1[i][j]*arg[j]+ret[i])%2;
  }
}

return ret;
}// fim da transformação

public static void main(String[] args){
int cripto[]= new int [128];
int temp1 []= new int[8];

int chave[]= {0,0,1,0, 1,0,1,1, 0,1,1,1, 1,1,1,0, 0,0,0,1, 0,1,0,1, 0,0,0,1, 0,1,1,0,
  0,0,1,0, 1,0,0,0, 1,0,1,0, 1,1,1,0, 1,1,0,1, 0,0,1,0, 1,0,1,0, 0,1,1,0,
  1,0,1,0, 1,0,1,1, 1,1,1,1, 0,1,1,1, 0,0,0,1, 0,1,0,1, 1,0,0,0, 1,0,0,0,
  0,0,0,0, 1,0,0,1, 1,1,0,0, 1,1,1,1, 0,1,0,0, 1,1,1,1, 0,0,1,1, 1,1,0,0};

int mensagem[]={0,0,1,1, 0,0,1,0, 0,1,0,0, 0,0,1,1, 1,1,1,1, 0,1,1,0, 1,0,1,0, 1,0,0,0,
  1,0,0,0, 1,0,0,0, 0,1,0,1, 1,0,1,0, 0,0,1,1, 0,0,0,0, 1,0,0,0, 1,1,0,1,
  0,0,1,1, 0,0,0,1, 0,0,1,1, 0,0,0,1, 1,0,0,1, 1,0,0,0, 1,0,1,0, 0,0,1,0,
  1,1,1,0, 0,0,0,0, 0,0,1,1, 0,1,1,1, 0,0,0,0, 0,1,1,1, 0,0,1,1, 0,1,0,0};

int K[][]= E(chave);                    // (EXPAND KEY)
int Nr=9;

System.out.print("Texto em claro: ");
exibe(mensagem);
```

```
System.out.println();
System.out.print("Chave: ");
exibe(chave);

for (int bit=0;bit<128;bit++){
          cripto[bit]=(mensagem[bit]+K[0][bit])%2;// Soma Sub-chave
          // (ADD ROUND KEY Inicial)
          }
for (int r=1;r<Nr+1;r++){// Inicia as 9 iteracoes
      for (int byt=0;byt<16;byt++){
           for (int bit=0;bit<8;bit++)
                temp1[bit]=cripto[byt*8+bit];
           temp1=SBS(temp1);   // Substituicao (BYTE SUB)
           for (int bit=0;bit<8;bit++)
                cripto[byt*8+bit]=temp1[bit];
           }// fim do for byt
          // Multip por M avalanche (MIX COLUMNS+SHIFT ROWS)
          cripto=MC2(cripto,r);
          for (int bit=0;bit<128;bit++)
          cripto[bit]=(cripto[bit]+K[r][bit])%2;//Soma Sub-chave(ADD ROUND KEY )
       System.out.println();
          System.out.print("O bloco transformado após a "+r+"a iteração (hexadecimal) é: ");
          exibe(cripto);
       }// fim das iteracoes

for ( int byt=0;byt<16;byt++){
           for (int bit=0;bit<8;bit++)
                temp1[bit]=cripto[byt*8+bit];
           temp1=SBS(temp1);                        // Substituicao (BYTE SUB
//exterior)
        for (int bit=0;bit<8;bit++)
                cripto[byt*8+bit]=temp1[bit];
           }// fim do for byt
System.out.println();
System.out.print("O criptograma apos Sub Byte exterior (hexadecimal) e: ");
exibe(cripto);
cripto=PSR(cripto);    // Permutacao   (SHIFT ROWS exterior)
System.out.println();
System.out.print("O criptograma apos Shift Rows exterior (hexadecimal) e: ");
```

```
exibe(cripto);

for (int bit=0;bit<128;bit++)
          cripto[bit]=(cripto[bit]+K[10][bit])%2; // Soma Sub-chave (ADD
//ROUND KEY FINAL)
System.out.println();
System.out.print("O criptograma (hexadecimal) e: ");
exibe(cripto);


}// ******************* fim do metodo RIJNDAEL***********/
}
```

12.4 APÊNDICE 4: MATRIZES DE AVALANCHE DA TRANSFORMAÇÃO LINEAR DO RIJNDAEL (r = 1,2...,6)

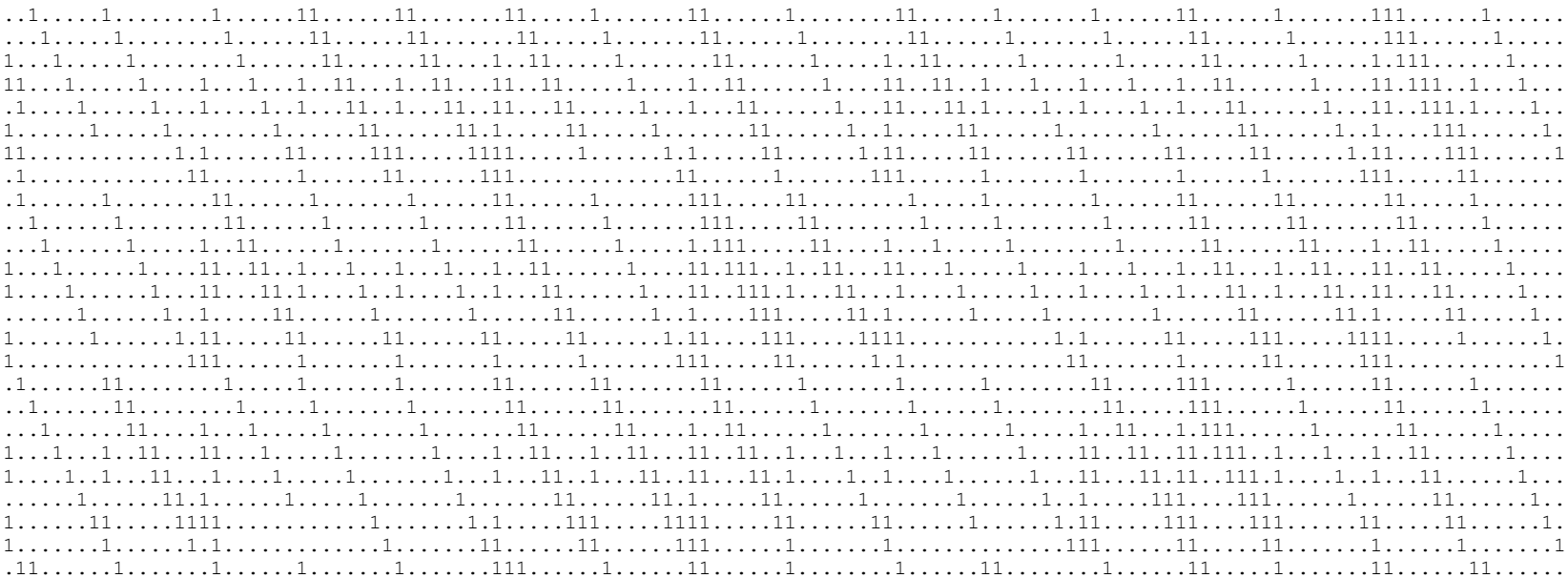
12.4.1 MATRIZ DE DEPENDÊNCIAS [D_1]

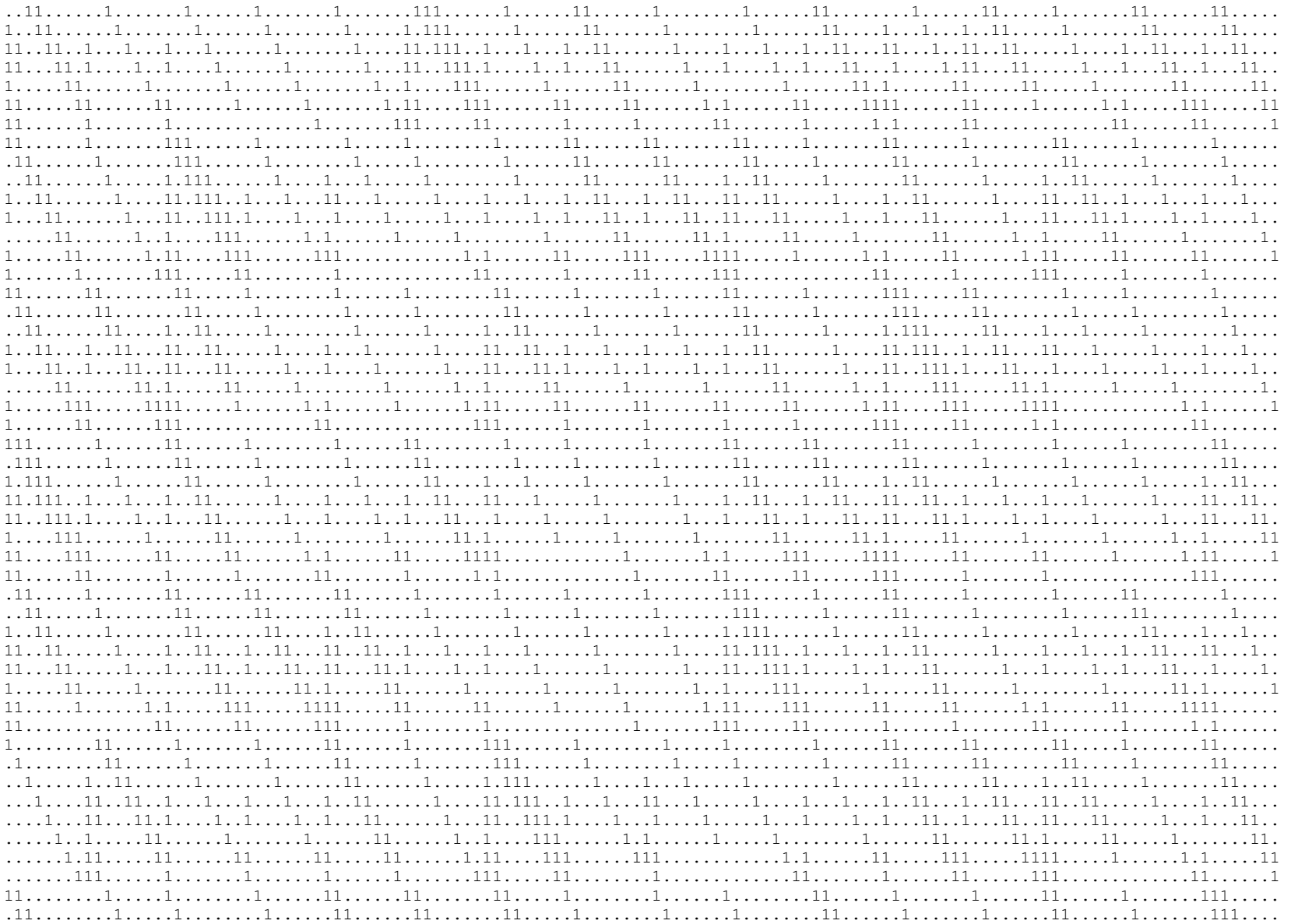
A matriz encontra-se em 12.3.2.

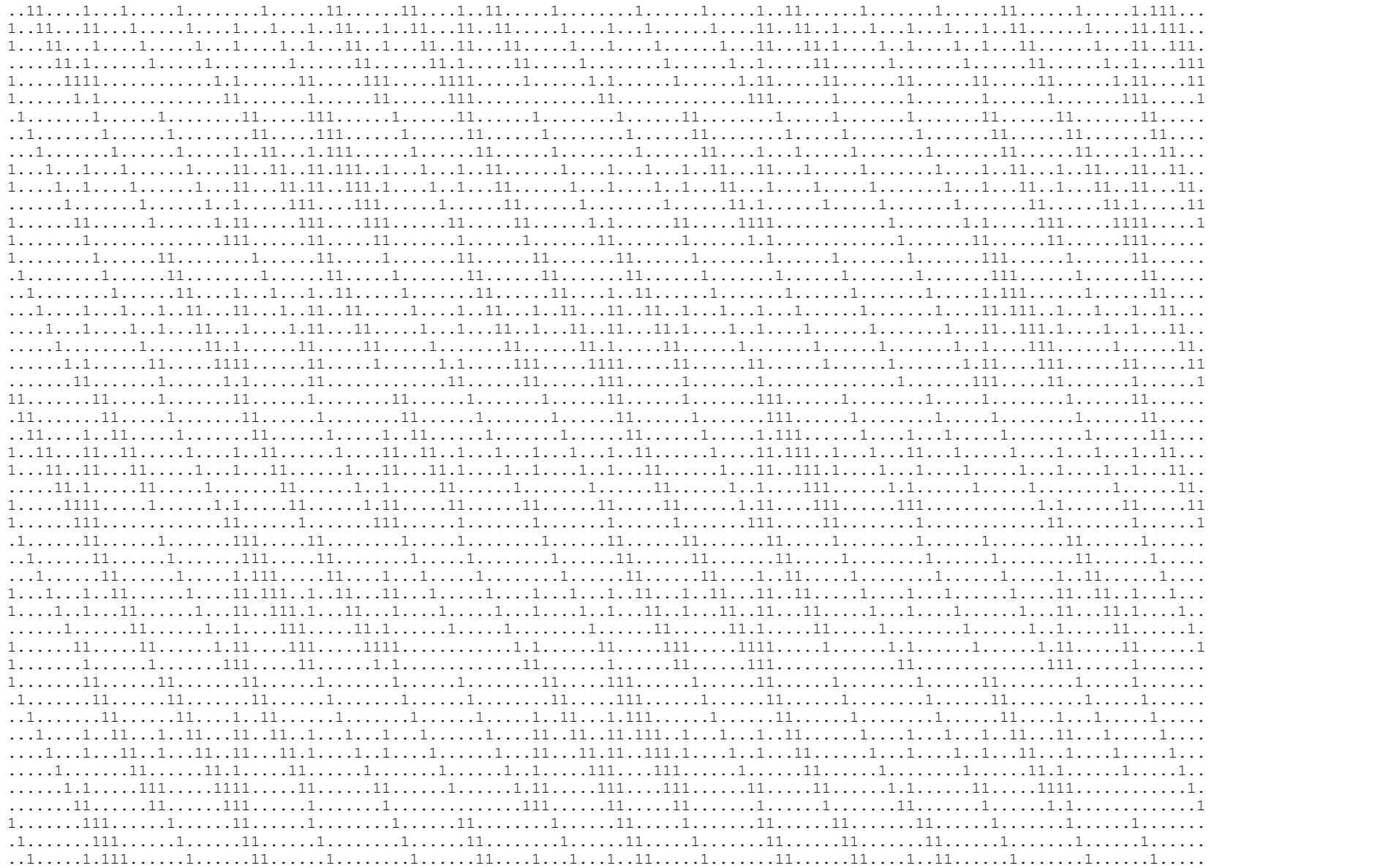
Limite Inferior do Poder de Avalanche = 5

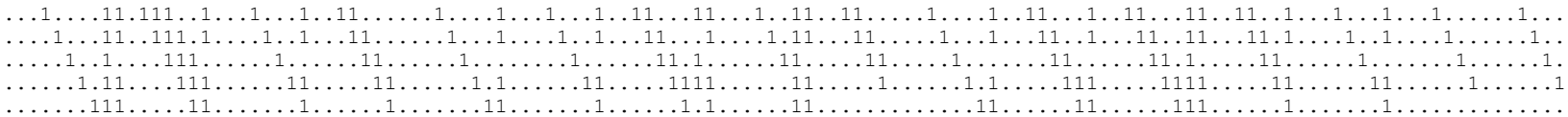
Limite Superior do Poder de Avalanche = 7

12.4.2 MATRIZ DE AVALANCHE [D_2] (duas iterações)



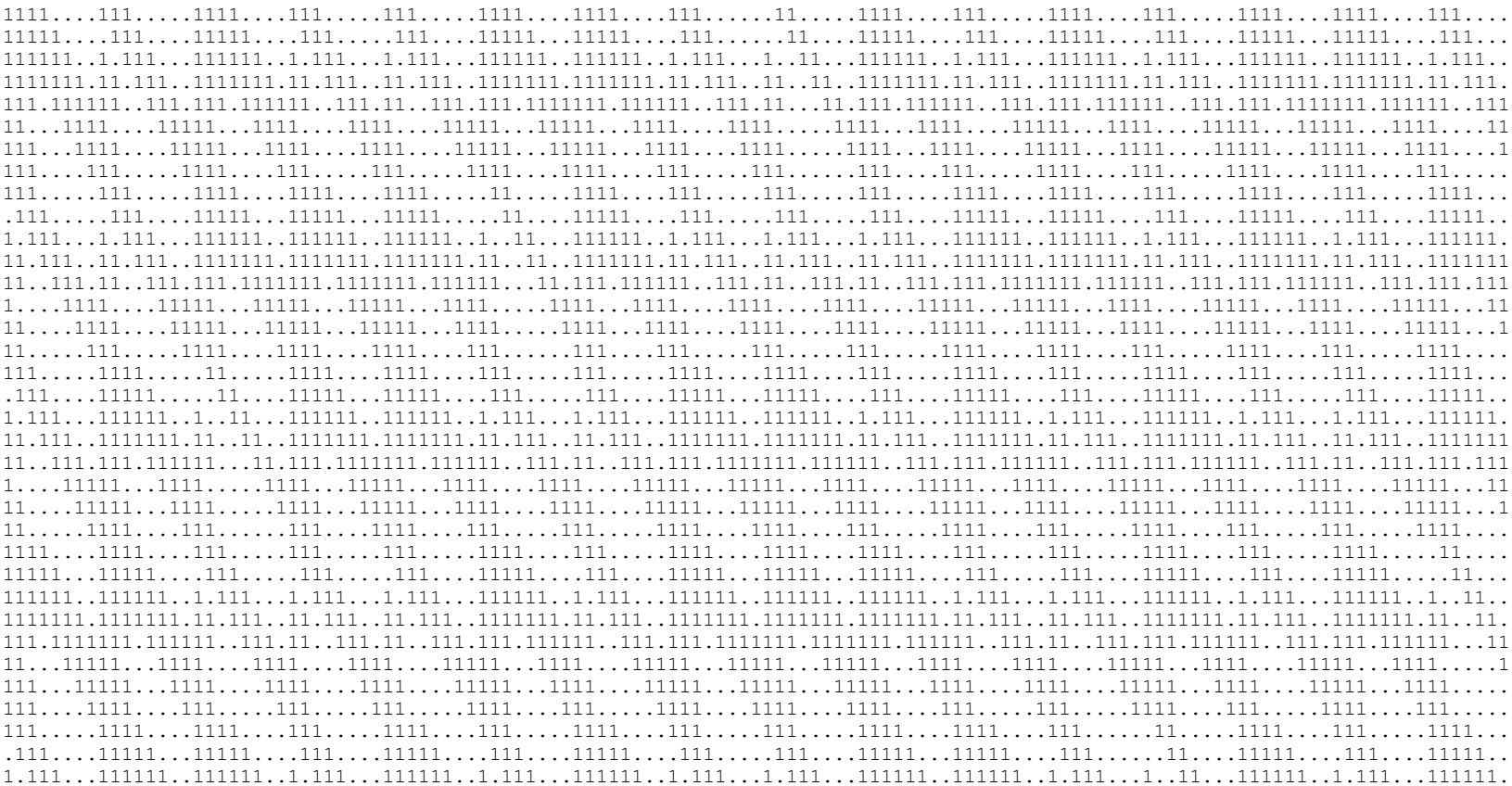


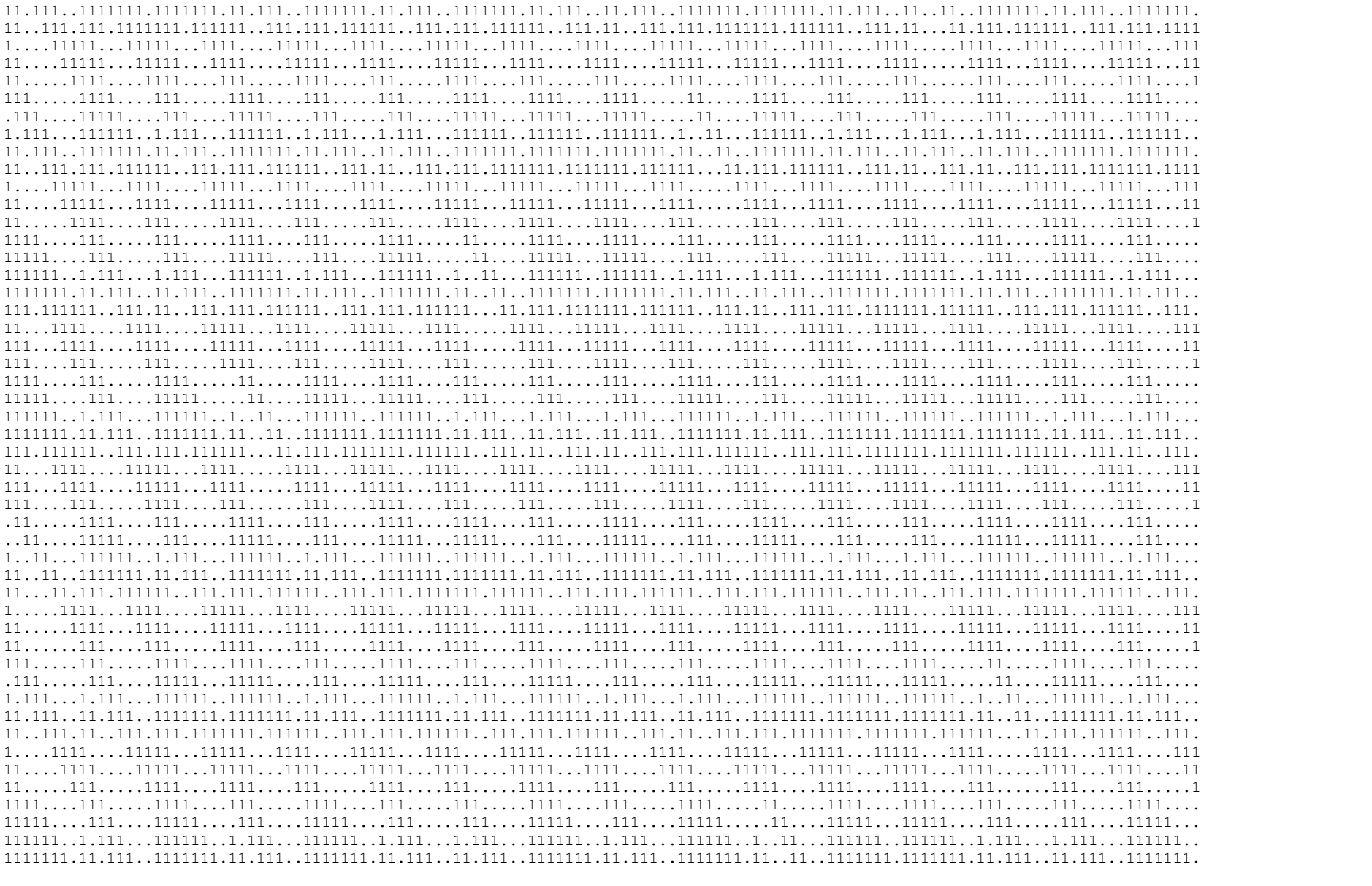


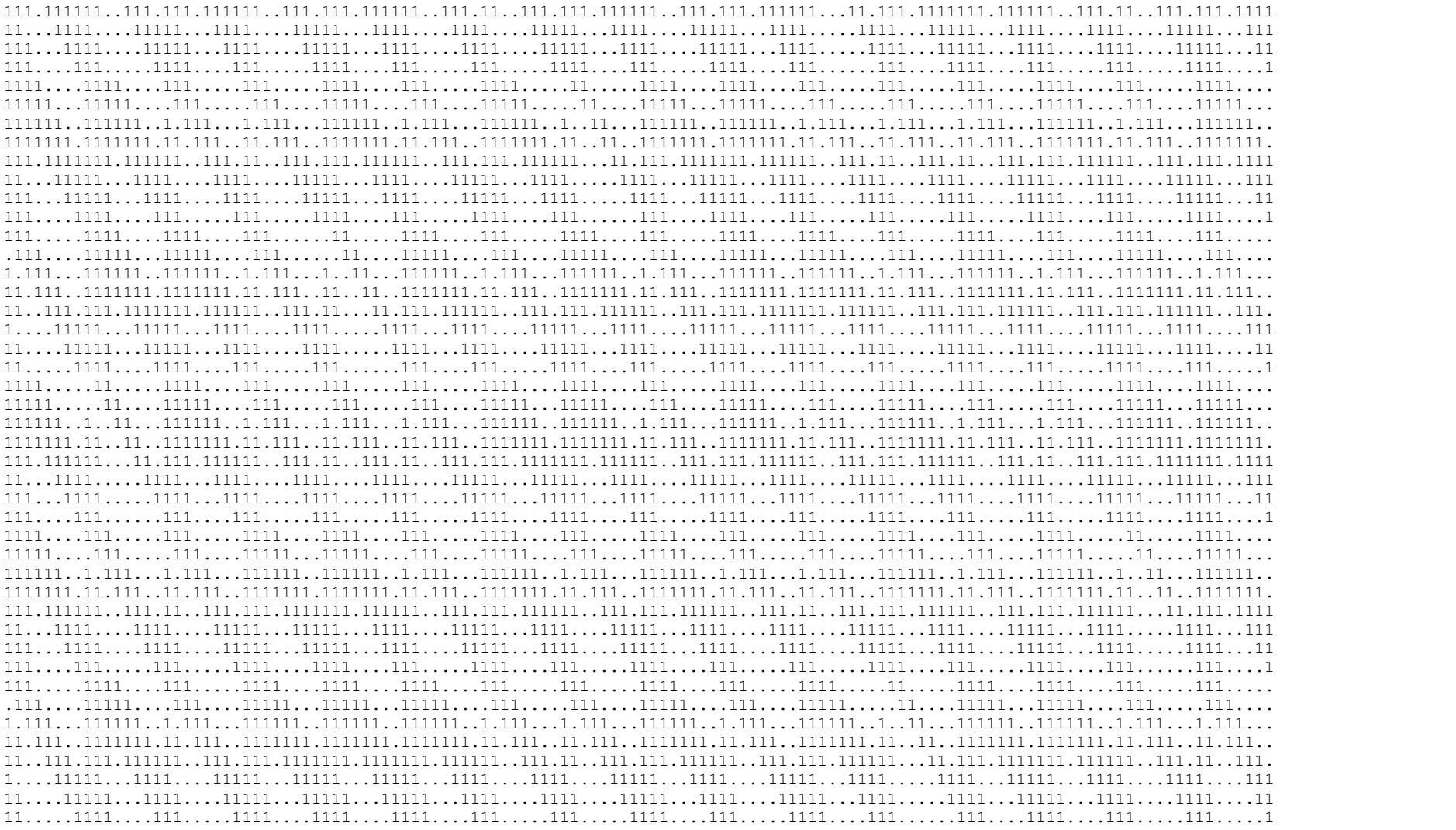


```
Limite Inferior do Poder de Avalanche = 24
Limite Superior do Poder de Avalanche = 40
```

12.4.3 MATRIZ DE AVALANCHE [D_3] (três iterações)



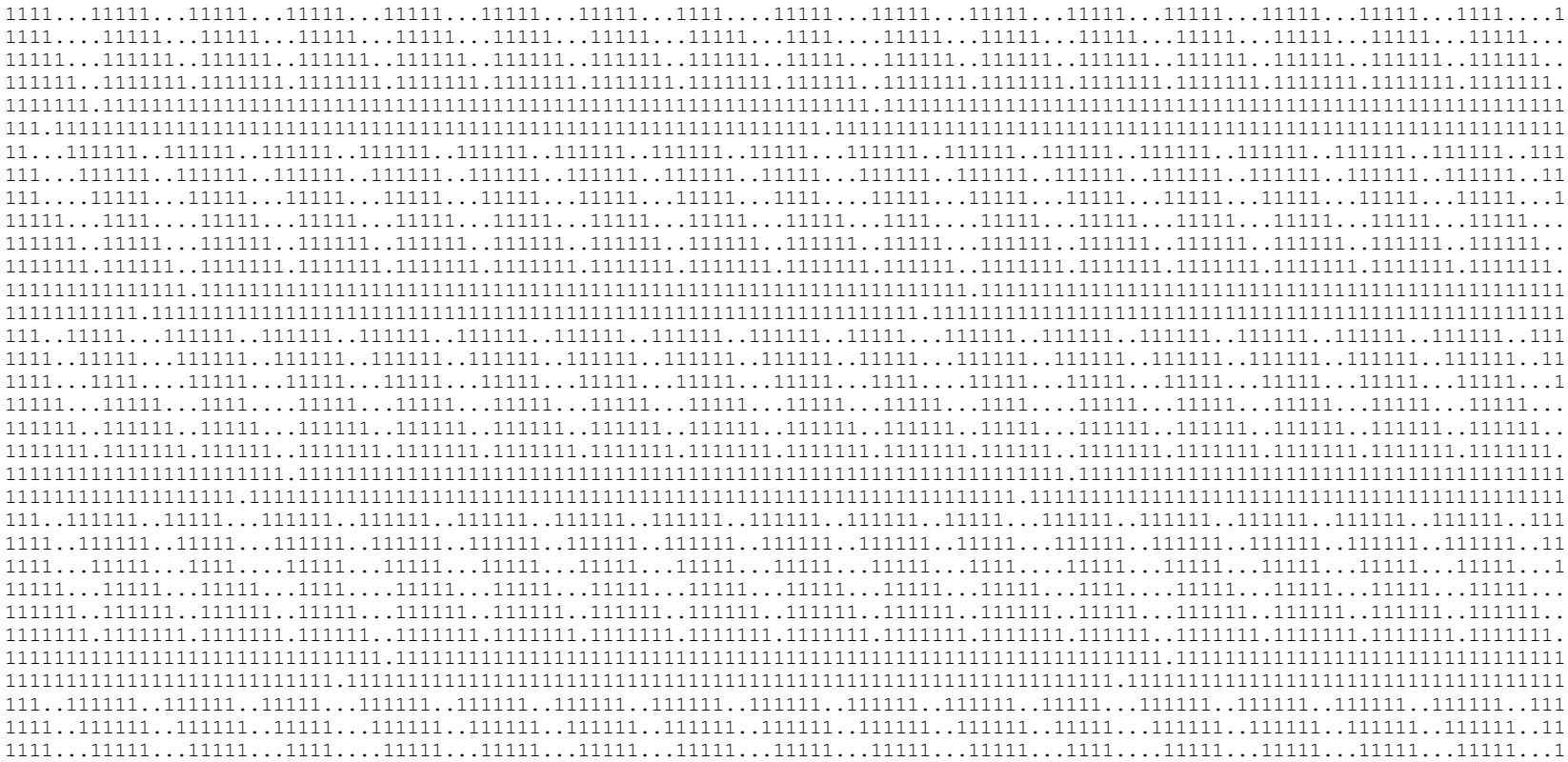




Limite Inferior do Poder de Avalanche = 55

Limite Superior do Poder de Avalanche = 95

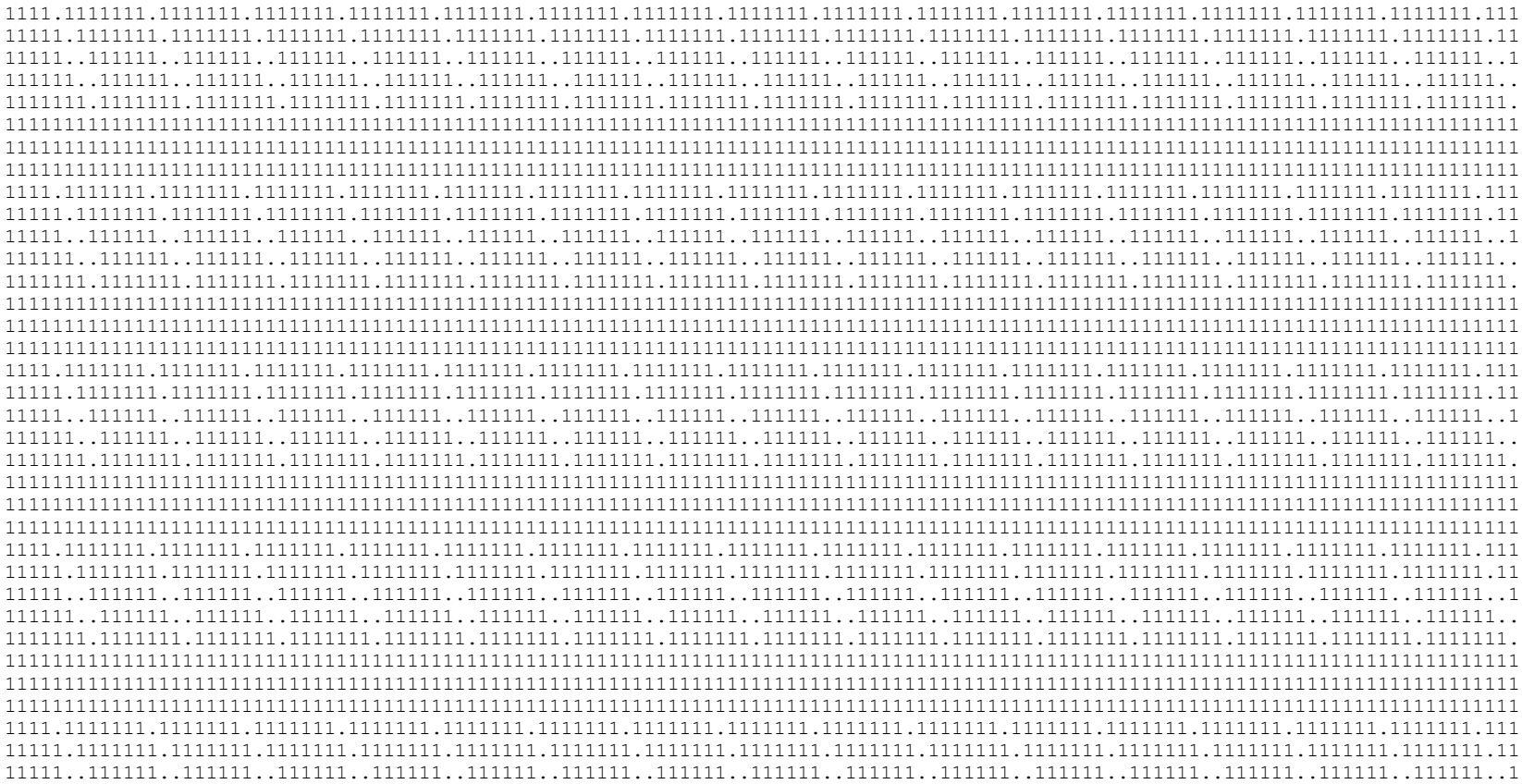
12.4.4 MATRIZ DE AVALANCHE [D_4] (quatro iterações)



Limite Inferior do Poder de Avalanche = 78

Limite Superior do Poder de Avalanche = 126

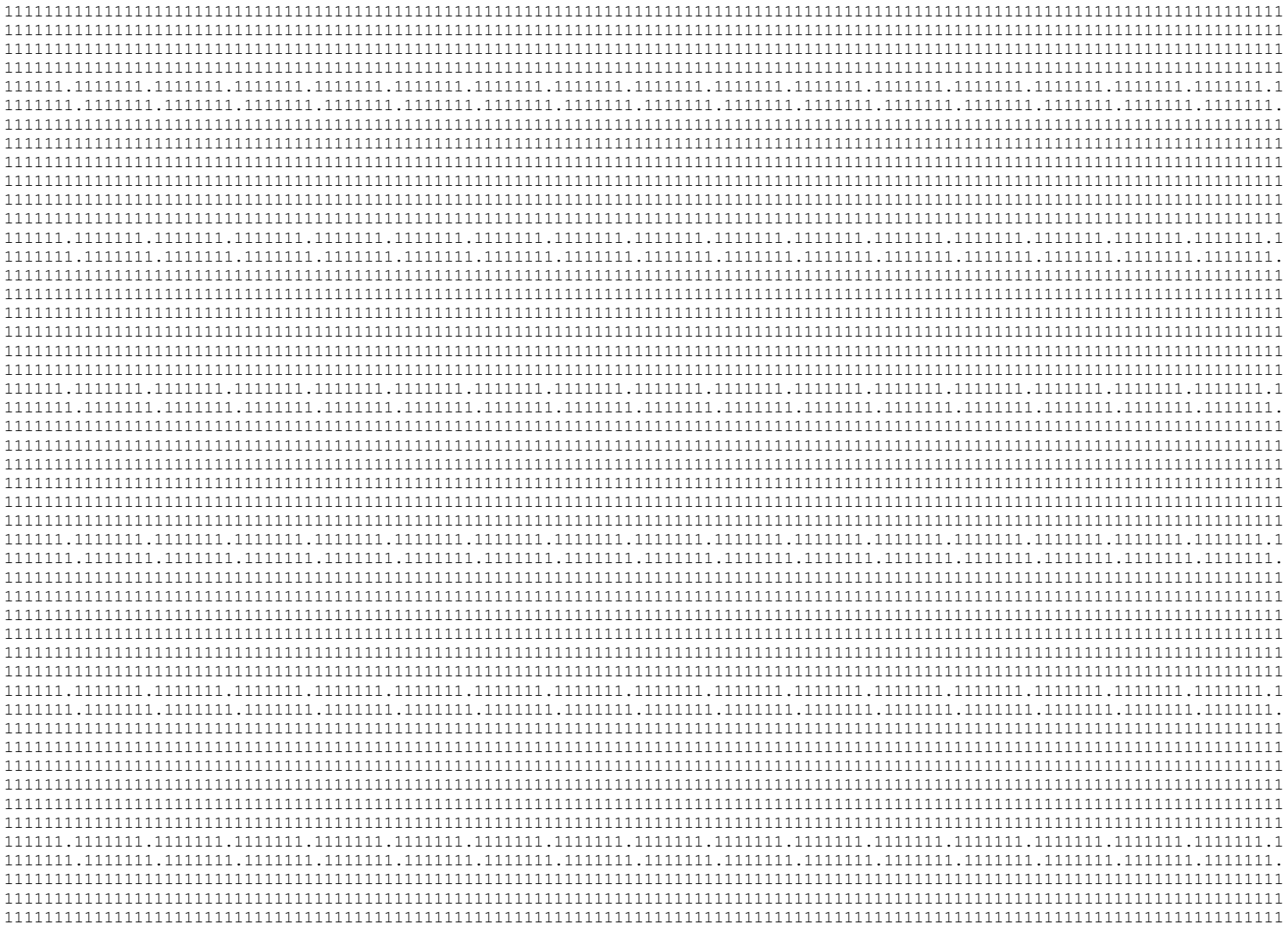
12.4.5 MATRIZ DE AVALANCHE [D_5] (cinco iterações)

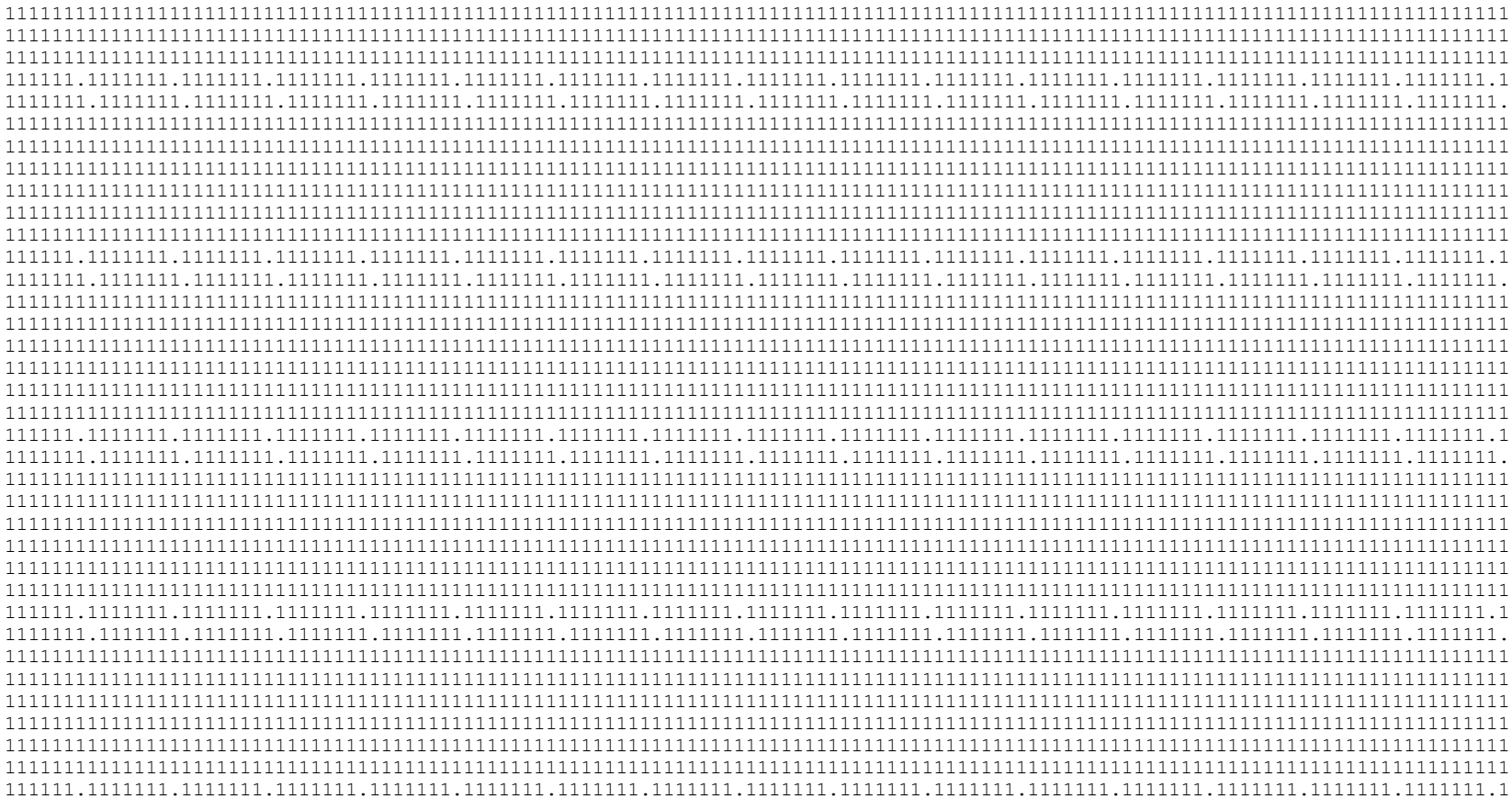


Limite Inferior do Poder de Avalanche 5 = 96

Limite Superior do Poder de Avalanche 5 = 128

12.4.6 MATRIZ DE AVALANCHE [D_6] (seis iterações)





Limite Inferior do Poder de Avalanche 6 = 112
Limite Superior do Poder de Avalanche 6 = 128

12.4.7 MATRIZ DE AVALANCHE [D_7] (sete iterações)

A matriz foi omitida por concisão. Trata-se de uma matriz *completa*, como pode ser visto pois seu LIPA=128.

12.4.8 CÓDIGO FONTE (EM JAVA)[213]

```
/*****************************************************************************************************
*   CÓDIGO UTILIZADO PARA GERAR O AS MATRIZES DE AVALANCHE DO RIJNDAEL APRESENTADAS EM D.1 A D.7     *
*   AUTORES: JORGE DE A. LAMBERT, ANDERSON R. FERREIRA, FILLIPE M.P. NAPOLITANO                      *
*   IME, 15 OUT 2003                                                                                *
*****************************************************************************************************/

public class rt_avalanche_apd_B {

    public static void main(String[] args) {

/*************************************************************************
*                                                                        *
*  definicao da matriz de transformação  32x32  (fonte) e da matriz      *
*  de permutação. Vide (LAMBERT, 2003)                                   *
*                                                                        *
*************************************************************************/

String temp;
int i,j,k,c;
int MC[][] = {
{0, 1, 0, 0, 0, 0, 0, 0,  1, 1, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 0},{
0, 0, 1, 0, 0, 0, 0, 0,  0, 1, 1, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0},{
0, 0, 0, 1, 0, 0, 0, 0,  0, 0, 1, 1, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0},{
1, 0, 0, 0, 1, 0, 0, 0,  1, 0, 0, 1, 1, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0},{
1, 0, 0, 0, 0, 1, 0, 0,  1, 0, 0, 0, 1, 1, 0, 0,  0, 0, 0, 0, 1, 0, 0, 0,  0, 0, 0, 0, 1, 0, 0, 0},{
0, 0, 0, 0, 0, 0, 1, 0,  0, 0, 0, 0, 0, 1, 1, 0,  0, 0, 0, 0, 0, 1, 0, 0,  0, 0, 0, 0, 0, 1, 0, 0},{
1, 0, 0, 0, 0, 0, 0, 1,  1, 0, 0, 0, 0, 0, 1, 1,  0, 0, 0, 0, 0, 0, 1, 0,  0, 0, 0, 0, 0, 0, 1, 0},{
1, 0, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 1,  0, 0, 0, 0, 0, 0, 0, 1,  0, 0, 0, 0, 0, 0, 0, 1},{

1, 0, 0, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0,  1, 1, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 0},{
0, 1, 0, 0, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0,  0, 1, 1, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0},{
0, 0, 1, 0, 0, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0,  0, 0, 1, 1, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0},{
```

[213] Inicialmente as MDR eram consideradas D_r para r+1 iterações. O código fonte não foi adaptado para D_r = matriz com r iterações. Portanto, ao ser executado, as MDR serão numeradas a partir de D_0 e não de D_1.

```
0, 0, 0, 1, 0, 0, 0, 0,  1, 0, 0, 0, 1, 0, 0, 0,  1, 0, 0, 1, 1, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0},{
0, 0, 0, 0, 1, 0, 0, 0,  1, 0, 0, 0, 0, 1, 0, 0,  1, 0, 0, 0, 1, 1, 0, 0,  0, 0, 0, 0, 1, 0, 0, 0},{
0, 0, 0, 0, 0, 1, 0, 0,  0, 0, 0, 0, 0, 0, 1, 0,  0, 0, 0, 0, 0, 1, 1, 0,  0, 0, 0, 0, 0, 1, 0, 0},{
0, 0, 0, 0, 0, 0, 1, 0,  1, 0, 0, 0, 0, 0, 0, 1,  1, 0, 0, 0, 0, 0, 1, 1,  0, 0, 0, 0, 0, 0, 1, 0},{
0, 0, 0, 0, 0, 0, 0, 1,  1, 0, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 1,  0, 0, 0, 0, 0, 0, 0, 1},{

1, 0, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0,  1, 1, 0, 0, 0, 0, 0, 0},{
0, 1, 0, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0,  0, 1, 1, 0, 0, 0, 0, 0},{
0, 0, 1, 0, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0,  0, 0, 1, 1, 0, 0, 0, 0},{
0, 0, 0, 1, 0, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0,  1, 0, 0, 0, 1, 0, 0, 0,  1, 0, 0, 1, 1, 0, 0, 0},{
0, 0, 0, 0, 1, 0, 0, 0,  0, 0, 0, 0, 1, 0, 0, 0,  1, 0, 0, 0, 0, 1, 0, 0,  1, 0, 0, 0, 1, 1, 0, 0},{
0, 0, 0, 0, 0, 1, 0, 0,  0, 0, 0, 0, 0, 1, 0, 0,  0, 0, 0, 0, 0, 0, 1, 0,  0, 0, 0, 0, 0, 1, 1, 0},{
0, 0, 0, 0, 0, 0, 1, 0,  0, 0, 0, 0, 0, 0, 1, 0,  1, 0, 0, 0, 0, 0, 0, 1,  1, 0, 0, 0, 0, 0, 1, 1},{
0, 0, 0, 0, 0, 0, 0, 1,  0, 0, 0, 0, 0, 0, 0, 1,  1, 0, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 1},{

1, 1, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 0,  1, 0, 0, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0},{
0, 1, 1, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0,  0, 1, 0, 0, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0},{
0, 0, 1, 1, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0,  0, 0, 1, 0, 0, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0},{
1, 0, 0, 1, 1, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0,  0, 0, 0, 1, 0, 0, 0, 0,  1, 0, 0, 0, 1, 0, 0, 0},{
1, 0, 0, 0, 1, 1, 0, 0,  0, 0, 0, 0, 1, 0, 0, 0,  0, 0, 0, 0, 1, 0, 0, 0,  1, 0, 0, 0, 0, 1, 0, 0},{
0, 0, 0, 0, 0, 1, 1, 0,  0, 0, 0, 0, 0, 1, 0, 0,  0, 0, 0, 0, 0, 1, 0, 0,  0, 0, 0, 0, 0, 0, 1, 0},{
1, 0, 0, 0, 0, 0, 1, 1,  0, 0, 0, 0, 0, 0, 1, 0,  0, 0, 0, 0, 0, 0, 1, 0,  1, 0, 0, 0, 0, 0, 0, 1},{
1, 0, 0, 0, 0, 0, 0, 1,  0, 0, 0, 0, 0, 0, 0, 1,  0, 0, 0, 0, 0, 0, 0, 1,  1, 0, 0, 0, 0, 0, 0, 0}};

int Pbites[] = {
0, 1, 2, 3, 4, 5, 6, 7,
40,  41,  42,  43,  44,  45,  46,  47,
80,  81,  82,  83,  84,  85,  86,  87,
120, 121, 122, 123, 124, 125, 126, 127,
32,  33,  34,  35,  36,  37,  38,  39,
72,  73,  74,  75,  76,  77,  78,  79,
112, 113, 114, 115, 116, 117, 118, 119,
24,  25,  26,  27,  28,  29,  30,  31,
64,  65,  66,  67,  68,  69,  70,  71,
104, 105, 106, 107, 108, 109, 110, 111,
16,  17,  18,  19,  20,  21,  22,  23,
56,  57,  58,  59,  60,  61,  62,  63,
96,  97,  98,  99,  100, 101, 102, 103,
8, 9, 10,  11,  12,  13,  14,  15,
```

```
48,   49,   50,   51,   52,   53,   54,   55,
88,   89,   90,   91,   92,   93,   94,   95};

/*************************************************************************
*                                                                        *
*  construçao da matriz 128x128 equivalente a Mix Columns                *
*                                                                        *
*************************************************************************/
int M[][] = new int[128][128];//cria a matriz nula

    for (k=0;k<4;k++){// laco que copia os blocos de MC(32x32) na diagonal principal
        for (i=0;i<32;i++){//
              for (j=0;j<32;j++){//
                  M[k*32+i][k*32+j]=MC[i][j];
                }// fim do for j
        }// fim do for i
    }// fim do for k

/*************************************************************************
*  construçao da matriz 128x128 equivalente a aplicacao de Mix Columns   *
*  sobre o bloco permutado segundo a permutação Pbites (ShiftRows)       *
*      Esta é a Matrizaes de Avalanche de Primeira                       *
*************************************************************************/

int Nr=7; // numero de rounds

int LIPA [] = new int [Nr];//Limite Inferior do Poder de Avalanche

int LSPA [] = new int [Nr];//Limite Superior do Poder de Avalanche

int Avalanche[][][] = new int[128][128][Nr];//cria a matriz nula
    for (i=0;i<128;i++){//
        for (j=0;j<128;j++){//
            Avalanche[i][Pbites[j]][0]=M[i][j];
        }// fim do for j
    }// fim do for i

// exibição da matriz de Avalanche de Primeira
System.out.println("Matriz de Avalanche de Primeira      ");
```

```
LIPA [0]= 128;//Atribui valor máximo a LIPA[0]
LSPA [0]= 0;//Atribui valor mínimo a LSPA[0]
    for (i=0;i<128;i++){// laco que gera impressao da matriz 128x128
       temp = "";
       c=0;// acumulador de peso hamming da linha i
        for (j=0;j<128;j++){//
                if (Avalanche[i][j][0]==0){
                  temp=temp+".";
                }
                  else{
                   temp=temp+"1";
                   c=c+1;//soma 1 ao peso hamming da linha
                  }
          }// fim do for j
           if (c < LIPA[0]) {
               LIPA[0]=c;//LIPA[0] recebe o menor peso hamming de todas as linhas da matriz
           }
           if (c > LSPA[0]) {
               LSPA[0]=c;//LSPA[0] recebe o maior peso hamming de todas as linhas da matriz
           }
       System.out.println(temp); // exibe a linha i de Avalanche
     }// fim do for i

 temp="Limite Inferior do Poder de Avalanche de Primeira       = "+LIPA[0];
 System.out.println(temp); // exibe LIPA[0]
 temp="Limite Superior do Poder de Avalanche de Primeira       = "+LSPA[0];
 System.out.println(temp); // exibe LSPA[0]

/*************************************************************************
*        Construção das Matrizes de Avalanche de       Superior          *
*                          e  cálculo dos respectivos                     *
*               Limites Inferiores de Poder de Avalanche                *
*************************************************************************/
for (int ordem_menos_um=1;ordem_menos_um < Nr;ordem_menos_um ++){
// Montagem da matriz de ordem  ordem_menos_um + 1 (ordem 2, 3, ...Nr)
for (i=0;i<128;i++){// perccorro a linha i da matriz de avalanche de ordem 1
   for (j=0;j<128;j++){//
            if (Avalanche[i][j][0] == 1){// se na matriz de avalanche de primeira 0rdem o j-esimo bite da entrada
afeta o i-esimo bite da saida
```

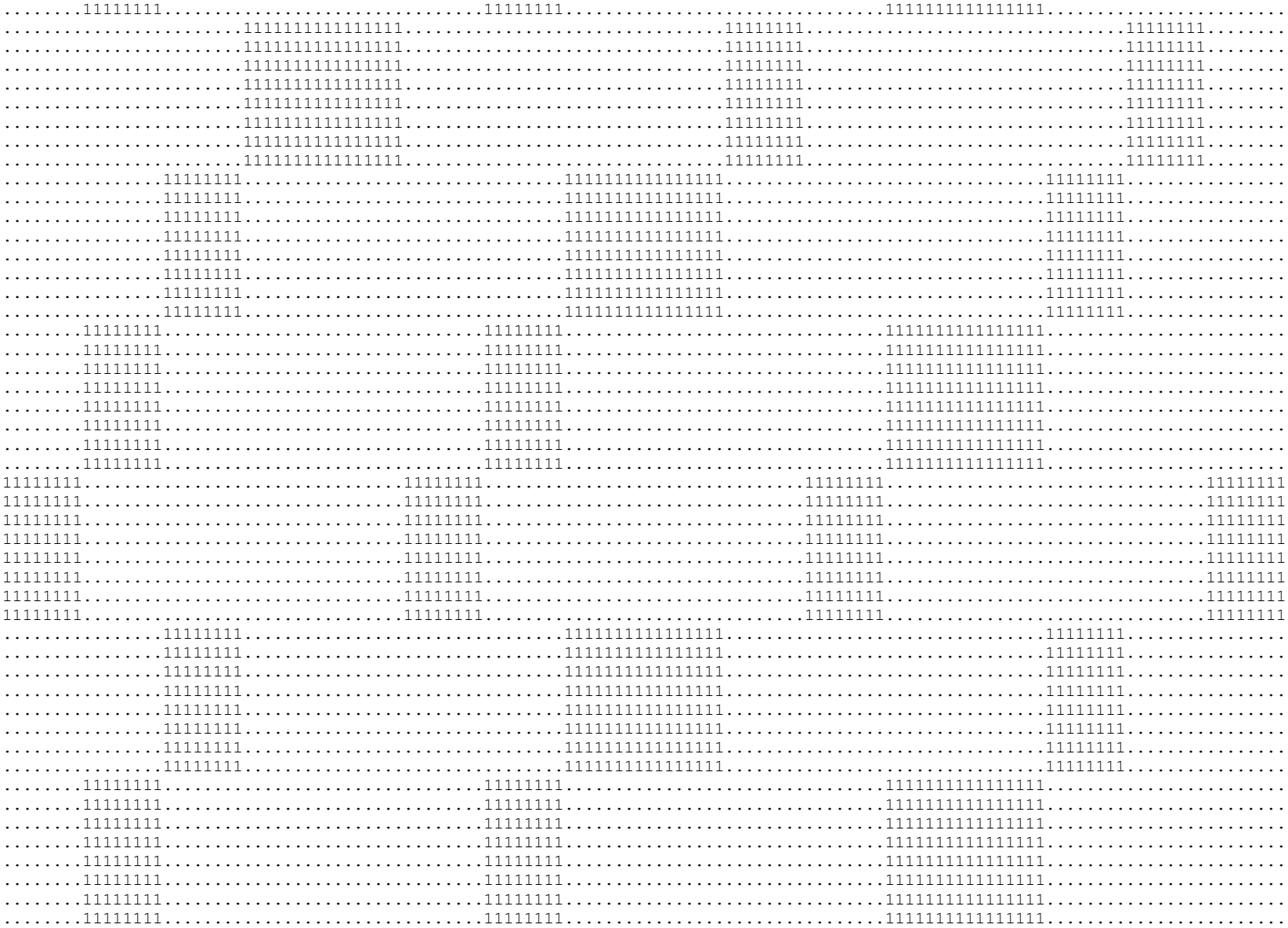
```
                    for (c=0; c<128;c++){//percorre a linha j da matriz de avalanche anterior
                    // a ideia é: se o j-esimo bite da entrada interfere no i-esimo bite da saída na matriz atual,
                    // todos os bites que interferiram no j-ésimo bite da saida na matriz anterior interferem
                    // no i-ésimo bite da saida atual por recorrencia
                            if (Avalanche[j][c][ordem_menos_um-1]==1) Avalanche[i][c][ordem_menos_um]=1;// seta na
linha i da matriz atual todos os
                            // bites que estão setados na linha j da matriz anterior
                    }//fim do for c
                }// fim do if
        }// fim do for j
    }// fim do for i
/*****************************************************************************
*  Exibicao das Matrizes de Avalanche de Ordem Superior a Um                 *
*****************************************************************************/
temp="Matriz de Avalanche de Ordem "+(ordem_menos_um+1);
System.out.println(temp);
LIPA [ordem_menos_um]= 128;//Atribui valor máximo a LIPA[ordem_menos_um]
LSPA [ordem_menos_um]= 0;//Atribui valor mínimo a LSPA[ordem_menos_um]
    for (i=0;i<128;i++){// laco que gera impressao da matriz 128x128
       temp = "";
       c=0;// acumulador de peso hamming da linha i
        for (j=0;j<128;j++){//
                if (Avalanche[i][j][ordem_menos_um]==0){
                  temp=temp+".";
                }
                  else{
                   temp=temp+"1";
                   c=c+1;//soma 1 ao peso hamming da linha
                  }
          }// fim do for j
            if (c<LIPA[ordem_menos_um]) {
               LIPA[ordem_menos_um]=c;//LIPA[ordem_menos_um] recebe o menor peso hamming de todas as linhas da
matriz
            }
            if (c>LSPA[ordem_menos_um]) {
               LSPA[ordem_menos_um]=c;//LSPA[ordem_menos_um] recebe o menor peso hamming de todas as linhas da
matriz
            }
       System.out.println(temp); // exibe a linha i de Avalanche
```

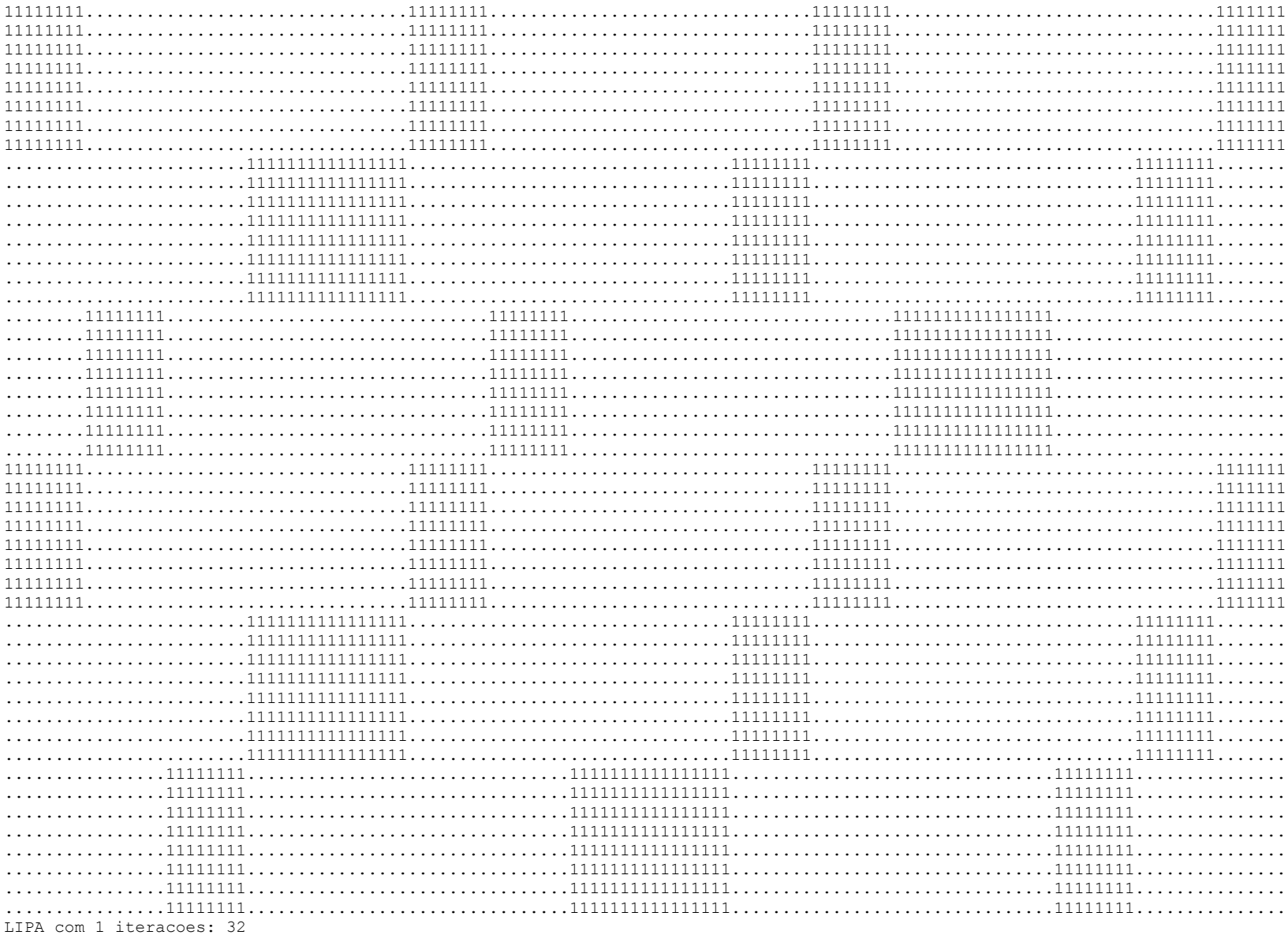
```
     }// fim do for i
        temp="Limite Inferior do Poder de Avalanche Ordem "+(ordem_menos_um+1)+" = "+LIPA[ordem_menos_um];
        System.out.println(temp); // exibe LIPA[ordem_menos_um]
        temp="Limite Superior do Poder de Avalanche Ordem "+(ordem_menos_um+1)+" = "+LSPA[ordem_menos_um];
        System.out.println(temp); // exibe LSPA[ordem_menos_um]
        }// fim de for ordem_menos_um
} //fim do main
}
```

12.4.9 MATRIZ DE DEPENDÊNCIAS ESTATÍSTICAS (*w=2000, 1 iteração*)

MDE (binária) da saída em relação a TEXTO CLARO FIPS com 1 iteração :

```
11111111.............................11111111.............................11111111.............................11111111
11111111.............................11111111.............................11111111.............................11111111
11111111.............................11111111.............................11111111.............................11111111
11111111.............................11111111.............................11111111.............................11111111
11111111.............................11111111.............................11111111.............................11111111
11111111.............................11111111.............................11111111.............................11111111
11111111.............................11111111.............................11111111.............................11111111
11111111.............................11111111.............................11111111.............................11111111
.......................111111111111111.............................11111111.............................11111111........
.......................111111111111111.............................11111111.............................11111111........
.......................111111111111111.............................11111111.............................11111111........
.......................111111111111111.............................11111111.............................11111111........
.......................111111111111111.............................11111111.............................11111111........
.......................111111111111111.............................11111111.............................11111111........
.......................111111111111111.............................11111111.............................11111111........
.......................111111111111111.............................11111111.............................11111111........
.............11111111.............................111111111111111.............................11111111.............
.............11111111.............................111111111111111.............................11111111.............
.............11111111.............................111111111111111.............................11111111.............
.............11111111.............................111111111111111.............................11111111.............
.............11111111.............................111111111111111.............................11111111.............
.............11111111.............................111111111111111.............................11111111.............
.............11111111.............................111111111111111.............................11111111.............
.............11111111.............................111111111111111.............................11111111.............
.......11111111.............................11111111.............................111111111111111.....................
.......11111111.............................11111111.............................111111111111111.....................
.......11111111.............................11111111.............................111111111111111.....................
.......11111111.............................11111111.............................111111111111111.....................
.......11111111.............................11111111.............................111111111111111.....................
.......11111111.............................11111111.............................111111111111111.....................
.......11111111.............................11111111.............................111111111111111.....................
```




LIPA com 1 iteracoes: 32

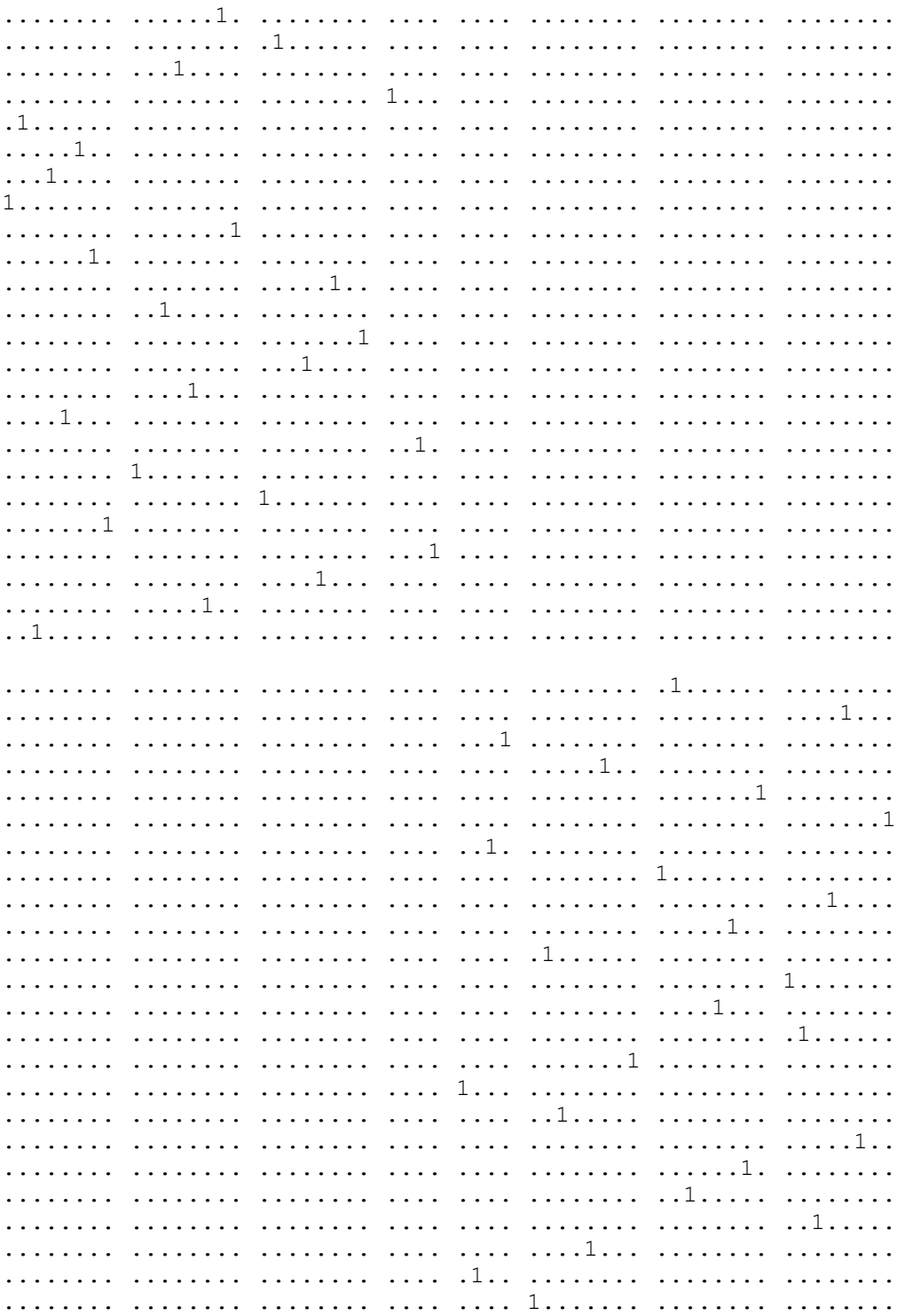
LSPA com 1 iteracoes: 32
As MDE para mais de 1 iteração são completas.

12.5 APÊNDICE 5: RESULTADOS DA ANÁLISE DO DES COM AS MATRIZES DE AVALANCHE

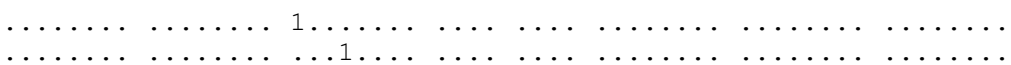
12.5.1 RELAÇÕES DE DEPENDÊNCIA DAS SUB-CHAVES EM RELAÇÃO AO BLOCO $C_0|D_0$

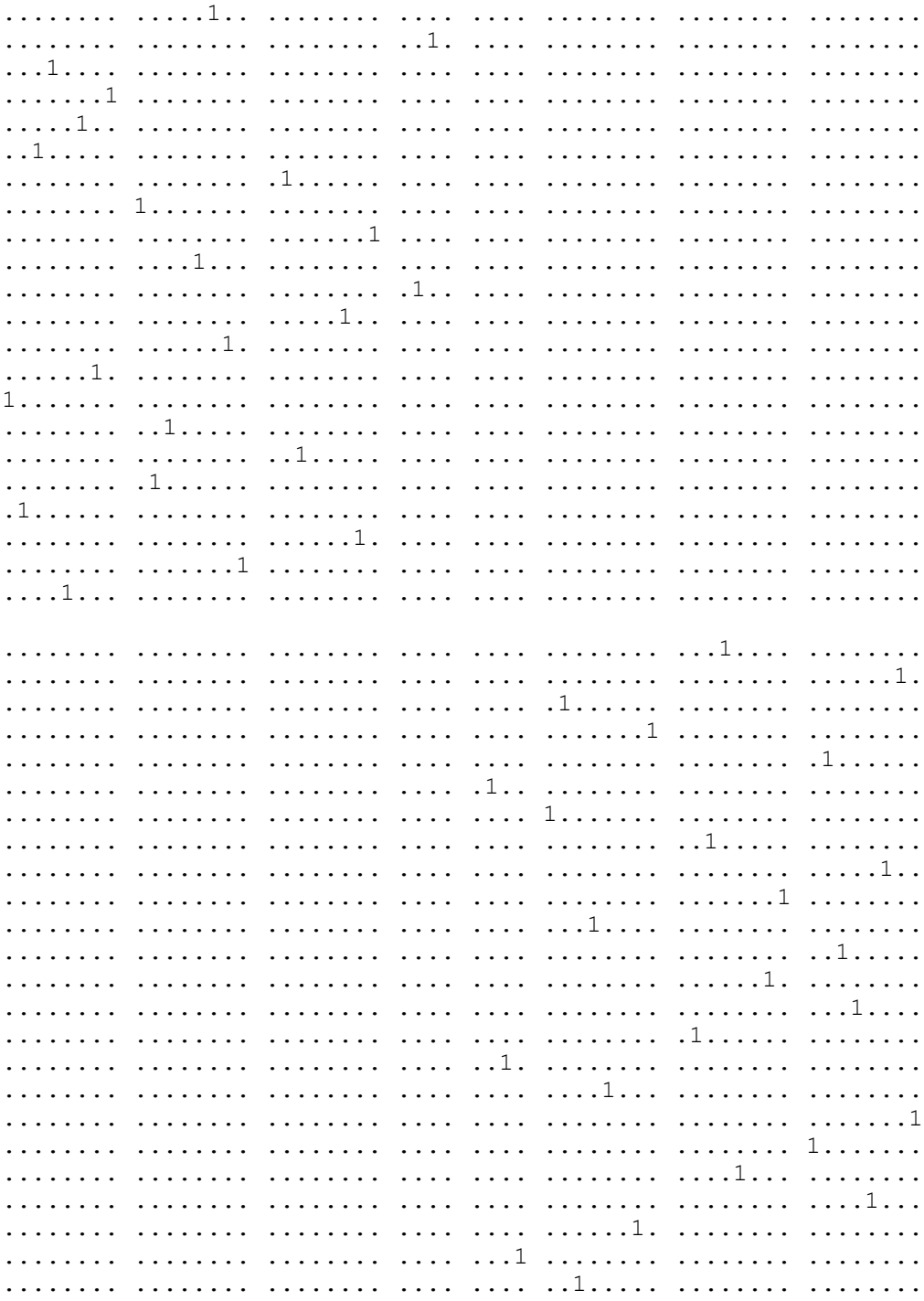
```
DESCONSIDERANDO A APLICACAO DE PC1, C[0]D[0]=
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55

Bites de K(56) na sub-chave  K 1 :
14 17 11 24 1 5   3 0 15 6 21 10   23 19 12 4 26 8   16 7 27 20 13 2
41 52 31 37 47 55   30 40 51 45 33 48   44 49 39 28 34 53   46 42 50 36 29 32
Bites de K(56) na sub-chave  K 2 :
15 18 12 25 2 6   4 1 16 7 22 11   24 20 13 5 27 9   17 8 0 21 14 3
42 53 32 38 48 28   31 41 52 46 34 49   45 50 40 29 35 54   47 43 51 37 30 33
Bites de K(56) na sub-chave  K 3 :
16 19 13 26 3 7   5 2 17 8 23 12   25 21 14 6 0 10   18 9 1 22 15 4
43 54 33 39 49 29   32 42 53 47 35 50   46 51 41 30 36 55   48 44 52 38 31 34
Bites de K(56) na sub-chave  K 4 :
17 20 14 27 4 8   6 3 18 9 24 13   26 22 15 7 1 11   19 10 2 23 16 5
44 55 34 40 50 30   33 43 54 48 36 51   47 52 42 31 37 28   49 45 53 39 32 35
Bites de K(56) na sub-chave  K 5 :
18 21 15 0 5 9   7 4 19 10 25 14   27 23 16 8 2 12   20 11 3 24 17 6
45 28 35 41 51 31   34 44 55 49 37 52   48 53 43 32 38 29   50 46 54 40 33 36
Bites de K(56) na sub-chave  K 6 :
19 22 16 1 6 10   8 5 20 11 26 15   0 24 17 9 3 13   21 12 4 25 18 7
46 29 36 42 52 32   35 45 28 50 38 53   49 54 44 33 39 30   51 47 55 41 34 37
Bites de K(56) na sub-chave  K 7 :
20 23 17 2 7 11   9 6 21 12 27 16   1 25 18 10 4 14   22 13 5 26 19 8
47 30 37 43 53 33   36 46 29 51 39 54   50 55 45 34 40 31   52 48 28 42 35 38
Bites de K(56) na sub-chave  K 8 :
21 24 18 3 8 12   10 7 22 13 0 17   2 26 19 11 5 15   23 14 6 27 20 9
48 31 38 44 54 34   37 47 30 52 40 55   51 28 46 35 41 32   53 49 29 43 36 39
Bites de K(56) na sub-chave  K 9 :
22 25 19 4 9 13   11 8 23 14 1 18   3 27 20 12 6 16   24 15 7 0 21 10
49 32 39 45 55 35   38 48 31 53 41 28   52 29 47 36 42 33   54 50 30 44 37 40
Bites de K(56) na sub-chave  K 10 :
23 26 20 5 10 14   12 9 24 15 2 19   4 0 21 13 7 17   25 16 8 1 22 11
50 33 40 46 28 36   39 49 32 54 42 29   53 30 48 37 43 34   55 51 31 45 38 41
Bites de K(56) na sub-chave  K 11 :
24 27 21 6 11 15   13 10 25 16 3 20   5 1 22 14 8 18   26 17 9 2 23 12
51 34 41 47 29 37   40 50 33 55 43 30   54 31 49 38 44 35   28 52 32 46 39 42
Bites de K(56) na sub-chave  K 12 :
25 0 22 7 12 16   14 11 26 17 4 21   6 2 23 15 9 19   27 18 10 3 24 13
52 35 42 48 30 38   41 51 34 28 44 31   55 32 50 39 45 36   29 53 33 47 40 43
Bites de K(56) na sub-chave  K 13 :
26 1 23 8 13 17   15 12 27 18 5 22   7 3 24 16 10 20   0 19 11 4 25 14
53 36 43 49 31 39   42 52 35 29 45 32   28 33 51 40 46 37   30 54 34 48 41 44
Bites de K(56) na sub-chave  K 14 :
27 2 24 9 14 18   16 13 0 19 6 23   8 4 25 17 11 21   1 20 12 5 26 15
54 37 44 50 32 40   43 53 36 30 46 33   29 34 52 41 47 38   31 55 35 49 42 45
Bites de K(56) na sub-chave  K 15 :
0 3 25 10 15 19   17 14 1 20 7 24   9 5 26 18 12 22   2 21 13 6 27 16
55 38 45 51 33 41   44 54 37 31 47 34   30 35 53 42 48 39   32 28 36 50 43 46
Bites de K(56) na sub-chave  K 16 :
1 4 26 11 16 20   18 15 2 21 8 25   10 6 27 19 13 23   3 22 14 7 0 17
28 39 46 52 34 42   45 55 38 32 48 35   31 36 54 43 49 40   33 29 37 51 44 47
```

12.5.1.1 DEPENDÊNCIAS DA SUB-CHAVE K1 EM RELAÇÃO À CHAVE(56 BITES)

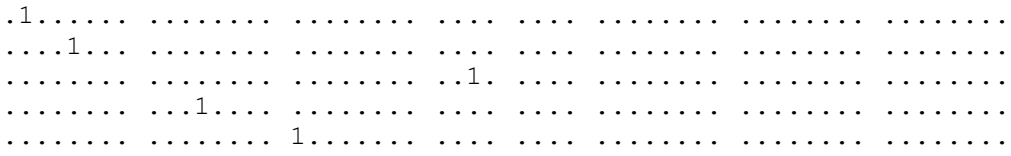


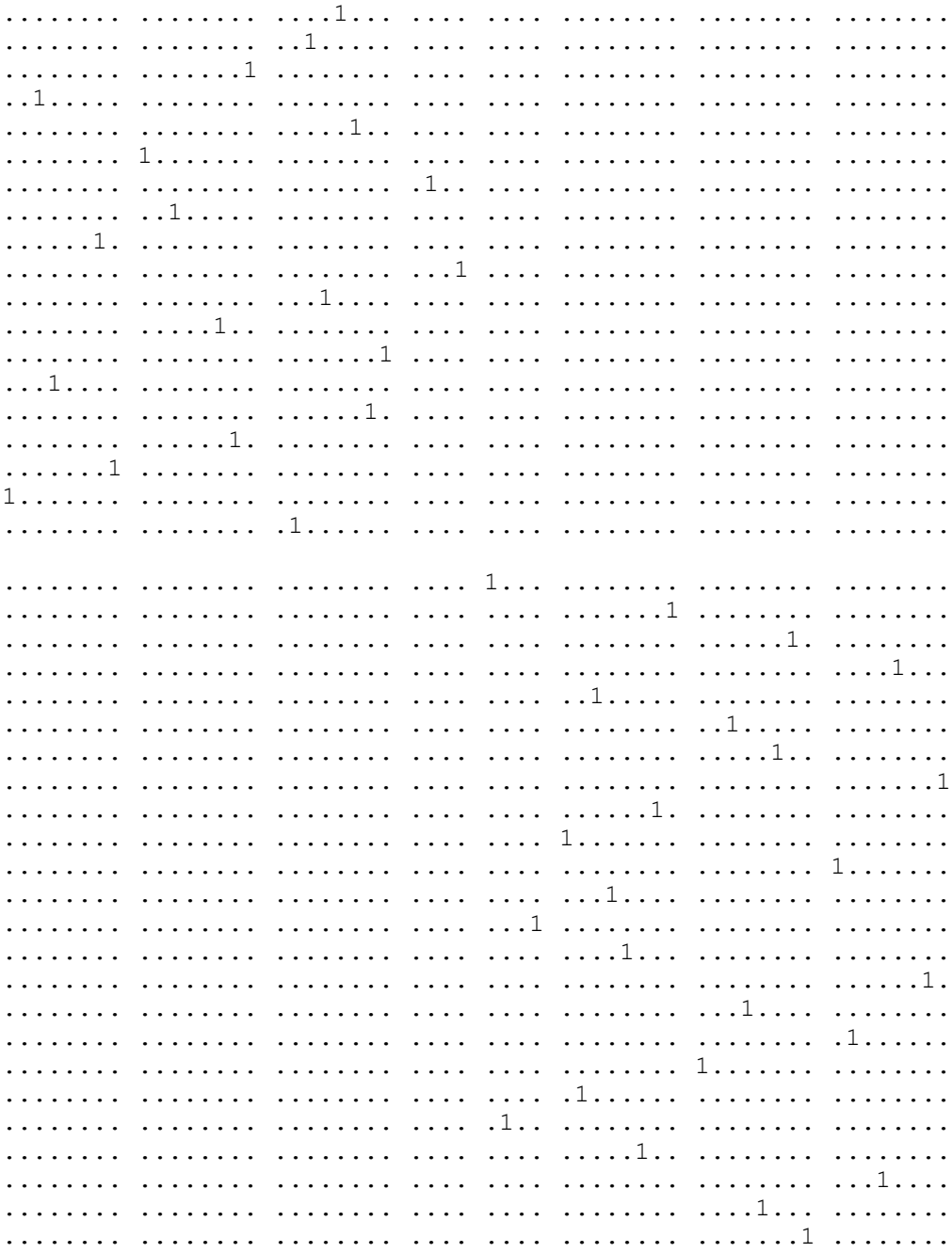
12.5.1.2 DEPENDÊNCIAS DA SUB-CHAVE K3 EM RELAÇÃO À CHAVE(56 BITES)





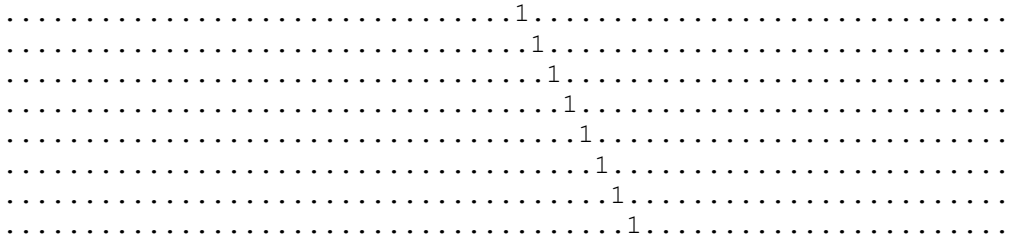
12.5.1.3 DEPENDÊNCIAS DA SUB-CHAVE K16 EM RELAÇÃO À CHAVE(64 BITES)

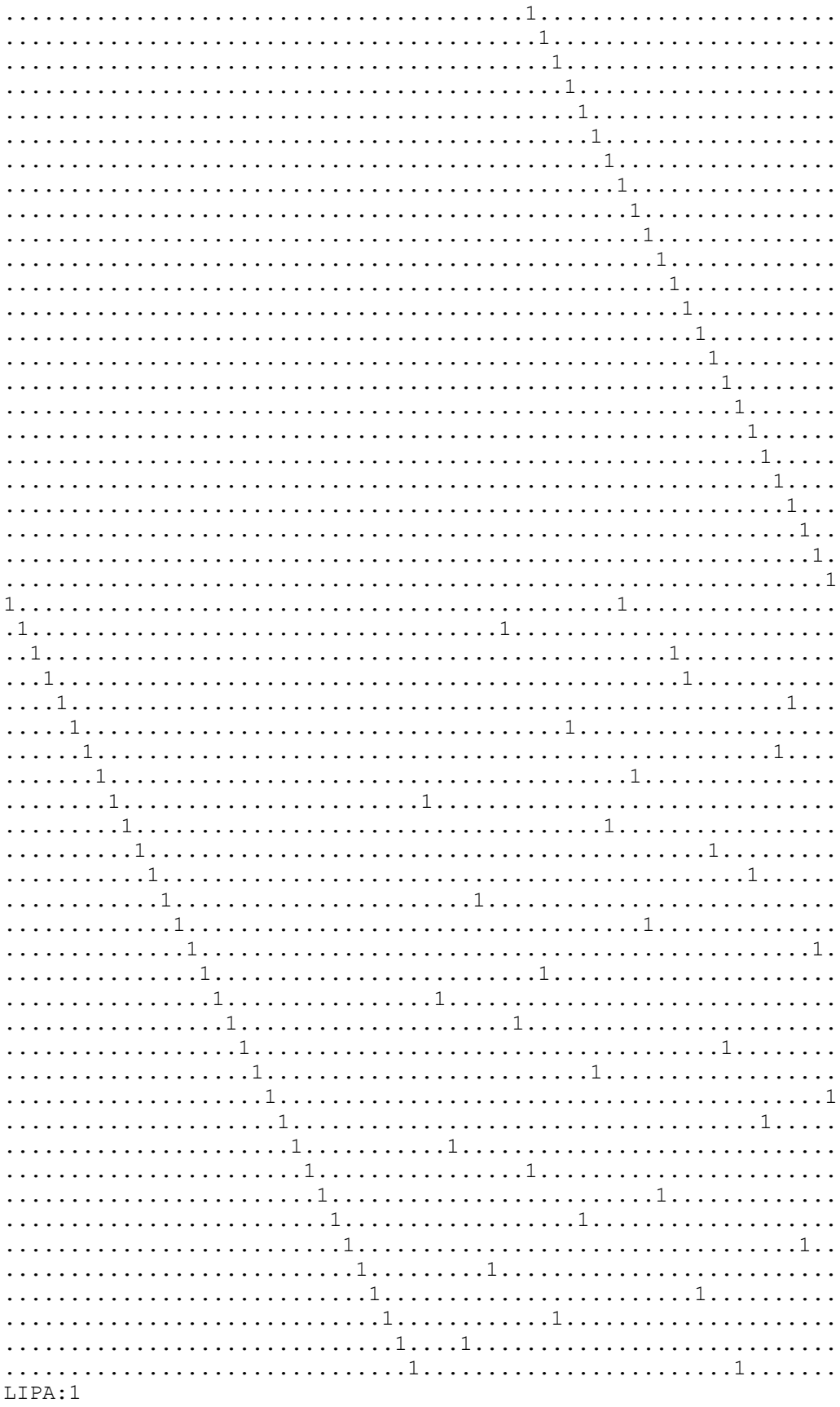




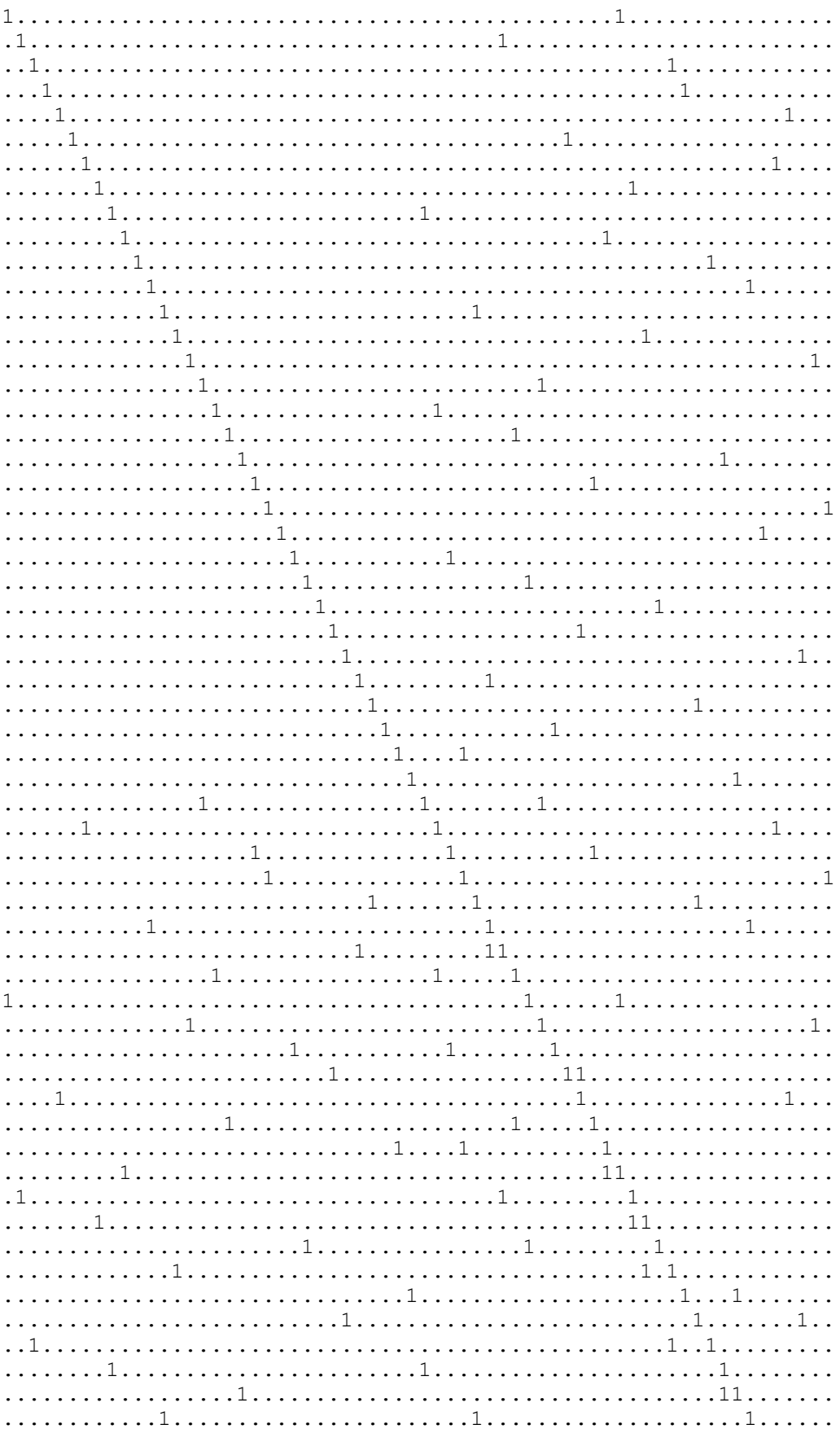
12.5.2 ANÁLISE DA TL DO DES COM AS MATRIZES DE AVALANCHE

12.5.2.1 MATRIZ DE DEPENDÊNCIAS [D_1] DA TL DO DES




LIPA:1
LSPA:2

12.5.2.2 MDR [D_2] (dois passos)



```
.............................1............1.............1.....
.....1.....................................1.............1....
......................1.....................................1.1...
..........1.............................................1......1..
...1.....................................................1.........1.
.........................1.........................1............1
LIPA: 2
LSPA: 3
```

12.5.2.3 MDR [D_{16}] (16 passos)

```
1........1..........1..11.....1....1....1.....11..1............1
.11.......111.............111.......111............1.11..1...1..
.11....1...111.............11.......111.........11.1.1...1......
...1....1.....11..1............11........1..........1..11.....1.
....1.1.........11.1.11......1...11....1..1..1............111...
....11............1.1.11..1...1....1....1..1111............1.1...
....111.............1.11..1...1....1.......1111............111...
.1.....1..111.............111.......111.........1....11..1...1..
...1....1.....11..1............11........1..........1..11.....1.
1........1..........1..11.....1....1....1.....11..1............1
.11....1..1111.............1........111.........11.1..1..1......
..1....1..1111.............1.1.......1............11.1.11..1...1..
.11....1..1.11.............1.1.......1.1..........11.1.11......1..
.1.....1...111.............111.......111.........11...1...1...1..
...1....1.....11..1............11........1..........1..11.....1.
...1....1.....11..1............11........1..........1..11.....1.
....111.........1....11..1...1...11.......111.............111...
....111.........11...1...1...1...1.....1..111.............111...
...1....1.....11..1............11........1..........1..11.....1.
....111.........11.1.1...1.......1.....1...111............111...
1........1..........1..11.....1....1....1.....11..1............1
.....11.........11.1.11......1...11....1..11.1............11....
....111.........11.1..1..1.......11....1...111.............11...
1........1..........1..11.....1....1....1.....11..1............1
1........1..........1..11.....1....1....1.....11..1............1
....1............11.1.11..1...1...11....1..1.11.............1.1...
.11....1..11.1.............11.........11.........11.1..1..1...1..
..1.......1111.............111.......11............1.1.11..1...1..
.11....1..1..1.............111........11.........11.1.11......1..
.....11.........11.1..1..1...1...11....1..1111.............1....
1........1..........1..11.....1....1....1.....11..1............1
...1....1.....11..1............11........1..........1..11.....1.
...1....1.....11..1............11........1..........1..11.....1.
....111.............1.11..1...1...11.......1111............111...
....111.........11.1.1...1.......11....1...111............111...
1........1..........1..11.....1....1....1.....11..1............1
.11....1..1..1.............111.......111.........11.1.11......1..
..1....1..1111.............1.1.......11..........11.1.11..1...1..
..1.......1111.............111.......111..........1.1.11..1...1..
....111.........1....11..1...1...11....1..111.............111...
1........1..........1..11.....1....1....1.....11..1............1
...1....1.....11..1............11........1..........1..11.....1.
....111.........11.1..1..1.......11....1..1111.............11...
....1............11.1.11..1...1...11....1..1111............1.1...
....1.1.........11.1.11......1...11....1..1.11............111...
....111.........11...1...1...1...1.....1..1111............111...
1........1..........1..11.....1....1....1.....11..1............1
```

```
1........1..........1..11.....1....1....1.....11..1...........1
.11.......111.............111.......111.........1..1.11..1...1..
.1.....1..111.............111.......111.........11...11..1...1..
1........1..........1..11.....1....1....1.....11..1...........1
.1.....1...111.............111.......111.........11.1.1...1...1..
...1....1.....11..1.............11........1..........1..11.....1.
.11....1..11.1.............11.........11.........11.1.11..1...1..
.11....1...111.............11.......111.........11.1.11..1......
...1....1.....11..1.............11........1..........1..11.....1.
...1....1.....11..1.............11........1..........1..11.....1.
.11....1..1.11.............1.1.......1.1.........11.1.11..1...1..
.....11.........11.1..1..1...1...11....1..1111.............11....
....11...........1.1.11..1...1....1....1..1111.............111...
.....11.........11.1.11......1...11....1..11.1.............111...
.11....1..1111.............1........111.........11.1..1..1...1..
...1....1.....11..1.............11........1..........1..11.....1.
1........1..........1..11.....1....1....1.....11..1...........1
LIPA: 12
LSPA: 17
```

12.6 APÊNDICE 6: ENTRADAS INVARIANTES NAS CAIXAS DE SUBSTITUIÇÃO DO DES

Este apêndice é baseado em (LAMBERT, 2003d).

O principal objetivo das caixas de substituição é inserir não linearidade na função a fim de confundir o criptoanalista. O argumento (entrada) de 48 bites é dividido em 8 cadeias de 6 bites [$b_1, b_2...b_6$], sendo cada uma operada por uma caixa de substituição - *S1, S2,...*ou *S8*. Cada caixa é composta por 64 células dispostas em 4 linhas numeradas de 0 a 3 e em 16 colunas numeradas de 0 a 15. O conteúdo de cada célula é um número decimal de 0 a 15. As caixas foram montadas de tal forma que em cada linha tenhamos presentes todos os números decimais de 0 a 15, desordenados, como pode ser visto na caixa *S1* a seguir (SCHNEIER, 1996, p. 276) (MENEZES, 1996, p. 260)(XEXEO, 1983)(LIMA, 1999):

TAB 12.6.1: Tabela de Substituição S1 do DES

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
01	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
10	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
11	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Caixa de Substituição *S1* : na primeira coluna temos $\boldsymbol{b_1b_6}$, indicando a linha, e na primeira linha temos $\boldsymbol{b_2b_3b_4b_5}$, indicando a coluna.

A representação binária do conteúdo da célula localizada na linha $\boldsymbol{b_1b_6}$ (base 2) e coluna $\boldsymbol{b_2b_3b_4b_5}$ (base 2) é o resultado da operação *S1*([$b_1, b_2...b_6$]). Assim, *S1*(110010) = 1100, pois na linha 10_2 (2_{10}) e coluna 1001_2 (9_{10}) temos o número $12_{10} = 1100_2$. Desta forma, cada cadeia de seis bites retorna uma cadeia de 4 bites, e o argumento de 48 bites retorna, consequentemente, um vetor de 32 bites (quatro para cada seis).

Para cada saída de uma caixa ***Sn***, existem quatro possíveis entradas de 6 bites capazes de gerá-la, o que faz com que a função *S* ***não seja*** bijetora. Por exemplo, para a caixa *S1*, a saída $1100_2=12_{10}$ pode ser conseqüência de qualquer uma das quatro seguintes entradas: 010110 , 010101, 110010 ou 10001, pois *S1*(110010)=*S1*(010101)=*S1*(110010)=*S1*(100011)=1100. Entretanto, para cada caixa de substituição existem casos em que o valor da saída pode determinar um ou mais bites da entrada.

Considerações válidas para as oito caixas de substituição:

a) Nas colunas 0 a 7, o bite ***b2*** é sempre igual a 0, e nas colunas 8 a 15, ***b2*** é sempre 1
b) Nas colunas 0, 1, 2, 3, 8, 9, 10 e 11, o bite ***b3*** é sempre igual a 0, e nas demais colunas ***b3*** é sempre 1;
c) Nas colunas 0, 1, 4, 5, 8, 9, 12 e 13, o bite ***b4*** é sempre igual a 0, e nas demais colunas ***b4*** é sempre 1;
d) Nas colunas pares, o bite ***b5*** é sempre igual a 0, e nas colunas impares ***b3*** é sempre 1;

Em conseqüência, no caso particular da caixa *S1*, podemos verificar 10 possíveis saídas da caixa *S1* que apresentam entradas invariantes, como pode ser visto nas ilustrações a seguir:

	0000	***0***001	***0***010	***0***011	***0***100	***0***101	***0***110	***0***111	1000	1001	1010	1011	1100	1101	1110	1111
00	14	**4**	13	**1**	**2**	15	11	8	**3**	**10**	6	12	**5**	9	0	7
01	0	15	7	**4**	14	**2**	13	**1**	**10**	6	12	11	9	**5**	**3**	8
10	**4**	**1**	14	8	13	6	**2**	11	15	12	9	7	**3**	**10**	**5**	0
11	15	12	8	**2**	**4**	9	**1**	7	**5**	11	**3**	14	**10**	0	6	13

Saída = 1, 2 ou **4** ⇒ Como todas as ocorrências do valor 1(,2 e 4) encontram-se na região hachurada, o bite ***b_2=0*** é uma entrada invariante e fica determinado pela saída = 1(,2 ou 4).
Saída = 3, 5 ou **10** ⇒ Como todas as ocorrências do valor 3 (,5 e 10) encontram-se na região não hachurada, o bite ***b_2=1*** é uma entrada invariante e fica determinado pela saída = 3(,5 ou 10).

FIG. 12.6.1 Regiões da caixa *S1* que determinam ***b_2.***

	0***0***00	0***0***01	0***0***10	0***0***11	0100	0101	0110	0111	1***0***00	1***0***01	1***0***10	1***0***11	1100	1101	1110	1111
00	14	4	13	1	2	15	11	8	3	10	6	**12**	5	9	0	7
01	0	15	7	4	14	2	13	1	10	6	**12**	11	9	5	3	8
10	4	1	14	8	13	6	2	11	15	**12**	9	7	3	10	5	0
11	15	**12**	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Saída = 12 ⇒ Como todas as ocorrências do valor 12 encontram-se na região hachurada, o bite ***b_3=0*** é uma entrada invariante e fica determinado pela saída = 12.

FIG. 12.6.2 Regiões da caixa *S1* que determinam ***b_3.***

	00**0**0	00**0**1	0010	0011	01**0**0	01**0**1	0110	0111	10**0**0	10**0**1	1010	1011	11**0**0	11**0**1	1110	1111
00	14	4	13	1	2	15	11	**8**	3	**10**	6	12	5	9	0	**7**
01	0	15	**7**	4	14	2	13	1	**10**	6	12	11	9	5	3	**8**
10	4	1	14	**8**	13	6	2	11	15	12	9	**7**	3	**10**	5	0
11	15	12	**8**	2	4	9	1	**7**	5	11	3	14	**10**	0	6	13

Saída = 7 ou **8** ⇒ Como todas as ocorrências do valor 7(8) encontram-se na região não hachurada, o bite $b_4=1$ é uma entrada invariante e fica determinado pela saída = 7(8).

Saída = 10 ⇒ Como todas as ocorrências do valor 10 encontram-se na região hachurada, o bite $b_4=0$ é uma entrada invariante e fica determinado pela saída = 10.

FIG. 12.6.3 Regiões da caixa *S1* que determinam b_4.

	000**0**	0001	001**0**	0011	010**0**	0101	011**0**	0111	100**0**	1001	101**0**	1011	110**0**	1101	111**0**	1111
00	14	4	13	1	2	15	11	8	**3**	10	6	12	5	9	0	7
01	0	15	7	4	14	2	13	1	10	6	12	11	9	5	**3**	8
10	4	1	14	8	13	6	2	11	15	12	9	7	**3**	10	5	0
11	15	12	8	2	4	9	1	7	5	11	**3**	14	10	0	6	13

Saída= 3 ⇒ Como todas as ocorrências do valor 3 encontram-se na região hachurada, o bite $b_5=0$ é uma entrada invariante e fica determinado pela saída = 3.

FIG. 12.6.4 Regiões da caixa *S1* que determinam b_5.

Analisando as dezesseis possíveis saídas para cada caixa de substituição, podemos montar a TAB. 12.6.2, que apresenta os bites que ficam determinados caso ocorram saídas favoráveis (saídas que apresentam entradas invariantes).

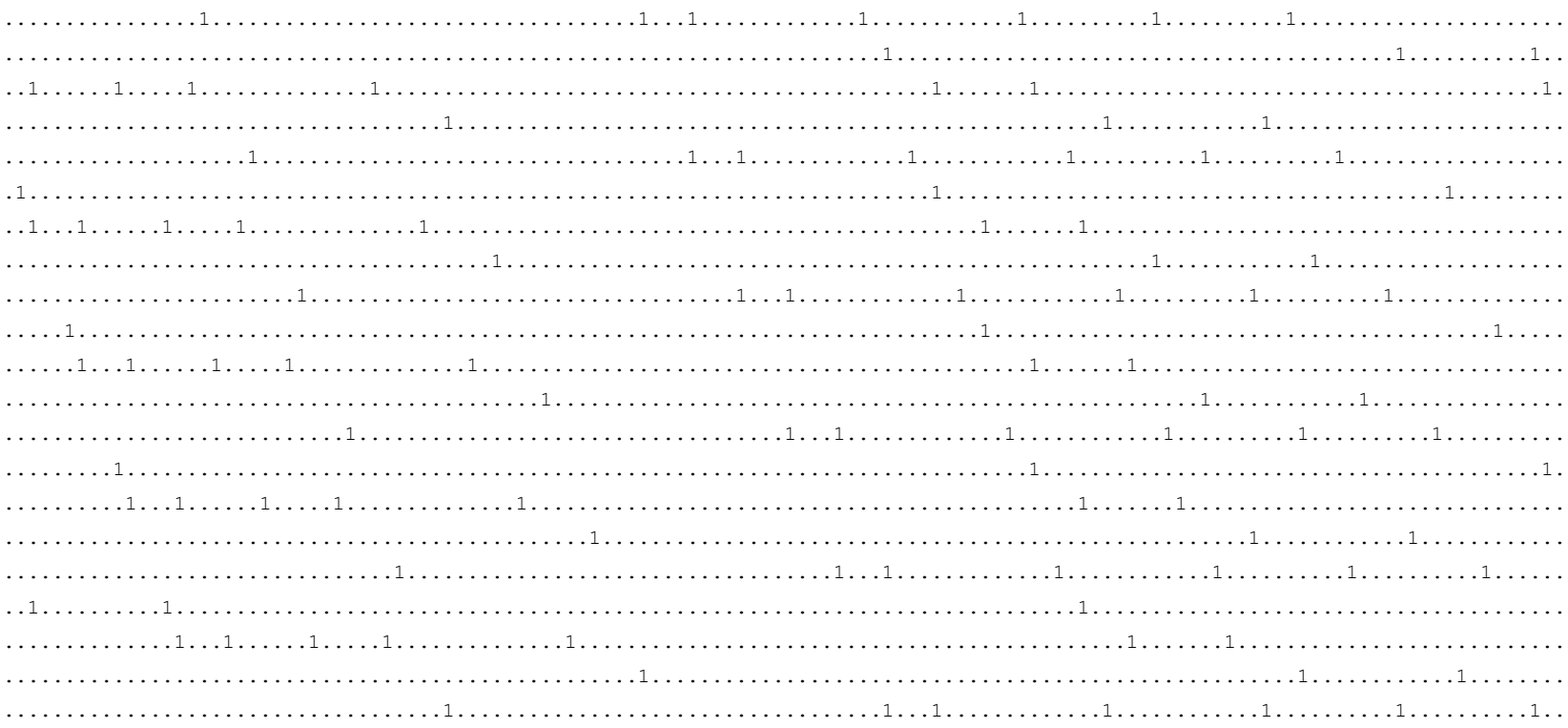
TAB. 12.6.2: Bites da entrada de 48 bites que ficam determinados (entradas invariantes) em cada caixa, para cada possível saída favorável (XEXEO, 1983, p. 6).

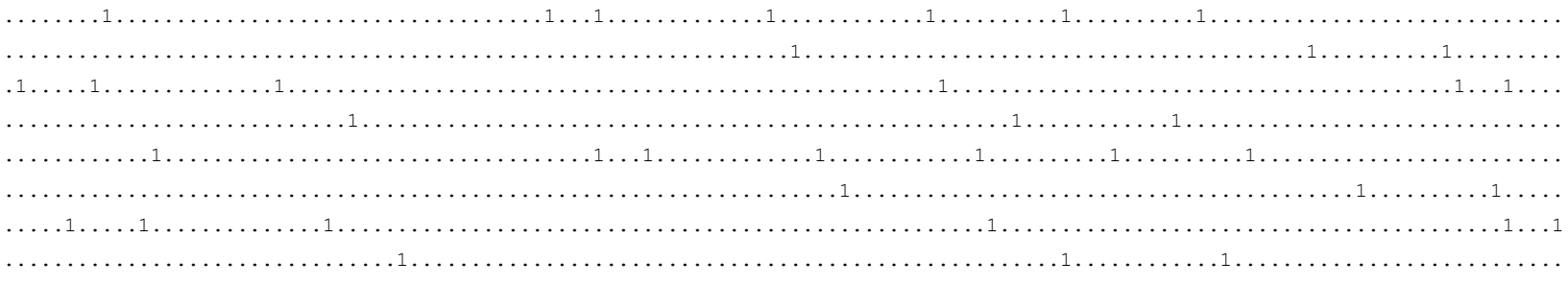
Saída	S1	S2	S3	S4	S5	S6	S7	S8
0		b10=0	b14=0	b20=0	b26=1	b32=1,b34=0		b44=1
1	b2=0			b22=0	b26=0			b44=0
2	b2=0		b14=0, 17=1	b20=1	b26=0	b32=0		
3	b2=1,b5=0				b26=1			
4	b2=0	b8=0		b20=1		b33=0	b38=0,b40=0	b44=0
5	b2=1	b8=1			b26=1		b38=0	b44=1
6			b14=0	b20=0				
7	b4=1	b9=0	b17=1		b26=0		b40=1	
8	b4=1					b33=1	b40=1	b46=1
9		b8=1	b14=0		b26=1,b29=1	b32=0,b35=0		
10	b2=1, b4=0							
11			b14=1,b16=0		b26=0	b32=1	b38=0,b39=0	
12	b3=0	b8=1	b14=1		b27=0	b32=0		
13			b15=0	b20=0	b27=1	b32=1	b38=0	
14			b16=1	b22=1				
15	b4=0				b26=1,b27=0	b32=0		b46=0

12.7 APÊNDICE 7: RESULTADOS DA ANÁLISE DO SERPENT COM AS MATRIZES DE AVALANCHE

12.7.1 ANÁLISE DA TL DO SERPENT COM AS MATRIZES DE AVALANCHE

12.7.1.1 MATRIZ DE DEPENDÊNCIAS [D_1] DA TL DO SERPENT

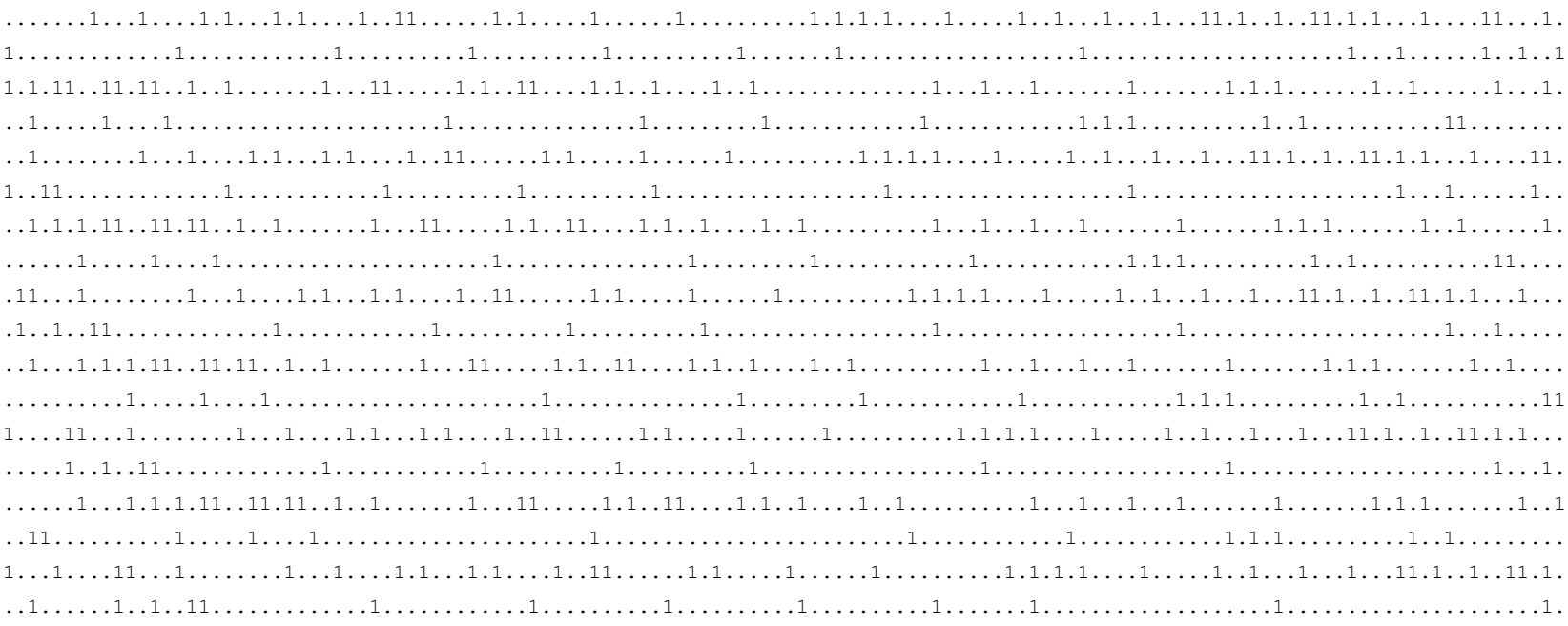


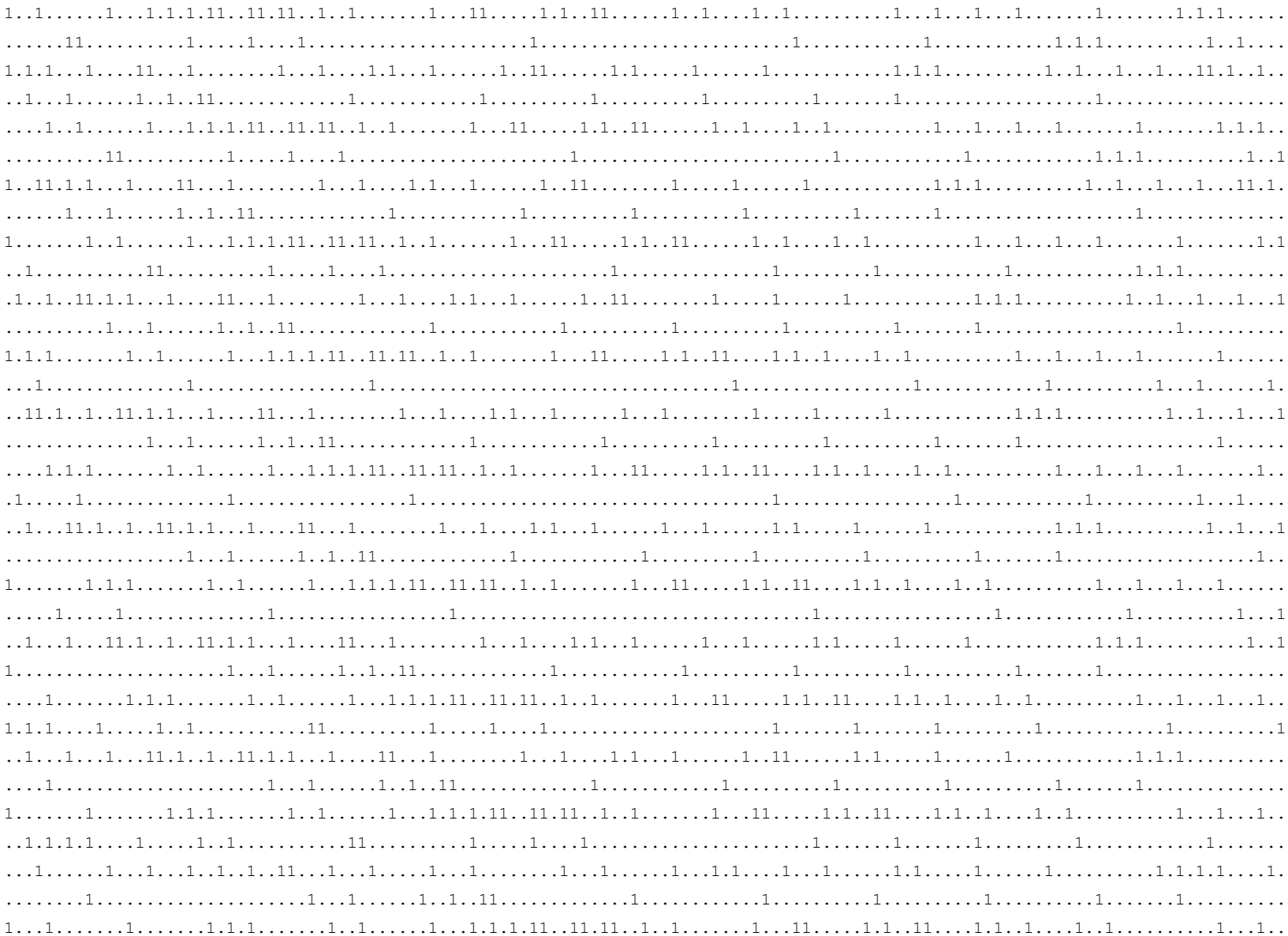


LIPA=2

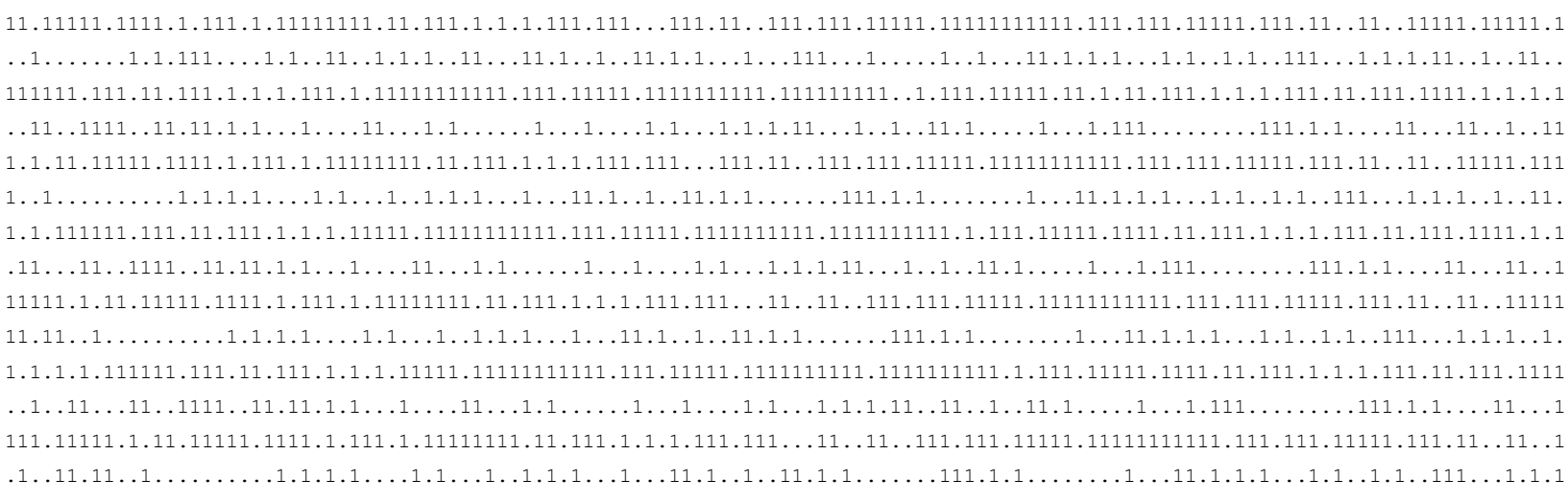
LSPA=7

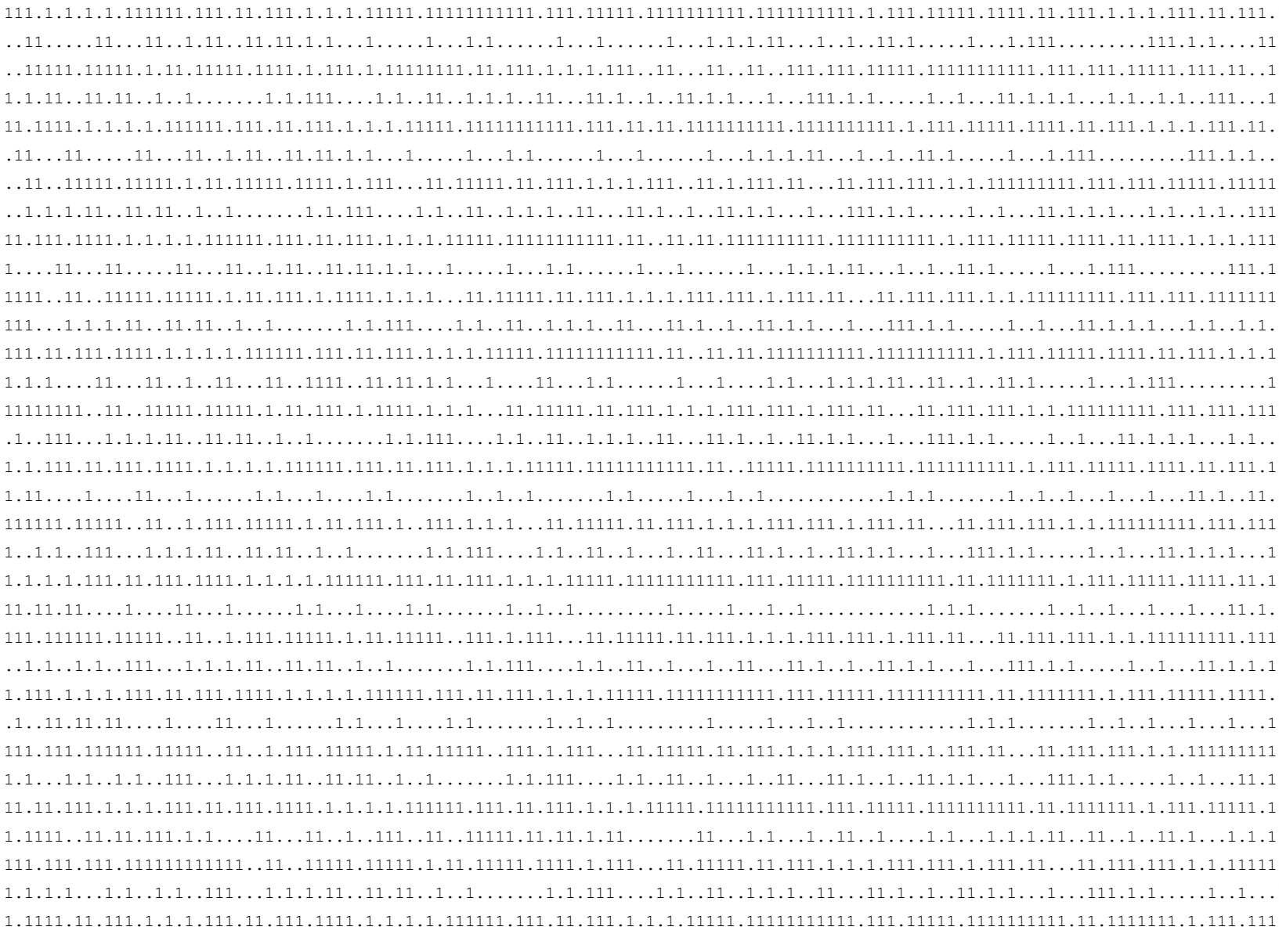
12.7.1.2 MATRIZ DE DEPENDÊNCIAS [D_2] DA TL DO SERPENT (DOIS PASSOS)

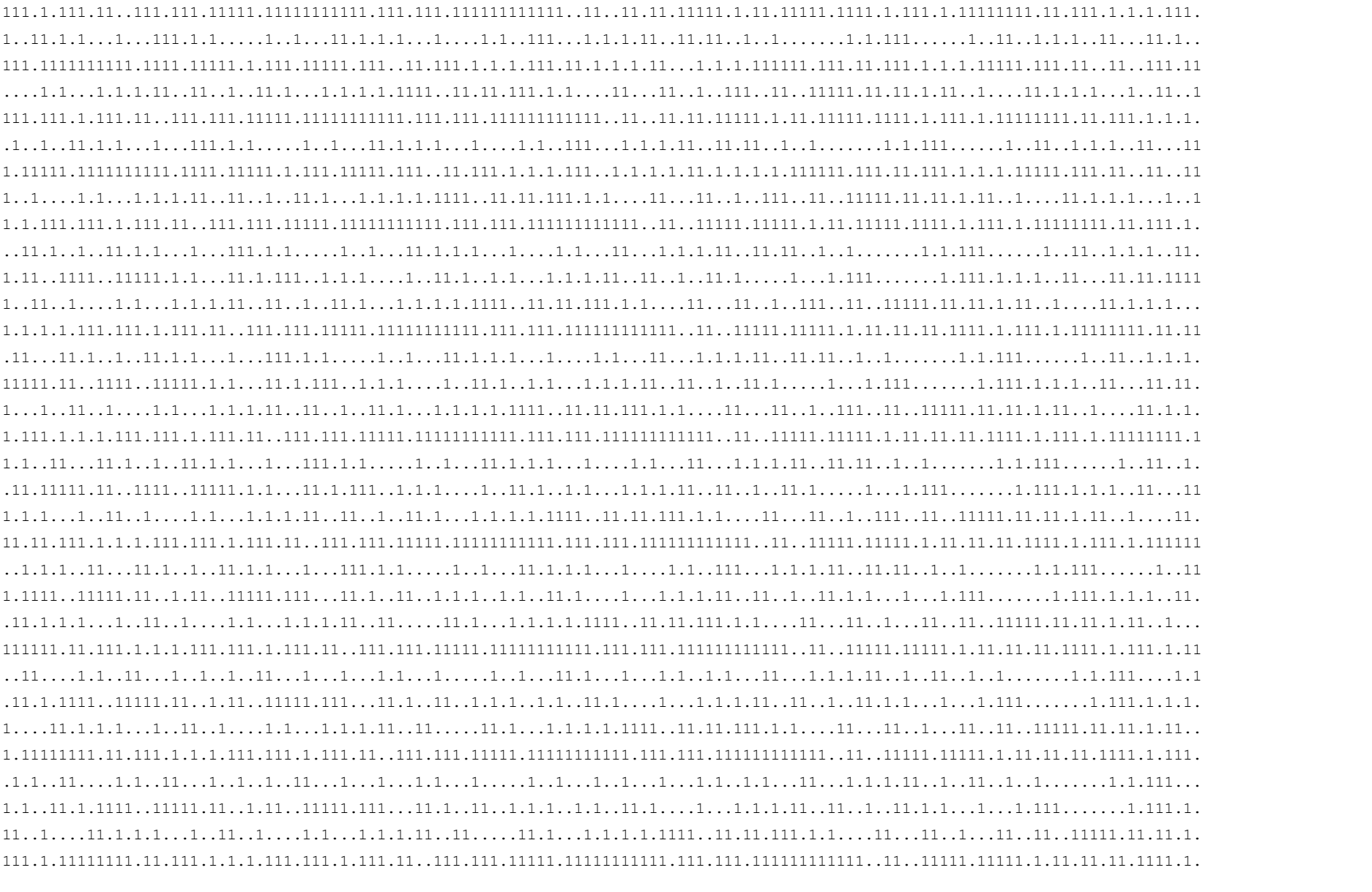


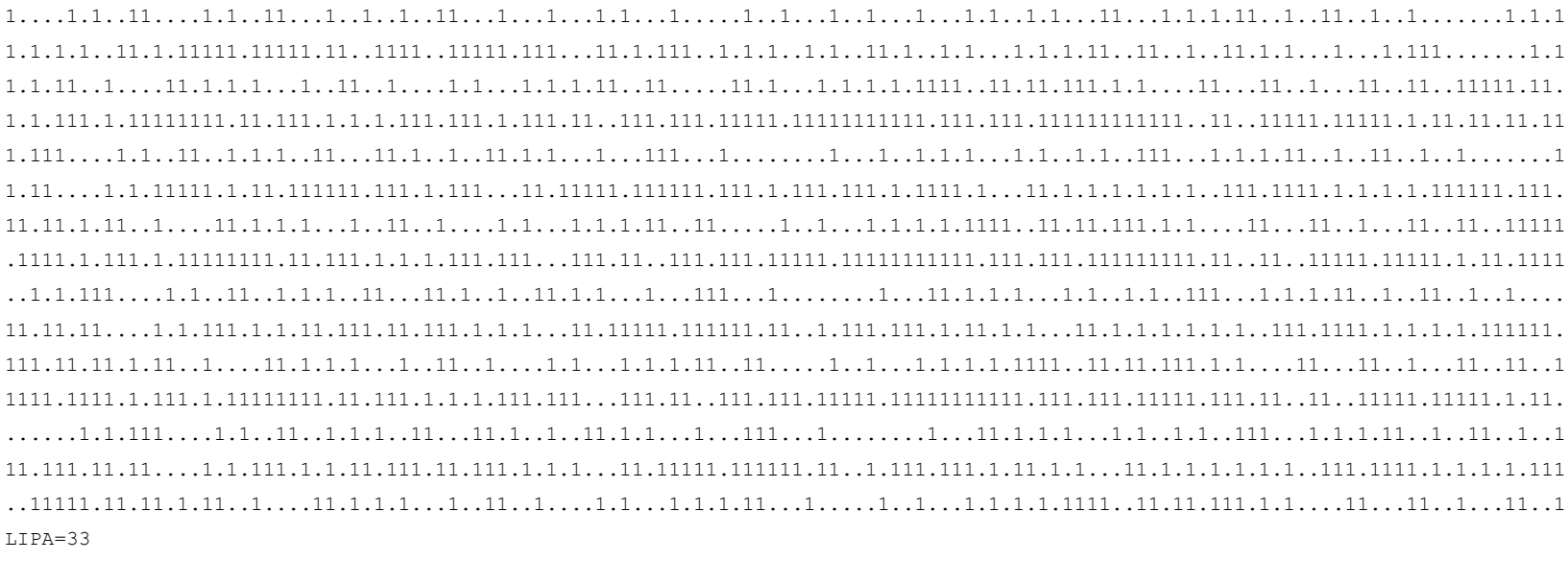


12.7.1.3 MATRIZ DE DEPENDÊNCIAS [D_3] DA TL SERPENT (TRÊS PASSOS)





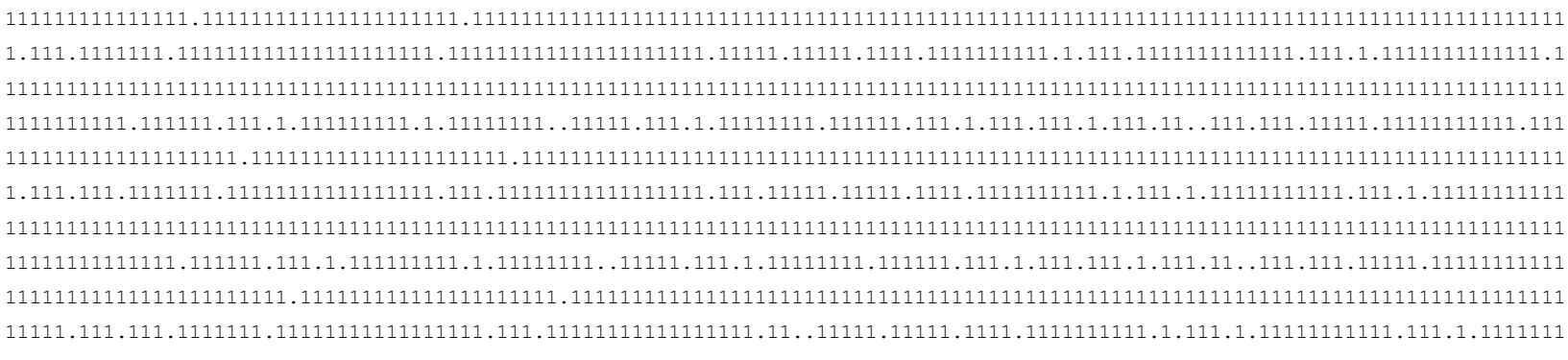




LIPA=33

LSPA=98

12.7.1.4 MATRIZ DE DEPENDÊNCIAS [D_4] DA TL DO SERPENT (QUATRO PASSOS)



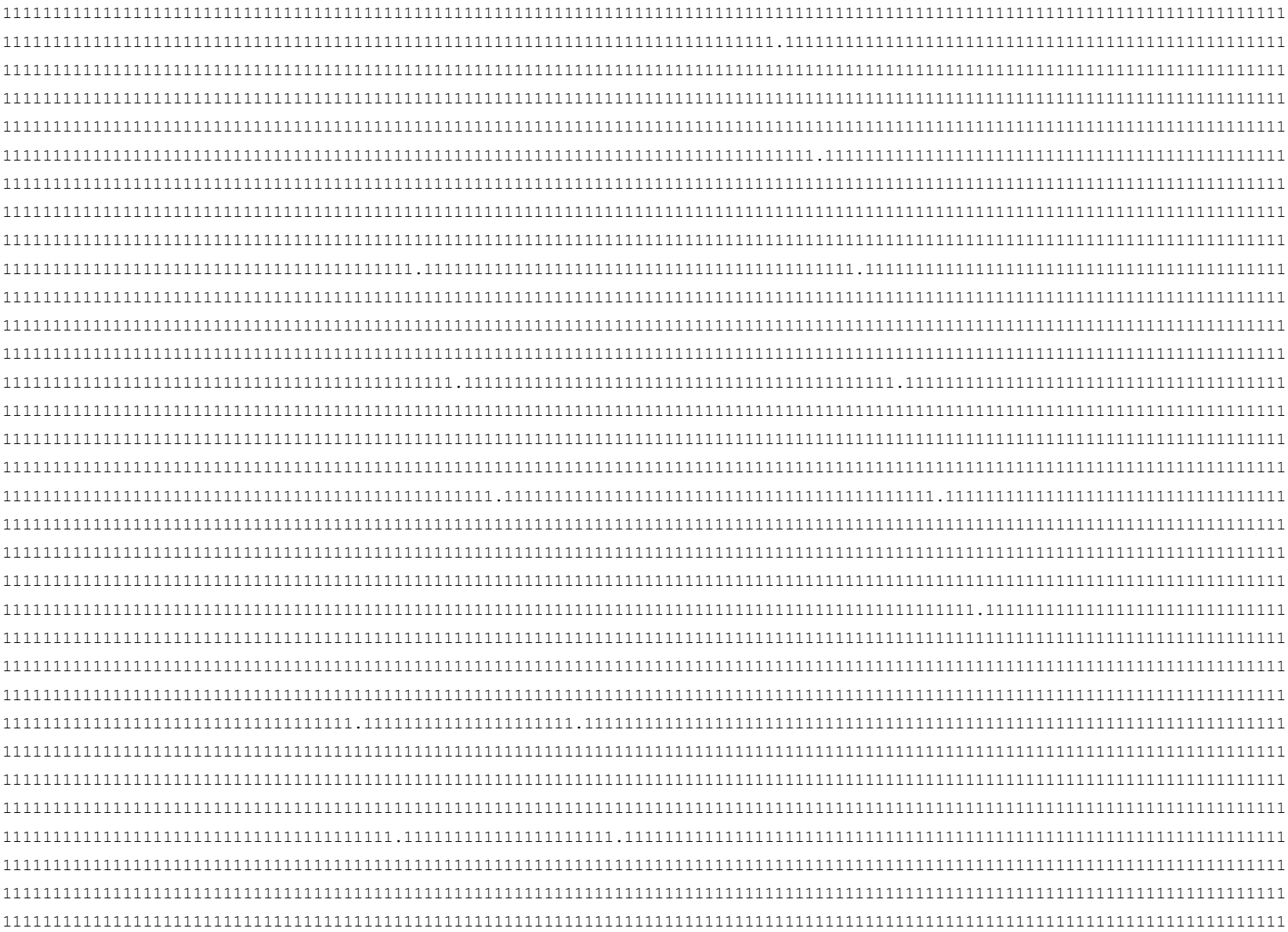
1111111111111.111.11111.1111111111111111111111111111111111111111111111111111.1.1111111111111111111.1111111111111111111.111.111.
1111111111111111111111111111111111111111111111111111111111111111.11111111111111111111.11111111111111111111111111111111111111111111
1.1.111.1111111111111.111.1.1111111111111.111.111.1111111.11111111111111111111.11111111111111111111.11111.11111.1111.11.111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111.1111111111111.111.11111.1111111111111111111111111111111111111111111111111.1.11111111111111111111.11111111111111111111.111.
1111111111111111111111111111111111.1111111.11111111111111111111.11111111111111111111.1111111111111111111111111111111111111111111
11111.1.111.1111111111111.111.1.1111111111111.111.111.1111111.11111111111111111111.111111111111.1111111.11111.11111.1111.11111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111.111.1111111111111.111.11111.1111111111111111111111111111111111111111111111111.1.11111111111111111111.11111111111111111111.
1111111111111111111111111111111111.1111111.11111111111111111111.11111111111111111111.1111111111111111111111111111111111111111111
111111111.1.111.1111111111111.111.1.1111111111111.111.111.1111111.11111111111111111111.111111111111.1111111.11111.11111.1111.1
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111.111.111.1111111111111.111.11111.1111111111111111111111111111111111111111111111111.1.11111111111111111111.11111111111111111111
1111111111111111111111111111111111.1111111.11111111111111111111.11111111111111111111.1111111111111111111111111111111111111111111
11.1111111111.1.111.1111111111111.111.1.1111111111111.111.111.1111111.11111111111111111111.111111111111.1111111.11111.11111.11
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111.111.111.1111111111111.111.11111.1111111111111111111111111111111111111111111111111111.1.11111111111111111111.11111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111.11111111111111111111.1111111111111111111111111111111111
1.1111.1111111111.1.111.1111111111111.111.1.1111111111111.111.111.1111111.11111111111111111111.111111111111.1111111.11111.1111
1111111111111111111111111111111111111111111111111111111111111111111.11111111111111111.111.1111111111111111111111111111111111111111111111111111111111
1111111111.111.111.1111111111111.111.11111.1111111111111111111111111111111111111111111111111111111111.1.11111111111111111111.1111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111.11111111111111111111.1111111111111111111111111111111111
11111.1111.1111111111.1.111.1111111111111.111.1.1111111111111.111.111.1111111.11111111111111111111.111111111111.1111111.11111.
1111111111111111111111111111111111111111111111111111111111111111111.11111111111111111.111.1111111111111111111111111111111111111111111111111111111111
11111111111111.111.111.1111111111111.111.11111.1111111111111111111111111111111111111111111111111111111111.1.11111111111111111111.111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111.11111111111111111111.1111111111111111111111111111111111
111.11111.1111.1111111111.1.111.1111111111111.111.1.1111111111111.111.111.1111111.11111111111111111111.111111111111.1111111.11
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111.11111111111111111111.1111111111111111111111111111111111
11111111111111111111.111.111.1111111111111.111.11111.1111111111111111111111111111111111111111111111111111111111.1.11111111111111111111.11111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1.11111.11111.1111.1111111111.1.111.1111111111111.111.1.1111111111111.111.111.1111111.11111111111111111111.111111111111.111111
1111111111111111111.11111111111111111111.11111111111111111111111111111111111111111111111111111111111111.111111111111111111111.111111111111111111111111111111111111
1111111111111111111.111.111.1111111111111.111.11111.1111111111111111111111111111111111111111111111111111111111.1.11111111111111111111.11111111111111111111111111111111111111111111

1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111.11111111111111111111.1111111111111
11111.11111.11111.1111.1111111111.1.111.1111.11111111.111.1.1111111111111.111.111.1111111.1111111.111111111111.11111111111111
1111111111111111111111.11111111111111111111.111111111111111111111111.1111111111111111.111.11111111111111111111111111111111111
11111.11111111111111111111.111.111.111111111111.111.11111.1111111111111111111111111111111111111111111111111.1.111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111.11111111111111111111.111111111
111111111.11111.11111.1111.1111111111.1.111.1111.1111111.111.1.1111111111111.111.111.1111111.111.111.111111111111.11111111111
11111111111111111111111111.11111111111111111111.111111111111111111111111.1111111111111111.111.11111111111111111111111111111
111111111.11111111111111111111.111.111.111111111111.111.11111.1111111111111111111111111111111111111111111111111.1.11111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111.11111111111111111111.111111
1111111111111.11111.11111.1111.1111111111.1.111.1111.1111111.111.1.1111111111111.111.111.1111111.111.111.111111111111.1111111
111111111111111111111111111111111111111111111111111111111111111111111111111111.1111111111111111.111.1111111111111111111111111
1111111111111.11111111111111111111.111.111.111111111111.111.11111.1111111111111111111111111111111111111111111111111.1.1111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111.11111111111111111111.11
111111.1111111111.1.111.11111.1111.111111.111.1.111.1111.1111111.111.1.1111111111111.111.111.1111111.111.111.111111111111.1111
11111111111111111.111111111111111.1.11111111.111111111111.111111111111111.111111111111.1.111.111.111111111111.11111111111111
1111111111111111.11111111111111111111.111.111.111111111111.111.11111.1111111111111111111111111111111111111111111111111.1.1111
1.111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111.1111111111111111111
111111111.111111111.1.111.11111.1111.111111.111.1.111.1111.1111111.111.1.1111111111111.111.111.1111111.111.111.111111111111.
11111111111111111111.111111111111111.1.11111111.111111111111.111111111111111.111111111111.1.111.111.111111111111.11111111111
11111111111111111111.11111111111111111111.111.111.111111111111.111.11111.1111111111111111111111111111111111111111111111111.1.
11111.11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111.11111111111111
111.111111111.111111111.1.111.11111.1111.111111.111.1.111.1111.1111111.111.1.1111111111111.111.111.1111111.111.111.111111111
1111111111111111111111111.111111111111111.1.11111111.111111111111.111111111111111.111111111111.1.111.111.111111111111.1111111
1.1.11111111111111111111.11111111111111111111.111.111.111111111111.111.11111.1111111111111111111111111111111111111111111111111
111111111.11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111.11111111111
111111.11111111111111111111.11111.11111.1111.1111111111.1.111..111.1111111.111.1.1111111111111.111.111.1111111.111.111.111111
1111111111111111111.111111111.11111111111.111.111111111111111111111111111111111111111111.111111111111.1.11111111111111111111.1111
11111.1.111111111111.1111111.111111111111111111111.111.111.111111111111.111.11111.111111111111111111111111111111111111111111111.
111111111111.11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111.1111111
1111111111.1111111.11.111111111.1.111.1.111.1111.111111.111.1.111.111111111111.111.1.111111111111.111.111.1111111.111111111
11111111111111111111111.111111111.11111111111.111.111111111111111111111111111111111111111111.111111111111.1.111111111111111111111.
111.11111.1.111111111111.1111111.111111111111111111111.111.111.111111111111.111.11111.1111111111111111111111111111111111111111
1111111111111111.1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111.1111

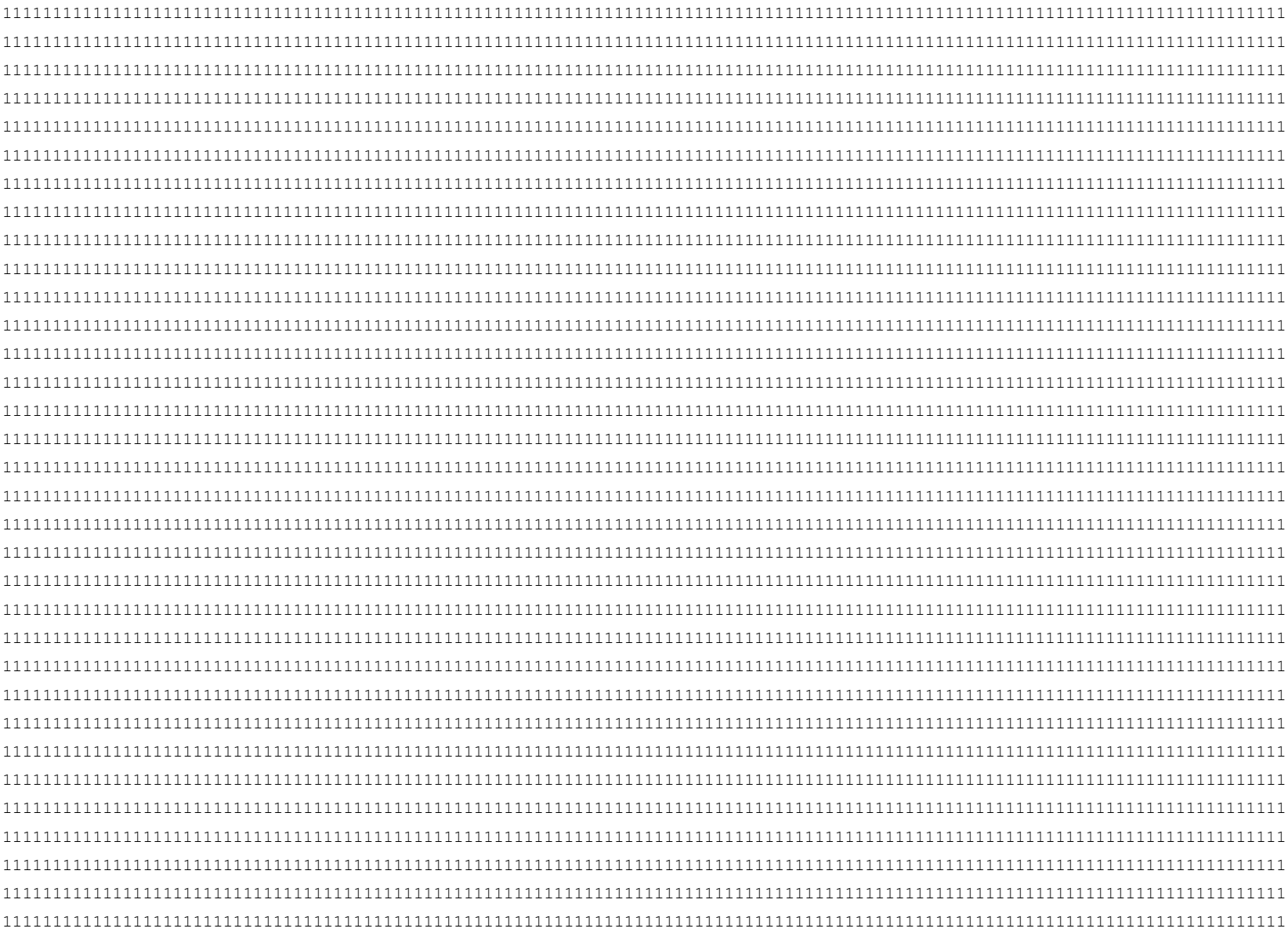
```
11111111111111.1111111.11.111.111111.1.1.1.1.111.1111.111111.111.1.111.1111111111111.111.1.1111111111111.111.111.1111111.111111
111.11111111111111111111111.111111111.1111111111.111.11111111111111111111111111111111111111111.111111111111.1.11111111111111111
1111111.11111.1.111111111111.1111111.11111111111111111111.111.111.111111111111.111.11111.11111111111111111111111111111111111
11111111111111111111.11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111.
11111111111111111.1111111.11.111.111111.1.1.1.1.111.1111.11111..111.1.111.111111111111.111.1.111111111111.111.111.1111111.11
1111111.11111111111111111111111111111111.11111111111111.11111111111111111111111111111111111111111.111111111111.1.11111111111111
1111111111.11111.1.111111111111.1111111.11111111111111111111.111.111.111111111111.111.11111.11111111111111111111111111111111111
111.11111111111111111111.11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1.1111111111111111.111.1111111111111111.111.11111.11111.1111.1.111.1111.1.111.111111111111.111.1.111111111111.111.111.111111
11111111111111111111111111111111.11111111111111111111.11111111111111111111111111111111111111111111111111111.11111111111111111
11111111111111.11111.1.111111111111.111.111.11111111111111111111.111.111.111111111111.111.11111.11111111111111111111111111111
11111111111111111111111111111111111111.11111111111111111111.11111111111111111111111111111111111111111111111111111.11111111111111
1111111111111111.11111.1.111111111111.111.111.11111111111111111111.111.111.111111111111.111.11111.11111111111111111111111111111
1111111111.11111111111111111111.11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1.1111111.11111111111111111.111.11111111111111111.111.11111.11111.1111.1.11111111.1.111.111111111111.111.1.111111111111.111.11
11111111111111111111111111111111111111111.11111111111111111111.11111111111111111111111111111111111111111111111111111.1111111111
11111111111111111111111.11111.1.111111111.111.111.111.11111111111111111111.111.111.111111111111.111.11111.11111111111111111111
LIPA=92
LSPA=128
```

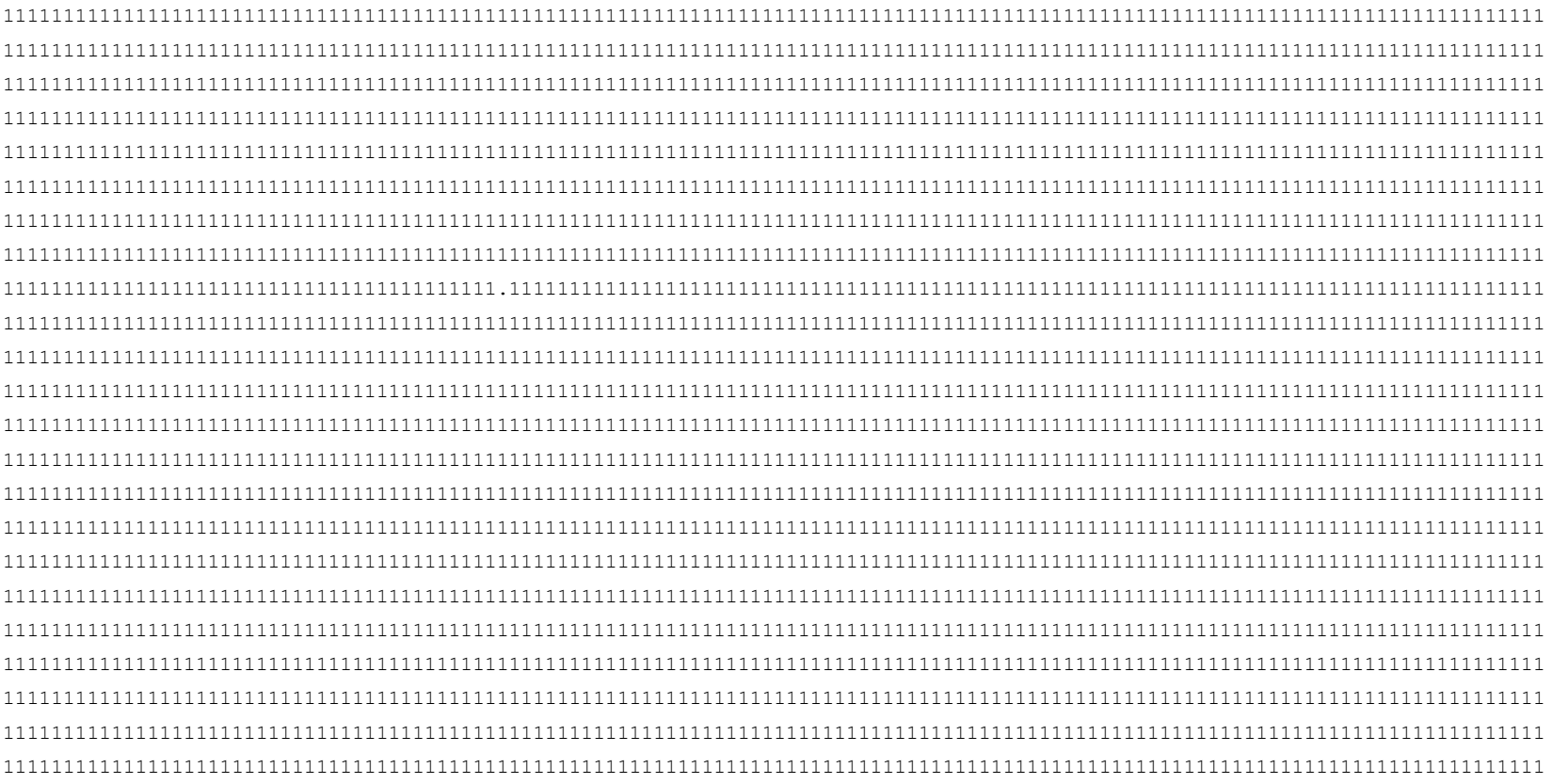
12.7.1.5 MATRIZ DE DEPENDÊNCIAS [D_5] DA TL DO SERPENT (CINCO PASSOS)

```
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
```



1111111111111111111111111111111111111111111.111111111111111111111.1111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111





LIPA=126 , LSPA=128

12.7.1.6 MATRIZ DE DEPENDÊNCIAS [D_6] DA TL DOSERPENT (SEIS PASSOS)

A matriz é completa.

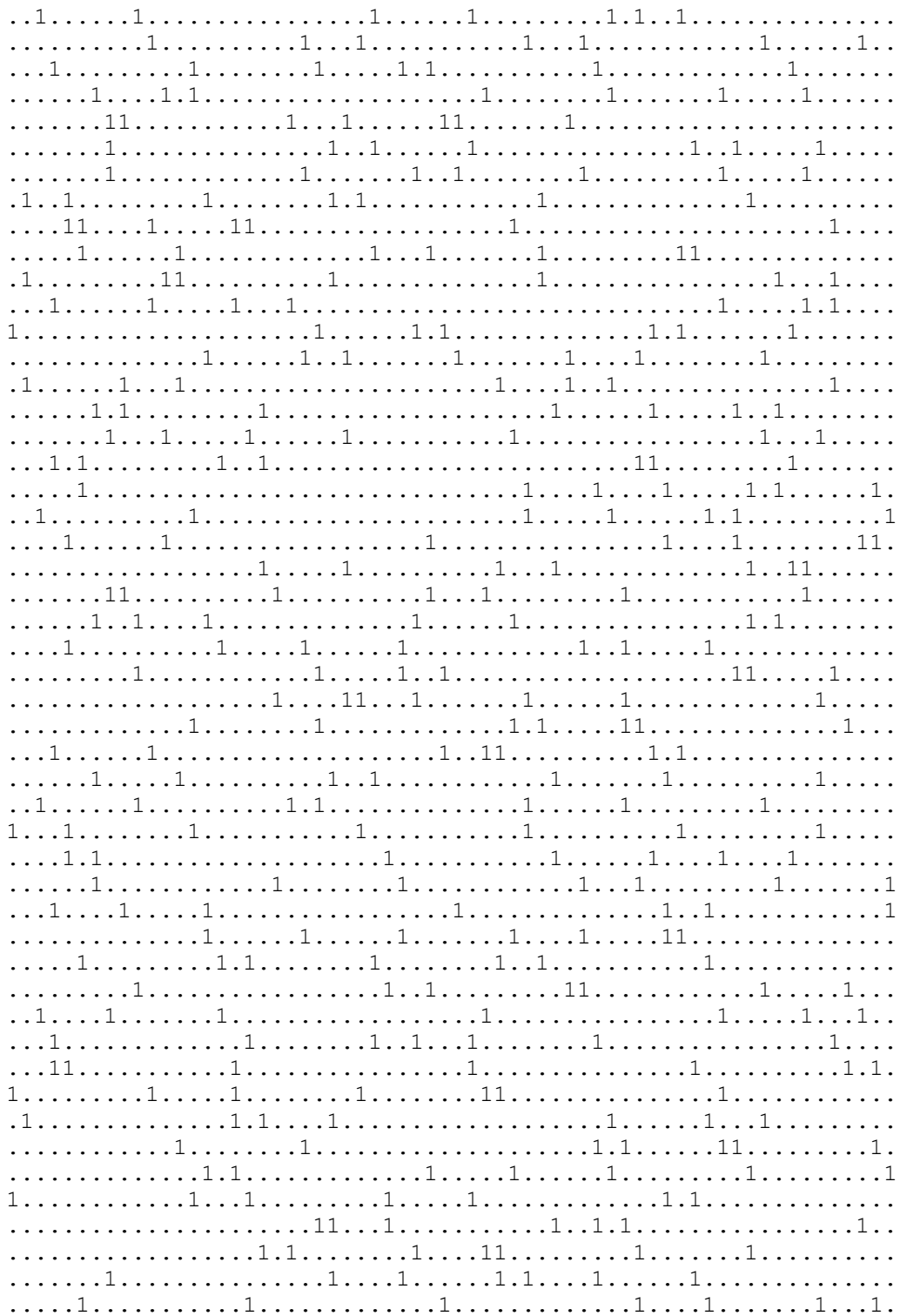
LIPA=128

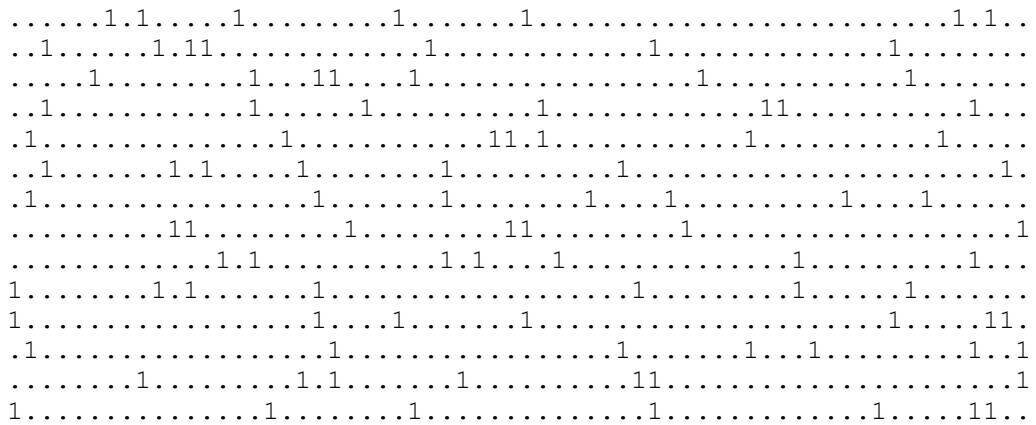
LSPA=128

12.8 **APÊNDICE 8: COMPLEMENTO DO CIFRADOR ALPHA**

12.8.1 MATRIZ QUE DEFINE A TL CIFRADOR ALPHA (“.” = “0”)

Cada um dos 64 bites da saída é função de exatamente sete bites da entrada. Esta matriz define o sistema de equações visto no método L_alpha no item 12.8.3 deste apêndice.





12.8.2 EXEMPLO DE EXECUÇÃO COM *n = t = 128* e *Nr = 10*

Alpha (passo a passo), considerando uma chave de 128 bites expandida pela função de expansão de chaves do AES.

```
Cifrando L[0]|R[0] = 3243f6a8 885a308d 313198a2 e0370734 ...

Sub-chave K0:  2b7e1516 28aed2a6 abf71588 09cf4f3c
Sub-chave K1:  a0fafe17 88542cb1 23a33939 2a6c7605
Sub-chave K2:  f2c295f2 7a96b943 5935807a 7359f67f
Sub-chave K3:  3d80477d 4716fe3e 1e237e44 6d7a883b
Sub-chave K4:  ef44a541 a8525b7f b671253b db0bad00
Sub-chave K5:  d4d1c6f8 7c839d87 caf2b8bc 11f915bc
Sub-chave K6:  6d88a37a 110b3efd dbf98641 ca0093fd
Sub-chave K7:  4e54f70e 5f5fc9f3 84a64fb2 4ea6dc4f
Sub-chave K8:  ead27321 b58dbad2 312bf560 7f8d292f
Sub-chave K9:  ac7766f3 19fadc21 28d12941 575c006e
Sub-chave K10:  d014f9a8 c9ee2589 e13f0cc8 b6630ca6

L[1] :                                          1a4f8db4 c899d592
R[0]:                                           313198a2 e0370734
L_alpha(R[0]):                                  f1b880b7 f3f0008b
L_alpha(R[0] + Kd[0]:                           5a4f953f fa3f4fb7
N_alpha(L_alpha(R[0]+Kd[0]):                    67db038a e515d268
R[1]=L[0] + N_alpha(L_alpha(R[0]+Kd[0]):        5598f522 6d4fe2e5
L[2] :                                          f5620b35 e51bce54
R[1]:                                           5598f522 6d4fe2e5
L_alpha(R[1]):                                  701e3313 b4115cd1
L_alpha(R[1] + Kd[1]:                           53bd0a2a 9e7d2ad4
N_alpha(L_alpha(R[1]+Kd[1]):                    26d7f4e5 aed7b8bf
R[2]=L[1] + N_alpha(L_alpha(R[1]+Kd[1]):        3c987951 664e6d2d
L[3] :                                          ce5aeca3 1cd8d46e
R[2]:                                           3c987951 664e6d2d
L_alpha(R[2]):                                  afa5d064 d7b18759
L_alpha(R[2] + Kd[2]:                           f690501e a4e87126
N_alpha(L_alpha(R[2]+Kd[2]):                    47cde19f 8dc4d190
R[3]=L[2] + N_alpha(L_alpha(R[2]+Kd[2]):        b2afeaaa 68df1fc4
L[4] :                                          8f2fadd7 2fc9e1fa
R[3]:                                           b2afeaaa 68df1fc4
```

```
L_alpha(R[3]):                                  86442712 58c7b463
L_alpha(R[3] + Kd[3]:                           98675956 35bd3c58
N_alpha(L_alpha(R[3]+Kd[3]):                    9b685624 9755cec4
R[4]=L[3] + N_alpha(L_alpha(R[3]+Kd[3]):        5532ba87 8b8d1aaa
L[5] :                                          ba761fc6 23df41d5
R[4]:                                           5532ba87 8b8d1aaa
L_alpha(R[4]):                                  857179ed 9046e400
L_alpha(R[4] + Kd[4]:                           33005cd6 4b4d4900
N_alpha(L_alpha(R[4]+Kd[4]):                    c3044af7 af5da709
R[5]=L[4] + N_alpha(L_alpha(R[4]+Kd[4]):        4c2be720 809446f3
L[6] :                                          98fa21d8 fc17db74
R[5]:                                           4c2be720 809446f3
L_alpha(R[5]):                                  c303e2e8 21197203
L_alpha(R[5] + Kd[5]:                           09f15a54 30e067bf
N_alpha(L_alpha(R[5]+Kd[5]):                    56a1d649 cdcaa0c0
R[6]=L[5] + N_alpha(L_alpha(R[5]+Kd[5]):        ecd7c98f ee15e115
L[7] :                                          815f6af5 ff1edfe8
R[6]:                                           ecd7c98f ee15e115
L_alpha(R[6]):                                  2d504f29 9860462d
L_alpha(R[6] + Kd[6]:                           f6a9c968 5260d5d0
N_alpha(L_alpha(R[6]+Kd[6]):                    b4eef934 37e796e0
R[7]=L[6] + N_alpha(L_alpha(R[6]+Kd[6]):        2c14d8ec cbf04d94
L[8] :                                          62402fe2 94af8467
R[7]:                                           2c14d8ec cbf04d94
L_alpha(R[7]):                                  52170291 b0039078
L_alpha(R[7] + Kd[7]:                           d6b14d23 fea54c37
N_alpha(L_alpha(R[7]+Kd[7]):                    6f0c54fb 722acef5
R[8]=L[7] + N_alpha(L_alpha(R[7]+Kd[7]):        ee533e0e 8d34111d
L[9] :                                          04814d2f 38b9abcf
R[8]:                                           ee533e0e 8d34111d
L_alpha(R[8]):                                  411a0023 253ca8fb
L_alpha(R[8] + Kd[8]:                           7031f543 5ab181d4
N_alpha(L_alpha(R[8]+Kd[8]):                    09323fc3 b87cc720
R[9]=L[8] + N_alpha(L_alpha(R[8]+Kd[8]):        6b721021 2cd34347
L[10] :                                         c70576d2 35299f66
R[9]:                                           6b721021 2cd34347
L_alpha(R[9]):                                  5fa74d14 ebb3a385
L_alpha(R[9] + Kd[9]:                           77766455 bcefa3eb
N_alpha(L_alpha(R[9]+Kd[9]):                    5b8e48d5 fedbec3d
R[10]=L[9] + N_alpha(L_alpha(R[9]+Kd[9]):       5f0f05fa c66247f2
L[11] :                                         8f1bfc52 0f8c627b
R[10]:                                          5f0f05fa c66247f2
L_alpha(R[10]):                                 5b974ac5 a3eb7246
L_alpha(R[10] + Kd[10]:                         baa8460d 15887ee0
N_alpha(L_alpha(R[10]+Kd[10]):                  b86a8ee3 5946bb53
R[11]=L[10] + N_alpha(L_alpha(R[10]+Kd[10]):  7f6ff831 6c6f2435
Texto cifrado : 7f6ff831 6c6f2435 8f1bfc52 0f8c627b

Decifrando L[0]|R[0] = 7f6ff831 6c6f2435 8f1bfc52 0f8c627b ...
Sub-chave K0:  2b7e1516 28aed2a6 abf71588 09cf4f3c
Sub-chave K1:  a0fafe17 88542cb1 23a33939 2a6c7605
Sub-chave K2:  f2c295f2 7a96b943 5935807a 7359f67f
Sub-chave K3:  3d80477d 4716fe3e 1e237e44 6d7a883b
Sub-chave K4:  ef44a541 a8525b7f b671253b db0bad00
Sub-chave K5:  d4d1c6f8 7c839d87 caf2b8bc 11f915bc
Sub-chave K6:  6d88a37a 110b3efd dbf98641 ca0093fd
Sub-chave K7:  4e54f70e 5f5fc9f3 84a64fb2 4ea6dc4f
Sub-chave K8:  ead27321 b58dbad2 312bf560 7f8d292f
Sub-chave K9:  ac7766f3 19fadc21 28d12941 575c006e
Sub-chave K10:  d014f9a8 c9ee2589 e13f0cc8 b6630ca6
L[1] :                                                  5f0f05fa c66247f2
```

```
R[0]:                                                   8f1bfc52 0f8c627b
R[0]+Ke[0]:                                             5f0f05fa c66247f2
L_alpha(R[0]+Ke[0]):                                    5b974ac5 a3eb7246
L_alpha(R[0]+Ke[0]) + Kd[0]:                            baa8460d 15887ee0
N_alpha(L_alpha(R[0]+Ke[0]) + Kd[0]):                   b86a8ee3 5946bb53
R[1]=L[0] + N_alpha(L_alpha(R[0]+Ke[0]) + Kd[0]):       c70576d2 35299f66
L[2] :                                                  6b721021 2cd34347
R[1]:                                                   c70576d2 35299f66
R[1]+Ke[1]:                                             6b721021 2cd34347
L_alpha(R[1]+Ke[1]):                                    5fa74d14 ebb3a385
L_alpha(R[1]+Ke[1]) + Kd[1]:                            77766455 bcefa3eb
N_alpha(L_alpha(R[1]+Ke[1]) + Kd[1]):                   5b8e48d5 fedbec3d
R[2]=L[1] + N_alpha(L_alpha(R[1]+Ke[1]) + Kd[1]):       04814d2f 38b9abcf
L[3] :                                                  ee533e0e 8d34111d
R[2]:                                                   04814d2f 38b9abcf
R[2]+Ke[2]:                                             ee533e0e 8d34111d
L_alpha(R[2]+Ke[2]):                                    411a0023 253ca8fb
L_alpha(R[2]+Ke[2]) + Kd[2]:                            7031f543 5ab181d4
N_alpha(L_alpha(R[2]+Ke[2]) + Kd[2]):                   09323fc3 b87cc720
R[3]=L[2] + N_alpha(L_alpha(R[2]+Ke[2]) + Kd[2]):       62402fe2 94af8467
L[4] :                                                  2c14d8ec cbf04d94
R[3]:                                                   62402fe2 94af8467
R[3]+Ke[3]:                                             2c14d8ec cbf04d94
L_alpha(R[3]+Ke[3]):                                    52170291 b0039078
L_alpha(R[3]+Ke[3]) + Kd[3]:                            d6b14d23 fea54c37
N_alpha(L_alpha(R[3]+Ke[3]) + Kd[3]):                   6f0c54fb 722acef5
R[4]=L[3] + N_alpha(L_alpha(R[3]+Ke[3]) + Kd[3]):       815f6af5 ff1edfe8
L[5] :                                                  ecd7c98f ee15e115
R[4]:                                                   815f6af5 ff1edfe8
R[4]+Ke[4]:                                             ecd7c98f ee15e115
L_alpha(R[4]+Ke[4]):                                    2d504f29 9860462d
L_alpha(R[4]+Ke[4]) + Kd[4]:                            f6a9c968 5260d5d0
N_alpha(L_alpha(R[4]+Ke[4]) + Kd[4]):                   b4eef934 37e796e0
R[5]=L[4] + N_alpha(L_alpha(R[4]+Ke[4]) + Kd[4]):       98fa21d8 fc17db74
L[6] :                                                  4c2be720 809446f3
R[5]:                                                   98fa21d8 fc17db74
R[5]+Ke[5]:                                             4c2be720 809446f3
L_alpha(R[5]+Ke[5]):                                    c303e2e8 21197203
L_alpha(R[5]+Ke[5]) + Kd[5]:                            09f15a54 30e067bf
N_alpha(L_alpha(R[5]+Ke[5]) + Kd[5]):                   56a1d649 cdcaa0c0
R[6]=L[5] + N_alpha(L_alpha(R[5]+Ke[5]) + Kd[5]):       ba761fc6 23df41d5
L[7] :                                                  5532ba87 8b8d1aaa
R[6]:                                                   ba761fc6 23df41d5
R[6]+Ke[6]:                                             5532ba87 8b8d1aaa
L_alpha(R[6]+Ke[6]):                                    857179ed 9046e400
L_alpha(R[6]+Ke[6]) + Kd[6]:                            33005cd6 4b4d4900
N_alpha(L_alpha(R[6]+Ke[6]) + Kd[6]):                   c3044af7 af5da709
R[7]=L[6] + N_alpha(L_alpha(R[6]+Ke[6]) + Kd[6]):       8f2fadd7 2fc9e1fa
L[8] :                                                  b2afeaaa 68df1fc4
R[7]:                                                   8f2fadd7 2fc9e1fa
R[7]+Ke[7]:                                             b2afeaaa 68df1fc4
L_alpha(R[7]+Ke[7]):                                    86442712 58c7b463
L_alpha(R[7]+Ke[7]) + Kd[7]:                            98675956 35bd3c58
N_alpha(L_alpha(R[7]+Ke[7]) + Kd[7]):                   9b685624 9755cec4
R[8]=L[7] + N_alpha(L_alpha(R[7]+Ke[7]) + Kd[7]):       ce5aeca3 1cd8d46e
L[9] :                                                  3c987951 664e6d2d
R[8]:                                                   ce5aeca3 1cd8d46e
R[8]+Ke[8]:                                             3c987951 664e6d2d
L_alpha(R[8]+Ke[8]):                                    afa5d064 d7b18759
L_alpha(R[8]+Ke[8]) + Kd[8]:                            f690501e a4e87126
N_alpha(L_alpha(R[8]+Ke[8]) + Kd[8]):                   47cde19f 8dc4d190
```

```
R[9]=L[8] + N_alpha(L_alpha(R[8]+Ke[8]) + Kd[8]):     f5620b35 e51bce54
L[10] :                                               5598f522 6d4fe2e5
R[9]:                                                 f5620b35 e51bce54
R[9]+Ke[9]:                                           5598f522 6d4fe2e5
L_alpha(R[9]+Ke[9]):                                  701e3313 b4115cd1
L_alpha(R[9]+Ke[9]) + Kd[9]:                          53bd0a2a 9e7d2ad4
N_alpha(L_alpha(R[9]+Ke[9]) + Kd[9]):                 26d7f4e5 aed7b8bf
R[10]=L[9] + N_alpha(L_alpha(R[9]+Ke[9]) + Kd[9]):    1a4f8db4 c899d592
L[11] :                                               313198a2 e0370734
R[10]:                                                1a4f8db4 c899d592
R[10]+Ke[10]:                                         313198a2 e0370734
L_alpha(R[10]+Ke[10]):                                f1b880b7 f3f0008b
L_alpha(R[10]+Ke[10]) + Kd[10]:                       5a4f953f fa3f4fb7
N_alpha(L_alpha(R[10]+Ke[10]) + Kd[10]):              67db038a e515d268
R[11]=L[10] + N_alpha(L_alpha(R[10]+Ke[10]) + Kd[10]):3243f6a8 885a308d
Texto decifrado: 3243f6a8 885a308d 313198a2 e0370734
```

12.8.3 CÓDIGO FONTE EM JAVA UTILIZADO PARA GERAR O ITEM 12.8.2

```
public class alpha_vs_2 {
public static void main(String[] args) {

/****************** comentarios iniciais ***************************
 * Este codigo foi criado para realizar um teste na versao 2 do    *
 * cifrador Alpha com vetor de teste (mesmo do FIPS 197)           *
 * nesta versao a transformação N_alpha usa os quatro primeiros    *
 * bites do baite a esquerda para um deslocamento complementar na  *
 * tabela (12x8 bits). Esta versao suporta chaves de 128 bites e   *
 * ate 17 iterações(definido por Nr)                               *
 * A F Exp Ch é a mesma do Rijndael, com 17 valores de R Con       *
 * Para usar mais de 17 iteracoes, aumentar as R Con ou mudar a    *
 * F Exp Ch                                                        *
 * chaves independentes podem ser usadas desde que altere-se a     *
 * F Exp Ch                                                        *
 * Autor: Jorge de A. Lambert                                      *
 * Jan/04                                                          *
 ******************************************************************/

int chave[]= {0,0,1,0, 1,0,1,1, 0,1,1,1, 1,1,1,0, 0,0,0,1, 0,1,0,1,
0,0,0,1,
              0,1,1,0, 0,0,1,0, 1,0,0,0, 1,0,1,0, 1,1,1,0, 1,1,0,1,
0,0,1,0,
              1,0,1,0, 0,1,1,0, 1,0,1,0, 1,0,1,1, 1,1,1,1, 0,1,1,1,
0,0,0,1,
              0,1,0,1, 1,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,1, 1,1,0,0,
1,1,1,1,
              0,1,0,0, 1,1,1,1, 0,0,1,1, 1,1,0,0};//exemplo apendice A e B
do FIPS 197

int txtclaro[]={0,0,1,1, 0,0,1,0, 0,1,0,0, 0,0,1,1, 1,1,1,1, 0,1,1,0,
1,0,1,0,
               1,0,0,0, 1,0,0,0, 1,0,0,0, 0,1,0,1, 1,0,1,0, 0,0,1,1,
0,0,0,0,
               1,0,0,0, 1,1,0,1, 0,0,1,1, 0,0,0,1, 0,0,1,1, 0,0,0,1,
1,0,0,1,
               1,0,0,0, 1,0,1,0, 0,0,1,0, 1,1,1,0, 0,0,0,0, 0,0,1,1,
0,1,1,1,
               0,0,0,0, 0,1,1,1, 0,0,1,1, 0,1,0,0};//exemplo apendice B do
FIPS

int cifrado[] = alpha(txtclaro,chave,1,11);
int decifrado [] = alpha(cifrado,chave,0,11);

}// fim de main

 /*********************************************************************
 *****                    metodo alpha     -      cifra           ******
 *****       argumento = int[128] binario  e int[128] binario     ******
 *****         retorna = int [128] binario = criptograma          ******
 *********************************************************************/
 public static int[] alpha(int MSG[], int KEY[], int cifra, int Nr){
 int L_anterior[]= new int [64];
 int R_anterior[]= new int [64];
 int L[]= new int [64];
 int R[]= new int [64];
 int temp [] =  new int[4];
 int temp1 [] = new int[8];
```

```
 int bit,byt,w,i,j;
 int cripto [] =  new int[128];

 if (cifra==1){//
     System.out.print("Cifrando L[0]|R[0] = ");
    }
else {
     System.out.print("Decifrando L[0]|R[0] = ");
    }
w=exibe(MSG,128);
System.out.println("");

/*********************************************************
 *                expande a chave                        *
 *********************************************************/
int K[][]= Expande(KEY,Nr); // (EXPAND KEY)
int Ke[][]= new int [Nr][64];
int Kd[][]= new int [Nr][64];
// aqui a chave ja expandida

/*********************************************************
 *                    exibe sub-chaves                   *
 *********************************************************/
for (i=0;i<Nr;i++){//para gerar ilustração das sub-chaves
    System.out.print("Sub-chave K"+i+":  ");
    for (j=0;j<32;j++){// exibe sub-chaves
                temp[0]=K[i][4*j];
                temp[1]=K[i][(4*j)+1];
                temp[2]=K[i][(4*j)+2];
                temp[3]=K[i][(4*j)+3];
                System.out.print(hexa(temp));
         if ((j+1)%8==0){
                System.out.print(" ");
             }//fim do if
    }// fim do for j
    System.out.println("");
    }// fim do loop de exibicao das sub-chaves


/*********************************************************************
 *                    CIFRA ENTRADA                                  *
 *********************************************************************/


/*********************************************************
 *              atribui Ke, Kd, L0 e R0                  *
 *********************************************************/
for (j=0;j<64;j++){
        L_anterior[j] = MSG[j];//L zero
        R_anterior[j] = MSG[j+64];//R zero
        }// fim do for j

for (i=0;i<Nr;i++){
        if (cifra==1){// se vai cifrar, sub-chaves na ordem normal
           for (j=0;j<64;j++){
                Ke[i][j] = K[i][j];
                Kd[i][j] = K[i][j+64];
                }// fim do for j
            }//fim do if cifra=1
        if (cifra==0){// se vai decifrar, sub-chaves na ordem inversa
           for (j=0;j<64;j++){
```

```
                Ke[i][j] = K[Nr-1-i][j];
                Kd[i][j] = K[Nr-1-i][j+64];
                }// fim do for j
            }//fim do if cifra=0
        }// fim do for i

// estao atribuidos todos os Ke e Kd, L0 e R0


/*********************************************************
 *                          iterações                    *
 *********************************************************/
int temporario[] = new int [64];
for (i=0;i<Nr;i++){// versao com Nr iteracoes (p=0)
    // calcula L[i]=R[i-1]+Ke[i-1]
    for (j=0;j<64;j++){
        L[j] = (R_anterior[j]+Ke[i][j])%2;
        temporario [j] =L[j];
        }
    System.out.print("L["+(i+1)+"] :
");
    exibe(temporario,64);
    System.out.println(" ");
    //_____________________________________________________________
    // atribui R(i-1) ao temporario ("estado")
    for (j=0;j<64;j++){
        temporario[j] = R_anterior[j];
        }
    System.out.print("R[" + i + "]:
");
    exibe(temporario,64);
    System.out.println(" ");

    //_____________________________________________________________
    if (cifra==0){//soma Ke ao temporario, se decifracao
        for (j=0;j<64;j++){
        temporario[j]=(temporario[j]+Ke[i][j])%2;
            }
        System.out.print("R[" + i + "]+Ke["+ i+"]:
");
        exibe(temporario,64);
        System.out.println(" ");
    }
    //_____________________________________________________________

    temporario = L_alpha(temporario);
    if (cifra==0){
        System.out.print("L_alpha(R[" + i + "]+Ke["+ i+"]):
");
    }
    else{
        System.out.print("L_alpha(R[" + i + "]):
");
    }
    exibe(temporario,64);
    System.out.println(" ");
    //_____________________________________________________________
    for (j=0;j<64;j++){// chavei-a o vetor temporario para entrada na TNL
        temporario[j]=(temporario[j]+Kd[i][j]) % 2;
        }
    if (cifra==0){
```

```
        System.out.print("L_alpha(R[" + i + "]+Ke["+ i+"]) + Kd["+i+"]:
");
    }
    else{
        System.out.print("L_alpha(R[" + i + "] + Kd["+i+"]:
");
    }
    exibe(temporario,64);
    System.out.println(" ");
    //_____________________________________________________________
    int [] baite = new int [8];
    int [] delta_linha = new int [8];
    delta_linha [0] =
temporario[56]*8+temporario[57]*4+temporario[58]*2+temporario[59];
    for (j=0;j<7;j++){
         delta_linha[(j+1)] = temporario[j*8]*8 + temporario[(j*8)+1]*4 +
temporario[(j*8)+2] *2 + temporario[(j*8)+3];
         }
    for (j=0;j<8;j++){
        for (int k=0;k<8;k++){
            baite [k]= temporario[8*j+k];
            }
        baite = N_alpha (baite, delta_linha[j]);
        for (int k=0;k<8;k++){
           temporario[8*j+k]= baite[k];
            }
        }
    if (cifra==0){
        System.out.print("N_alpha(L_alpha(R[" + i + "]+Ke["+ i+"]) +
Kd["+i+"]):                ");
    }
    else{
        System.out.print("N_alpha(L_alpha(R[" + i + "]+Kd["+i+"]):
");
    }
    exibe(temporario,64);
    System.out.println(" ");

    //_____________________________________________________________________
    //calcula R[i]
    for (j=0;j<64;j++){
        R[j] = (L_anterior[j] + temporario[j]) %2;
        L_anterior[j]=L[j];
        R_anterior[j]=R[j];
        }
    if (cifra==0){
        System.out.print("R["+(i+1)+"]=L["+i+"] + N_alpha(L_alpha(R[" + i
+ "]+Ke["+ i+"]) + Kd["+i+"]):  ");
    }
    else{
        System.out.print("R["+(i+1)+"]=L["+i+"] + N_alpha(L_alpha(R[" + i
+ "]+Kd["+i+"]):       ");
    }
    exibe(R,64);
    System.out.println(" ");
}// fim das iteracoes ************************************

for (j=0;j<64;j++){//faz cripto=R|L (trocado intencionalmente)
     cripto[j] = R[j];//
     cripto[j+64] = L[j];//
     }// fim do for j
```

```
if (cifra==1){//
     System.out.println("Texto cifrado");
    }
else {
     System.out.println("Texto decifrado");
}
w=exibe(cripto,128);
System.out.println("");

return cripto;
}// ******************** fim do metodo alpha  ****************************



/**********************************************************************
 *****            metodo expansão da chave - (EXPAND KEY)         ******
 **********           argumento = int[128] binario                ******
 ***********          retorna = int [Nr][128] binario             ******
 **********************************************************************/
public static int[][] Expande(int KEY[],int Nr){//modificada para gerar
ate 17 sub-chaves
int XK[][] = new int [Nr][128];
int temp1[]= new int[8];
int temp2[]= new int[8];
int temp3[]= new int[8];
int temp4[]= new int[8];
int temptemp[]= new int[8];
String temp;
int i,j,c,c2;
int RCON [][]={{0,0,0,0,0,0,0,0},// este byte nao é usado (apenas para
coerencia dos indices)
               {0,0,0,0,0,0,0,1},// este byte e o RCon[1] pela notação do
FIPS 197 (Nk=4)
               {0,0,0,0,0,0,1,0},// este byte e o RCon[2]
               {0,0,0,0,0,1,0,0},// este byte e o RCon[3]
               {0,0,0,0,1,0,0,0},// este byte e o RCon[4]
               {0,0,0,1,0,0,0,0},// este byte e o RCon[5]
               {0,0,1,0,0,0,0,0},// este byte e o RCon[6]
               {0,1,0,0,0,0,0,0},// este byte e o RCon[7]
               {1,0,0,0,0,0,0,0},// este byte e o RCon[8]
               {0,0,0,1,1,0,1,1},// este byte e o RCon[9]
               {0,0,1,1,0,1,1,0},// este byte e o RCon[10]
               {0,1,1,0,1,1,0,0},// este byte e o RCon[11]
               {1,1,0,1,1,0,0,0},// este byte e o RCon[12]
               {1,0,1,0,1,0,1,1},// este byte e o RCon[13]
               {0,1,0,0,1,1,0,1},// este byte e o RCon[14]
               {1,0,1,1,0,1,1,0},// este byte e o RCon[15]
               {0,1,1,1,0,1,1,1},// este byte e o RCon[16]
               {1,1,1,0,1,1,1,0},// este byte e o RCon[17]
               };//foram cradas até o r_con 17 para possibilitar 17
iteracoes do Alpha



for (j=0;j<128;j++)// atribui a chave aa primeira sub-chave
    XK[0][j]=KEY[j];
for (i=1;i<Nr;i++){// i do FIPS para chave do round i. no ARK inicial i=0.
    for (c=0;c<8;c++){//atribui a word W[i-1,3] a temp
        temp1[c]=XK[i-1][c+96];
        temp2[c]=XK[i-1][c+104];
```

```
        temp3[c]=XK[i-1][c+112];
        temp4[c]=XK[i-1][c+120];
        }//fim do for c


temptemp=temp1;temp1=temp2;temp2=temp3;temp3=temp4;temp4=temptemp;//ROTWOR
D
    //BYTESUB WORD
    temp1=N_alpha(temp1,0);
    temp2=N_alpha(temp2,0);
    temp3=N_alpha(temp3,0);
    temp4=N_alpha(temp4,0);
    //fim de BYTESUB WORD
     for (c=0;c<8;c++){//soma RCon e faz XOR para gerar W[i,0],
     // que é a primeira WORD(de 32 bits) da sub-chave i
        temp1[c]=(temp1[c]+RCON[i][c])%2;
        XK[i][c]=(temp1[c]+XK[i-1][c])%2;
        XK[i][c+8]=(temp2[c]+XK[i-1][c+8])%2;
        XK[i][c+16]=(temp3[c]+XK[i-1][c+16])%2;
        XK[i][c+24]=(temp4[c]+XK[i-1][c+24])%2;
        }
     for (j=1;j<4;j++){//para gerar as WORDS W[i,1],W[i,2],W[i,3]
        for (c=0;c<32;c++){//
            c2=j*32+c;
            XK[i][c2]=(XK[i-1][c2]+XK[i][c2-32])%2;
            }//fim do for c
        }//fim do for j
     }//fim do for i

return XK;
}// ***************** fim do metodo expand key  ************************


/**********************************************************************
 *****         metodo N_alpha = (BYTESUB-rijndael)                ******
 *****         argumento = baite bin                              ******
 *****         retorna = baite bin                                ******
 **********************************************************************/
 public static int [] N_alpha(int [] xy, int delta_linha){//byte xy e o
que sera substituido por zw
 int lin,col, substituto;
 int BS_decimal [][] = {
{99,124,119,123,242,107,111,197,48,1,103,43,254,215,171,118},
{202,130,201,125,250,89,71,240,173,212,162,175,156,164,114,192},
{183,253,147,38,54,63,247,204,52,165,229,241,113,216,49,21},
{4,199,35,195,24,150,5,154,7,18,128,226,235,39,178,117},
{9,131,44,26,27,110,90,160,82,59,214,179,41,227,47,132},
{83,209,0,237,32,252,177,91,106,203,190,57,74,76,88,207},
{208,239,170,251,67,77,51,133,69,249,2,127,80,60,159,168},
{81,163,64,143,146,157,56,245,188,182,218,33,16,255,243,210},
{205,12,19,236,95,151,68,23,196,167,126,61,100,93,25,115},
{96,129,79,220,34,42,144,136,70,238,184,20,222,94,11,219},
{224,50,58,10,73,6,36,92,194,211,172,98,145,149,228,121},
{231,200,55,109,141,213,78,169,108,86,244,234,101,122,174,8},
{186,120,37,46,28,166,180,198,232,221,116,31,75,189,139,138},
{112,62,181,102,72,3,246,14,97,53,87,185,134,193,29,158},
{225,248,152,17,105,217,142,148,155,30,135,233,206,85,40,223},
{140,161,137,13,191,230,66,104,65,153,45,15,176,84,187,22},
};
 lin=xy[0]*8+xy[1]*4+xy[2]*2+xy[3];//clacula linha de entrada na caixa
(caso hexa)
```

```
 lin =(lin + delta_linha)%16;
 col=xy[4]*8+xy[5]*4+xy[6]*2+xy[7];//calcula coluna de entrada na caixa
(caso hexa)
 substituto = BS_decimal[lin][col];
 int zw [] = CG2a8(substituto);
 return zw;
}// ******************* fim do metodo SUBSTITUICAO *********************


/*************************************************
 **********      metodo L_alpha                 ******
 **********   obs: foi implementado bit a bit  ******
 **********  argumento = int[64] binario       ******
 **********   retorna = int[64] binario        ******
 *************************************************/
 public static int[] L_alpha(int e[]){//
 int s[]= new int [64];

 s[20] = ( e[62] + e[47] + e[11] + e[4] + e[30] + e[61] + e[52]) % 2;
 s[47] = ( e[20] + e[35] + e[45] + e[29] + e[18] + e[34] + e[53]) % 2;
 s[11] = ( e[20] + e[10] + e[59] + e[3] + e[16] + e[57] + e[51]) % 2;
 s[4] = ( e[20] + e[40] + e[7] + e[32] + e[24] + e[8] + e[31]) % 2;
 s[30] = ( e[20] + e[22] + e[37] + e[9] + e[54] + e[2] + e[44]) % 2;
 s[61] = ( e[20] + e[60] + e[1] + e[38] + e[46] + e[50] + e[63]) % 2;
 s[52] = ( e[20] + e[19] + e[25] + e[15] + e[5] + e[56] + e[43]) % 2;
 s[35] = ( e[47] + e[28] + e[36] + e[21] + e[14] + e[48] + e[41]) % 2;
 s[45] = ( e[47] + e[49] + e[27] + e[0] + e[33] + e[13] + e[17]) % 2;
 s[29] = ( e[47] + e[58] + e[39] + e[12] + e[23] + e[26] + e[6]) % 2;
 s[18] = ( e[47] + e[55] + e[42] + e[62] + e[37] + e[5] + e[53]) % 2;
 s[34] = ( e[47] + e[8] + e[14] + e[3] + e[63] + e[50] + e[32]) % 2;
 s[53] = ( e[47] + e[15] + e[48] + e[60] + e[2] + e[33] + e[22]) % 2;
 s[10] = ( e[11] + e[59] + e[12] + e[38] + e[55] + e[1] + e[23]) % 2;
 s[59] = ( e[11] + e[56] + e[0] + e[49] + e[9] + e[39] + e[19]) % 2;
 s[3] = ( e[11] + e[6] + e[57] + e[51] + e[13] + e[34] + e[43]) % 2;
 s[16] = ( e[11] + e[7] + e[36] + e[17] + e[24] + e[58] + e[54]) % 2;
 s[57] = ( e[11] + e[10] + e[31] + e[32] + e[42] + e[21] + e[63]) % 2;
 s[51] = ( e[11] + e[55] + e[12] + e[2] + e[40] + e[26] + e[9]) % 2;
 s[40] = ( e[4] + e[33] + e[49] + e[16] + e[60] + e[3] + e[62]) % 2;
 s[7] = ( e[4] + e[53] + e[25] + e[38] + e[23] + e[14] + e[1]) % 2;
 s[32] = ( e[4] + e[46] + e[56] + e[27] + e[6] + e[39] + e[51]) % 2;
 s[24] = ( e[4] + e[15] + e[21] + e[41] + e[28] + e[50] + e[44]) % 2;
 s[8] = ( e[4] + e[36] + e[5] + e[16] + e[59] + e[10] + e[17]) % 2;
 s[31] = ( e[4] + e[25] + e[58] + e[37] + e[48] + e[0] + e[13]) % 2;
 s[22] = ( e[30] + e[44] + e[19] + e[34] + e[7] + e[8] + e[57]) % 2;
 s[37] = ( e[30] + e[40] + e[54] + e[27] + e[41] + e[9] + e[60]) % 2;
 s[9] = ( e[30] + e[26] + e[5] + e[48] + e[38] + e[49] + e[12]) % 2;
 s[54] = ( e[30] + e[58] + e[1] + e[33] + e[17] + e[31] + e[46]) % 2;
 s[2] = ( e[30] + e[22] + e[28] + e[3] + e[13] + e[56] + e[42]) % 2;
 s[44] = ( e[30] + e[36] + e[53] + e[16] + e[63] + e[43] + e[14]) % 2;
 s[60] = ( e[61] + e[19] + e[55] + e[32] + e[24] + e[0] + e[62]) % 2;
 s[1] = ( e[61] + e[37] + e[21] + e[10] + e[54] + e[41] + e[25]) % 2;
 s[38] = ( e[61] + e[2] + e[57] + e[34] + e[7] + e[15] + e[51]) % 2;
 s[46] = ( e[61] + e[27] + e[44] + e[23] + e[39] + e[42] + e[22]) % 2;
 s[50] = ( e[61] + e[24] + e[6] + e[59] + e[8] + e[32] + e[14]) % 2;
 s[63] = ( e[61] + e[60] + e[40] + e[16] + e[25] + e[0] + e[54]) % 2;
 s[19] = ( e[52] + e[50] + e[2] + e[63] + e[37] + e[13] + e[43]) % 2;
 s[25] = ( e[52] + e[22] + e[28] + e[53] + e[9] + e[59] + e[31]) % 2;
 s[15] = ( e[52] + e[46] + e[39] + e[18] + e[55] + e[6] + e[8]) % 2;
 s[5] = ( e[52] + e[23] + e[33] + e[49] + e[58] + e[7] + e[26]) % 2;
 s[56] = ( e[52] + e[41] + e[1] + e[57] + e[27] + e[36] + e[19]) % 2;
 s[43] = ( e[52] + e[44] + e[51] + e[12] + e[42] + e[62] + e[21]) % 2;
```

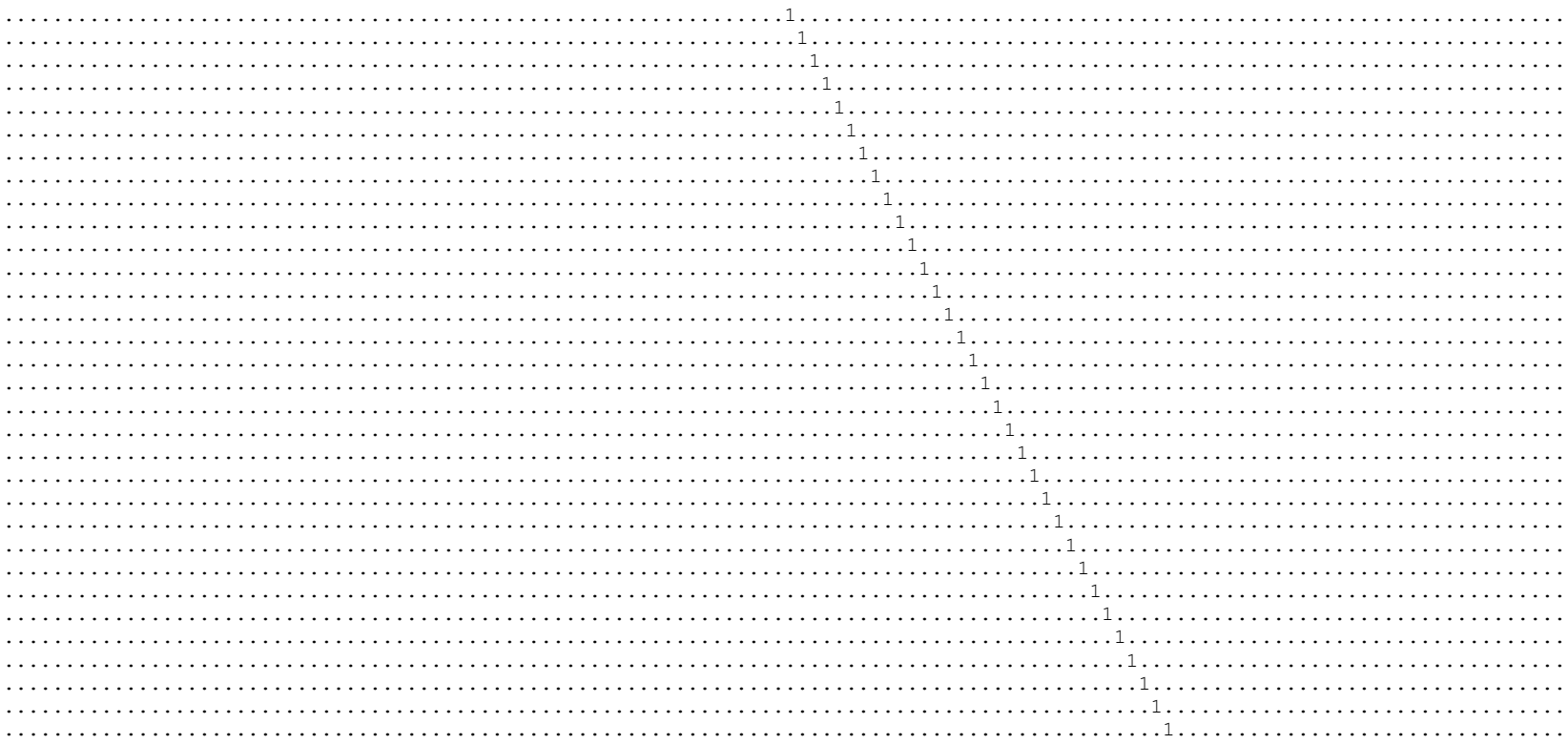
```
 s[28] = ( e[35] + e[34] + e[31] + e[3] + e[48] + e[10] + e[46]) % 2;
 s[36] = ( e[35] + e[5] + e[50] + e[38] + e[15] + e[26] + e[17]) % 2;
 s[21] = ( e[35] + e[56] + e[18] + e[39] + e[57] + e[53] + e[24]) % 2;
 s[14] = ( e[35] + e[1] + e[12] + e[43] + e[8] + e[59] + e[40]) % 2;
 s[48] = ( e[35] + e[28] + e[7] + e[23] + e[49] + e[37] + e[42]) % 2;
 s[41] = ( e[35] + e[10] + e[0] + e[51] + e[25] + e[34] + e[16]) % 2;
 s[49] = ( e[45] + e[62] + e[17] + e[5] + e[50] + e[31] + e[58]) % 2;
 s[27] = ( e[45] + e[13] + e[44] + e[36] + e[22] + e[38] + e[60]) % 2;
 s[0] = ( e[45] + e[43] + e[9] + e[2] + e[48] + e[26] + e[33]) % 2;
 s[33] = ( e[45] + e[6] + e[55] + e[19] + e[28] + e[41] + e[63]) % 2;
 s[13] = ( e[45] + e[14] + e[24] + e[32] + e[40] + e[54] + e[21]) % 2;
 s[17] = ( e[45] + e[15] + e[3] + e[56] + e[18] + e[46] + e[5]) % 2;
 s[58] = ( e[29] + e[60] + e[27] + e[13] + e[34] + e[49] + e[15]) % 2;
 s[39] = ( e[29] + e[42] + e[17] + e[3] + e[59] + e[26] + e[33]) % 2;
 s[12] = ( e[29] + e[48] + e[0] + e[31] + e[46] + e[22] + e[56]) % 2;
 s[23] = ( e[29] + e[14] + e[55] + e[36] + e[53] + e[6] + e[9]) % 2;
 s[26] = ( e[29] + e[19] + e[37] + e[58] + e[24] + e[44] + e[25]) % 2;
 s[6] = ( e[29] + e[21] + e[41] + e[7] + e[57] + e[51] + e[32]) % 2;
 s[55] = ( e[18] + e[12] + e[62] + e[27] + e[38] + e[10] + e[2]) % 2;
 s[42] = ( e[18] + e[43] + e[16] + e[50] + e[23] + e[54] + e[1]) % 2;
 s[62] = ( e[18] + e[20] + e[40] + e[28] + e[63] + e[39] + e[8]) % 2;
 return s;
}// ************        fim do metodo Lalpha           *****************

/*********************************************************************
 *****            metodo hexa - converte bin-hexa                 ******
 **********           argumento = int[4] binario                   ******
 ***********          retorna = um caracter hexa                   ******
 *********************************************************************/
public static String hexa(int bin[]){
String [] Simbolos =
{"0","1","2","3","4","5","6","7","8","9","a","b","c","d","e","f"};
String h= Simbolos[(8*bin[0]+4*bin[1]+2*bin[2]+bin[3])];
return h;
}// ******************** fim do metodo hexa  ***************************


/*********************************************************************
 *****                metodo exibe - exibe  array                 ******
 **********           argumento = int[128] binario                 ******
 ***********          retorna = um caracter hexa                   ******
 *********************************************************************/
public static int exibe(int vetor[], int tamanho){
int temp[]=new int [4];
int h=0;
for (int j=0;j<tamanho;j++){// exibe criptograma hexa
        if ((j%4)==0){
               temp[0]=vetor[j];
               temp[1]=vetor[j+1];
               temp[2]=vetor[j+2];
               temp[3]=vetor[j+3];
               System.out.print(hexa(temp));
               }
         if (j>0){
               if (((j+1)%32)==0) System.out.print(" ");
               }
    }// fim do for j de exibicao do criptograma hexa
return h;
}// ******************** fim do metodo exibe *************************/
```

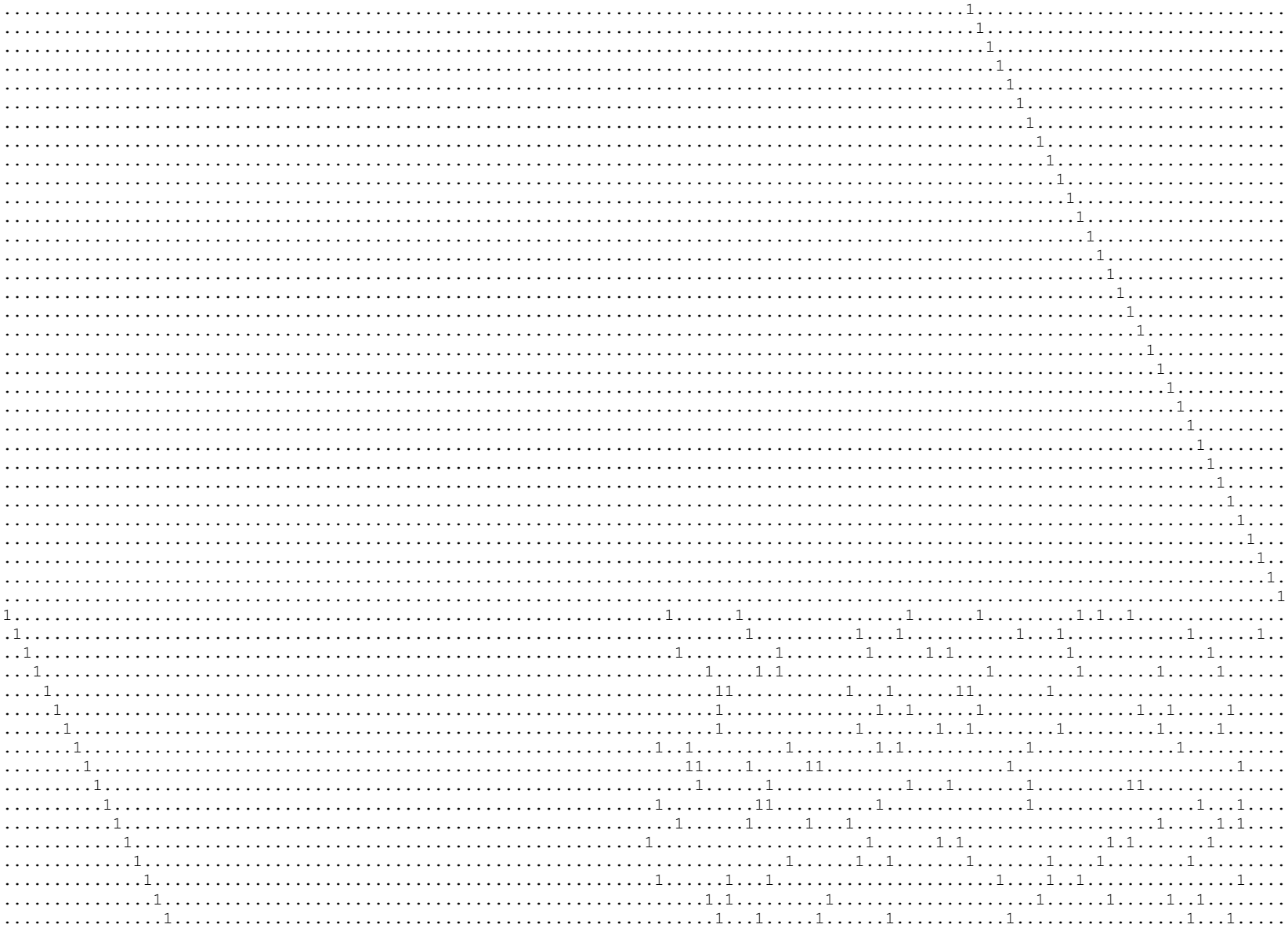
```
/*********************************************************************
 ***** metodo CG2a8- converte inteiro ate 255 em polinomio        ******
 *****                            argumento = int                 ******
 *****          retorna = um polinomio do CG(2^8)                 ******
 *********************************************************************/
public static int [] CG2a8(int inteiro){
int polinomio[]={0,0,0,0,0,0,0,0};
int p2=128;
for (int i=0;i<8;i++){
    if (inteiro>(p2-1)){
        polinomio[i]=1;
        inteiro=inteiro-p2;
    }
    p2=p2/2;
}
return polinomio;
}// fim do metodo CG2a8


}// fim da classe alpha_vs_2
```

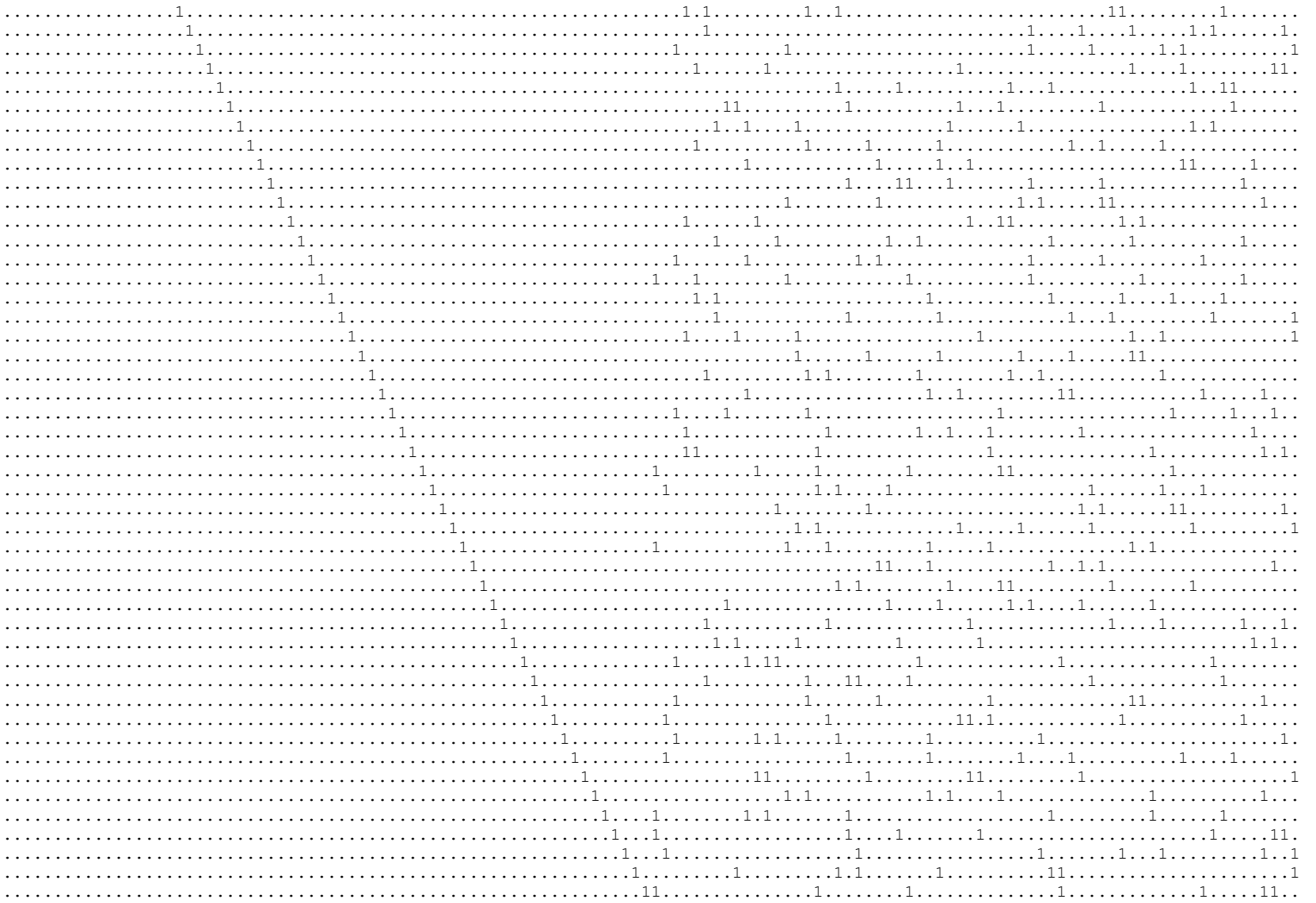
12.9 **APÊNDICE 9: AVALIAÇÃO DO CIFRADOR ALPHA**

12.9.1 MATRIZES DE AVALANCHE DA TL ALPHA

12.9.1.1 MATRIZ DA TL = MATRIZ DE DEPENDÊNCIAS [D_1] - 1 ITERAÇÃO

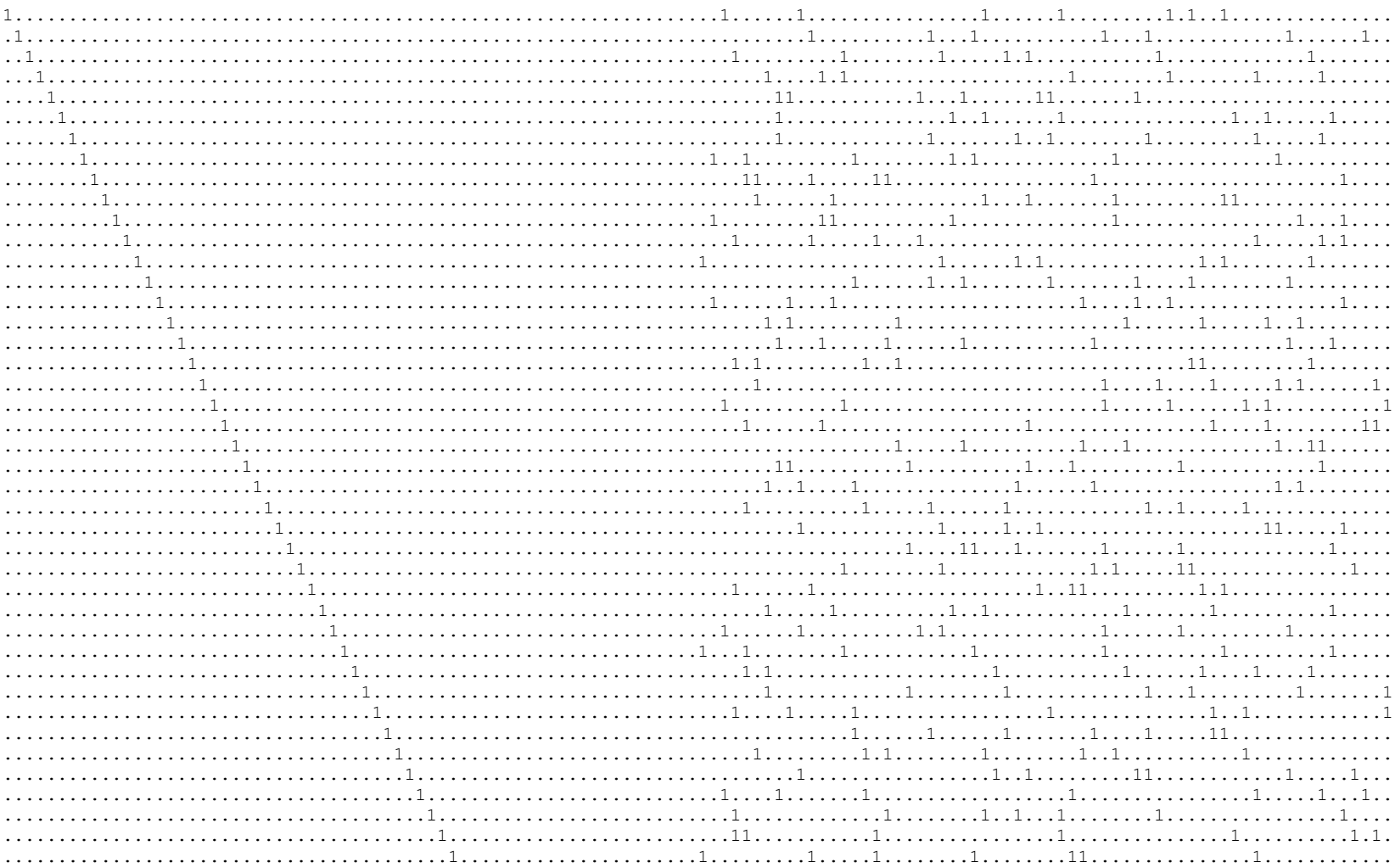


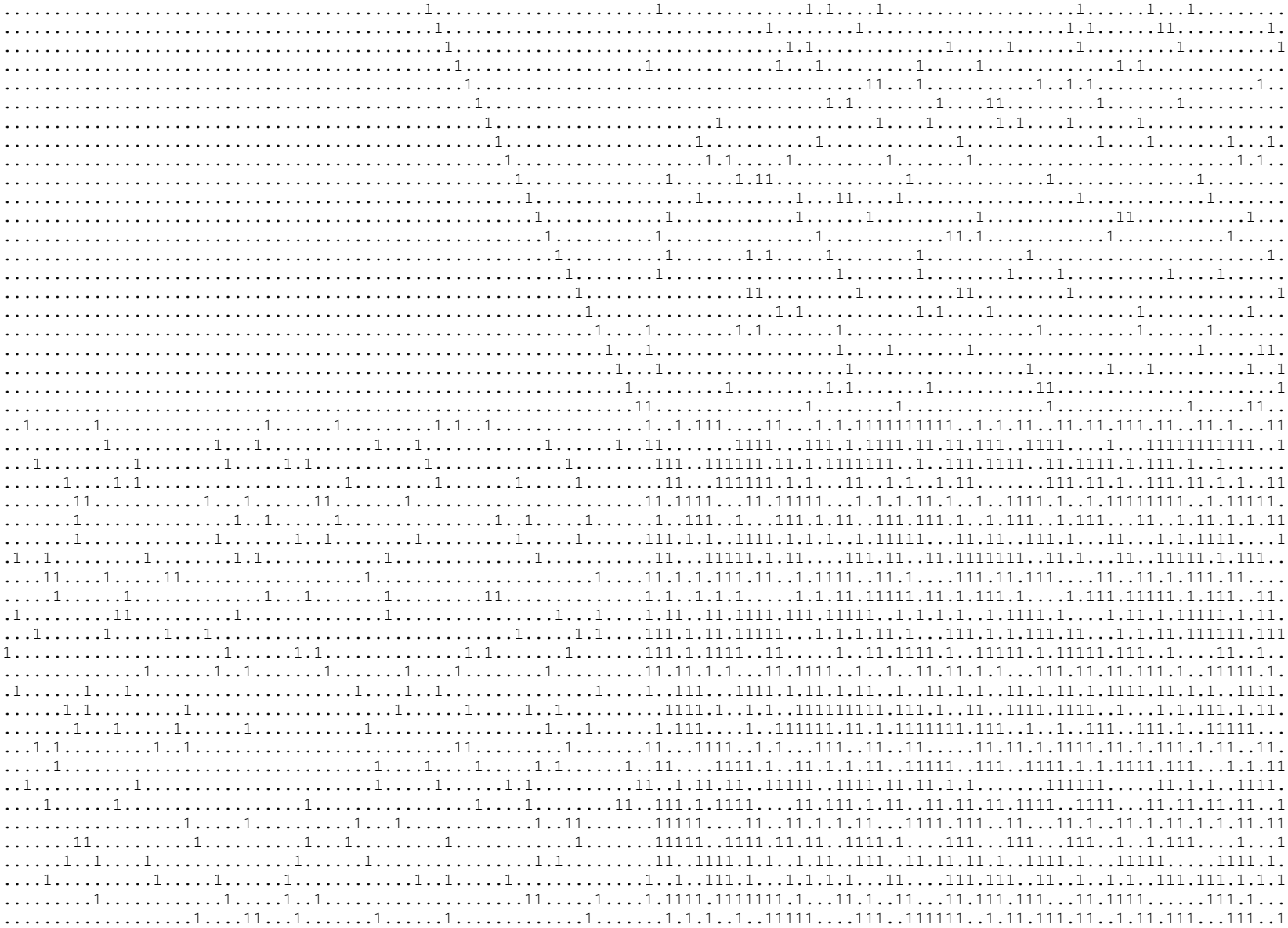


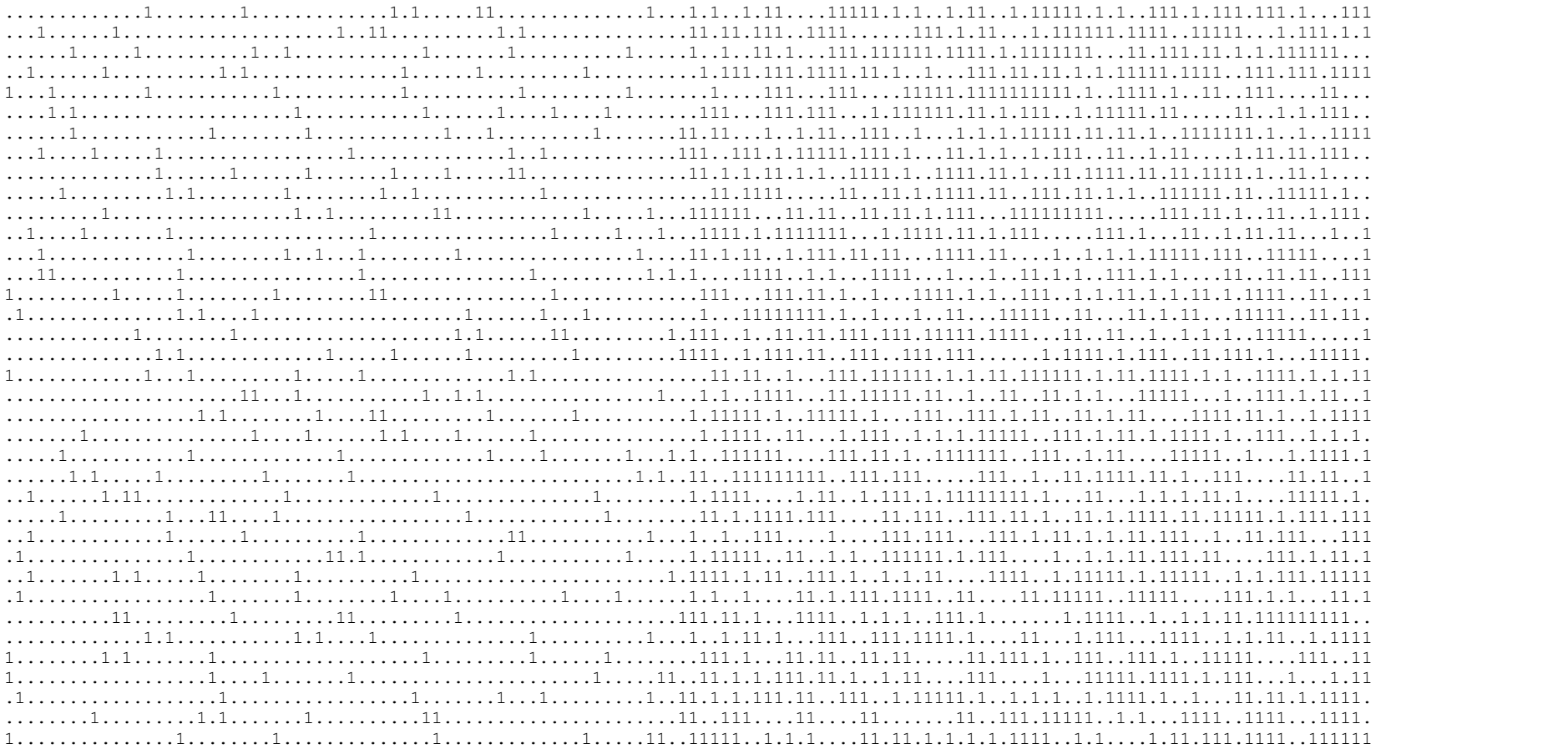


Limite Inferior do Poder de Avalanche 1 iteracao = 1
Limite Superior do Poder de Avalanche 1 iteracao = 8

12.9.1.2 MDR PARA 2 ITERAÇÕES:



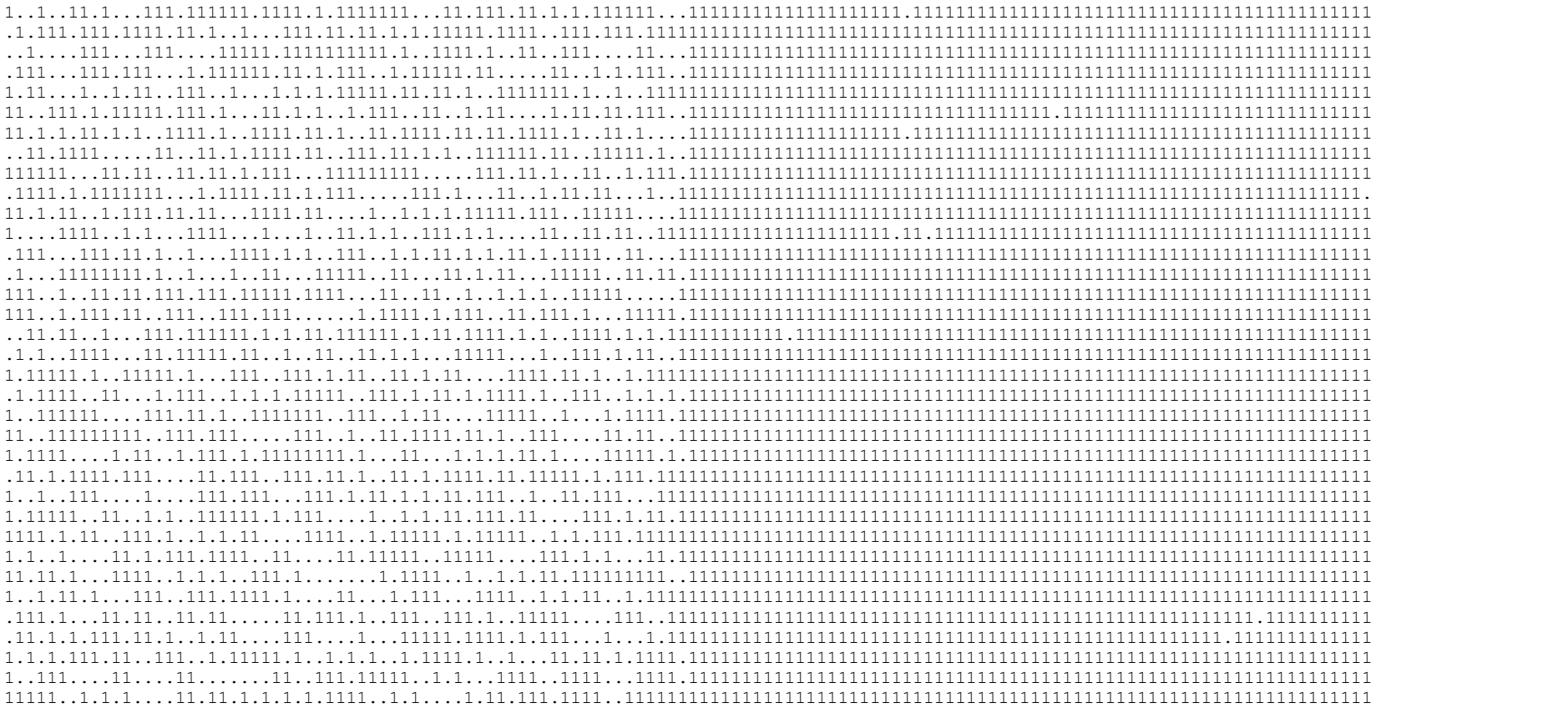




Limite Inferior do Poder de Avalanche 2 = 8

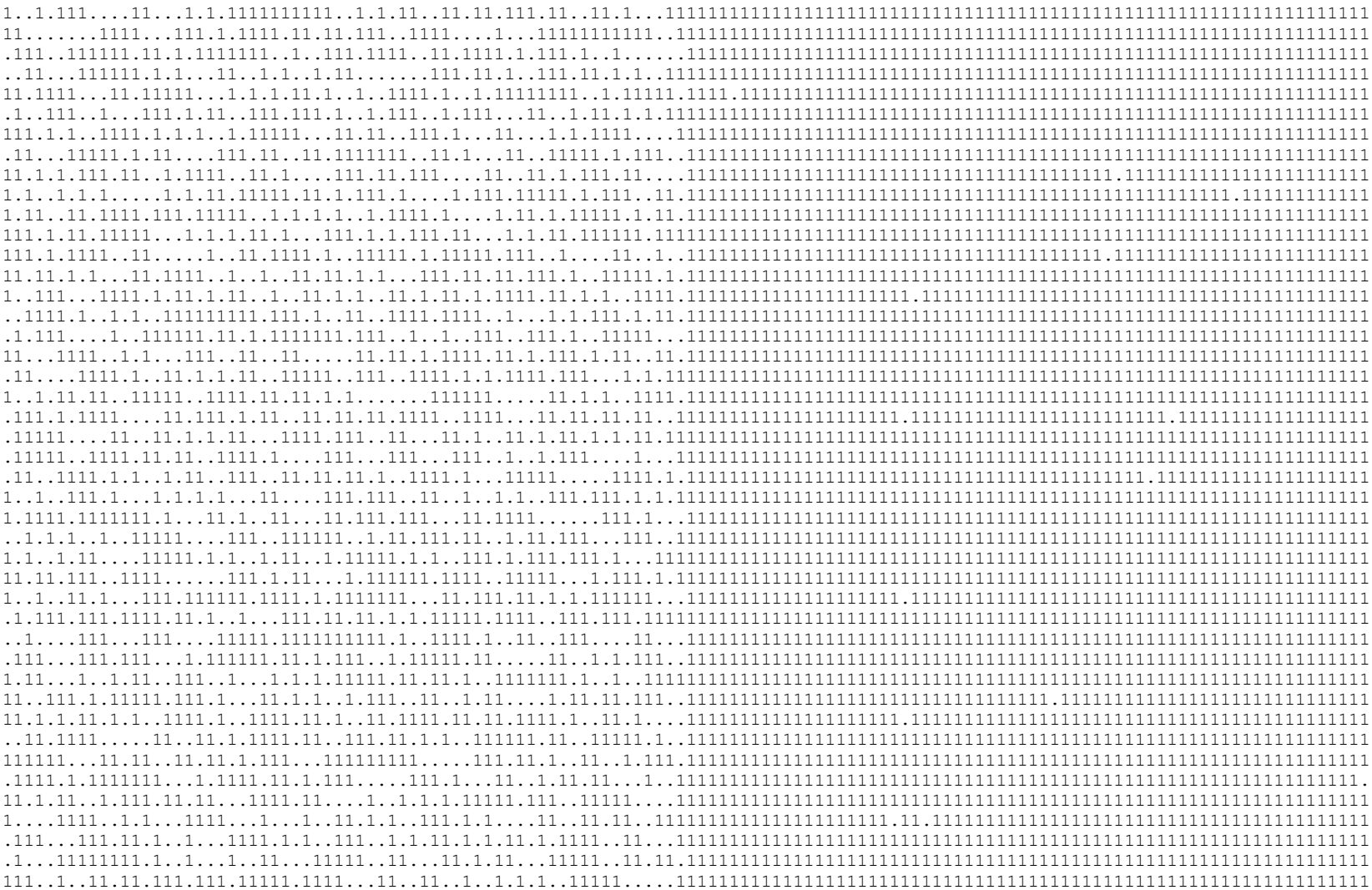
Limite Superior do Poder de Avalanche 2 = 50

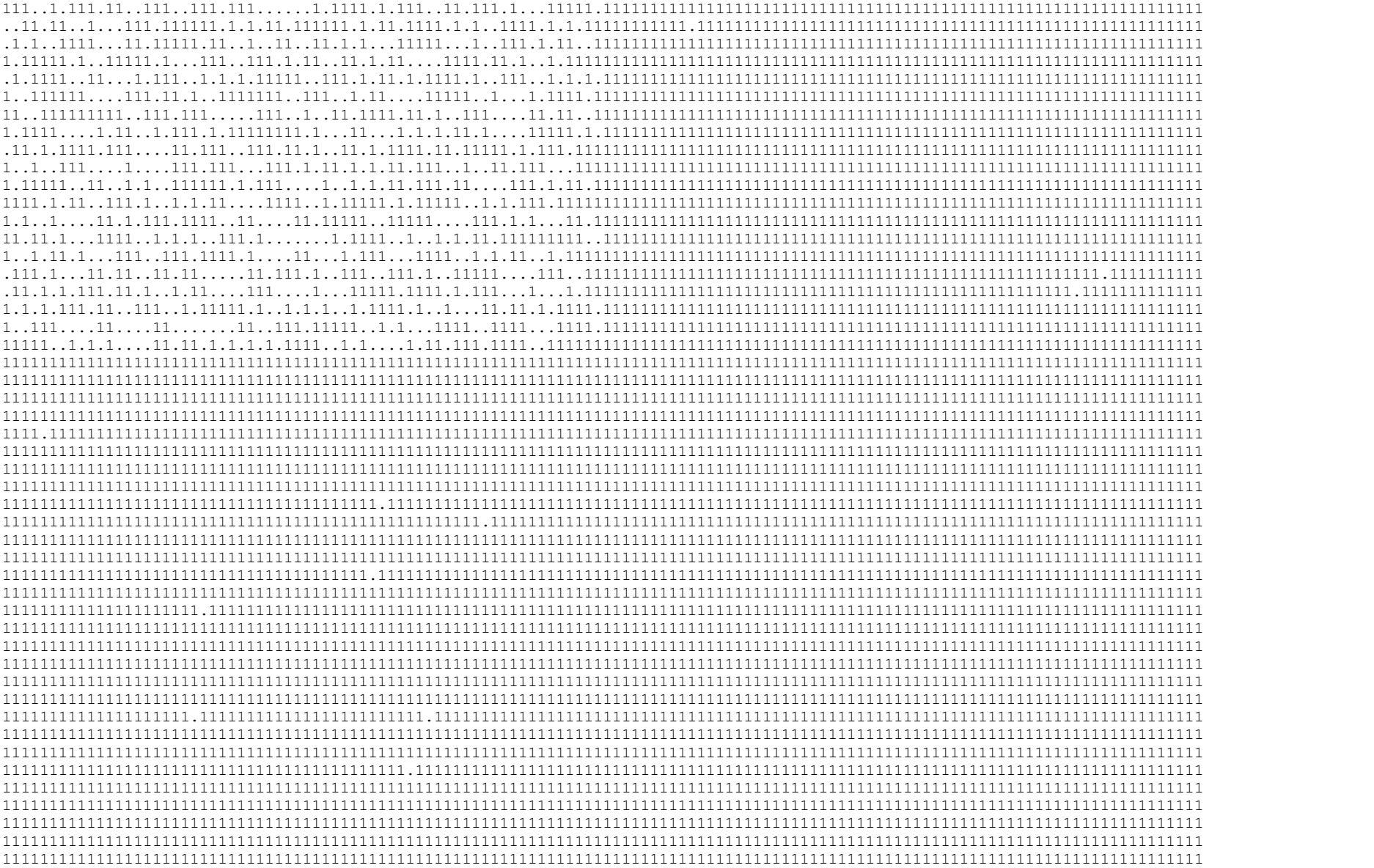
12.9.1.3 MDR PARA 3 ITERAÇÕES:

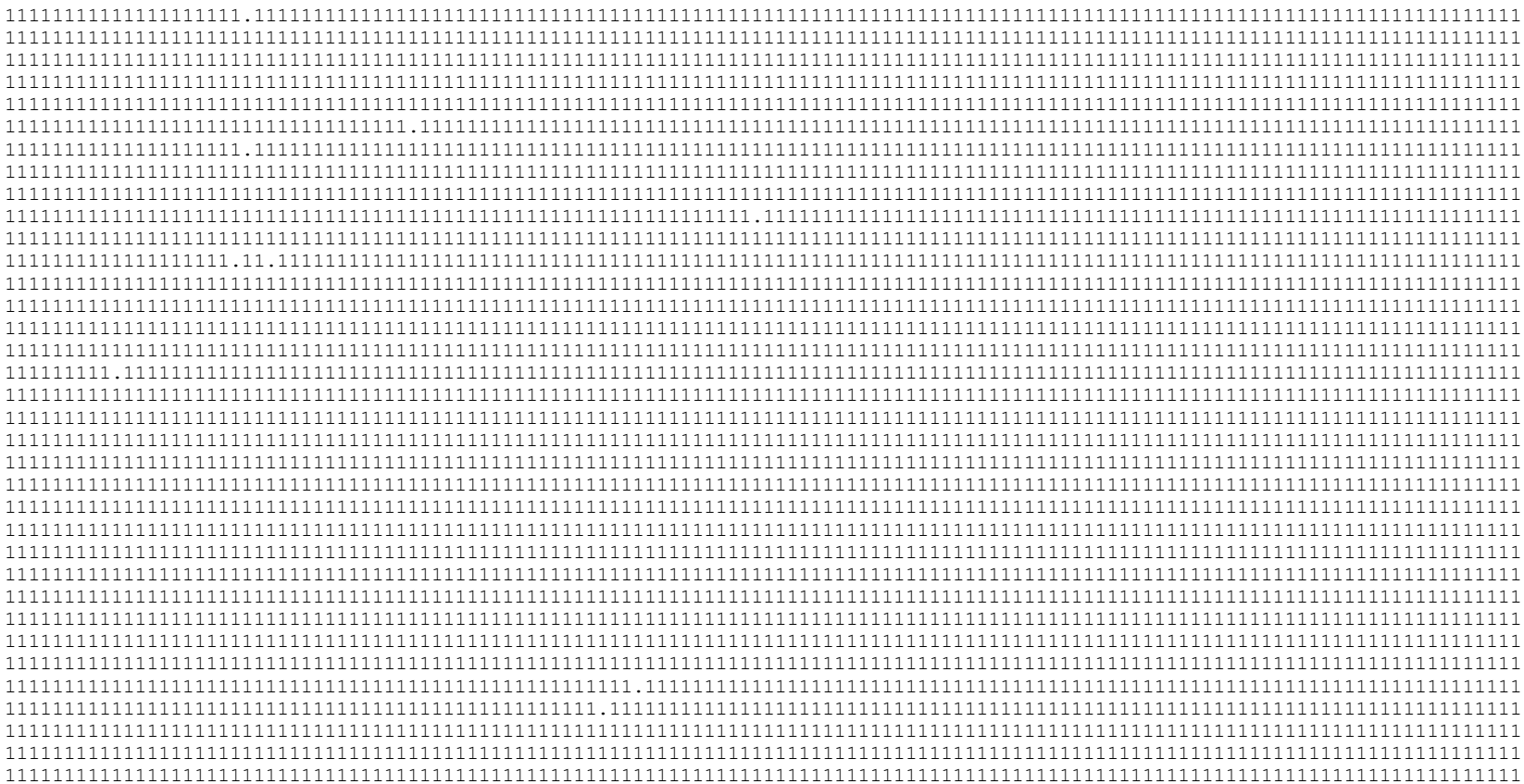


Limite Inferior do Poder de Avalanche 3 = 38
Limite Superior do Poder de Avalanche 3 = 107

12.9.1.4 MDR PARA 4 ITERAÇÕES:

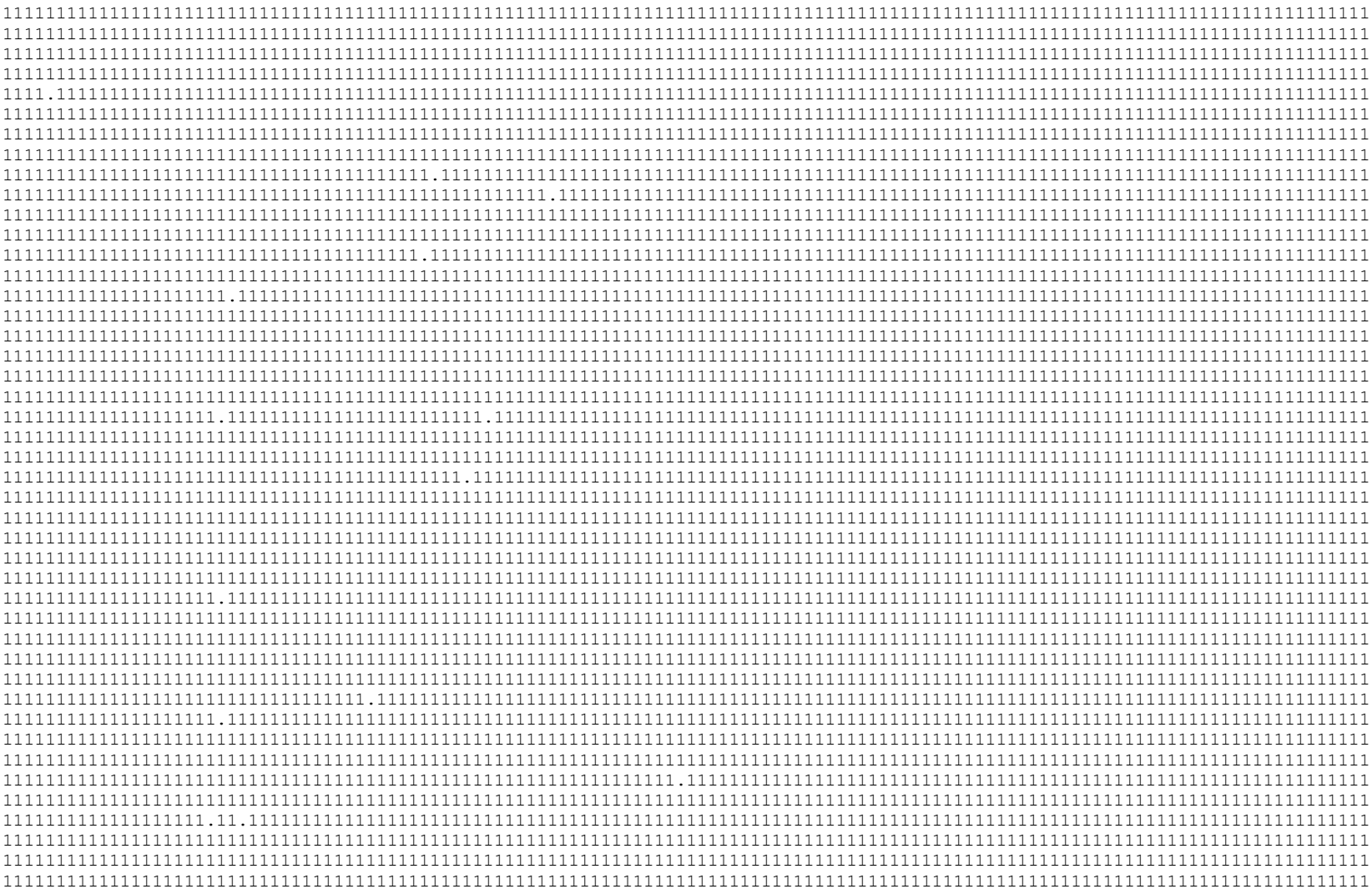


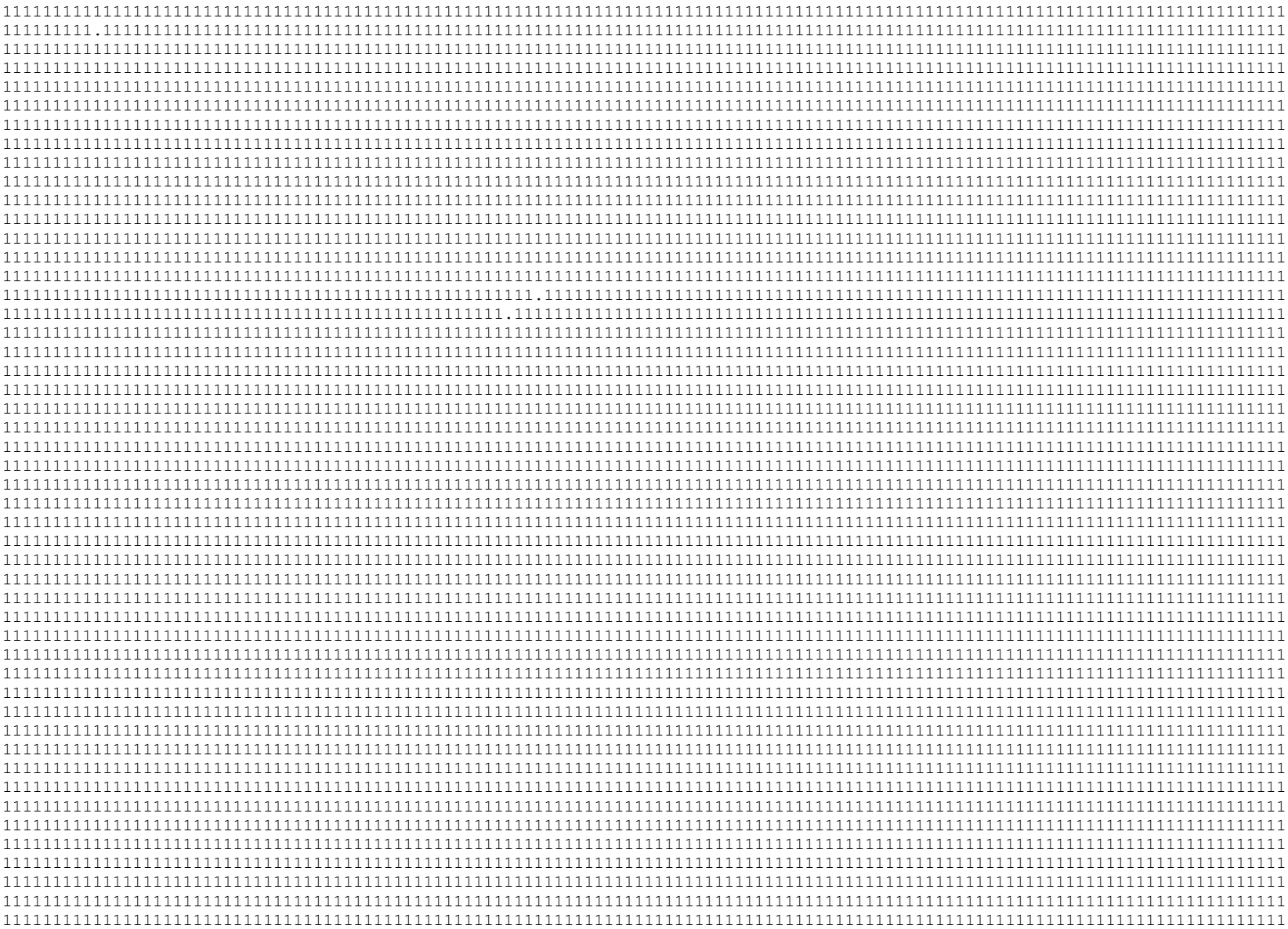


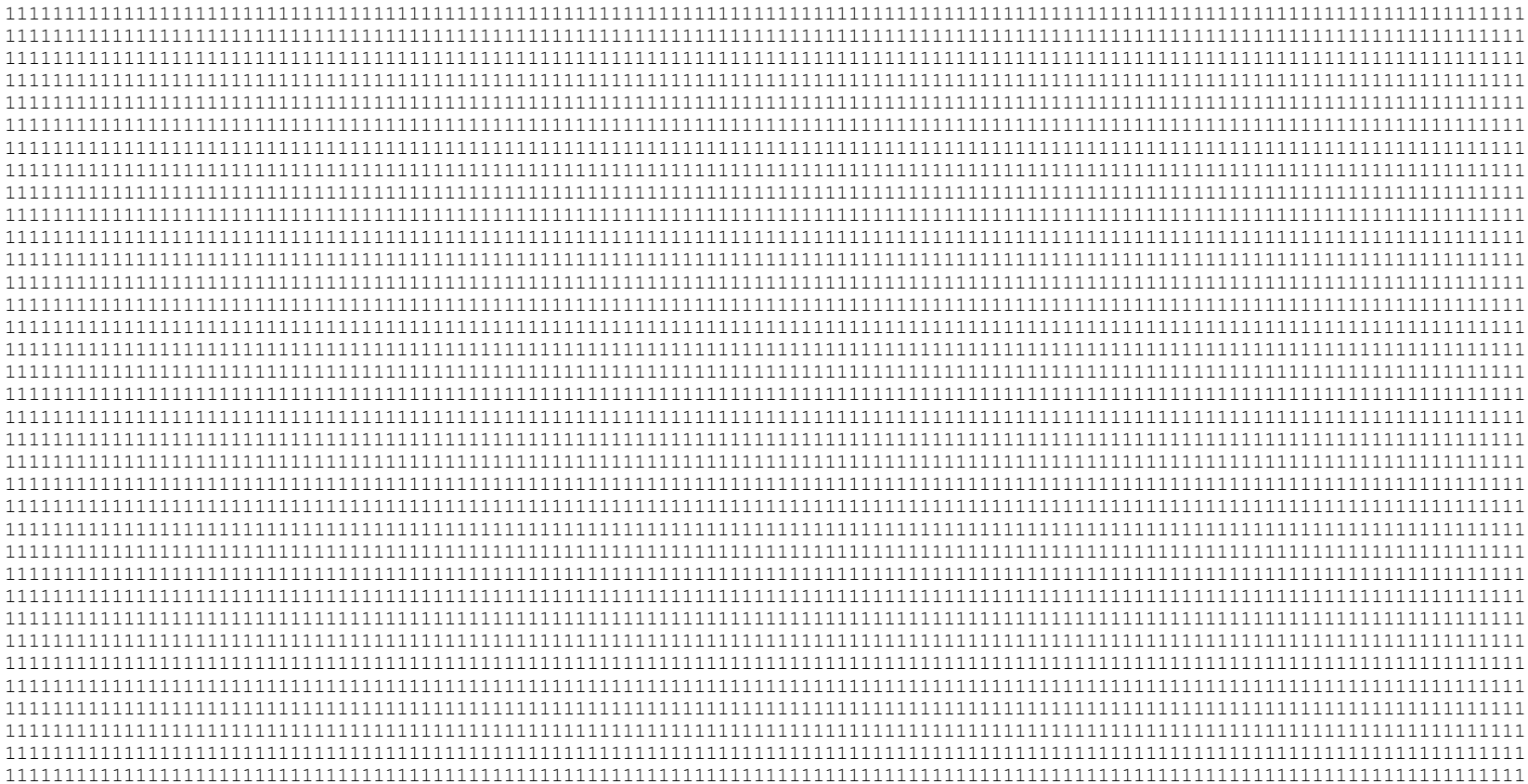


Limite Inferior do Poder de Avalanche 4 = 93
Limite Superior do Poder de Avalanche 4 = 128

12.9.1.5 MDR PARA 5 ITERAÇÕES:







```
Limite Inferior do Poder de Avalanche 5 = 126
Limite Superior do Poder de Avalanche 5 = 128
```

12.9.1.6 MDR PARA 6 ITERAÇÕES:

```
A matriz é completa.
Limite Inferior do Poder de Avalanche 6 = 128
Limite Superior do Poder de Avalanche 6 = 128
```

12.9.1.7 CÓDIGO FONTE (MDR)

O método Alpha foi omitido por encontrar-se no encontra-se no apêndice 8, em 12.8.3.

```
public class Alpha_2_avalanche {

public static void main(String[] args) {

/********************** comentarios iniciais ******************************
 * Este codigo foi criado para construir as MDR da TL da versão 2 do       *
 * cifrador Alpha. A primeira matriz de dependências é construida pelo     *
 * processo da MDE (retirando a TNL e executando uma iteracao do Alpha,    *
 * sem a inversão dos lados no final.                                      *
 * A partir da primeira MDE, as demais sao geradas por recorrência.        *
 * Autor: Jorge de A. Lambert                                              *
 * Jan/04                                                                  *
 ***************************************************************************/

int chave[]= {0,0,1,0, 1,0,1,1, 0,1,1,1, 1,1,1,0, 0,0,0,1, 0,1,0,1, 0,0,0,1,
              0,1,1,0, 0,0,1,0, 1,0,0,0, 1,0,1,0, 1,1,1,0, 1,1,0,1, 0,0,1,0,
              1,0,1,0, 0,1,1,0, 1,0,1,0, 1,0,1,1, 1,1,1,1, 0,1,1,1, 0,0,0,1,
              0,1,0,1, 1,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,1, 1,1,0,0, 1,1,1,1,
              0,1,0,0, 1,1,1,1, 0,0,1,1, 1,1,0,0};//exemplo apendice A e B do FIPS

int Nr=6;
int MDR [][][] = new int [Nr+1][128][128];//pq serao 9 iteracoes
int txtclaro [] = new int [128];
int cripto1 [] = new int [128];
int cripto2 [] = new int [128];
int afetados [] = new int [128];
int D1[][]=new int[128][128];

for (int afeta=0;afeta<128;afeta ++){
     for (int w=0;w<1;w++){
         txtclaro = gerabloco(0);
         cripto1 = alpha_modificado(txtclaro,chave);//MMMMMMMMM
```

```
          txtclaro[afeta]=(txtclaro[afeta]+1)%2;//complementa um bite
          cripto2 = alpha_modificado(txtclaro,chave);//MMMMMMMMM
          afetados = diferenca(cripto1, cripto2);
          for (int i=0; i<128; i++){//
               D1[i][afeta]=(D1[i][afeta]+afetados[i]);
               }//i
          }//w
      }//afeta
     for (int i=0;i<128;i++){
         for (int j=0;j<128;j++){
              if (D1[i][j]>0){
                     D1[i][j]=1;
                     }//if
              else{
                     D1[i][j]=0;
                     }//else
                }//j
       }
// aqui ja foi gerada a primeira matriz de dependencias da TL
MDR = avalanche(D1, Nr);
}// fim de main
/**************************************************************************
*******                         metodo AVALANCHE                      ******
*******              argumento = 1 vetores int [128][128]             ******
*******                  retorna =   int [128][128][Nr]               ******
*                    Construçao das Matrizes de Avalanche                  *
*                      e    calculo dos respectivos                       *
*          Limites Inferiores de Potencial de Avalanche                   *
**************************************************************************/
public static int [][][] avalanche(int D1 [][], int Nr){//
int LIPA []=new int[Nr+1];
int LSPA []=new int[Nr+1];
int Avalanche [][][]= new int [Nr+1][128][128];
String temp="";
for (int i=0 ; i<128 ; i++){
    for (int j=0; j<128; j++){
        Avalanche [1][i][j] = D1[i][j];//atribui a matriz TL à Avalanche de primeira ordem
        }//  fim do for j
    }// fim do for i
```

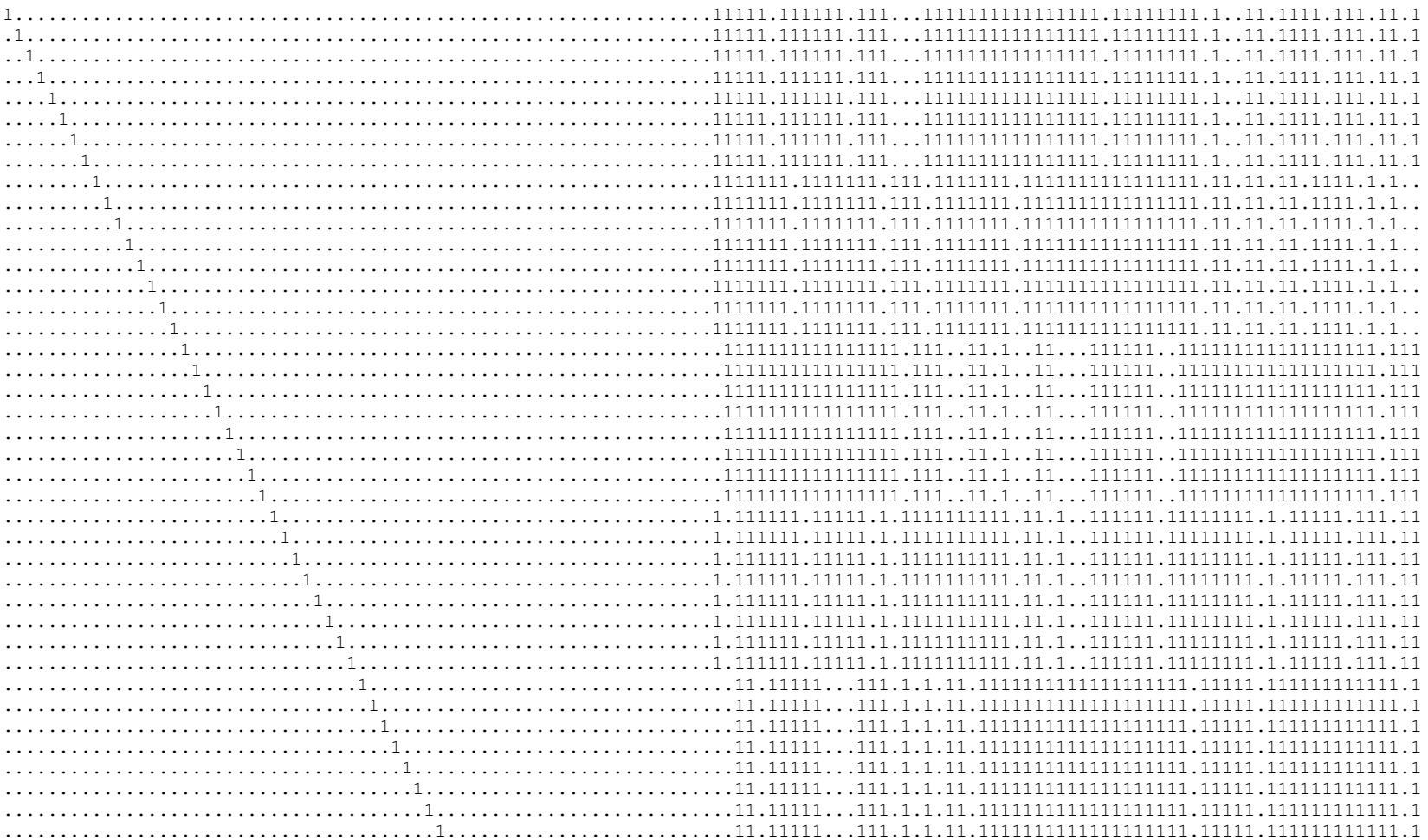
```
//////////////////////////// EXIBIÇÃO ////////////////////////////////////
System.out.println("");
System.out.println("");
temp= "Matriz da TL = Matriz de Dependencias [D1] - 1 iteração";
System.out.println(temp);
LIPA[1]=128;
LSPA[1]=0;

for (int i=0;i<128;i++){// laco que gera impressao da matriz 128x128
       temp = "";
       int c=0;// acumulador de peso hamming da linha i
       for (int j=0;j<128;j++){//
                if (Avalanche[1][i][j]==0){
                  temp=temp+".";
                }// fim do if
                  else{
                   temp=temp+"1";
                   c=c+1;//soma 1 ao peso hamming da linha
                }// fim do else
          }// fim do for j
            if (c<LIPA[1]) {
               LIPA[1]=c;//LIPA[0] recebe o menor peso hamming de todas as linhas da matriz
           }// fim do if
            if (c>LSPA[1]) {
               LSPA[1]=c;//LSPA[ordem_menos_um] recebe o menor peso hamming de todas as linhas da matriz
           }// fim do if
       System.out.println(temp); // exibe a linha i de Avalanche
     }// fim do for i
        temp="Limite Inferior do Poder de Avalanche 1 iteracao = " + LIPA[1];
        System.out.println(temp); // exibe LIPA[ordem_menos_um]
        temp="Limite Superior do Poder de Avalanche 1 iteracao = " + LSPA[1];
        System.out.println(temp); // exibe LSPA[ordem_menos_um]

//////////////////////////////////////////////////////////////////////////

for (int iteracoes=2; iteracoes < Nr+1; iteracoes ++) {
    // Montagem das MDR
    for (int i=0;i<128;i++){// perccorro a linha i da matriz de avalanche que esta sendo montada (ordem =
ordem_menos_um + 1)
```

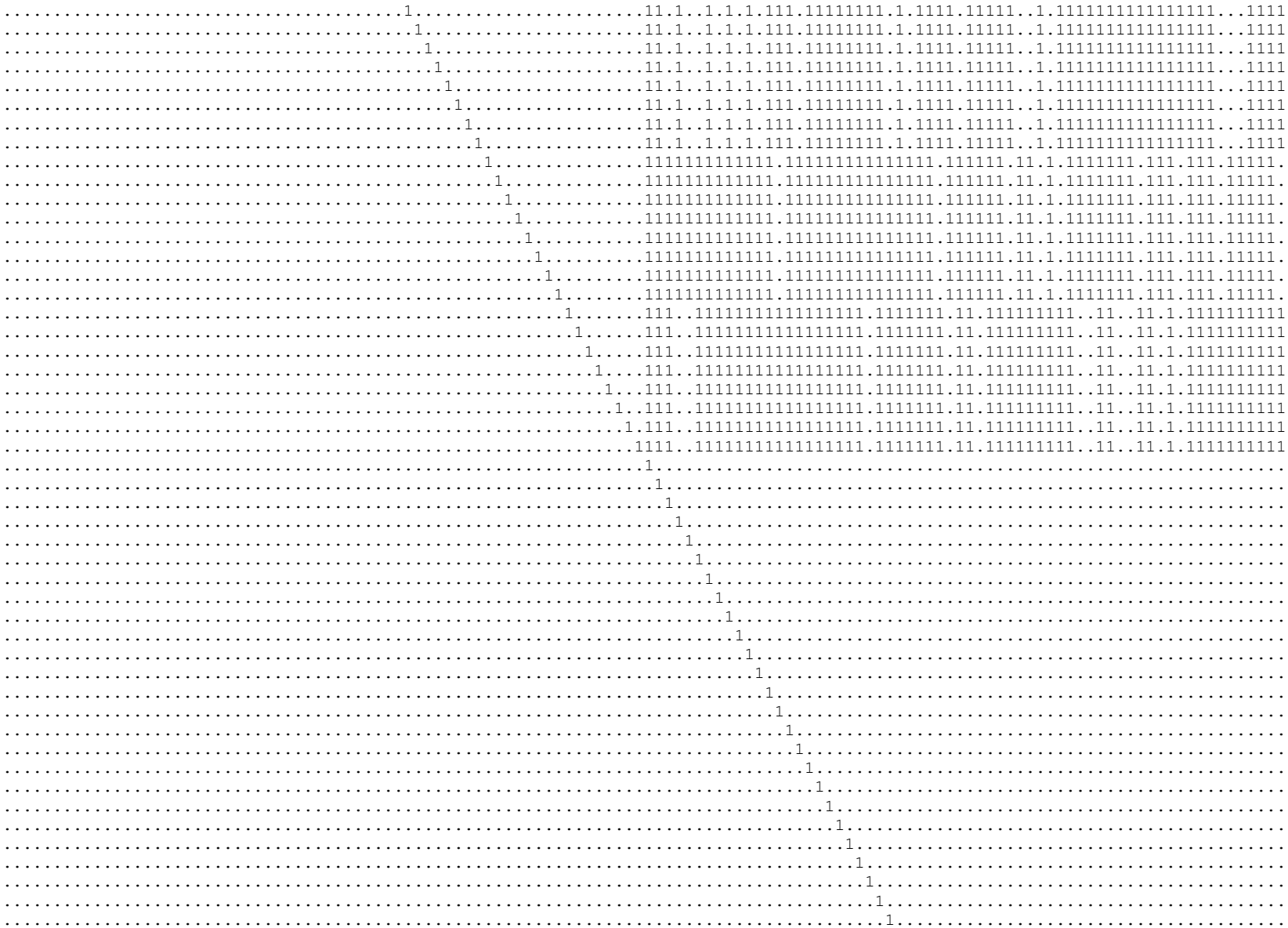
```
        for (int j=0;j<128;j++){//
            if (D1[i][j] > 0){// se na matriz de dependencias o j-esimo bite da entrada afeta o i-esimo bite da
saida
                    for (int c=0; c<128;c++){//percorre a linha j da mantriz de avalanche anterior
                    // a ideia é: se o j-esimo bite da entrada anterior interfere no i-esimo bite da saída anterior
(olho na de matriz de primeira ordem) ,
                    // todos os bites que interferiram no j-ésimo bite da saida na matriz anterior interferem
                    // no i-ésimo bite da saida atual por recorrencia
                           if (Avalanche[iteracoes-1][j][c]>0) Avalanche[iteracoes][i][c]=1;// seta na linha i da
matriz atual todos os
                           // bites que estão setados na linha j da matriz anterior
                        }//fim do for c
                }// fim do if
            }// fim do for j
        }// fim do for i
//montei agora exibo



/*************************************************************************
* Exibicao das Matrizes de Avalanche                                     *
*************************************************************************/
temp = "MDR para " + iteracoes + " iterações:";
System.out.println(temp);
LIPA [iteracoes]= 128;//Atribui valor máximo a LIPA[ordem_menos_um]
LSPA [iteracoes]= 0;//Atribui valor mínimo a LSPA[ordem_menos_um]
    for (int i=0;i<128;i++){// laco que gera impressao da matriz 128x128
       temp = "";
       int c=0;// acumulador de peso hamming da linha i
       for (int j=0;j<128;j++){//
            if (Avalanche[iteracoes][i][j]==0){
                  temp = temp + ".";
               }// fim do if
            else{
                  temp=temp+"1";
                  c=c+1;//soma 1 ao peso hamming da linha
               }// fim do else
         }// fim do for j
```

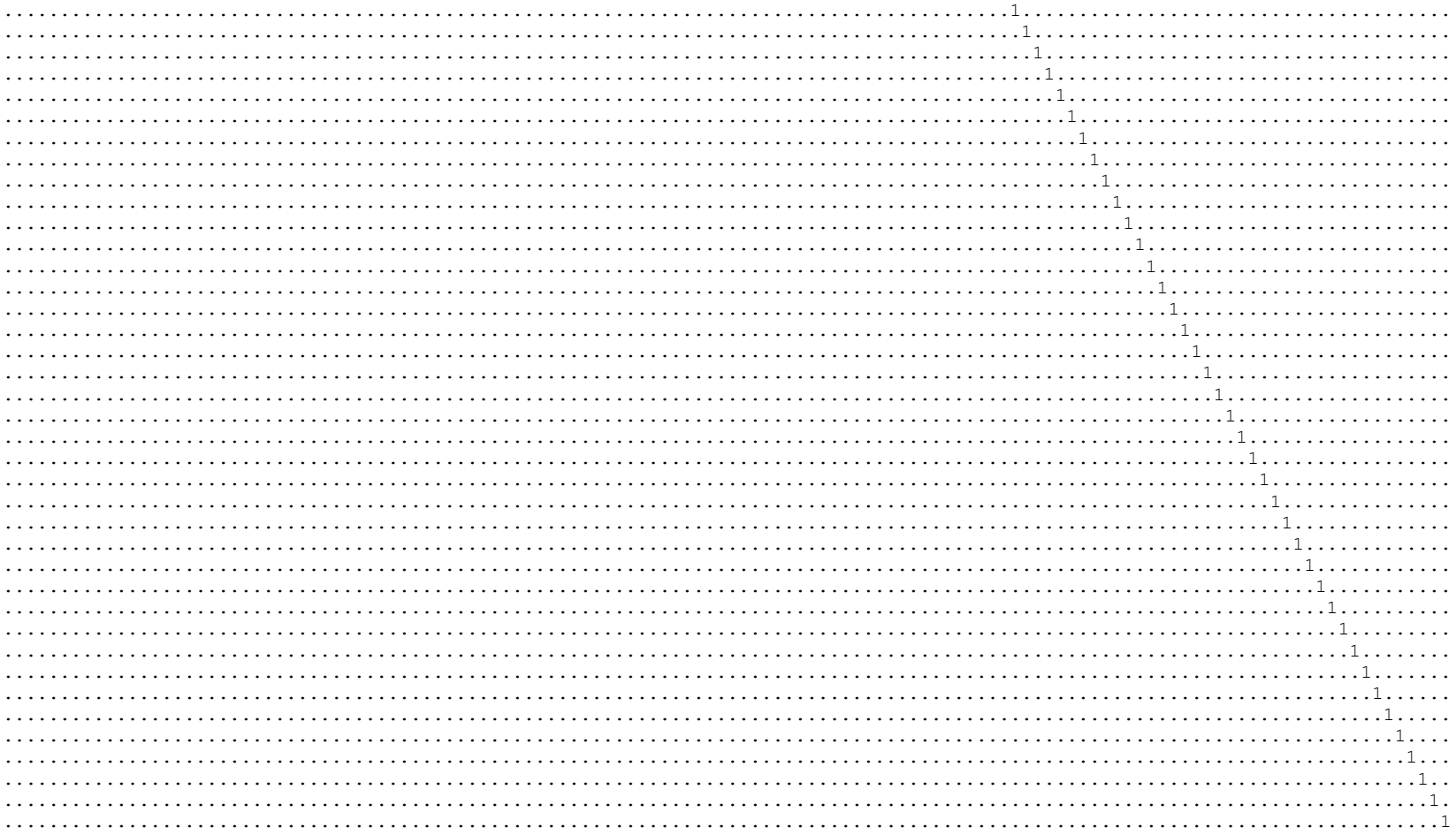
```
            if (c < LIPA[iteracoes]) {
               LIPA[iteracoes]=c;//LIPA[ordem_menos_um] recebe o menor peso hamming de todas as linhas da matriz
            }// fim do if
            if (c > LSPA[iteracoes]) {
               LSPA[iteracoes]=c;//LSPA[ordem_menos_um] recebe o menor peso hamming de todas as linhas da matriz
            }// fim do if
      System.out.println(temp); // exibe a linha i de Avalanche
    }// fim do for i

       temp="Limite Inferior do Poder de Avalanche "+ iteracoes + " = " + LIPA[iteracoes];
       System.out.println(temp); // exibe LIPA[ordem_menos_um]
       temp="Limite Superior do Poder de Avalanche "+ iteracoes + " = " + LSPA[iteracoes];
       System.out.println(temp); // exibe LSPA[ordem_menos_um]
       System.out.println("");
       System.out.println("");
}// fim de for iteracoes

return Avalanche;
}// ******************** fim do metodo avalanche  *********************/

}// fim da classe alpha_2_avalanche
```

12.9.2 MATRIZES DE DEPENDÊNCIAS ESTATÍSTICAS DO CIFRADOR ALPHA

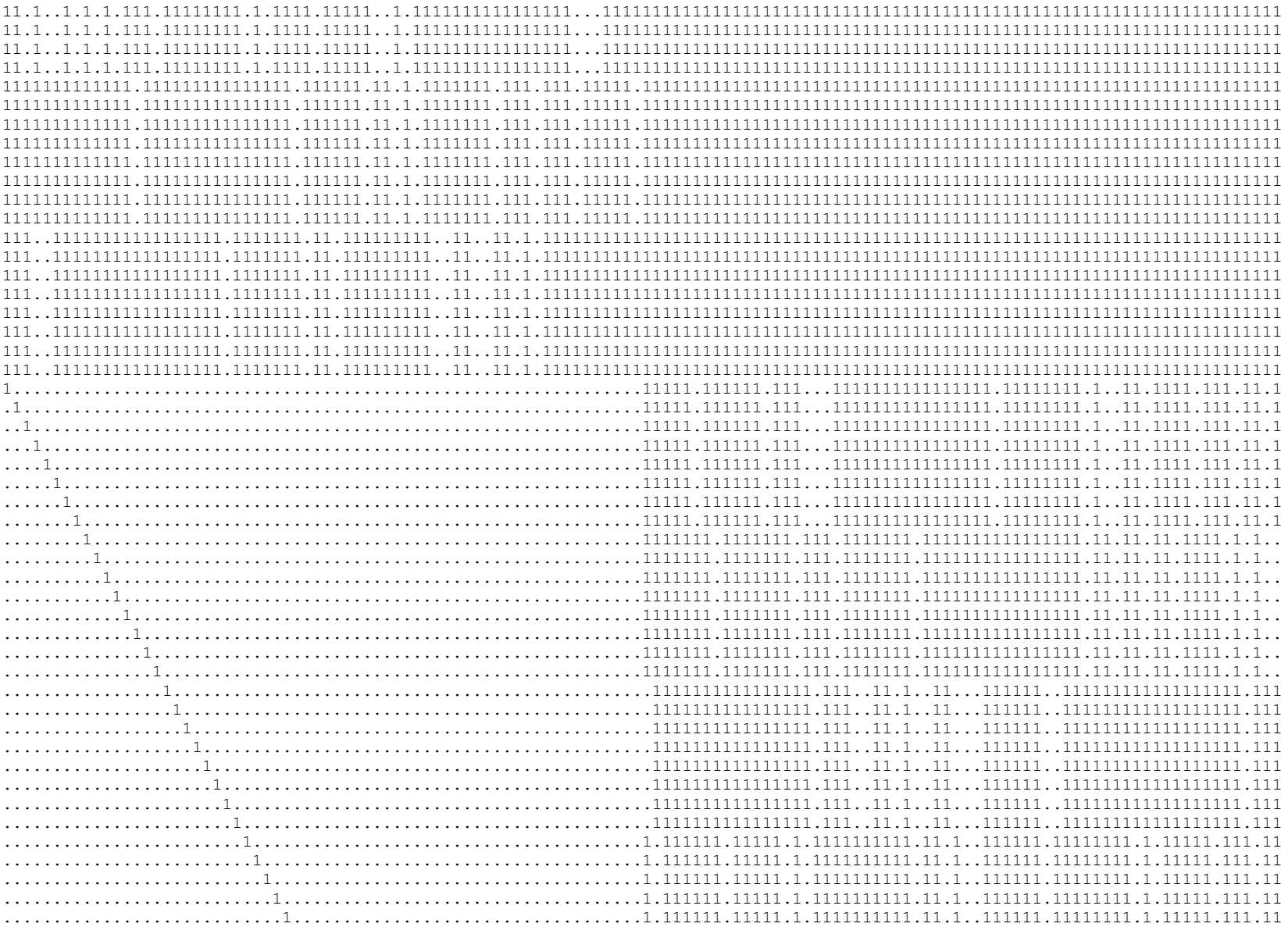
12.9.2.1 MDE (BINÁRIA) DA SAIDA EM RELACAO A TXT CLARO COM 1 ITERACOES (W=2000):

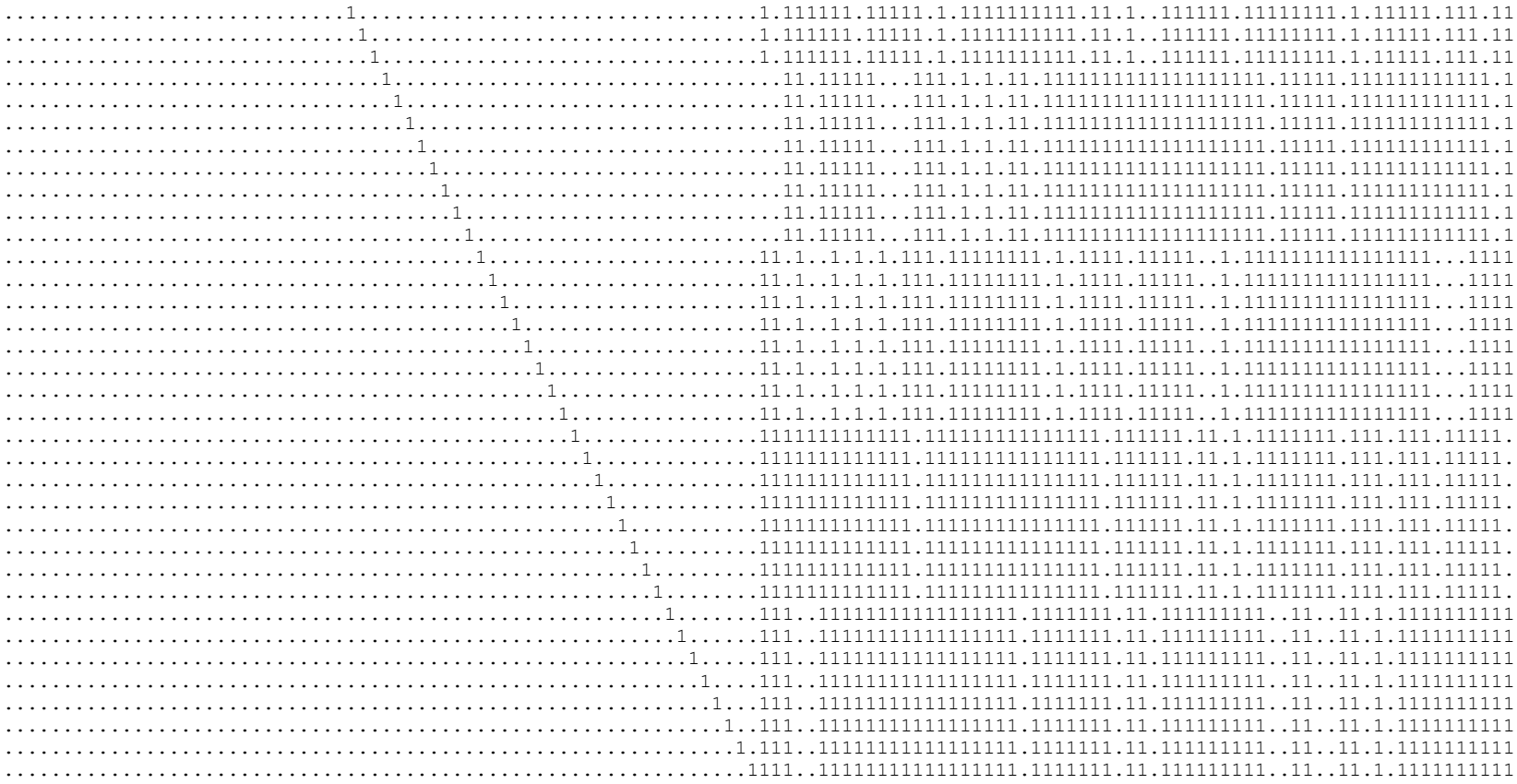





LIPA com 1 iteracoes: 1
LSPA com 1 iteracoes: 56

12.9.2.2 MDE (BINARIA) DA SAIDA EM RELACAO A TXT CLARO COM 2 ITERACOES :

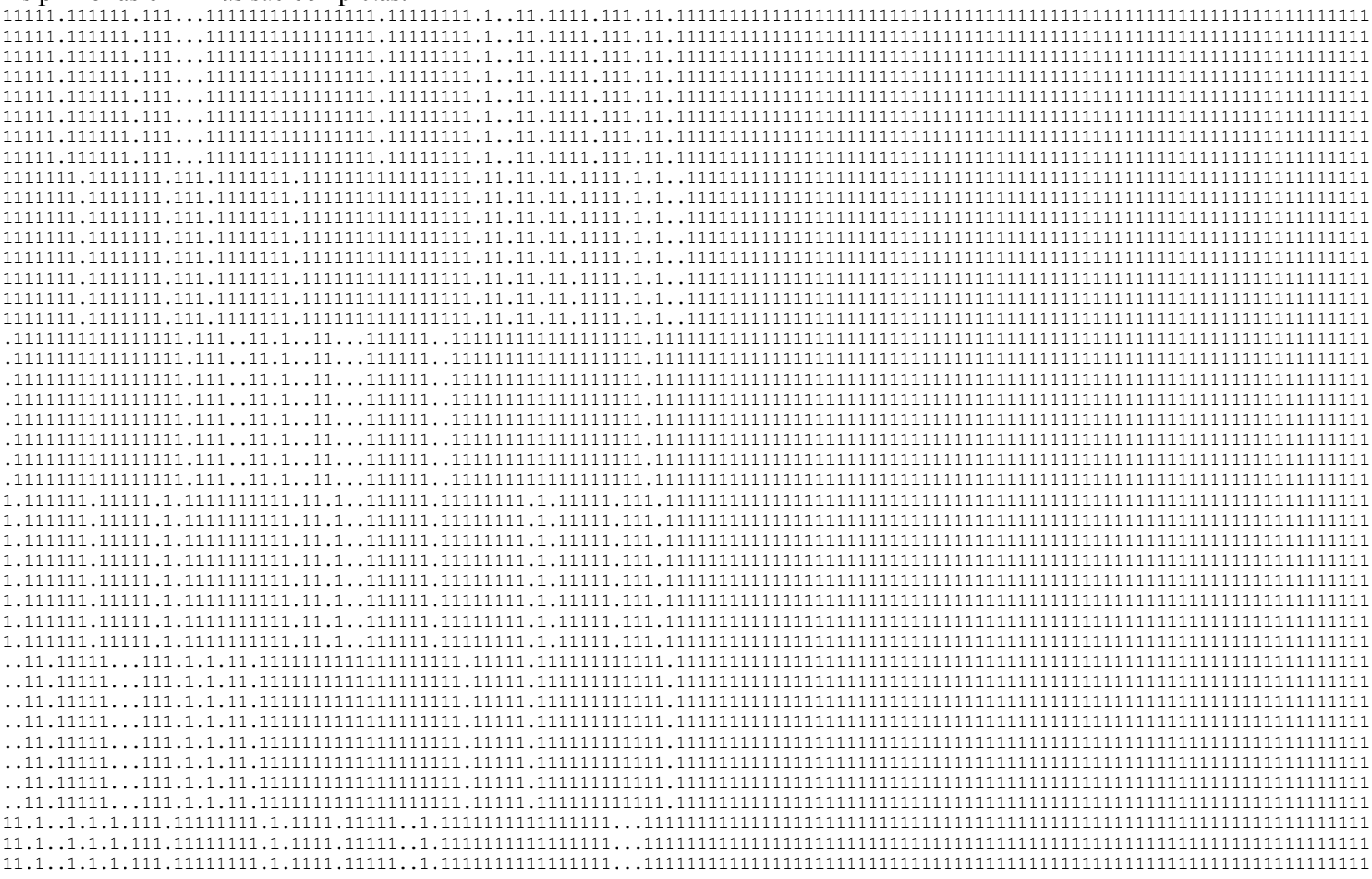




LIPA com 2 iterações: 49
LSPA com 2 iterações: 119

12.9.2.3 MDE (BINARIA) DA SAÍDA EM RELAÇÃO A TEXTO EM CLARO COM 3 ITERAÇÕES :

As primeiras 64 linhas são completas.



```
11.1..1.1.1.111.11111111.1.1111.11111..1.1111111111111111...11111111111111111111111111111111111111111111111111111111111111111111
11.1..1.1.1.111.11111111.1.1111.11111..1.1111111111111111...11111111111111111111111111111111111111111111111111111111111111111111
11.1..1.1.1.111.11111111.1.1111.11111..1.1111111111111111...11111111111111111111111111111111111111111111111111111111111111111111
11.1..1.1.1.111.11111111.1.1111.11111..1.1111111111111111...11111111111111111111111111111111111111111111111111111111111111111111
11.1..1.1.1.111.11111111.1.1111.11111..1.1111111111111111...11111111111111111111111111111111111111111111111111111111111111111111
1111111111111.111111111111111.111111.11.1.1111111.111.111.11111.1111111111111111111111111111111111111111111111111111111111111111
1111111111111.111111111111111.111111.11.1.1111111.111.111.11111.1111111111111111111111111111111111111111111111111111111111111111
1111111111111.111111111111111.111111.11.1.1111111.111.111.11111.1111111111111111111111111111111111111111111111111111111111111111
1111111111111.111111111111111.111111.11.1.1111111.111.111.11111.1111111111111111111111111111111111111111111111111111111111111111
1111111111111.111111111111111.111111.11.1.1111111.111.111.11111.1111111111111111111111111111111111111111111111111111111111111111
1111111111111.111111111111111.111111.11.1.1111111.111.111.11111.1111111111111111111111111111111111111111111111111111111111111111
1111111111111.111111111111111.111111.11.1.1111111.111.111.11111.1111111111111111111111111111111111111111111111111111111111111111
1111111111111.111111111111111.111111.11.1.1111111.111.111.11111.1111111111111111111111111111111111111111111111111111111111111111
111..11111111111111111.1111111.11.111111111..11..11.1.11111111111111111111111111111111111111111111111111111111111111111111111111
111..11111111111111111.1111111.11.111111111..11..11.1.11111111111111111111111111111111111111111111111111111111111111111111111111
111..11111111111111111.1111111.11.111111111..11..11.1.11111111111111111111111111111111111111111111111111111111111111111111111111
111..11111111111111111.1111111.11.111111111..11..11.1.11111111111111111111111111111111111111111111111111111111111111111111111111
111..11111111111111111.1111111.11.111111111..11..11.1.11111111111111111111111111111111111111111111111111111111111111111111111111
111..11111111111111111.1111111.11.111111111..11..11.1.11111111111111111111111111111111111111111111111111111111111111111111111111
111..11111111111111111.1111111.11.111111111..11..11.1.11111111111111111111111111111111111111111111111111111111111111111111111111
111..11111111111111111.1111111.11.111111111..11..11.1.11111111111111111111111111111111111111111111111111111111111111111111111111
LIPA com 3 iterações: 112
LSPA com 3 iterações: 128
```

12.9.2.4 MDE (BINARIA) DA SAÍDA EM RELAÇÃO A TEXTO EM CLARO COM 4 ITERAÇÕES :

A matriz é completa.

12.9.2.5 PERTURBAÇÕES MÍNIMAS, MÉDIAS, E MÁXIMAS POR LINHAS, W = 4000, ATÉ 8 ITERAÇÕES

		PERTURBAÇÕES MINIMAS								PERTURBAÇÕES MÁXIMAS								PERTURBAÇÕES MÉDIAS							
iterações		1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Lin	0	1	12	39	45	41	45	41	44	1	37	78	85	84	82	86	85	1	25,11	56,09	64,09	63,89	64,18	64,12	63,96
Lin	1	1	13	37	44	44	44	43	46	1	38	73	83	84	87	84	85	1	25,13	56,04	64,05	63,93	63,98	63,99	64,08
Lin	2	1	16	41	46	45	45	45	42	1	43	79	82	87	83	84	82	1	29,28	60,33	64,00	63,95	63,99	64,09	64,04
Lin	3	1	15	42	45	46	42	45	44	1	43	82	86	84	85	85	86	1	29,34	60,11	63,91	63,87	63,92	63,93	63,97
Lin	4	1	11	34	40	45	45	44	44	1	31	68	83	85	82	85	84	1	21,17	52,17	64,08	63,98	64,02	64,01	63,99
Lin	5	1	15	37	41	41	44	46	44	1	38	73	85	85	83	84	87	1	25,51	56,58	63,94	63,96	64,08	64,03	64,06
Lin	6	1	21	46	41	43	45	41	42	1	46	85	83	87	84	83	83	1	33,19	64,31	63,93	64,09	63,90	64,03	63,98
Lin	7	1	12	36	42	40	46	46	44	1	39	75	87	84	85	85	85	1	25,46	56,43	64,04	63,81	63,98	64,08	63,90
Lin	8	1	17	39	45	46	44	46	46	1	47	79	84	85	83	81	88	1	29,17	60,07	63,87	63,97	63,90	63,96	64,12
Lin	9	1	15	36	41	44	45	39	45	1	42	79	84	84	84	83	85	1	27,24	58,01	63,85	63,99	63,97	63,97	63,92
Lin	10	1	15	37	42	44	44	43	44	1	42	78	84	83	86	84	83	1	27,30	58,32	63,98	64,02	63,87	63,80	64,03
Lin	11	1	13	37	45	45	45	44	43	1	36	76	84	89	87	87	83	1	25,13	56,13	63,86	64,14	63,86	63,83	64,27
Lin	12	1	14	36	45	42	46	41	44	1	36	73	86	85	85	81	84	1	25,15	56,09	63,84	64,05	63,89	63,88	63,97
Lin	13	1	16	41	45	43	45	44	43	1	41	79	83	84	85	84	83	1	28,94	59,88	64,05	63,87	64,06	64,19	63,97
Lin	14	1	17	34	45	43	46	43	42	1	41	78	83	84	82	83	85	1	28,72	59,71	63,93	64,07	63,99	64,10	64,08
Lin	15	1	14	37	45	44	44	45	42	1	37	74	82	83	85	83	89	1	25,06	56,01	64,08	63,91	63,92	64,01	64,01
Lin	16	1	9	34	45	42	43	41	45	1	33	70	84	83	86	85	85	1	20,99	52,03	64,01	63,89	63,98	63,86	63,85
Lin	17	1	14	40	42	45	46	41	44	1	36	77	83	83	91	87	83	1	25,54	56,55	64,02	63,96	64,14	64,03	63,88
Lin	18	1	14	38	45	45	43	46	43	1	37	77	83	83	85	84	85	1	25,21	56,20	63,89	63,93	63,93	63,96	64,03
Lin	19	1	17	40	42	42	45	44	43	1	41	80	85	84	83	83	85	1	29,34	60,04	63,98	63,95	63,97	64,01	64,00
Lin	20	1	17	40	45	42	44	44	41	1	43	80	90	84	85	89	82	1	28,98	60,03	64,09	64,12	63,98	64,18	63,98
Lin	21	1	15	39	41	44	44	42	44	1	41	79	84	86	93	86	85	1	27,40	58,52	63,98	63,96	63,87	64,06	64,12
Lin	22	1	14	38	43	43	45	44	45	1	36	75	85	84	86	84	84	1	25,04	56,01	64,06	64,08	64,00	64,10	64,07
Lin	23	1	15	40	41	47	40	46	46	1	43	83	83	83	85	82	83	1	29,01	60,07	64,00	64,09	64,02	64,02	63,98
Lin	24	1	17	41	45	44	43	43	44	1	42	79	92	83	85	86	84	1	29,21	60,20	64,07	64,00	63,90	63,95	64,00
Lin	25	1	16	41	45	42	45	45	43	1	42	79	86	93	84	86	82	1	29,44	60,35	63,93	64,06	64,08	63,84	64,11
Lin	26	1	16	43	42	43	44	45	43	1	41	78	83	84	87	84	82	1	29,40	60,41	63,86	63,94	64,05	63,99	63,95
Lin	27	1	10	33	45	42	40	45	44	1	33	69	84	87	82	85	83	1	21,31	52,15	64,16	63,97	63,96	64,14	64,00
Lin	28	1	16	43	44	45	40	45	44	1	44	80	84	84	83	85	84	1	29,28	60,19	63,93	63,70	63,94	63,85	64,07

Lin	29	1	16	40	43	42	43	44	44	*1*	42	80	85	85	84	82	83	*1*	29,07	60,00	64,00	64,10	64,05	64,02	64,04
Lin	30	1	15	37	39	46	42	45	43	*1*	36	74	83	81	83	84	85	*1*	25,17	56,32	63,84	63,87	64,03	64,03	64,01
Lin	31	1	11	37	41	44	44	45	44	*1*	37	74	85	83	83	85	87	*1*	23,32	54,21	63,97	63,93	63,86	63,81	63,85
Lin	32	1	14	37	42	44	38	44	44	*1*	40	72	83	87	84	85	82	*1*	25,04	56,02	64,01	64,09	63,92	64,03	63,97
Lin	33	1	10	33	43	44	45	40	42	*1*	34	72	83	83	83	84	83	*1*	21,47	52,44	63,81	63,93	64,01	63,95	64,01
Lin	34	1	20	45	45	45	45	44	44	*1*	47	83	84	84	84	82	84	*1*	33,09	64,04	64,07	64,03	63,92	63,91	64,10
Lin	35	1	16	41	45	46	43	45	43	*1*	42	80	86	82	86	87	82	*1*	29,63	60,39	63,92	64,07	63,99	63,87	63,99
Lin	36	1	13	36	46	44	46	45	43	*1*	41	82	89	85	82	85	84	*1*	27,20	58,17	63,99	64,12	64,07	64,22	63,85
Lin	37	1	13	41	45	45	40	45	46	*1*	43	80	85	85	87	84	85	*1*	29,61	60,49	63,98	63,99	64,03	63,95	64,12
Lin	38	1	17	42	44	46	44	44	43	*1*	41	77	85	84	85	84	84	*1*	28,91	59,91	63,97	64,05	63,98	64,07	64,06
Lin	39	1	18	43	45	42	42	41	46	*1*	42	83	85	87	84	81	88	*1*	29,24	60,34	64,22	63,99	64,17	64,05	64,04
Lin	40	1	12	35	43	44	45	43	45	*1*	31	74	85	85	83	83	85	*1*	21,28	52,29	63,98	64,03	64,01	63,98	64,19
Lin	41	1	14	38	44	44	44	42	41	*1*	37	75	85	85	84	82	82	*1*	25,30	56,21	64,00	63,90	64,16	64,12	63,96
Lin	42	1	21	44	44	42	45	41	45	*1*	47	84	83	83	86	86	84	*1*	33,03	63,98	64,10	64,01	63,98	64,07	64,04
Lin	43	1	14	37	45	44	43	46	44	*1*	38	74	89	84	84	82	87	*1*	24,95	55,85	64,08	64,11	64,16	63,96	64,01
Lin	44	1	10	32	47	46	46	44	42	*1*	33	70	82	83	82	86	84	*1*	21,04	52,02	63,91	64,02	63,91	63,96	64,13
Lin	45	1	21	45	45	45	46	43	41	*1*	46	88	88	85	83	85	82	*1*	33,53	64,55	64,03	63,85	63,94	64,00	63,92
Lin	46	1	18	40	44	43	45	42	44	*1*	44	79	83	90	85	85	86	*1*	29,59	60,54	64,12	63,93	64,11	64,20	64,10
Lin	47	1	11	35	45	44	46	42	46	*1*	32	69	82	85	83	84	85	*1*	21,15	52,20	64,10	63,99	63,84	64,03	63,79
Lin	48	1	19	39	43	44	43	42	45	*1*	43	81	88	82	86	86	84	*1*	29,38	60,57	64,09	63,91	63,99	64,01	64,12
Lin	49	1	11	33	45	42	41	44	43	*1*	32	69	83	84	84	87	84	*1*	21,22	52,37	63,98	63,92	64,06	63,95	64,12
Lin	50	1	11	35	45	46	44	42	44	*1*	36	77	84	85	86	84	84	*1*	23,12	54,25	63,96	64,07	63,86	64,10	63,81
Lin	51	1	11	35	46	46	44	45	43	*1*	36	73	83	84	84	85	83	*1*	22,84	53,94	64,08	63,96	63,94	64,09	63,92
Lin	52	1	21	47	45	46	46	46	44	*1*	49	82	86	85	85	83	85	*1*	32,99	63,97	64,09	63,84	63,93	63,97	64,07
Lin	53	1	11	36	43	44	45	44	45	*1*	32	70	85	87	86	85	85	*1*	21,31	52,24	64,07	63,77	63,97	64,06	64,03
Lin	54	1	20	45	44	43	46	41	44	*1*	48	85	85	84	82	85	83	*1*	33,40	64,43	63,80	63,90	63,92	64,11	64,00
Lin	55	1	17	41	39	43	45	45	45	*1*	41	82	86	85	84	83	84	*1*	29,09	60,24	63,93	64,04	64,12	64,01	64,13
Lin	56	1	21	45	42	44	41	45	42	*1*	49	82	84	85	87	84	82	*1*	33,47	64,32	63,82	63,85	64,04	64,11	63,98
Lin	57	1	8	31	40	45	45	44	46	*1*	31	67	84	85	83	84	83	*1*	19,07	50,00	64,04	63,96	63,99	63,95	63,97
Lin	58	1	14	39	45	44	42	43	45	*1*	37	75	84	86	83	85	84	*1*	25,29	56,10	63,95	63,99	64,00	63,86	64,05
Lin	59	1	14	37	40	44	46	41	41	*1*	37	72	89	87	82	83	85	*1*	25,15	56,15	63,99	64,03	63,90	64,17	64,13
Lin	60	1	14	40	45	44	42	43	46	*1*	37	80	85	84	84	85	84	*1*	25,37	56,31	64,02	63,90	64,16	63,93	64,08
Lin	61	1	16	40	42	43	39	43	46	*1*	43	82	81	85	86	81	83	*1*	29,40	60,36	63,93	64,09	64,13	63,94	63,94
Lin	62	1	11	34	43	41	45	43	43	*1*	33	70	87	87	83	83	84	*1*	21,16	52,16	64,01	63,96	64,05	63,97	64,06
Lin	63	1	13	38	43	44	39	44	46	*1*	38	76	83	82	84	89	82	*1*	25,07	56,06	64,09	64,03	64,07	63,94	64,03
Lin	64	12	37	41	44	44	47	44	44	*37*	75	84	85	86	85	85	88	*25,20*	56,27	63,92	63,96	64,20	64,04	63,90	63,75

Lin	65	12	39	47	45	44	45	45	44	*37*	75	86	83	84	86	86	84	*25,15*	55,98	64,11	63,99	63,88	64,02	63,93	64,10
Lin	66	15	42	38	46	45	40	45	45	*42*	80	85	83	86	87	84	83	*29,26*	60,23	63,81	64,08	64,06	64,07	63,99	63,93
Lin	67	18	40	44	43	45	46	46	45	*42*	84	88	86	83	85	83	84	*29,26*	60,40	64,08	63,91	63,93	64,14	64,14	63,96
Lin	68	9	36	40	42	43	44	43	44	*32*	68	83	86	83	84	83	86	*21,21*	52,30	63,84	64,04	63,96	64,10	64,07	64,00
Lin	69	12	39	43	45	45	45	39	44	*39*	75	85	86	87	84	81	83	*25,47*	56,51	64,06	64,02	64,13	63,82	64,04	63,85
Lin	70	20	47	45	42	44	44	46	45	*47*	85	85	84	87	84	85	88	*33,19*	64,13	64,01	64,08	64,06	63,95	64,10	63,89
Lin	71	13	38	41	44	42	42	41	43	*38*	75	86	84	82	85	85	85	*25,38*	56,26	64,03	63,78	63,96	63,98	64,07	63,97
Lin	72	13	41	45	43	46	44	43	44	*42*	78	83	92	84	85	84	82	*29,16*	59,96	63,94	64,00	63,98	63,93	64,09	63,90
Lin	73	16	37	45	45	43	44	40	41	*41*	76	81	83	84	85	85	83	*27,08*	57,97	64,08	63,97	63,82	63,95	64,21	63,95
Lin	74	14	40	45	44	44	43	43	46	*44*	77	82	88	83	83	85	84	*27,38*	58,26	63,92	63,85	64,05	64,04	64,01	63,99
Lin	75	12	37	45	46	46	45	43	44	*38*	75	84	83	84	90	85	84	*25,15*	56,17	64,03	63,95	64,06	64,00	64,12	63,99
Lin	76	15	39	43	47	45	44	45	42	*39*	74	87	90	83	87	82	84	*25,02*	56,00	64,10	63,86	63,86	63,85	63,86	64,21
Lin	77	18	39	45	44	41	45	45	42	*41*	82	84	83	83	89	87	86	*29,00*	60,04	63,99	64,08	64,04	63,95	64,05	64,05
Lin	78	15	41	46	42	45	44	44	42	*42*	79	89	86	88	83	85	85	*28,80*	59,87	63,99	63,81	63,94	64,06	64,00	64,02
Lin	79	13	33	46	43	45	45	46	44	*37*	72	86	83	88	86	88	84	*25,07*	56,00	63,95	63,92	64,03	64,04	63,96	64,14
Lin	80	10	34	45	43	44	44	47	43	*32*	71	82	84	84	80	86	84	*20,87*	51,87	63,87	64,09	64,12	63,76	64,10	64,07
Lin	81	14	41	44	45	44	42	46	45	*37*	80	86	84	85	86	86	84	*25,56*	56,49	63,94	64,04	63,99	63,99	63,99	63,97
Lin	82	12	37	41	41	43	45	45	46	*37*	74	88	83	84	83	85	85	*25,20*	56,16	63,93	64,09	63,92	64,16	63,95	63,95
Lin	83	17	42	46	45	43	41	44	44	*42*	77	84	83	84	83	85	87	*29,22*	60,23	63,86	64,03	64,04	64,03	63,96	64,12
Lin	84	16	36	45	44	45	43	44	43	*43*	79	84	83	88	83	84	83	*28,98*	59,82	64,05	63,94	63,94	64,09	63,96	63,92
Lin	85	14	38	44	43	43	42	43	42	*41*	81	82	86	83	83	86	84	*27,36*	58,41	63,93	64,06	64,01	64,01	63,93	64,04
Lin	86	13	38	44	46	44	44	43	45	*37*	75	83	83	84	86	83	85	*25,01*	55,96	63,90	64,00	63,93	64,11	64,01	63,97
Lin	87	16	39	44	45	46	46	44	44	*41*	81	84	85	85	85	87	84	*29,07*	60,22	63,83	64,14	63,94	64,09	63,92	64,08
Lin	88	18	41	45	44	45	47	42	45	*42*	80	82	84	84	87	83	85	*29,30*	60,23	64,01	64,09	64,11	63,88	63,91	64,03
Lin	89	17	42	44	41	46	44	43	44	*41*	81	82	85	83	85	88	85	*29,36*	60,29	64,08	64,01	64,04	63,93	63,92	64,13
Lin	90	19	39	42	45	41	45	44	44	*42*	78	85	87	87	81	87	85	*29,32*	60,27	64,00	63,95	63,86	64,03	64,07	63,92
Lin	91	11	32	42	41	43	45	44	46	*32*	70	84	85	88	85	87	83	*21,33*	52,19	63,97	63,88	63,87	63,92	63,99	63,99
Lin	92	17	43	44	43	43	42	44	44	*42*	83	86	85	85	82	85	86	*29,19*	60,35	63,99	63,94	63,99	63,89	64,17	64,04
Lin	93	17	39	42	46	45	41	44	42	*41*	77	84	85	82	88	85	82	*29,04*	60,06	64,00	64,02	63,95	64,06	63,97	64,01
Lin	94	15	39	46	47	44	43	46	42	*36*	74	84	89	88	81	88	83	*25,22*	56,36	63,99	64,15	63,86	63,99	64,16	63,89
Lin	95	10	35	42	43	42	42	44	44	*36*	74	84	86	88	85	86	85	*23,37*	54,34	64,16	64,05	64,04	63,96	63,98	64,08
Lin	96	15	37	46	43	44	45	40	44	*39*	73	87	84	84	86	81	84	*25,06*	55,95	64,01	64,12	64,20	63,85	63,87	64,09
Lin	97	11	33	44	44	44	45	46	45	*32*	71	83	83	85	85	84	89	*21,34*	52,42	63,95	63,98	64,03	63,74	63,88	64,12
Lin	98	20	43	42	42	43	42	43	41	*48*	84	86	85	82	85	82	83	*33,16*	64,25	64,01	64,07	64,05	63,91	63,97	64,04
Lin	99	18	42	46	43	42	43	46	42	*41*	81	89	88	85	83	83	85	*29,54*	60,53	63,93	64,11	64,00	63,95	63,97	63,95
Lin	100	15	39	40	42	42	45	43	44	*43*	80	84	82	83	85	83	84	*27,24*	58,16	64,04	63,97	64,02	63,96	64,07	63,87

Lin	101	16	41	44	44	46	42	47	45	*42*	80	84	84	82	86	82	84	*29,55*	60,44	63,96	64,13	63,97	64,09	63,92	64,06
Lin	102	15	41	44	45	45	44	43	44	*41*	78	84	84	86	81	82	85	*28,74*	59,87	64,09	64,02	64,01	63,94	64,11	63,94
Lin	103	18	42	44	42	42	45	42	45	*43*	80	87	82	84	84	82	85	*29,31*	60,28	64,17	63,88	64,04	63,90	64,05	63,98
Lin	104	10	35	43	37	43	43	43	44	*33*	69	84	83	84	83	82	85	*21,28*	52,36	64,01	64,13	64,05	64,08	63,93	63,96
Lin	105	13	39	43	43	46	42	46	45	*39*	73	84	85	83	83	85	84	*25,36*	56,53	64,07	63,99	64,09	63,92	64,12	63,90
Lin	106	19	45	45	44	43	41	42	44	*47*	83	88	87	88	86	85	82	*32,99*	64,14	64,06	63,94	63,88	63,99	63,85	64,10
Lin	107	12	39	40	46	41	46	45	45	*37*	75	84	86	85	84	82	84	*24,84*	55,96	64,01	63,96	63,91	63,99	64,12	64,02
Lin	108	10	36	44	45	45	46	40	47	*32*	75	84	82	82	85	83	83	*20,99*	51,98	64,19	63,86	63,97	63,92	63,98	63,94
Lin	109	20	45	45	45	46	44	43	44	*47*	84	84	84	85	83	84	85	*33,53*	64,36	63,94	64,07	64,11	64,15	63,91	63,96
Lin	110	18	43	46	44	41	46	47	44	*43*	78	85	84	81	83	83	84	*29,62*	60,76	64,01	64,08	64,00	63,99	64,01	63,97
Lin	111	10	37	44	43	44	41	43	41	*33*	68	86	87	82	84	85	85	*21,14*	52,10	63,99	64,01	63,96	63,93	64,16	63,99
Lin	112	17	40	43	45	43	45	45	46	*45*	77	88	82	84	86	85	84	*29,47*	60,38	63,99	64,08	64,06	64,05	64,08	64,10
Lin	113	11	32	45	44	45	45	45	43	*32*	70	84	86	83	86	83	84	*21,19*	52,14	64,00	64,03	63,97	64,09	64,04	63,99
Lin	114	11	31	42	43	43	44	44	40	*40*	72	83	84	84	84	84	88	*23,19*	54,18	64,03	63,93	63,98	63,95	64,05	64,04
Lin	115	12	35	42	44	44	41	40	45	*39*	72	88	87	83	83	82	85	*23,00*	53,99	63,93	64,07	64,15	64,05	64,06	64,06
Lin	116	19	46	43	40	45	46	45	41	*46*	83	85	86	85	83	83	84	*32,96*	64,00	63,89	64,05	63,98	63,93	63,98	64,06
Lin	117	11	36	43	44	45	47	44	46	*31*	72	84	83	83	88	86	85	*21,27*	52,24	64,01	64,00	63,84	63,97	64,03	64,08
Lin	118	20	48	46	44	44	40	46	42	*47*	86	82	84	84	87	83	83	*33,37*	64,49	64,14	64,08	63,87	64,10	63,94	63,94
Lin	119	17	40	44	46	45	44	44	43	*44*	81	82	83	83	84	84	86	*29,12*	60,26	64,14	63,94	63,90	63,96	63,89	63,96
Lin	120	20	45	45	43	41	43	46	44	*49*	82	84	85	87	87	84	83	*33,36*	64,33	64,05	64,05	63,87	63,94	64,04	63,95
Lin	121	8	28	41	45	43	44	44	45	*32*	67	84	83	86	84	88	84	*19,21*	49,98	63,97	64,01	64,04	64,05	64,03	64,08
Lin	122	14	38	44	45	42	42	44	44	*37*	75	83	83	84	86	84	87	*25,16*	56,16	64,05	63,80	64,01	64,16	63,91	63,97
Lin	123	13	39	43	45	41	45	42	45	*37*	73	86	86	89	82	85	84	*25,11*	56,21	63,93	64,18	64,03	64,02	64,02	63,92
Lin	124	13	37	46	46	44	43	44	43	*38*	76	84	86	85	85	82	87	*25,36*	56,28	63,98	63,98	63,96	63,99	64,04	64,04
Lin	125	17	42	41	43	40	44	45	44	*41*	81	82	86	82	83	83	82	*29,11*	60,25	63,82	64,09	63,93	63,92	63,92	63,84
Lin	126	10	35	46	42	44	42	45	45	*33*	71	87	87	84	82	86	84	*21,24*	52,11	64,01	64,08	64,01	64,03	63,99	63,83
Lin	127	14	38	46	42	45	45	44	46	*37*	77	85	82	83	84	83	83	*25,11*	56,12	64,05	63,97	63,90	64,06	63,92	63,95

12.9.2.6 CÓDIGO FONTE (JAVA) UTILIZADO PARA GERAR OS ITENS 12.9.2.1 A 12.9.2.5

Foram omitidos os métodos que constam no apêndice 8.

```
public class  kit_testes_1_Alpha_vs_2 {
public static void main(String[] args) {

/************************ comentarios iniciais *************************
 * Este codigo foi criado para construir testes de MDEs da versao 2 do  *
 * cifrador Alpha com vetor de teste (mesmo do FIPS 197) analisá-las;   *
 * realizar testes de perturbação;                                      *
 * Autor: Jorge de A. Lambert                                           *
 * Jan/04                                                               *
 ************************************************************************/
 // As MDE são calculadas em relacao ao texto em claro.
 // Para calcular em relação a chave, basta trocar "txt claro" por "chave"
 // nas linhas marcadas com "#####"
 // Para calcular a matriz de dependencias da TL, substituir a TNL do
cifrador
 // pela identidade, usar w qualquer e Nr=1
 // Para usar o metodo GeraMDE em cifradores de blocos de tamanho
diferente de
 // 128 é necessario adaptar o metodo gerabloco e substituir os metodos
 // que sao particulares do cifrador em análise
/*
int chave[]= {0,0,1,0, 1,0,1,1, 0,1,1,1, 1,1,1,0, 0,0,0,1, 0,1,0,1,
0,0,0,1,
              0,1,1,0, 0,0,1,0, 1,0,0,0, 1,0,1,0, 1,1,1,0, 1,1,0,1,
0,0,1,0,
              1,0,1,0, 0,1,1,0, 1,0,1,0, 1,0,1,1, 1,1,1,1, 0,1,1,1,
0,0,0,1,
              0,1,0,1, 1,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,1, 1,1,0,0,
1,1,1,1,
              0,1,0,0, 1,1,1,1, 0,0,1,1, 1,1,0,0};//exemplo apendice A e B
do FIPS
*/
int chave[]= {0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,
0,0,0,0,
              0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,
0,0,0,0,
              0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,
0,0,0,0,
              0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,
0,0,0,0,
              0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0};//exemplo apendice A e B
do FIPS



int Nr=10;//numero de iterações
int n=128;
int w=4000;//numero de testes de edpendencia
int MDE [][][] = GeraMDE (128,chave,Nr, w, 1,1);
int lixo = estatistica_MDE(128, Nr+1, MDE, w);
double [][][] pert = perturbacao (128,chave,Nr+1, w);
```

```
System.out.println("PERTURBAÇÕES");

for (int i=0;i<n ;i ++){//exibe perturbacoes
    System.out. print("Lin, "+i+", Pert , ");
    for (int j=0;j<3;j++){
         System.out. print("    ");
         for (int r=1; r<Nr+1; r++){
            System.out. print(pert[r] [j] [i] + ",");
            }
         }
     System.out. println("");
    }

}// fim de main

/*************************************************************
 **********                    metodo GeraMDE               ******
 **********  argumento1 = tamanho do bloco                  ******
 **********  argumento2 = 1 vetor fixo (chave)              ******
 **********  argumento3 = numero de iteracoes maximo        ******
 **********  argumento4 = numero de testes de dependencias ******
 **********  argumento5 = 1 imprime MDE, 0 nao imprime      ******
 **********  argumento6 = 1 imprime MDE binaria, 0 nao imp.******
 **********  retorna =  MDE [Nr][n][n]                      ******
 *************************************************************/
public static int [][][] GeraMDE (int n,int chave [], int Nr, int wmax,
int imprimeMDE, int imprime_binaria){//
int txtclaro [] = new int [n];
int cripto1 [] = new int [n];
int cripto2 [] = new int [n];
int afetados [] = new int [n];
int PA [] = new int [n];
int LIPA = 0;
int LSPA = 0;
int MDE [][][] = new int [Nr+1][n][n];
for (int r=1; r<Nr+1; r++){// ate a MDE com 8 iteracoes
        for (int afeta=0;afeta<n;afeta ++){
            for (int w=0;w<wmax;w++){
                txtclaro = gerabloco(0);
                cripto1 =
alpha(txtclaro,chave,1,r);//######################
                txtclaro[afeta]=(txtclaro[afeta]+1)%2;//complementa um
bite
                cripto2 =
alpha(txtclaro,chave,1,r);//######################
                afetados = diferenca(n, cripto1, cripto2);
                for (int i=0; i<n; i++){//
                    MDE[r][i][afeta]=(MDE[r][i][afeta]+afetados[i]);
                    }//i
                }//w
            }//afeta
        if(imprimeMDE==1){
        System.out.println("MDE com relacao a TEXTO CLARO com "+r+"
iteracoes e w= "+wmax+": ");//#######
            for (int i=0;i<n;i++){
                for (int j=0;j<n;j++){
                    System.out.print(MDE[r][i][j]+", ");
                }
                System.out.println("");
            }// for i
```

```
        }// if
        if(imprime_binaria==1){
            System.out.println("MDE (binaria) da saida em relacao a TEXTO
CLARO  com "+r+" iteracoes : ");//########
            LSPA=0;
            LIPA=10000000;
            for (int i=0;i<n;i++){
                PA[i]=0;
                for (int j=0;j<n;j++){
                    if (MDE[r][i][j]>0){
                        System.out.print("1");
                        PA[i]=PA[i]+1;
                        }//if
                    else{
                        System.out.print(".");
                        }//else
                    }//j
                System.out.println("");
                if (PA[i]>LSPA){
                    LSPA=PA[i];
                }//if
                if (PA[i]<LIPA){
                    LIPA=PA[i];
                }//if
                }//i
                System.out.println("");
                System.out.println("LIPA com "+r+" iteracoes: "+LIPA);
                System.out.println("LSPA com "+r+" iteracoes: "+LSPA);
            }// if  imprime binaria
    }//r
return MDE;
}// ******************* fim do metodo Gera MDE  *********************

/*************************************************************
 **********             metodo perturbação                 ******
 **********  argumento1 = tamanho do bloco                 ******
 **********  argumento2 = 1 vetor fixo (chave)             ******
 **********  argumento3 = numero de iteracoes maximo       ******
 **********  argumento4 = numero de testes de dependencias ******
 **********  retorna =  double PERT [Nr+1][3][n]           ******
 *************************************************************/
public static double [][][] perturbacao (int n,int chave [], int
Nr_mais_um, int wmax){//
int txtclaro [] = new int [n];
int cripto1 [] = new int [n];
int cripto2 [] = new int [n];
int afetados [] = new int [n];
double pert [][][] = new double [Nr_mais_um][3][n];
double acumulador=0;
for (int r=1; r<Nr_mais_um; r++){// ate a MDE com 8 iteracoes
        for (int afeta=0;afeta<n;afeta ++){
            acumulador=0;
            pert[r][0][afeta]=10000;//armazenara a pert minima
            pert[r][1][afeta]=0;//armazenara a pert maxima
            for (int w=0;w<wmax;w++){
                pert[r][2][afeta]=0;//armazenara a pert media (de todas as
wmax calculadas)
                txtclaro = gerabloco(0);
                cripto1 =
alpha(txtclaro,chave,1,r);//####################
```

```
                txtclaro[afeta]=(txtclaro[afeta]+1)%2;//complementa um
bite
                cripto2 =
alpha(txtclaro,chave,1,r);//#####################
                afetados = diferenca(n, cripto1, cripto2);
                for (int i=0; i<n; i++){//
                    pert [r][2][afeta]=(pert [r][2][afeta]+afetados[i]);
                    }//i
                acumulador = acumulador + pert [r][2][afeta];
                if (pert [r][2][afeta]<pert [r][0][afeta]) pert
[r][0][afeta]=pert [r][2][afeta];
                if (pert [r][2][afeta]>pert [r][1][afeta]) pert
[r][1][afeta]=pert [r][2][afeta];
                }//w
                pert[r][2][afeta]=acumulador/wmax;
            }//afeta
    }//r
return pert;
}// **************** fim do metodo perturbação **************

}// fim da classe  kit_testes_1_Alpha_vs_2
```

12.9.3 TESTES DE ALEATORIEDADE DO NIST

12.9.3.1 CÓDIGO FONTE (EM JAVA)

Alguns métodos no código foram omitidos por se encontrarem no apêndice 8, em 12.8.3.

```
public class  testes_nist_alpha_vs2 {
public static void main(String[] args) {

/******************** comentarios iniciais ***************************
* Este codigo foi criado para realizar no cifrador Alpha_vs2 os testes *
* previstos pelo NIST (FIPS 140-1) – (MENEZES, 1996)                   *
* Jan/04                                                               *
************************************************************************/

int chave[]= {0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,
0,0,0,0,
              0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,
0,0,0,0,
              0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,
0,0,0,0,
              0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,
0,0,0,0,
              0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0};

int texto_claro[] = {0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,
0,0,0,0,
              0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,
0,0,0,0,
              0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,
0,0,0,0,
              0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,
0,0,0,0,
              0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0};
int Nr=4;//numero de iterações
int n=128;
int auxiliar = 0;
int cifrado[]= new int [128];
int bloco_teste [] = new int [20096];//20096=157*128. serao cifrados 157
blocos para gerar os 20000 bites
for (int bit_um=0; bit_um<128;bit_um ++){
    texto_claro[bit_um]=1;
    cifrado = alpha(texto_claro, chave, 1, Nr);
    for (int i=0;i<128;i++){
        bloco_teste[(bit_um*128) + i] = cifrado[i];
    }
    texto_claro [bit_um] = 0;
}
//neste ponto ja foram, gerados 128*128 = 16384  bites
chave[0] = 1;
for (int bit_um=0; bit_um<29;bit_um ++){
    texto_claro[bit_um]=1;
    cifrado = alpha(texto_claro, chave, 1, Nr);
    for (int i=0;i<128;i++){
        auxiliar=16384+(bit_um*128) + i;
        bloco_teste[auxiliar] = cifrado[i];
    }
    texto_claro [bit_um] = 0;
```

```
}
//exibe bloco

System.out.println("SEQUENCIA DE BITES GERADA PELO ALPHA VS2 COM "+Nr+"
ITERACOES");
System.out.println("PARA TESTES DO NIST [MENEZES_96, 183]");
for (int i=0;i<20000;i++){
    System.out.print(bloco_teste[i]);
    if ((i+1)%128==0) System.out.println( "");
}
System.out.println(" ");
System.out.println("
_____________________________________________________________");
System.out.println("
_____________________________________________________________");
System.out.println("Teste Monobit (9654 < aprovado < 10346)");
System.out.println("
_____________________________________________________________");
int n1 = monobit(bloco_teste);
System.out.print("RESULTADO:n1="+n1);
if ((n1<10346)&&(n1>9654)){
    System.out.println(" APROVADO!");
    }
else {
    System.out.println(" REPROVADO!");
    }
System.out.println("
_____________________________________________________________");
System.out.println("
_____________________________________________________________");
System.out.println("Teste Poker (1.03 < aprovado < 57.4)");
System.out.println("
_____________________________________________________________");
double x3 = poker(bloco_teste);
System.out.print("RESULTADO:X3="+x3);
if ((x3<57.4)&&(x3>1.03)){
    System.out.println(" APROVADO!");
    }
else {
    System.out.println(" REPROVADO!");
    }
System.out.println("
_____________________________________________________________");
System.out.println("
_____________________________________________________________");
System.out.println("Teste de Sequencias de 0");
System.out.println("
_____________________________________________________________");
int [] intervalos = {2267, 2733,1079,1421,502,748,223,402,90,223,90,223};
int z[] = sequencias(bloco_teste,0);
for (int i=1;i<7;i++){
    System.out.print("Tamanho "+i+": "+z[i]+ " ocorrencias. Valores
permitidos:"+
    intervalos[2*(i-1)]+" a "+intervalos[2*(i-1)+1]+". RESULTADO:");
if ((z[i]<intervalos[2*(i-1)+1])&&(z[i]>intervalos[2*(i-1)])){
    System.out.println(" APROVADO!");
    }
else {
    System.out.println(" REPROVADO!");
    }
}
```

```
System.out.println("
_____________________________________________________________");
System.out.println("Teste de Sequencias de 1");
System.out.println("
_____________________________________________________________");
int u[] = sequencias(bloco_teste,1);
for (int i=1;i<7;i++){
    System.out.print("Tamanho "+i+": "+z[i]+ " ocorrencias. Valores
permitidos:"+
           intervalos[2*(i-1)]+ " a "+intervalos[2*(i-1)+1]+".
RESULTADO:");
if ((z[i]<intervalos[2*(i-1)+1])&&(z[i]>intervalos[2*(i-1)])){
    System.out.println(" APROVADO!");
    }
else {
    System.out.println(" REPROVADO!");
    }
}
System.out.println("
_____________________________________________________________");
System.out.println("
_____________________________________________________________");
System.out.println("Maior Sequencia de 0");
System.out.println("
_____________________________________________________________");
int maior = maior_sequencia(bloco_teste,0);
    System.out.print("Valores permitidos: até 33. RESULTADO: "+maior);
if (maior<34){
    System.out.println(" APROVADO!");
    }
else {
    System.out.println(" REPROVADO!");
    }
System.out.println("
_____________________________________________________________");
System.out.println("
_____________________________________________________________");
System.out.println("Maior Sequencia de 1");
System.out.println("
_____________________________________________________________");
maior = maior_sequencia(bloco_teste,1);
    System.out.print("Valores permitidos: até 33. RESULTADO: "+maior);
if (maior<34){
    System.out.println(" APROVADO!");
    }
else {
    System.out.println(" REPROVADO!");
    }
System.out.println("
_____________________________________________________________");



}// fim de main

/*************************************************************
 *****                  metodo monobit -                  ******
 *****            argumento = int[128] binario            ******
 *****                   retorna = INT                    ******
 *************************************************************/
```

```
public static int monobit(int vetor[]){
int contador=0;
for (int i=0;i<20000;i++){
     if (vetor[i]==1) contador ++;
}
return contador;
}// ****************** fim do metodo MONOBIT  ***********************/

/*********************************************************
 *****                   metodo POKER                  ******
 *****             argumento = int[128] binario        ******
 *****                  retorna = double               ******
 *********************************************************/
public static double poker(int vetor[]){
double x3=0.0;
//obs: n=20000, m=4, k=20000/4=5000,
int inicio=0;
int i=0;
double somatorio=0;
int [] ni = new int[16];
    x3=16.0/5000.0;
for (int j=0;j<5000;j++){
    inicio=4*j;
    i = vetor[inicio]*8 + vetor[inicio+1]*4
+vetor[inicio+2]*2+vetor[inicio+3];
    ni[i]=ni[i]+1;
    }
for (i=0;i<16;i++){
    somatorio=somatorio+(ni[i]*ni[i]);
    }
    x3 = (x3*somatorio)-5000;
return x3;
}//*************         fim do metodo poker        **************


/*********************************************************
 *****               metodo SEQUENCIAS                ******
 *****             argumento = int[128] binario       ******
 *********************************************************/
public static int [] sequencias(int vetor[], int bit){
int [] uz = new int[7];
int posicao=0;
int tamanho=0;
while (posicao<20000){
    while (vetor[posicao]==bit){
            tamanho=tamanho+1;
            posicao=posicao+1;
        }//fim do while

    if (tamanho<6){
        uz[tamanho]=uz[tamanho]+1;
        }
    else{
        uz[6]=uz[6]+1;
        }
    tamanho=0;
    posicao=posicao+1;
    }
return uz;
}//*************         fim do metodo SEQUENCIAS        **************
```

```
/*********************************************************
 *****           metodo MAIOR SEQUENCIA              ******
 *****            argumento = int[128] binario       ******
 *********************************************************/
public static int maior_sequencia(int vetor[], int bit){
int posicao=0;
int tamanho=0;
int max=0;
int bit_barra=(bit+1)%2;
while (posicao<20000-max){
    while (vetor[posicao]==bit){
            tamanho=tamanho+1;
            posicao=posicao+1;
        }//fim do while
    if (tamanho>max){
        max=tamanho;
        }
    tamanho=0;
    posicao=posicao+1;
    }
return max;
}//*************   fim do metodo MAIOR SEQUENCIAS      *************


 /*********************************************************************
 *****                     metodo alpha     -      cifra           ******
*********************************************************************/
 VIDE APENDICE 8
}// ****************** fim do metodo alpha  ************************


/*********************************************************************
 *****            metodo expansão da chave - (EXPAND KEY)          ******
 *********************************************************************/
VIDE APENDICE 8
}// ************** fim do metodo expand key  ***********************


/*********************************************************************
 *****         metodo N_alpha = (BYTESUB-rijndael)
 *********************************************************************/
VIDE APENDICE 8
}// ******************fim do metodo SUBSTITUICAO ***********************


/*********************************************************************
********            metodo L_alpha                                ******
*********************************************************************

 VIDE APENDICE 8
// ************        fim do metodo Lalpha           *****************

/*********************************************************************
*****            metodo hexa - converte bin-hexa                  ******
*********************************************************************/
VIDE APENDICE 8
// ******************** fim do metodo hexa  ****************************


/*********************************************************************
 *****            metodo exibe 128 - exibe  array                  ******
```

```
 ***********************************************************************/
VIDE APENDICE 8
// ****************** fim do metodo exibe 128  ***********************/


/***********************************************************************
 ***** metodo CG2a8- converte inteiro ate 255 em polinomio        ******
 ***********************************************************************/
VIDE APENDICE 8
// fim do metodo CG2a8

}// fim da classe  testes_nist_alpha_vs2
```

12.10 APÊNDICE 10: TRANSFORMAÇÕES LINEARES CONSTRUÍDAS PELO MÉTODO DOS NÓS E SUAS RESPECTIVAS MATRIZES DE DEPENDÊNCIAS

12.10.1 GERAÇÃO DE UMA TL DE TAMANHO n=16 E m=4 COEFICIENTES NÃO NULOS POR EQUAÇÃO

```
Ultimo nivel : 3
Numero de grupos por nível: 0 1 3 1
Total de posicoes acumulado ate o nivel, inclusive: 1 4 13 16
Nr de posicoes no ultimo grupo: 3
OBS: O valor −1 indica que o campo ainda não foi preenchido.
```

12.10.1.1 RESULTADO PARCIAL (ALGORITMO DO ITEM 4.2 EXECUTADO ATE PASSO (iv))

```
nó  :  nivel, grp , pos, bit ,pai , f1 , f2 , f3...
Nó  0      0,   0,   0,   8,   0,   7,  10,   4,
Nó  1      1,   0,   0,   7,   8,  11,   2,  15,
Nó  2      1,   0,   1,  10,   8,   1,   9,   5,
Nó  3      1,   0,   2,   4,   8,  13,   3,  12,
Nó  4      2,   0,   0,  11,   7,   6,  14,   0,
Nó  5      2,   0,   1,   2,   7,  -1,  -1,  -1,
Nó  6      2,   0,   2,  15,   7,  -1,  -1,  -1,
Nó  7      2,   1,   0,   1,  10,  -1,  -1,  -1,
Nó  8      2,   1,   1,   9,  10,  -1,  -1,  -1,
Nó  9      2,   1,   2,   5,  10,  -1,  -1,  -1,
Nó  10     2,   2,   0,  13,   4,  -1,  -1,  -1,
Nó  11     2,   2,   1,   3,   4,  -1,  -1,  -1,
Nó  12     2,   2,   2,  12,   4,  -1,  -1,  -1,
Nó  13     3,   0,   0,   6,  11,  -1,  -1,  -1,
Nó  14     3,   0,   1,  14,  11,  -1,  -1,  -1,
Nó  15     3,   0,   2,   0,  11,  -1,  -1,  -1,
```

12.10.1.2 RESULTADO FINAL (ALGORITMO DO ITEM 7.2.4.1 EXECUTADO ATE PASSO (v))

```
nó  :  nivel, grp , pos, bit ,pai , f1 , f2 , f3...
Nó  0      0,   0,   0,   8,   0,   7,  10,   4,
Nó  1      1,   0,   0,   7,   8,  11,   2,  15,
Nó  2      1,   0,   1,  10,   8,   1,   9,   5,
Nó  3      1,   0,   2,   4,   8,  13,   3,  12,
Nó  4      2,   0,   0,  11,   7,   6,  14,   0,
Nó  5      2,   0,   1,   2,   7,   1,   5,  13,
Nó  6      2,   0,   2,  15,   7,   3,  12,   9,
Nó  7      2,   1,   0,   1,  10,   2,   6,  14,
Nó  8      2,   1,   1,   9,  10,   3,  15,   0,
Nó  9      2,   1,   2,   5,  10,  13,  12,   9,
Nó  10     2,   2,   0,  13,   4,   6,   1,   2,
Nó  11     2,   2,   1,   3,   4,  14,   5,  15,
Nó  12     2,   2,   2,  12,   4,   0,   9,  13,
Nó  13     3,   0,   0,   6,  11,   1,   5,  15,
Nó  14     3,   0,   1,  14,  11,   3,  12,   2,
Nó  15     3,   0,   2,   0,  11,   8,   6,   4,
```

Matriz da TL = Matriz de Dependências da TL

```
....1.1.1..1....
..1...1...1...1.
.1...1.1.....1..
....11........11
...1....1...11..
.........11.11..
.1...1.....1...1
..1.....1..1...1
1...1..1..1.....
1..1......1....1
.1...1..11......
1.....11......1.
1...1....1...1..
.11.1.1.........
..11.......11...
...1...1.1..1...
```

Limite Inferior do Potencial de Avalanche 0 = 4
Limite Superior do Potencial de Avalanche 0 = 4

Segunda Matriz de Avalanche (2 passos da TL)

```
11.111111.111111
.111.1.111.111.1
.11.1.1.11111111
..11...1111111..
111.1111.11..111
1111111.111..1.1
1.11..11.11.111.
11.11111.11.111.
.111111.11.111.1
.1.1111111.11.11
1.111.11.11.1111
.111111.1..11..1
11111.1.1.1111.1
.111.1111.111111
11..1111.1...111
1.1111..1111.111
```

Limite Inferior do Potencial de Avalanche 2 = 9
Limite Superior do Potencial de Avalanche 2 = 14

Terceira Matriz de Avalanche (3 passos da TL)

```
1111111111111111
1111111111111111
1111111111111111
111111111111.111
1111111111111111
1111111111111111
1111111111111111
1111111.11111111
1111111111111111
1111111111111111
1111111111111111
1111111111111111
1111111111111111
1111111111111111
1111111111111111
1111111111111111
```

```
Limite Inferior do Potencial de Avalanche  3 = 15
Limite Superior do Potencial de Avalanche  3 = 16

Quarta Matriz de Avalanche (4 passos da TL)
A matriz é completa.
Limite Inferior do Potencial de Avalanche  4 = 16
Limite Superior do Potencial de Avalanche  4 = 16
```

12.10.2 GERAÇÃO DE UMA TL DE TAMANHO n=64 E m=7 COEFICIENTES NÃO NULOS POR EQUAÇÃO

```
Ultimo nivel : 3
Numero de grupos por nível: 0 1 6 4
Total de posicoes acumulado ate o nivel, inclusive: 1 7 43 64
Nr de posicoes no ultimo grupo: 3
```

nó	:	nivel,	grp ,	pos,	bit ,	pai ,	f1 ,	f2 ,	f3...			
Nó	0	0,	0,	0,	40,	5,	1,	51,	16,	27,	34,	61,
Nó	1	1,	0,	0,	1,	40,	8,	41,	2,	25,	52,	38,
Nó	2	1,	0,	1,	51,	40,	57,	13,	18,	58,	7,	30,
Nó	3	1,	0,	2,	16,	40,	21,	32,	44,	10,	55,	46,
Nó	4	1,	0,	3,	27,	40,	63,	12,	49,	4,	22,	37,
Nó	5	1,	0,	4,	34,	40,	31,	53,	26,	19,	45,	36,
Nó	6	1,	0,	5,	61,	40,	62,	3,	9,	60,	15,	50,
Nó	7	2,	0,	0,	8,	1,	6,	29,	47,	23,	35,	0,
Nó	8	2,	0,	1,	41,	1,	33,	17,	56,	42,	54,	24,
Nó	9	2,	0,	2,	2,	1,	11,	20,	14,	43,	39,	48,
Nó	10	2,	0,	3,	25,	1,	28,	59,	5,	58,	13,	19,
Nó	11	2,	0,	4,	52,	1,	37,	49,	15,	55,	63,	21,
Nó	12	2,	0,	5,	38,	1,	35,	17,	33,	10,	60,	53,
Nó	13	2,	1,	0,	57,	51,	31,	50,	32,	12,	23,	59,
Nó	14	2,	1,	1,	13,	51,	14,	39,	22,	52,	26,	62,
Nó	15	2,	1,	2,	18,	51,	11,	56,	48,	28,	36,	20,
Nó	16	2,	1,	3,	58,	51,	9,	43,	30,	54,	38,	33,
Nó	17	2,	1,	4,	7,	51,	57,	0,	31,	17,	49,	15,
Nó	18	2,	1,	5,	30,	51,	60,	42,	3,	37,	44,	29,
Nó	19	2,	2,	0,	21,	16,	18,	11,	19,	50,	62,	22,
Nó	20	2,	2,	1,	32,	16,	48,	24,	56,	47,	13,	6,
Nó	21	2,	2,	2,	44,	16,	35,	59,	14,	46,	54,	4,
Nó	22	2,	2,	3,	10,	16,	28,	20,	39,	29,	7,	45,
Nó	23	2,	2,	4,	55,	16,	18,	36,	63,	53,	12,	58,
Nó	24	2,	2,	5,	46,	16,	23,	38,	26,	3,	47,	52,
Nó	25	2,	3,	0,	63,	27,	10,	21,	55,	57,	24,	9,
Nó	26	2,	3,	1,	12,	27,	32,	42,	5,	18,	44,	30,
Nó	27	2,	3,	2,	49,	27,	56,	4,	11,	36,	50,	6,
Nó	28	2,	3,	3,	4,	27,	15,	62,	46,	31,	20,	54,
Nó	29	2,	3,	4,	22,	27,	35,	0,	33,	24,	48,	21,
Nó	30	2,	3,	5,	37,	27,	60,	45,	13,	12,	23,	39,
Nó	31	2,	4,	0,	31,	34,	28,	43,	58,	7,	49,	22,
Nó	32	2,	4,	1,	53,	34,	9,	5,	29,	38,	52,	59,
Nó	33	2,	4,	2,	26,	34,	44,	14,	42,	17,	37,	63,
Nó	34	2,	4,	3,	19,	34,	55,	4,	30,	6,	32,	57,
Nó	35	2,	4,	4,	45,	34,	10,	46,	60,	48,	28,	43,
Nó	36	2,	4,	5,	36,	34,	0,	12,	38,	29,	52,	21,
Nó	37	2,	5,	0,	62,	61,	19,	15,	53,	37,	3,	26,

```
   Nó  38     2,   5,   1,   3,  61,  57,  17,  47,  35,  45,  25,
   Nó  39     2,   5,   2,   9,  61,  59,  49,  10,   7,  20,  54,
   Nó  40     2,   5,   3,  60,  61,  18,  56,  32,  44,  14,  31,
   Nó  41     2,   5,   4,  15,  61,   0,  62,  36,  30,  11,  53,
   Nó  42     2,   5,   5,  50,  61,  19,  46,   3,  33,  45,  55,
   Nó  43     3,   0,   0,   6,   8,   7,  23,  26,   9,  63,  22,
   Nó  44     3,   0,   1,  29,   8,  43,  58,  39,  50,  24,   5,
   Nó  45     3,   0,   2,  47,   8,  13,  33,  59,  52,  14,  28,
   Nó  46     3,   0,   3,  23,   8,  42,   4,  20,  19,  55,  57,
   Nó  47     3,   0,   4,  35,   8,  32,  10,   6,  31,  12,  36,
   Nó  48     3,   0,   5,   0,   8,  24,  56,  43,   7,  22,  48,
   Nó  49     3,   1,   0,  33,  41,  47,  26,  46,  49,  62,   4,
   Nó  50     3,   1,   1,  17,  41,  15,  37,  23,  58,   0,  53,
   Nó  51     3,   1,   2,  56,  41,   9,  18,  38,  30,  45,  63,
   Nó  52     3,   1,   3,  42,  41,  50,  11,  29,  21,  35,   3,
   Nó  53     3,   1,   4,  54,  41,  44,  13,  60,   6,  39,  47,
   Nó  54     3,   1,   5,  24,  41,  17,  25,   0,  31,  53,  22,
   Nó  55     3,   2,   0,  11,   2,  42,  63,   9,  38,  54,  47,
   Nó  56     3,   2,   1,  20,   2,  60,  18,  29,   5,  36,  10,
   Nó  57     3,   2,   2,  14,   2,  50,  33,  48,  62,  25,  12,
   Nó  58     3,   2,   3,  43,   2,  17,   4,  44,  56,  39,  19,
   Nó  59     3,   2,   4,  39,   2,  49,  24,   7,  42,  11,  45,
   Nó  60     3,   2,   5,  48,   2,   6,  21,  32,  26,  59,  52,
   Nó  61     3,   3,   0,  28,  25,  15,  37,  55,   5,  57,  20,
   Nó  62     3,   3,   1,  59,  25,  43,  14,  30,  23,  35,  13,
   Nó  63     3,   3,   2,   5,  25,  40,  46,  54,  58,   3,  18,
```

Matriz de Dependências da TL

```
.......11.............1.1.................1....1.......1.......
..1.....1..............1............1.11..........1...........
.1.........1..1.....1.................1...1....1..............
..................1.......1.........1.........1.1.........1...1..
...............1....1......1...1..............1.......1.......1.
...1..............1......1..............1.....1.......1...1.....
.......111............11..1.....................................1
1..............1.1.............1................1.1.....1......
11....1..............1.....1.....1...........1................
.......1..1.........1.............................1....1....1.1..
.......1........1...1.......11.........1.....1..................
..1......1.............................1...1...1.....1........1
.....1............1........1..1.1.........1.1...................
..............1.......1...1............1...........11.........1.
..1.........1............1.......1..............1.1...........1.
1..........1..................1.....1..............1.......11.
..........1..........1..........1.......1...1.1........1........
1..............1.......1.............1...1...........1....1.....
...........1........1......1......1...........1..1....1.......
....1.1.........................1.1.1...................1.1......
..1..1....1.......1..........1......1.......................1...
...........1....1.11..1.............................1...........1.
1.....................1..1..1.....1.1............1..............
....1...1..........11.....................1...........1.1......
1................1....1..1.....1.........1...........1..........
.1...1.......1.....1........1.............................11....
..............1..1...............1..1....1.1..................1
....1.......1.........1..............1..1........1.............1
.....1.........1....1....1...........1..................1.1......
.....1..1..............1..............1...1......1.......1.....
...1.........................1.......1....1.1......1........1...
.......1..............1.....1.....1........1.....1........1.....
......1......1..1.......1.......................11.......1.......
```

```
....1.....................1..............1....11.1............1.
...................1......1....1....1...1....1.......1.........
......1.1.1.1...................11...1.........................
1...........1........1.......1....1...1.............1...........
............11.........1...1...........1.....1..............1...
.1........1......1..............1.1.................1......1...
..1....1...1............1.................1..1...1..............
.1...1..........1.........1......1................1.........1..
.1..............1......1........1........1...........1.1.......
...1.......1.........1.......1.....1.....1........1.............
..1.1............1.1.....................1....1...........1.......
....1.........1.1...................1..........1.......1....1....
..........1..................1.....1........1..1.1...........1...
...1............1......1..1...........1........1....1...........
........1....11.............1....1...................1......1....
..1...1..............1....1.....1.....................1......1....
....1.1....1...............1........1.............1.....1.......
...1...............1.............1...........11........1.....1..
.......1.....1....1...........1.........1................11.....
.1.............1.....1..............1...........1.....1.......1
.....1...1...................1....1...1.............1......1....
......1......1.........................1.1..1..1............1...
............1...1.1..................1................1....1....1
.........1........1...........1.......1..1...1..................1
............1..........1.......11..................11.......1....
.........1.....................1..1....1....1.......1..1.........
.............11........1.1....1....1.......1.....................
...............1...1............11...........1...........1....1..
...1.....1.....1.........................1.........1.........1.1.
...1...........1...1......1..........1................1.......1..
.........11..........1..1..1.............................1.1......
Limite Inferior do Potencial de Avalanche  = 7
Limite Superior do Potencial de Avalanche  = 7
```

Segunda Matriz de Avalanche (2 passos da TL)

```
111.1.1..1.....1.111.1111111.11111.1..11.1..11.111.111..11.1...1
11...11...11.11111.111.11..111...111.1.1..11...111.1.1111.1111.1
..1.111111111....111.1..111..1..11..1.11111.11.1111.1.1.1..11.11
11.1.11.111.1111...1...1....1..1111.11..11.1..1.1.1111....111.1.
1.111111..1111.11.11..11..1.111...1.111111.11..1.1..11....1.1111
.1.1.11..1.1.1..11.11..1.1111.1..1111.11.1.111.11..11.1.111111..
11..1.11111...11.1.111.11..1.1.1.111.1....1.1..111.1..11.1.1.1.1
1...1.111..111.1..1...111..11.111.1.11..11.1....1111.1..1111.11.
..1.1111111.111....11.11111.1..111..1.111111....1.1.1..11111...1
1.111111.111.111111.1..1.1.11111...11..111.111.1.111....11..1.1.
1.1..1.11.11...1.11.11..11..11.11.1.11.11.11111.1111...1.11.1...
.1.1..111111.11..1..11..1..111...1.1...1.1.11..1111.1111.1.111..
...11.1....1111.1.1.111.11..11.....111..111.1.111111..1.1.111..1
1111...1...11111.111.1..1111..1..111.1..1.1.11..111..1.1.11..111
.111111....1.111..1111....111.1.11...1.1.111111111..11.1..11.11.
1.11.1.111..1..1...1.11.1.1..1....1..11.1.111..11.11111.1..11111
.1.11111...1111.1.111.111.1111....111.11.....1111.11111.1.11.111
11..11.111.111...1.11.111..1.11..11.1.11..11.1..1..1111111.1111.
1.1..111.11.11.1..1.11...11..11.1.1.111.111..1.1....1.11.1111..1
...1..1111..11.11.111.111.11.1.11...11..1.1.11111.11.11.1.111.11
11.1.1.11..11.1.1.1.11..11..11.11.1.1.111..1111.1.111.1.1.1..1..
1.111.1..111...1...111..1.111.1.1111111.1.1.11111..1.11111...1.1
1.1.1.111.111...1111.11.111....11...11..11.1..11111.11..1..1..11
1111111...111..11.1.11.1...1.1111.111....1....11..11.111.1111.11
11...1.111...1.1.1.1.1111..111...111.11..111....11..111.1.11....
..11111.11...111..1.1.11.11...1.1111.11111.1..1....11.11.11...1.
```

```
1.111....11111111..1.1.11111.1.1.1.111.111...11.1.1..111.1111.1.
11..111..11111.11.1.11.11..1..1111111..1..1.111.1.11..1111..111.
1111.1....1111..1.11...1.1.111111...1..11....11...11.11...111111
11111.11.1.1.....111..1111...111.1.1..1111111111.1.1.1111.1..1..
...111.11..1111.111..1.111.1.1111..1...111.11111..1...1.111111..
1.1.111..1.1...1.1.111..1111..11.1.111111..111..1111.11111......
1.1...11111..11..11..111.11.1.1111....1111..111....111.1...1..11
.1.11.1.1..1.11111.11..11.111..1.11.111...1.1.11..1.111.1..1.111
11..1111.11.1.1.11...11....1111.1.1..11...111.1.11.11..1.11111.1
11...11111..11..1.1.11111.11111.1.11..11..1111.111..1...1.1....1
.1...1.11.11...11111.11.1.11..1111.111.11.1111..111..1.11.1.1.11
..1.11.11.111.1...111.1.1.111.111.1..1.11.11111.11.11..111..1111
1.1.1111111.1.111.1.1..1.11.11.11.1.111111..1111.1..11..1.11.11.
11111.1..111..11.1..111..1.111.1..111.11.111..111111.11.11..1..1
..111..1111.11.1..11.11..11...111...111.11..111..11.1111.11.1.11
1.111.1.11.1.1.1.11..111.11..111...1.11111..1111.11.11....1.1.11
.111.11.11111...1111..1.11.....111.11.11..11.111..1...11111..111
111.1.11.1.1..111.1.1..11..1..111.11.111.111.11.11...111.111..11
..11..1.1.1.11111...11.1.111..1111.11.1111.11.111.1.1.11....1.1.
..111111......11111111.1.11.11.11...11111...11.1....11.111.1.1..
.1..1...1.1..111.1.111...1..1...1111.1..1.1.1111.1..11.1.1.111.1
111.111.....1111....1111.11..11..1.1.1.1.1.1..1111111..1.1....11
.1....1111.1.11111111111111...1...11.1.1..111..1111....11.....11
1.111..111..1..1..111111..11.111.11..11.111..111.1..1.11.....111
...11.1..11.1..1111....1.11.1.1.1.111.1.11.1.111111.11.1.11.1111
11.1.1...1.11.1111..1.11..111111111.1111..111...11111.1.11.1111.
1.1.1.1.111111..1.11.11111.1..1.....1.1111...1....1.11.1111.1111
.1.1.1.11.1..111.11111.1111...11.1.111.11..1.11..11..111..1111.1
.11.1..111.1.11.111...111.1.1..111.1...1..1.111..1.11.1.1..1.111
1....1...1111.....1.11..1..1111.111.1.1.1.111.1.1..11.1111.1....
.1.1...1.111.....1..11..1..111...11111....111.1.11.1.11111.111..
...111111....11.1.111.1111.11.1.1111....1.11111111.....1111..1..
.1111.11..1..1...1111.....1..11..1.1.1.1111.1.11.1.1.11.1111111.
.111111.1.1.111..1.11.1..11.11.111..11.1..1.1...1.111..111111.1.
..111.11.1.111111.1.1.1.11..1.1..1111.1.11.1.1111111..1.1.111.11
11.1.1.1..11..1111111....111..11111111......1111.1.1.11111.1.11.
1..1111..1.11111.1.....1.1.1.11.1.1111111.1.11.1..1.11.1.1.11111
1...1..1..111...11111.11.1..11.11...11.111...1...111.11...11.111
Limite Inferior do Potencial de Avalanche  2 = 32
Limite Superior do Potencial de Avalanche  2 = 42
```

Terceira Matriz de Avalanche (3 passos da TL)

```
11111111111111111111111111111111111111111111111111111111111111111
1.11111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111.11111111111111111111
11111111111111111111111111.11111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111
1111111111.11111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111.11111111111111
11111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111
```

1111.1111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111.1111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1.1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111.1111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111.1111111
1111111111111111111111111111111.1111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
11111111.1111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
11.11111.1111111111111111111111111111.1111111111111111111
1111111111111111111111111111111111.1111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111.1111111111111111111.1111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111.1111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1.1111111111111111111111111111.1111111111111.111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
Limite Inferior do Potencial de Avalanche 3 = 61
Limite Superior do Potencial de Avalanche 3 = 64

Quarta Matriz de Avalanche (4 passos da TL)
A matriz é completa.
Limite Inferior do Potencial de Avalanche 4 = 64
Limite Superior do Potencial de Avalanche 4 = 64

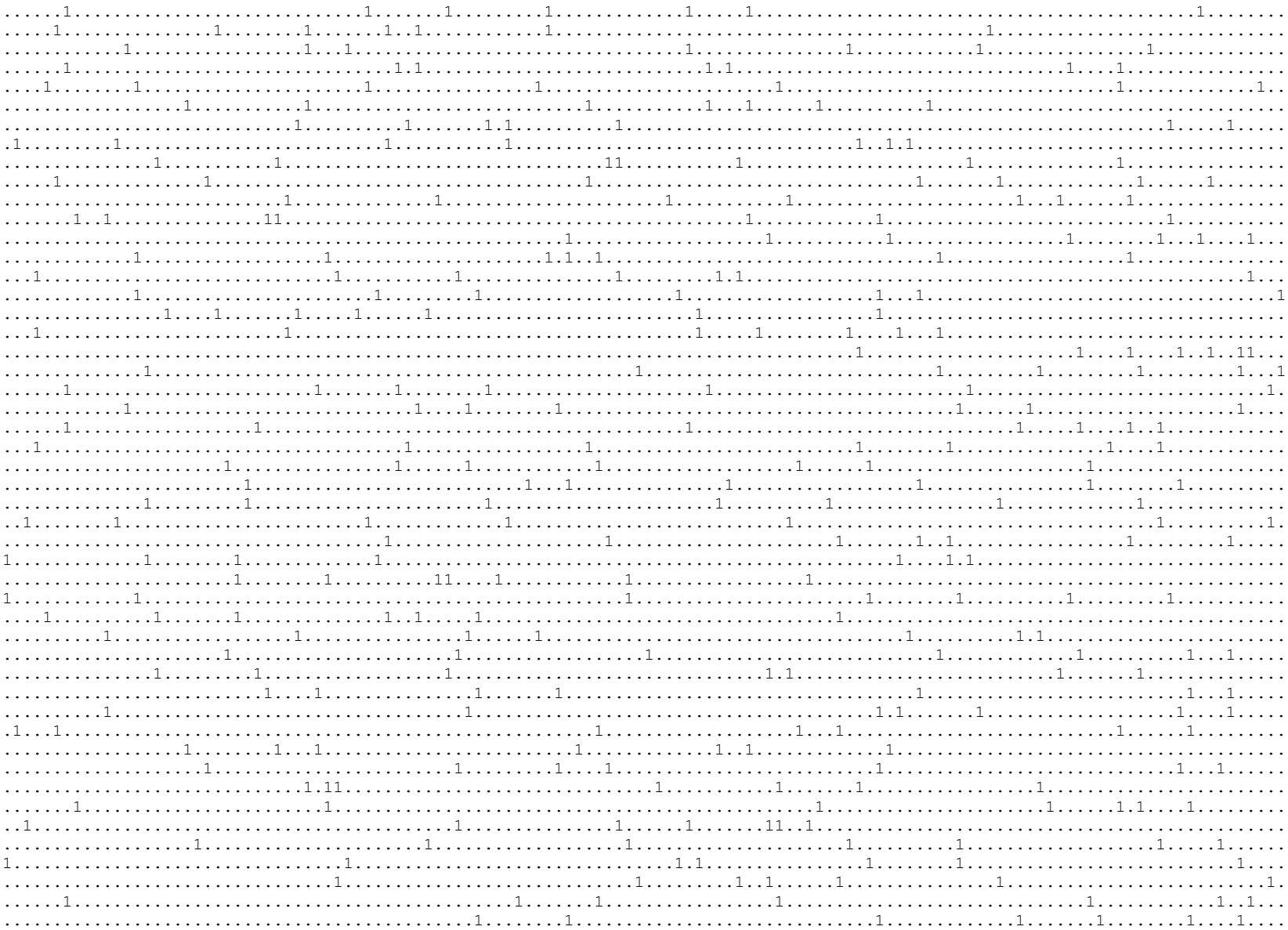
12.10.3 GERAÇÃO DE UMA TL DE TAMANHO *n=128* E *m=7* COEFICIENTES NÃO NULOS POR EQUAÇÃO

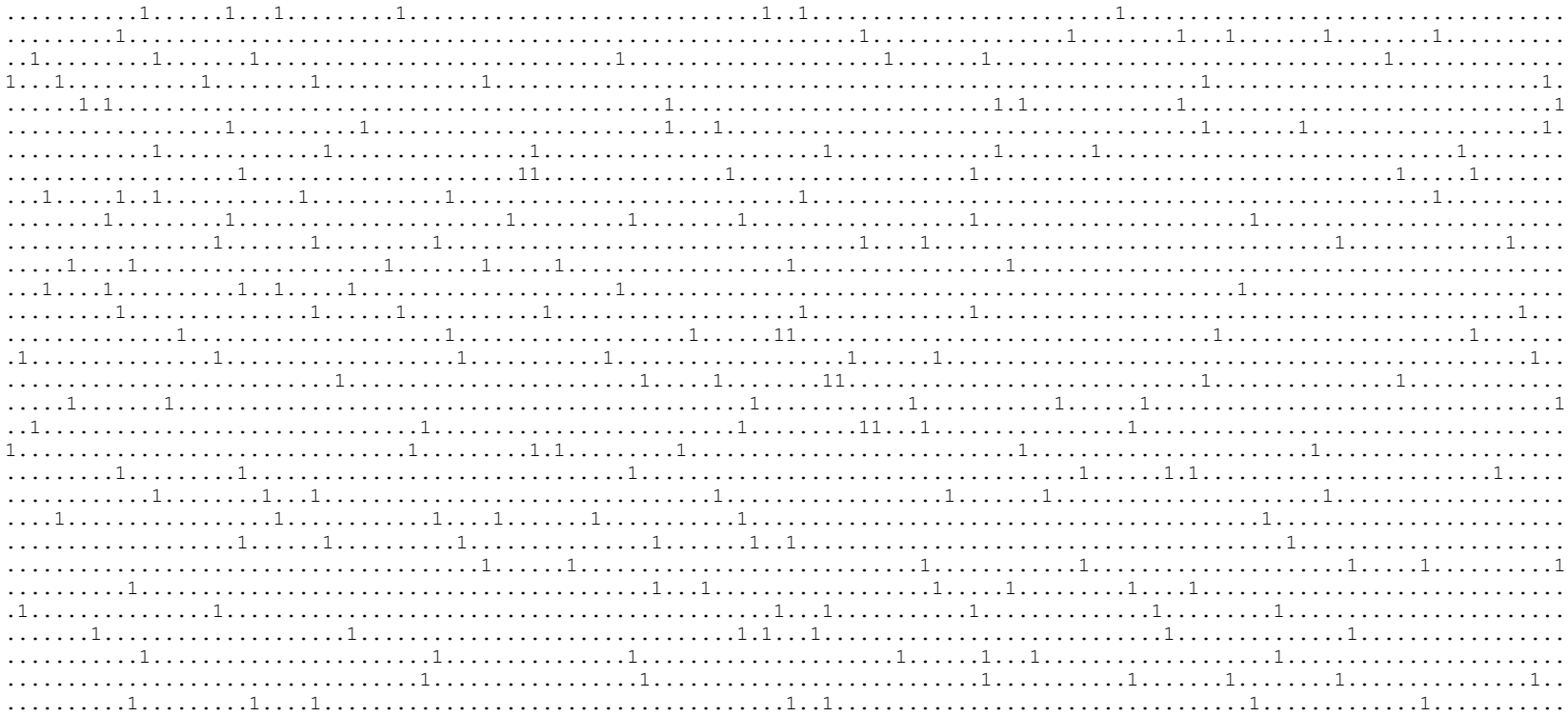
```
Ultimo nivel : 3
Numero de grupos por nível: 0 1 6 15
Total de posicoes acumulado ate o nivel, inclusive: 1 7 43 128
Nr de posicoes no ultimo grupo: 1

    nó  :  nivel, grp , pos, bit ,pai , f1 , f2 , f3...
   Nó  0      0,   0,   0, 119,  40, 103,  60,   4,  35,  22,  48,
   Nó  1      1,   0,   0, 103, 119,  67,  43,  81,  89,  12,  26,
   Nó  2      1,   0,   1,  60, 119, 124, 106,  76,  88,  56, 115,
   Nó  3      1,   0,   2,   4, 119, 100,   5,  24,  37,  52,  80,
   Nó  4      1,   0,   3,  35, 119, 107, 125,  11,  64,  47,  19,
   Nó  5      1,   0,   4,  22, 119,  72,  82, 109,  97,  31,  23,
   Nó  6      1,   0,   5,  48, 119,  44,  36,  68,   6,  54,  74,
   Nó  7      2,   0,   0,  67, 103,  14,  93, 123, 113,  63, 127,
   Nó  8      2,   0,   1,  43, 103, 116,  17,  58,  78, 104,   3,
   Nó  9      2,   0,   2,  81, 103,  10,  29,  46,  90, 101,  53,
   Nó  10     2,   0,   3,  89, 103,  85,  33,  65,  77,  30,  32,
   Nó  11     2,   0,   4,  12, 103,   1,  99, 105, 118,  69,  50,
   Nó  12     2,   0,   5,  26, 103,  13,  21, 120,  59,  84,  42,
   Nó  13     2,   1,   0, 124,  60,  95,  62,   7,  66, 110,  28,
   Nó  14     2,   1,   1, 106,  60, 102,  51,  79,  18,   8,  41,
   Nó  15     2,   1,   2,  76,  60,  38,  94,  83, 122, 112,  91,
   Nó  16     2,   1,   3,  88,  60,  45, 121, 117,  20,  87,  55,
   Nó  17     2,   1,   4,  56,  60,  96,  61,  15,  27, 111,  73,
   Nó  18     2,   1,   5, 115,  60,  71,   2,  34,  70,  92,  75,
   Nó  19     2,   2,   0, 100,   4,   0,  98,  39, 126,  25,  16,
   Nó  20     2,   2,   1,   5,   4,   9,  86,  57, 108,  49, 114,
   Nó  21     2,   2,   2,  24,   4,  40,  10,  55,  34, 110,  85,
   Nó  22     2,   2,   3,  37,   4,  68,  20,  74, 120,  30,  66,
   Nó  23     2,   2,   4,  52,   4,  53, 125,  77,  36, 111,  13,
   Nó  24     2,   2,   5,  80,   4,  41,  83,  23,  15,  47,  38,
   Nó  25     2,   3,   0, 107,  35,  70,  17,  75,  25, 123, 109,
   Nó  26     2,   3,   1, 125,  35,  51,  84,  11, 104,  80,  73,
   Nó  27     2,   3,   2,  11,  35,  33, 127,  52,  44,  71,  27,
   Nó  28     2,   3,   3,  64,  35,  16,  42,  29,  69,  87,  21,
   Nó  29     2,   3,   4,  47,  35,  37,  78,  49, 121,  14, 105,
   Nó  30     2,   3,   5,  19,  35,  65,  31, 126,  54,   9,  18,
   Nó  31     2,   4,   0,  72,  22,  86,  39, 108,  46,  79,  59,
   Nó  32     2,   4,   1,  82,  22,  93, 107, 122,  45,  64, 118,
   Nó  33     2,   4,   2, 109,  22,  19,  28,   3, 101,  50,   8,
   Nó  34     2,   4,   3,  97,  22,  32,  65,  18,  91,  62,  11,
   Nó  35     2,   4,   4,  31,  22,  72, 125,   7, 102, 110,  46,
   Nó  36     2,   4,   5,  23,  22,  82, 112,  30,  74,  53,  36,
   Nó  37     2,   5,   0,  44,  48,  84,  16,  51,  85,  15, 117,
   Nó  38     2,   5,   1,  36,  48,  90,  72, 111,  57,   2,  42,
   Nó  39     2,   5,   2,  68,  48, 126,  39,  70,  31,  96,   6,
   Nó  40     2,   5,   3,   6,  48,  77,  92,  97,  21, 108,  38,
   Nó  41     2,   5,   4,  54,  48, 122, 116,  40,  29,  50,  61,
   Nó  42     2,   5,   5,  74,  48,  14,  71,  82,  24, 113,  99,
   Nó  43     3,   0,   0,  14,  67,  41,  54, 109,  19,   8,  73,
   Nó  44     3,   0,   1,  93,  67, 123,  69,   0,  95,  86,  34,
   Nó  45     3,   0,   2, 123,  67,  63,  17, 104,  79,  94,   1,
   Nó  46     3,   0,   3, 113,  67,  27,  68,  52,  98, 114,  58,
   Nó  47     3,   0,   4,  63,  67,  13,  47,  37,  87, 127,  91,
   Nó  48     3,   0,   5, 127,  67, 102,  25, 116,  64,  20,  10,
   Nó  49     3,   1,   0, 116,  43, 107,   0,  45,  33,  55,  83,
```

```
Nó  50     3,   1,   1,  17,  43,  75, 121,  63,   2,  94,   9,
Nó  51     3,   1,   2,  58,  43,  66,  78, 101, 112, 105,  28,
Nó  52     3,   1,   3,  78,  43,  23,  32,  44,  62,  49,  80,
Nó  53     3,   1,   4, 104,  43, 120,  79,  19,  59,  42, 114,
Nó  54     3,   1,   5,   3,  43,  31, 109,  99,  95, 126,  51,
Nó  55     3,   2,   0,  10,  81,  34,  86,  11,   1,  64,  13,
Nó  56     3,   2,   1,  29,  81,  27,  16,  61,  71,  38,  53,
Nó  57     3,   2,   2,  46,  81,  82,  98, 120,  90,  78,  47,
Nó  58     3,   2,   3,  90,  81, 118, 104,   7, 113,  32, 111,
Nó  59     3,   2,   4, 101,  81,  96, 127,   8,  54,   6,  83,
Nó  60     3,   2,   5,  53,  81,  30,  74,  92,  70,  18,  58,
Nó  61     3,   3,   0,  85,  89,  46, 117, 122,  10,  87,  97,
Nó  62     3,   3,   1,  33,  89,  52,  57,  17,  45,  39, 107,
Nó  63     3,   3,   2,  65,  89,  75,  28,   3,  93,  69,  84,
Nó  64     3,   3,   3,  77,  89,  23,  94,  37,  14,  96,   0,
Nó  65     3,   3,   4,  30,  89,  29,  63,  73,  44, 105, 121,
Nó  66     3,   3,   5,  32,  89, 112,  49,  68,  66, 110,  57,
Nó  67     3,   4,   0,   1,  12,  33, 125, 101,  15,  90,  40,
Nó  68     3,   4,   1,  99,  12,  80,   2, 113,  72,  50,  20,
Nó  69     3,   4,   2, 105,  12,  24,   3,   9, 117,  65,  36,
Nó  70     3,   4,   3, 118,  12,  85,  21,  58,  25,  77, 108,
Nó  71     3,   4,   4,  69,  12,  55, 123,  41,  95, 102,  46,
Nó  72     3,   4,   5,  50,  12,  97,  30,  34,  84, 114,  68,
Nó  73     3,   5,   0,  13,  26,  93,  50, 104,  20, 127,  78,
Nó  74     3,   5,   1,  21,  26,   1,  62,   8,  92,  98,  42,
Nó  75     3,   5,   2, 120,  26,  53,  61,  19,  64,  37, 105,
Nó  76     3,   5,   3,  59,  26,  27,  10,  87,   7, 116,  74,
Nó  77     3,   5,   4,  84,  26, 122,  55, 118,  47,  91,  31,
Nó  78     3,   5,   5,  42,  26,  99,  23,  15,  69,  38,  83,
Nó  79     3,   6,   0,  95, 124,   6,  77, 108,  59, 121,  51,
Nó  80     3,   6,   1,  62, 124,  71,  73,  45,   3,  61,  33,
Nó  81     3,   6,   2,   7, 124,  24,  14, 101,  16, 110,  90,
Nó  82     3,   6,   3,  66, 124, 123,  85, 117, 107, 120, 112,
Nó  83     3,   6,   4, 110, 124,  44,  32,  25,  65,  79,   9,
Nó  84     3,   6,   5,  28, 124,  17,  49,  82,  91,  96,   2,
Nó  85     3,   7,   0, 102, 106,  58,  18,  98,  29, 126,  54,
Nó  86     3,   7,   1,  51, 106,  72,  39,   6,  41,  70, 111,
Nó  87     3,   7,   2,  79, 106, 116,  62,  95,  13,  86,   0,
Nó  88     3,   7,   3,  18, 106,  11,  21, 109, 118,  66,  94,
Nó  89     3,   7,   4,   8, 106, 125,  59,  52,  75,  36,  40,
Nó  90     3,   7,   5,  41, 106,  80,  28, 102,  93,   7, 107,
Nó  91     3,   8,   0,  38,  76,  16,  55, 114,  70,  29,  41,
Nó  92     3,   8,   1,  94,  76,  99, 126,  33,  63,  73,  83,
Nó  93     3,   8,   2,  83,  76,  15,  25, 113, 105,  78,  44,
Nó  94     3,   8,   3, 122,  76,  10,  53,  57,  92,  82,  97,
Nó  95     3,   8,   4, 112,  76,  37,   1,  17, 125,  69,  49,
Nó  96     3,   8,   5,  91,  76,   2,  77,  68,  61,  45,  80,
Nó  97     3,   9,   0,  45,  88,  96,  20,   8,  92,  34,  28,
Nó  98     3,   9,   1, 121,  88, 110, 127, 116,  46,  75,  39,
Nó  99     3,   9,   2, 117,  88, 122,   9,  19,  51,  95,  97,
Nó  100    3,   9,   3,  20,  88, 104, 114,   0,  66,  62,  86,
Nó  101    3,   9,   4,  87,  88,  27,  18,  57,  31,  71,  74,
Nó  102    3,   9,   5,  55,  88,  11,  90,   1,  50,  38,  85,
Nó  103    3,  10,   0,  96,  56, 123, 118, 109, 101,  47,  87,
Nó  104    3,  10,   1,  61,  56,  93,  59,  54, 112,  32,  13,
Nó  105    3,  10,   2,  15,  56,  23,  30,   6, 121,  79,  65,
Nó  106    3,  10,   3,  27,  56,  42, 102, 111,   7,  21,  84,
Nó  107    3,  10,   4, 111,  56,  36,  64, 120,  99,  14,  63,
Nó  108    3,  10,   5,  73,  56,  24, 117,  72,  52,  91, 108,
Nó  109    3,  11,   0,  71, 115,  40,   3,  94,  85,  58, 110,
Nó  110    3,  11,   1,   2, 115,  75,  18, 122,  32,  14,  49,
```

```
Nó  111    3,  11,   2,  34, 115,  28, 113,  66,  44,  98,  55,
Nó  112    3,  11,   3,  70, 115,   6,  25, 107, 112, 101,  68,
Nó  113    3,  11,   4,  92, 115,  62,  19,  95,  42,  84, 121,
Nó  114    3,  11,   5,  75, 115,  11,  78,  36, 126,  50,   2,
Nó  115    3,  12,   0,   0, 100,  16,  37,  27,  46,  57,  94,
Nó  116    3,  12,   1,  98, 100, 108,  70,  87,  96, 117,   9,
Nó  117    3,  12,   2,  39, 100,  73,  23,   3,  24,  40,  63,
Nó  118    3,  12,   3, 126, 100,  80,  34, 125,  52, 109,  92,
Nó  119    3,  12,   4,  25, 100, 116,  15,  72,  97,  71, 102,
Nó  120    3,  12,   5,  16, 100,  69,  47,  17,  53,  77, 104,
Nó  121    3,  13,   0,   9,   5, 123,   0,   8,  90,  33,  29,
Nó  122    3,  13,   1,  86,   5,  83, 118,  59,  79,   1, 111,
Nó  123    3,  13,   2,  57,   5,  20,  99,  58, 113, 120,  91,
Nó  124    3,  13,   3, 108,   5,  31,  82,  64,  39,  10,  45,
Nó  125    3,  13,   4,  49,   5,  54,  98,  30,  21,  41,  38,
Nó  126    3,  13,   5, 114,   5,  13,  74,  61, 127,  93,  86,
Nó  127    3,  14,   0,  40,  24, 119,   7, 114, 105,  51,  65,
```

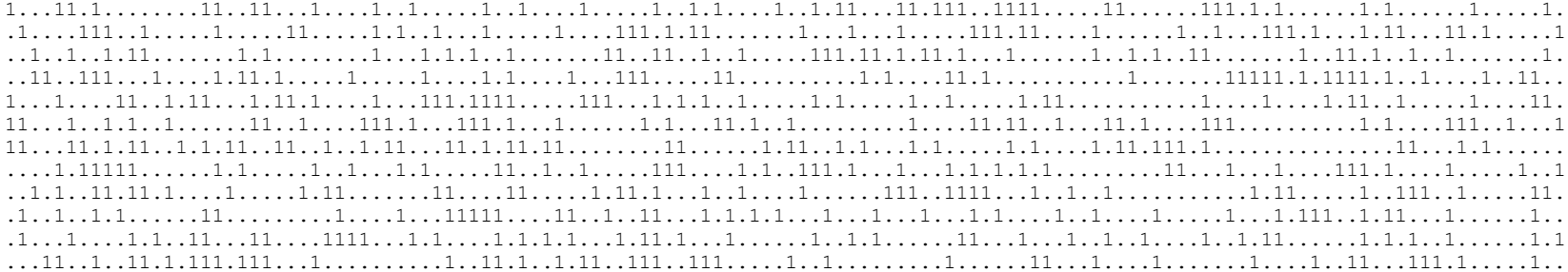
Matriz de Dependências da TL (128 bites)

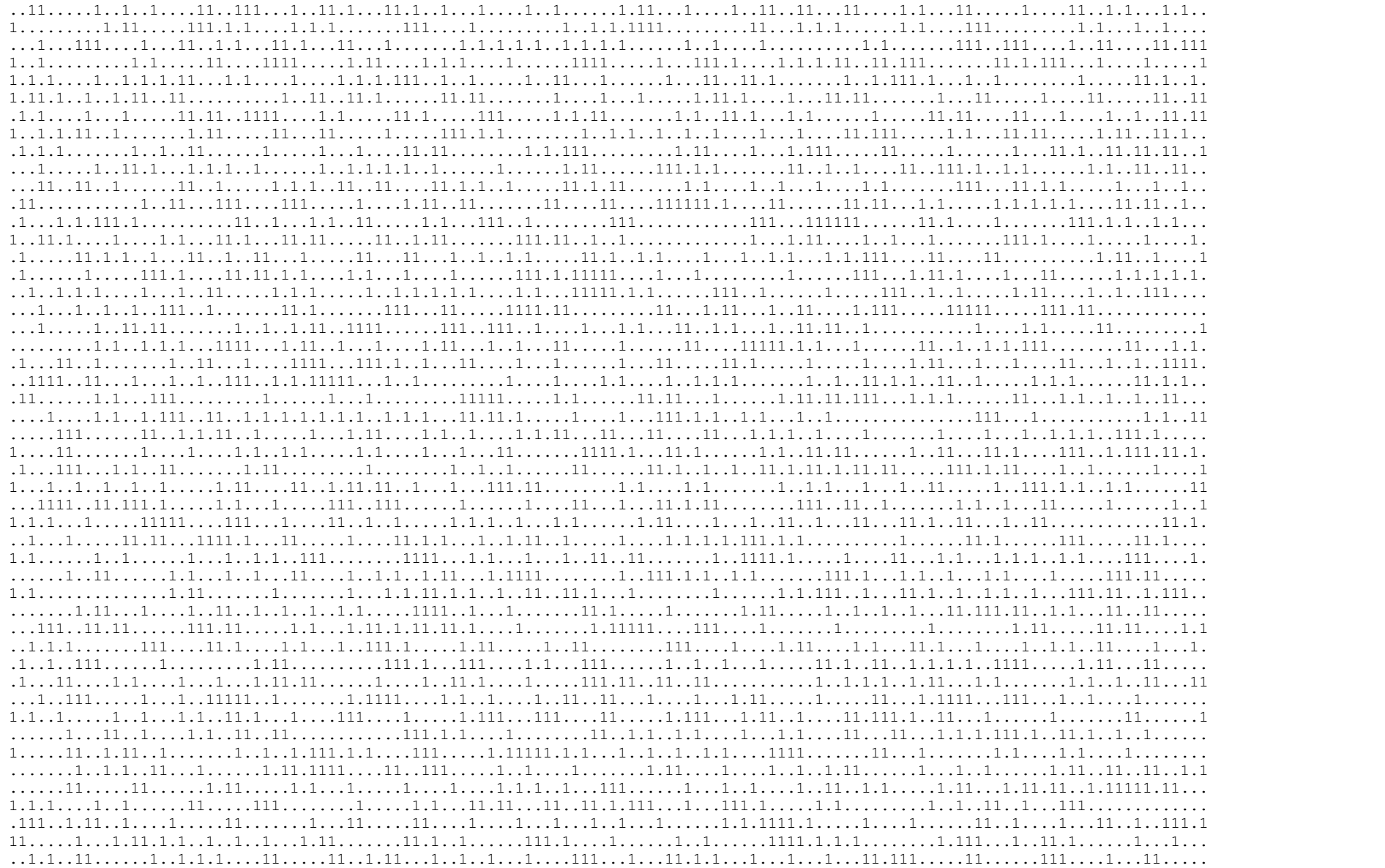


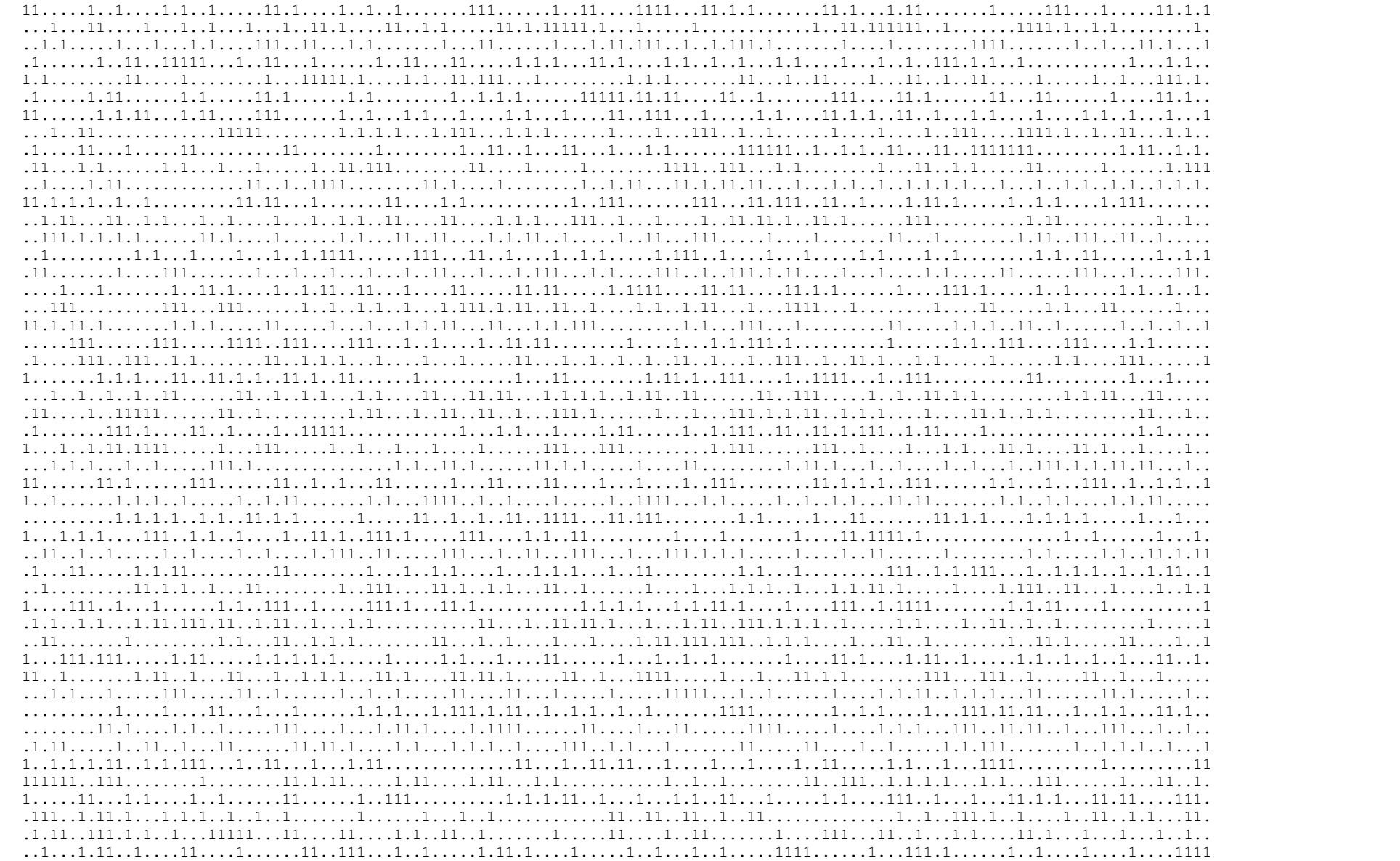


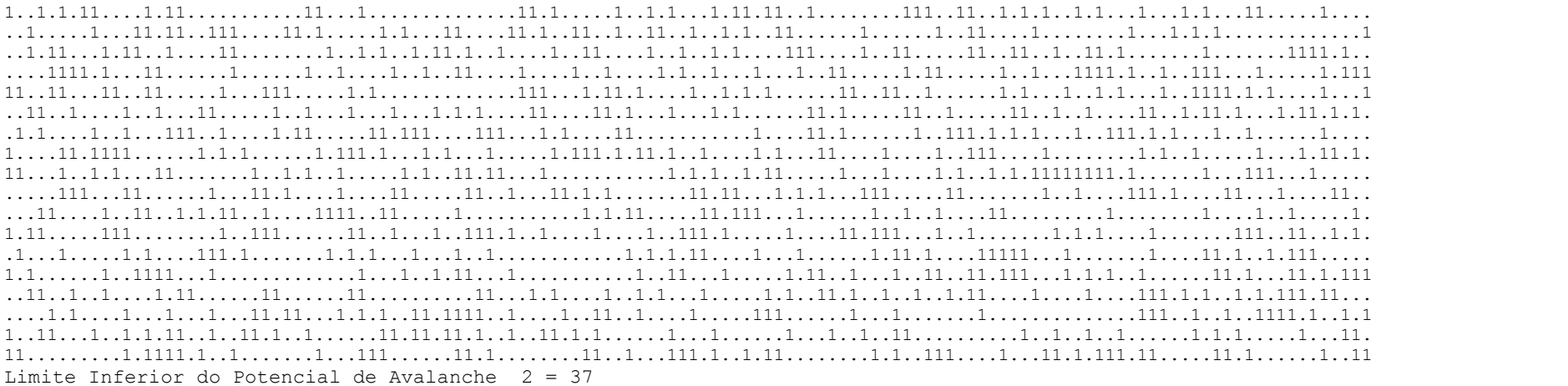
Limite Inferior do Potencial de Avalanche 0 = 7
Limite Superior do Potencial de Avalanche 0 = 7

Segunda Matriz de Avalanche (2 passos da TL)



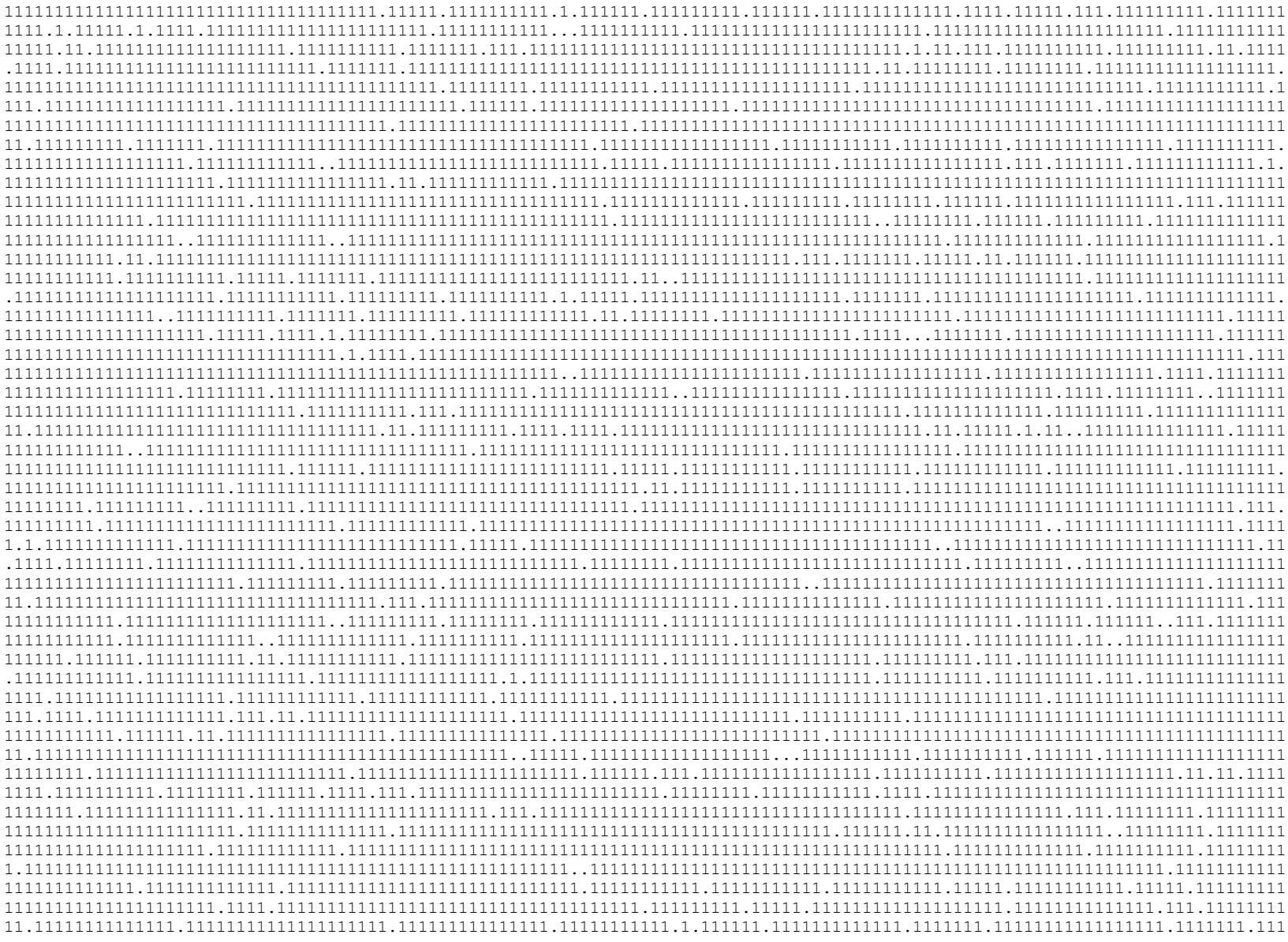


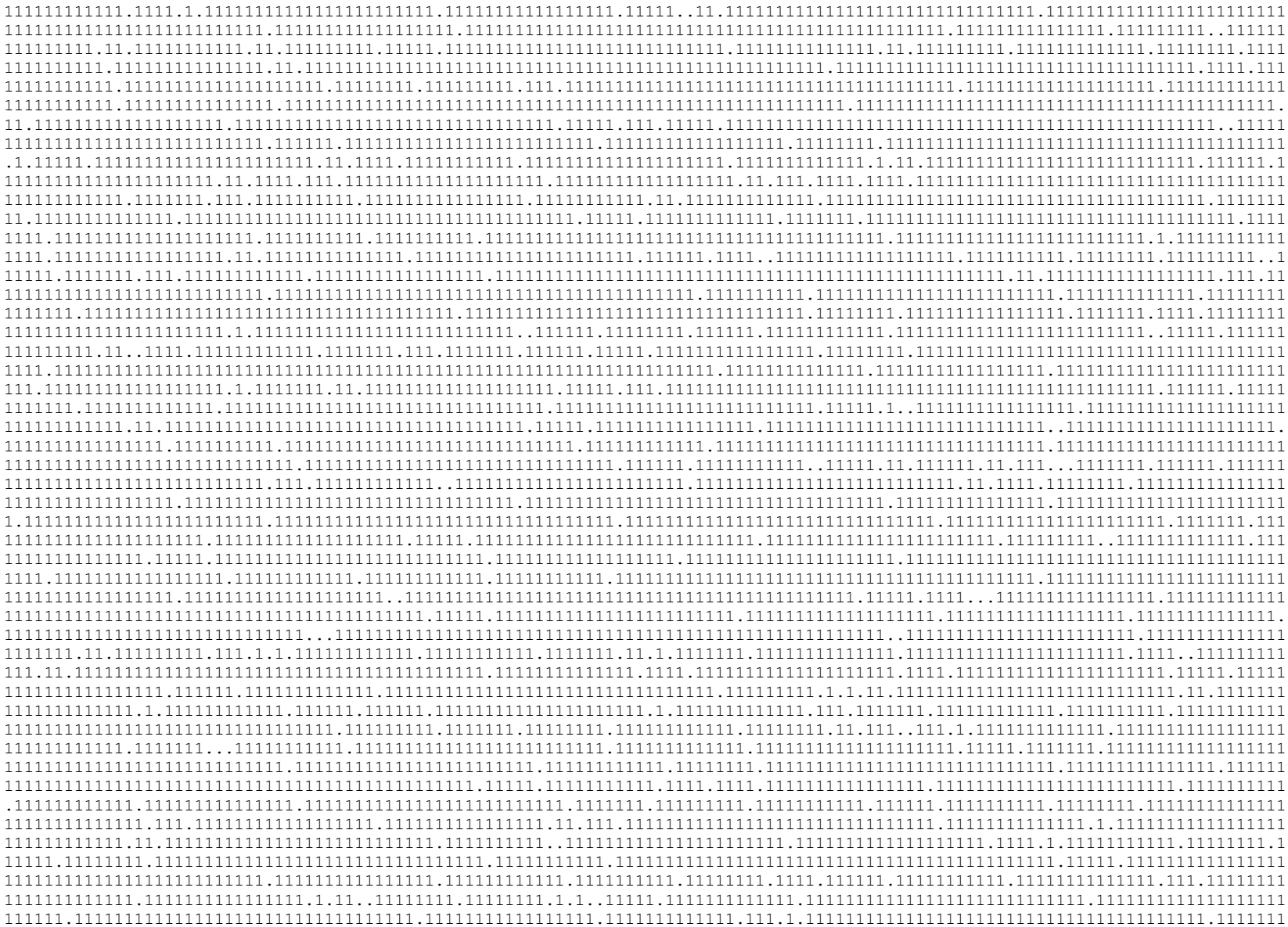




Limite Inferior do Potencial de Avalanche 2 = 37
Limite Superior do Potencial de Avalanche 2 = 46

Terceira Matriz de Avalanche (3 passos da TL)





1111111.11111.11111111.1111111.1.111.11111111111111111111111111111111111111.111111.1111111111111111.1.11111111111111.11.1111111
1111111111111111111..11111.11111111111.1.11111.1111111111111111111.1111111111.11.1111111111.111.11111111111111111111111111
11111111111..1111111111111111111111.11111.1111111.11111111111111111111111111.11111111111111111111111111111111.1111..1111111
11111111111111111111111111111111.111111111.1111111111111111111111.1111111111111.111111111.111111.11111111111111111.11111.111
1111111111.11.1111111.1111111111.11111111111111111111111111111.111.11111111111111111111.11111111111111.1.11111111111.111111111111
11.1111.111111.111111111.11111111111.111111111111.1111111111111111111111.111111111.1.111111111111.11111111111111111111111111111111
Limite Inferior do Potencial de Avalanche 3 = 112
Limite Superior do Potencial de Avalanche 3 = 126

Quarta Matriz de Avalanche (4 passos da TL)

Esta matriz foi omitida por economia de espaço. A matriz é *completa* ($D_{i,j}$=1 para todo par (i,j)).

Limite Inferior do Potencial de Avalanche 4 = 128
Limite Superior do Potencial de Avalanche 4 = 128

12.10.4 CÓDIGO FONTE

Código utilizado para construir transformações lineares - Método dos Nós - e suas respectivas matrizes de dependências (em linguagem java).

```
/*****************************************************************************
 * CRIADO POR JORGE LAMBERT PARA CONSTRUIR TRANSFORMAÇÕES LINEARES EM FORMA DE *
 * MATRIZES n x n COM HMIN=HMAX=m                                              *
 * ENTRA n>15 E m > 3                                                          *
 * RETORNA MATRIZ DE TRANSFORMAÇÃO LINEAR                                      *
 * Created on 03 de Novembro de 2003, 15:20                                    *
 *****************************************************************************/

public class arvore_avalanche_n_m {

public static void main(String[] args) {
//INICIALIZA VARIAVEIS
final int n=16;// maior que 15, entrada. se mudar, corrigir na geracao de k1 tambem
final int m=4;// maior que 3,entrada
final int ultimo_nivel = calcula_ultimo_nivel(n,m);// proximo nivel depois do que contem o no n. para 16 era 4
final int tam_nivel [] = new int [ultimo_nivel+1];// {1,3,9,27,81} = (m-1)^nivel
final int pos_final_nivel [] = new int [ultimo_nivel+1];//para n=16 {0,3,13,40,48}
final int total_acumulado [] = new int [ultimo_nivel+1];//para 16 = {1,4,14,41,48}
final int grupos_no_nivel [] = new int [ultimo_nivel+1];
int valores_proibidos [] = new int [m*m+m];//variavel que armazenara os valores de bites vizinhos
int nr_posicoes_ult_grupo=m-1;
int tentativas=0;
int no [][]= new int [n][m+4];//[ordem=linha=0 a n-1][niv, grupo, posicao, bite escolhido , pai, filho 0, filho 1, ...filho m-2]
                              // tenho m-1 filhos distintos e um pai em cada no
// OBSERVACAO: Seja s(i), e(i) os i-esimos bites da saida e entrada da TL.
// As expressões boolenas são s(BITE ESCOLHIDO) = (e(PAI) + e(FILHO_0),+ e(FILHO_1) + ... + e(FILHO_(m-2))) % 2

int i, j, k, aux_1, aux_2, qvp, aleatorio_inteiro, fase, y;
int [] utz_bytes_bits = new int [2+(n/8)+n+2];//n,m,nr utilizacoes dos n/8 baites, dos n bites, e dois campos p/ bite e baite
escolhido
double aleatorio;

/***************** CALCULOS E ATRIBUICOES INICIAIS**********************/

String gn="";
String ta="";
for (int nivel=0;nivel<ultimo_nivel+1;nivel++){//calculos iniciais
    total_acumulado [nivel]= (potencia((m-1),(nivel+1))-1)/(m-2);
```

```
    if (nivel==ultimo_nivel){
        grupos_no_nivel[ultimo_nivel]= (n-total_acumulado [nivel-1])/(m-1);
        total_acumulado [nivel]=n;
        if (((n-1)%(m-1))>0){// caso em que (n-1) nao é multiplo de (m-1) e terei um grupo incompleto no final
            grupos_no_nivel[ultimo_nivel]=grupos_no_nivel[ultimo_nivel]+1;
            nr_posicoes_ult_grupo= (n-1)%(m-1);
        }
    }
    else grupos_no_nivel[nivel] = potencia((m-1),nivel)/(m-1);//=(m-1)elevado a(nivel-1)
    ta=ta+total_acumulado[nivel]+" ";
    gn=gn+grupos_no_nivel[nivel]+" ";
}// fim do for nivel
System.out.println("Gera TL de  tamanho "+ n +" (" + n + " equacoes) e " +m+ " coeficientes não nulos por linha (equação).");
System.out.println("Ultimo nivel : "+ultimo_nivel);
System.out.println("Numero de grupos por nível: "+gn);
System.out.println("Total de posicoes acumulado ate o nivel, inclusive: "+ta);
System.out.println("Nr de posicoes no ultimo grupo: "+nr_posicoes_ult_grupo);;
utz_bytes_bits [0]=n;//armazena n em utz_bytes_bits p/ que possa ser transportado p/ dentro do metodo escolhe_bite
utz_bytes_bits [1]=m;//armazena m em utz_bytes_bits p/ que possa ser transportado p/ dentro do metodo escolhe_bite

for (i=0;i<n;i++){
    for (j=0;j<m+4;j++){
        no [i][j]=-1;//inicializa todos os campos com flag de vazio=-1
    }//fim do for j
}// fim do for i


/*****************************************************************************
*                   PREENCHIMENTO DA ESTRUTURA ARVORE DA TL                  *
*****************************************************************************/
no[0][0]=0;//nivel do 0-esimo bite escolhido
no[0][1]=0;//grupo do 0-esimo bite escolhido
no[0][2]=0;//posicao do bite 0-esimo bite escolhido dentro do nivel

// preenchimento de todos os campos 3(bite) (indice da linha da TL que receberá a expressao booleana formada pelos filhos),
//e dos campos 0(nivel)
for (i=0;i<n;i++){
     utz_bytes_bits = escolhe_bite(utz_bytes_bits, 0);//escolhe um bite
     no[i][3]=utz_bytes_bits[2+(n/8)+n+1];//atribui ao campo bite o valor escolhido
     utz_bytes_bits = incrementa_nr_utz (no[i][3], utz_bytes_bits);
     for (j=ultimo_nivel;j>-1;j--){// inicio do for j
         if (i<total_acumulado[j]) no[i][0]=j;
          }// fim do for j
     }//fim do for i
```

```
for (i=2;i<(2+n/8+n);i++){
    utz_bytes_bits[i]=0;//anula numero de utilizacoes, pois a numeração feita foi apenas o cmapo linha (lado esquero da equação
booleana)
    }
//aqui tenho todos os campos 3(bite) e 0(nivel) preenchidos
//*************************************************************************

// preenchimento de todos os campos 1(grupo)2(posicao)
for (i=1;i<n;i++){
     float g = (i-total_acumulado[no[i][0]-1])/(m-1);
     int gint = (int) g;
     no [i][1] = gint;
     aux_1 =  i-total_acumulado[no[i][0]-1];
     no [i][2] = aux_1 - ((m-1)* no[i][1]);
     }//fim do for i
//aqui tenho todos os campos 0(nivel)  1(grupo)  2(posicao)  3(bite)
//*************************************************************************

//definicao dos pais do nivel 1 (e reciprocamente, filhos do nivel 0)
for (i=1;i<m;i++){
    no[i][4]=no[0][3];// o campo 'pai' do no i recebe o numero do bite no nó 0
    utz_bytes_bits = incrementa_nr_utz (no[0][3], utz_bytes_bits);
    no[0][4+i]=no[i][3];
    utz_bytes_bits = incrementa_nr_utz (no[i][3], utz_bytes_bits);
    }// fim do for i
no[0][4]=no[n-1][3];//o pai do primeiro no é o último no
utz_bytes_bits = incrementa_nr_utz (no[n-1][3], utz_bytes_bits);
// aqui tenho os campos pai e filho dos niveis 0 e 1


// preenchimento dos campos 4(pai) dos niveis 2 em diante
for (i=m;i<n;i++){
        aux_2= total_acumulado[no[i][0]-2]+no[i][1];
        no[i][4]=no[aux_2][3];//preenche campo pai do no i
        utz_bytes_bits = incrementa_nr_utz (no[aux_2][3], utz_bytes_bits);
        // agora vou ao no pai e coloco este como filho no primeiro campo filho nao preenchido
        aux_1=0;
        for (j=m+3;j>4;j--){//procura um campo filho desocupado (flag '-1')
             if(no[aux_2][j]==-1) aux_1=j;
            }//fim do for j
        if (aux_1 !=0 ){
          no[aux_2][aux_1]=no[i][3];//preeencho no nó pai o respectivo campo deste filho
          utz_bytes_bits = incrementa_nr_utz (no[i][3], utz_bytes_bits);
        }
```

```
     }//fim do for i

//Até aqui todos os valores apareceram como filhos exatamente uma vez

y=imprime_booleanas(no, n ,m);//o valor de y nao sera usado
// y=mostra_utilizacoes_bites(utz_bytes_bits);//linha de controle

/*****************************************************************************
                Preenchimento dos campos filho ainda não preenchidos.
                A partir de agora o critério de preenchimento muda:
             i) cada filho tem que ser diferente dos valores proibidos;
           ii) o bite escolhido é sempre um dos menos utilizados até o momento
 Os valores proibidos são: o pai(1 valor), os tios (m-2 valores) e os (m-1)x(m-1) primos.
               Temos então 1+m-2+(m-1)x(m-1) = m(m-1) valores proibidos
*****************************************************************************/

no[n-1][5]=no[0][3];//ultimo no recebe o primeiro no como filho
utz_bytes_bits = incrementa_nr_utz (no[0][3], utz_bytes_bits);
qvp =m*m+m;//quantidade de valores proibidos m*(m-1). foi utilizada a dimensao m*m porque alguns valores se repetem


for (i=0;i<qvp;i++) valores_proibidos [i]=-1;// preenche todos os campos com flag
for (i=m;i<n;i++){//laço de preenchimento dos filhos
                  //comeco no nó m pois ate o nó m-1(nivel 2) estao todos preenchidos com certeza,
                  //embora seja possivel que alguns nós tambem estejam completos entre m e m+(m-1)^2
    for (j=5;j<(m+4);j++){//verifico campos referentes a filhos
        if (no[i][j]==-1){// se o campo filho está vazio, entro no bloco de preenchimento(este)
            System.out.println("Preenchendo o filho "+j+" do nó "+i+"." );
// y=imprime_booleanas(no, n ,m);//o valor de y nao sera usado
// y=mostra_utilizacoes_bites(utz_bytes_bits);//tirar depois
            valores_proibidos [0]=no[i][3];//o primeiro valor proibido é o do proprio bite: diagonal principal=0
            for (int p=0;p<qvp;p++) valores_proibidos [p]=-1;// preenche todos os campos com flag '-1'
            aux_2=1;//marcador de posicao em valores_proibidos
            System.out.print("Valores proibidos : ");
            for (k=(i-3);k<(i+1);k++){//percorre os nós de i-(m-1) até i para preencher valores_proibidos
               if ((no[k][0]==no[i][0])&& (no[k][1]==no[i][1])){//if mesmo grupo do mesmo nivel
                    for (int z=3;z<(m+4);z++){//proibe os valores do avo, dos tios e dos primos
                       if (z==4)z=5;//nao exclui os pais
                       valores_proibidos [aux_2] = no[k][z];
                       System.out.print(valores_proibidos[aux_2]+" ");
                       aux_2=aux_2+1;
                       if (k==i) /*&& !(no[i][z]==-1)*/ {//proibe valores dos irmaos
                            valores_proibidos[aux_2]=no[k][z];
                            aux_2 = aux_2+1;
```

```
                        }//fim do if ( (k==i) && (no[i][z]!=-1) )
                    }//fim do for z
                }//fim do if mesmo grupo
            }//fim do for k. tenho o vetor com os aux_2 valores proibidos para o filho no[i][j];

            boolean permitido=false;
            tentativas=0;
            while (permitido == false){
                tentativas=tentativas+1;
//                System.out.println("Tentativas: "+tentativas);//linha de controle
                if (tentativas == (45*n)){//diminui exigencias
                       for (int p=1;p<qvp;p++) valores_proibidos [p]=-1;// preenche todos os campos com flag '-1'
                            aux_2=1;//marcador de posicao em valores_proibidos
                                for (int z=5;z<m+4;z++){//proibe os valores do avo, dos tios e dos primos
                                    valores_proibidos [aux_2] = no[i][z];
                                    System.out.print(aux_2+" ");
                                    aux_2 = aux_2 + 1;
                                }//fim do for z
                }// fim do  if (tentativas ==(80*n))
                if (tentativas>(50*n)-1){
                       System.out.println("Mais de "+50*n+" tentativas sem sucesso. Reinicie o programa.");
                }
                fase=0;
                if (tentativas>(10*n)-1){
                       fase=1;
    //                   System.out.println("Fase 2.");//linha de controle
                }
                if (tentativas>(30*n)-1){
                       fase=2;
      //                 System.out.println("Fase 3.");// linha de controle
                }
                if (tentativas>(40*n)-1){
                       fase=3;
        //                 System.out.println("Fase 3."); // linha de controle
                }
                utz_bytes_bits = escolhe_bite(utz_bytes_bits, fase);
                no[i][j] = utz_bytes_bits[2+n/8+n+1];
                permitido = true;
                for (k=0;k<aux_2;k++){//testa se o valor escolhido esta entre os valores proibidos
                    if (valores_proibidos[k]==no[i][j]) permitido = false;
                    }//fim do for k
                }//fim do while
            utz_bytes_bits = incrementa_nr_utz (no[i][j], utz_bytes_bits);
      }//fim do if (no[i][j]=-1)
```

```
    }//fim do for j
}//fim do for i


y=imprime_booleanas(no, n ,m);//o valor de y nao sera usado
//y=mostra_utilizacoes_bites(utz_bytes_bits);// linha de controle

//Construcao da TL com base nas Expressoes Booleanas (nos nós)
int TL[][] = new int [n][n];
for (i=0;i<n;i++){
    for (j=4;j<m+4;j++){
        TL [no[i][3]][no[i][j]]=1;
    }
}

int Avalanche [][][] = avalanche(TL, n, 4);


}// FIM DE MAIN



/**************************************************************************
*                            metodo escolhe_bite                          *
*                      argumento = vetor int [n/8+n+2]                    *
*                       retorna   = vetor int [n/8+n+2]                   *
*   os dois primeiros sao n e m                                           *
*   os seguintes n/8 campos são os nr de utilizacoes dos baites           *
*   os seguintes n campos sao os nr de utilizacoes dos bites              *
*   os últimos 2 campos sao para o baite e bite escolhido retornarem      *
**************************************************************************/
public static int [] escolhe_bite(int entrada[], int fase){//
int n=entrada [0];
int m=entrada [1];
int nrbytes=n/8;
int auxiliar=0;
int relaxa = 0;
int baite=-1;
double aleatorio=0;
int aleatorio_inteiro=0;
int k=0;
boolean permitido=false;
//escolhe um baite entre os menos usados
int menor_nr_utilizacoes=8*m;//inicializa com o valor maximo de utilizacoes para um baite
```

```
for (int c1=2;c1<nrbytes+2;c1++){//laco para ver qual o menor numero de utilizacoes por baite
     if (entrada[c1] < menor_nr_utilizacoes){
         menor_nr_utilizacoes = entrada [c1];
//            System.out.println("men nr utz "+menor_nr_utilizacoes); // linha de controle
         }//fim do if
     }//fim do for c1
//aqui tenho em menor_nr_utz armazenada a quantidade de vezes que o(s) baite(s) menos usado(s) foi(ram) usado(s)
//if (menor_nr_utilizacoes==(m*8)){//testa se todos foram usados 8m vezes. caso positivo, fim
//    System.out.println("Todos os baites foram usados "+ 8*m +"  vezes!");
//    return entrada;
//    }
while (permitido==false){
    baite=-1;// inicio com flag
    aleatorio = Math.random()*(nrbytes);
    aleatorio_inteiro = (int) aleatorio;
//            System.out.println("aleatorio_inteiro "+aleatorio_inteiro); // linha de controle
           permitido=false;
           auxiliar = menor_nr_utilizacoes + fase + 1;
           for (relaxa=menor_nr_utilizacoes;relaxa<auxiliar;relaxa++){
               if (entrada [2 + aleatorio_inteiro] == relaxa) permitido = true;
           } // fim do for
           if ((aleatorio_inteiro == entrada[2+n/8+n]) && (k<2*n)){
               permitido = false;
               k=k+1; // conta o numerro de vezes que o baite foi consideradp invalido
                                 // por ser igual ao anterior
           }
           if (permitido){
               //testa se o baite sorteado é um dos menos utilizados e
               //é diferente do ultimo sorteado (pode ocorrer qdo todos foram utilizados
               //a mesma qtd de vezes)
               baite = aleatorio_inteiro;
               entrada [2+nrbytes+n] = baite;//qual o baite que sera usado
           }//fim do if permitido
}// fim do while  baite

// aqui tenho o baite escolhido
// agora escolho um dos bites menos usados dentro do baite menos usado(escolhido)
        menor_nr_utilizacoes = m+1;//inicializa com o valor maximo de utilizacoes para um bite
        int aux = 2 + nrbytes + baite * 8 ;//posicao do bite inicial do baite. o 2 e pq as 2 pos iniciais sao para n e m
                                  //o nr bytes porque as posicoes seguintes sao para armazenar as utz dos baites
                                  //em entrada [aux] esta armazenada a quantidade de vezes que o bit 0 do baite baite foi usado
        for (int c1=aux;c1<aux+8;c1++){//laco para ver qual o menor numero de utilizacoes por bite dentro do baite
             if (entrada [c1] < menor_nr_utilizacoes){
               menor_nr_utilizacoes = entrada[c1];
```

```
                }//fim do if
              }//fim do for c1
//        if ( menor_nr_utilizacoes <(m-fase)) menor_nr_utilizacoes = menor_nr_utilizacoes + fase;//relaxa condicao do bite menos
usado
        int bite_escolhido=-1;
        int bite_do_baite=-1;
        relaxa=0;
        while (bite_escolhido < 0){
      //      System.out.println("Entrou do while esolhe bite!"); // linha de controle
            aleatorio = Math.random()*8;
            aleatorio_inteiro = (int) aleatorio;
            bite_do_baite = aleatorio_inteiro;
            auxiliar = aux + aleatorio_inteiro;
            relaxa = menor_nr_utilizacoes;
            if ( menor_nr_utilizacoes <(m-fase)){;//relaxa condicao do bite menos usado
                relaxa = menor_nr_utilizacoes + (int) (Math.random()*(fase+1));//se fase = 0, nada muda
            }
            if (entrada [auxiliar] == relaxa){
                bite_escolhido = 8 * baite + bite_do_baite;
                entrada [ 2 + nrbytes + n + 1 ] = bite_escolhido;//define bite será utilizado
            }//fim do if
        }// fim do while  bite
 //           System.out.println("Saiu do while esolhe bite!"); // linha de controle
// aqui tenho o bite escolhido


return entrada;
}// ******************** fim do metodo escolhe bite  *********************/



/*************************************************************************
*                             metodo imprime_booleanas                   *
*************************************************************************/
public static int imprime_booleanas(int no[][],int n,int m){//
System.out.println("   ");
System.out.println("   nó  :  nivel, grp , pos, bit ,pai , f1 , f2 , f3...");
for (int i=0;i<n;i++){
    String temp="   Nó  "+i+"  ";
    if (i<100) temp=temp+" ";//ajusta alinhamento
    if (i<10) temp=temp+" ";//ajusta alinhamento
    for (int j=0;j<m+4;j++){
            if (no[i][j]<100) temp=temp+" ";//ajusta alinhamento
            if (no[i][j]<10) temp=temp+" ";//ajusta alinhamento
```

```
        temp= temp + no[i][j]+ ", ";
     }
System.out.println(temp);
}
    return n;
}// ******************** fim do metodo imprime_booleanas *********************/



/*************************************************************************
*                      metodo potencia                                   *
*                      argumento = int a, int b                          *
*                      retorna   = a elevado a b                         *
*          (criado para nao ter que usar variavel double em Math.pow     *
*************************************************************************/
public static int potencia(int a, int b){//
int pot=1;
for (int i=0;i<b;i++) pot=pot*a;
return pot;
}// ******************** fim do metodo potencia  *********************/


/*************************************************************************
*                      metodo calcula_ultimo_nivel                       *
*                      argumento = int n, int m, (n > m)                 *
*                      retorna   = nivel em que se encontra o m*n        *
*                      em uma pg de razao m-1                            *
*************************************************************************/
public static int calcula_ultimo_nivel(int n, int m){//
int ult_niv=0;
int soma_termos=0;
int level=-1;
while (soma_termos<n){
    level=level+1;
    soma_termos = (potencia((m-1),(level+1))-1)/(m-2);
    }
return level;
}// ******************** fim do metodo calcula_ultimo_nivel   *********************/

/*************************************************************************
*                      metodo incrementa_nr_utz                          *
*                      argumento = int bit, int []                       *
*                      retorna   = utz_bytes_bits atualizado             *
*************************************************************************/
```

```
public static int [] incrementa_nr_utz (int a, int [] b){//
int n=b[0];
int m=b[1];
int posicao_do_baite = (int) (2+a/8);
int posicao_do_bite = 2+n/8+a;
b[posicao_do_baite]=b[posicao_do_baite]+1;
b[posicao_do_bite]=b[posicao_do_bite]+1;
return b;
}// ******************** incrementa_nr_utz    *********************/

/*************************************************************************
*                     metodo mostra_utilizacoes_bites                    *
*                     argumento = int []                                 *
*                     retorna   = int []                                 *
*                     em uma pg de razao m-1                             *
*************************************************************************/
public static int mostra_utilizacoes_bites(int utz[]){//
int n=utz[0];
int aux=0;
String temp;
for (int i=0;i<n;i++){
   aux=utz[2+n/8+i];
   System.out.println("O Bit "+i+" foi utilizado "+aux+" vezes.");
   }// fim do for i
return aux;
}// ****************** fim do metodo mostra_utilizacoes_bites   ******************/


/*************************************************************************
*                           metodo AVALANCHE                             *
*                     argumento = 1 vetores int [n][n]                   *
*                        retorna =   int [n][n][Nr]                      *
*                   Construçao das Matrizes de Avalanche                 *
*                     e    calculo dos respectivos                       *
*             Limites Inferiores de Potencial de Avalanche               *
*************************************************************************/
public static int [][][] avalanche(int trans [][], int n, int Nr){//

int LIPA []=new int[Nr];
int LSPA []=new int[Nr];
int Avalanche [][][]= new int [n][n][Nr];
String temp="";// apenas declaracao da variavel
for (int i=0 ; i<n ; i++){
    for (int j=0; j<n; j++){
```

```
        Avalanche [i][j][0] = trans[i][j];//atribui a matriz TL à Avalanche de primeira ordem
        }//  fim do for j
    }// fim do for i

//////////////////////////// EXIBIÇÃO ////////////////////////////

System.out.println("");
System.out.println("");
temp= "Matriz de Dependências da TL ";
System.out.println(temp);
LIPA[0]=n;
LSPA[0]=0;

for (int i=0;i<n;i++){// laco que gera impressao da matriz 128x128
       temp = "";
       int c=0;// acumulador de peso hamming da linha i
        for (int j=0;j<n;j++){//
                if (Avalanche[i][j][0]==0){
                  temp=temp+".";
                }// fim do if
                  else{
                   temp=temp+"1";
                   c=c+1;//soma 1 ao peso hamming da linha
                }// fim do else
          }// fim do for j
            if (c<LIPA[0]) {
               LIPA[0]=c;//LIPA[0] recebe o menor peso hamming de todas as linhas da matriz
           }// fim do if
            if (c>LSPA[0]) {
               LSPA[0]=c;//LSPA[ordem_menos_um] recebe o menor peso hamming de todas as linhas da matriz
           }// fim do if
       System.out.println(temp); // exibe a linha i de Avalanche
     }// fim do for i

        temp="Limite Inferior do Potencial de Avalanche Ordem 0 = " + LIPA[0];
        System.out.println(temp); // exibe LIPA[ordem_menos_um]
        temp="Limite Superior do Potencial de Avalanche Ordem 0 = " + LSPA[0];
        System.out.println(temp); // exibe LSPA[ordem_menos_um]

////////////////////////////////////////////////////////////////////////////////

for (int o_menos_1=1; o_menos_1< Nr; o_menos_1 ++) {

    // Montagem da matriz de ordem  ordem_menos_um + 1 (ordem 2, 3, ...Nr)
```

```
    for (int i=0;i<n;i++){// perccorro a linha i da matriz de avalanche que esta sendo montada (ordem =ordem_menos_um + 1)
        for (int j=0;j<n;j++){//
            if (Avalanche[i][j][0] == 1){// se na matriz Avalanche de orimeira ordem o j-esimo bite da entrada afeta o i-esimo
bite da saida
                    for (int c=0; c<n;c++){//percorre a linha j da mantriz de avalanche anterior
                    // a ideia é: se o j-esimo bite da entrada anterior interfere no i-esimo bite da saída anterior (olho na de
matriz de primeira ordem) ,
                    // todos os bites que interferiram no j-ésimo bite da saida na matriz anterior interferem
                    // no i-ésimo bite da saida atual por recorrencia
                           if (Avalanche[j][c][o_menos_1-1]==1) Avalanche[i][c][o_menos_1]=1;//seta na lin i da matriz os
                           // bites que estão setados na linha j da matriz anterior
                    }//fim do for c
                }// fim do if
        }// fim do for j
    }// fim do for i
//montada a matriz



/*****************************************************************************
* Exibicao das Matrizes de Avalanche (para mais de um passo da TL)           *
*****************************************************************************/
temp= "Matriz de Avalanche de Ordem "+(o_menos_1+1);
System.out.println(temp);
LIPA [o_menos_1]= n;//Atribui valor máximo a LIPA[ordem_menos_um]
LSPA [o_menos_1]= 0;//Atribui valor mínimo a LSPA[ordem_menos_um]
    for (int i=0;i<n;i++){// laco que gera impressao da matriz nxn
       temp = "";
       int c=0;// acumulador de peso hamming da linha i
        for (int j=0;j<n;j++){//
                if (Avalanche[i][j][o_menos_1]==0){
                  temp=temp+".";
                }// fim do if
                  else{
                   temp=temp+"1";
                   c=c+1;//soma 1 ao peso hamming da linha
                }// fim do else
          }// fim do for j
            if (c<LIPA[o_menos_1]) {
               LIPA[o_menos_1]=c;//LIPA[ordem_menos_um] recebe o menor peso hamming de todas as linhas da matriz
           }// fim do if
            if (c>LSPA[o_menos_1]) {
               LSPA[o_menos_1]=c;//LSPA[ordem_menos_um] recebe o menor peso hamming de todas as linhas da matriz
           }// fim do if
```

```
        System.out.println(temp); // exibe a linha i de Avalanche
      }// fim do for i

         temp="Limite Inferior do Potencial de Avalanche Ordem "+(o_menos_1+1)+" = "+LIPA[o_menos_1];
         System.out.println(temp); // exibe LIPA[ordem_menos_um]
         temp="Limite Superior do Potencial de Avalanche Ordem "+(o_menos_1+1)+" = "+LSPA[o_menos_1];
         System.out.println(temp); // exibe LSPA[ordem_menos_um]
         System.out.println("");
         System.out.println("");
}// fim de for ordem_menos_um

return Avalanche;

}// ******************** fim do metodo avalanche  *********************/

}//fim da classe arvore

//ALGUNS MELHORAMENTOS QUE PODEM SER FEITOS NO CÓDIGO FONTE
//corrigir variavel utz_baytes bites para ser operada de dentro do método escolhe bites
//fazer chamada de metodos sem ter que transportar valores para dentro e para fora
//melhorar geração de aleatórios
```

12.11 APÊNDICE 11: QUADRADOS LATINOS E MATRIZES DE DISTRIBUIÇÕES LINEARES E DIFERENCIAIS

12.11.1 RESULATDOS DE DISTRIBUIÇÃO LINEAR E DIFERENCIAL

A seguir são apresentados os resultados dos testes de distribuição diferencial e linear para os 10 QL com melhores características escolhidos em um universo de 9730. É apresentado apenas o resultado completo do primeiro QL e o resumo dos demais.

12.11.1.1 TABELA NÚMERO 0

```
Tabela de Substituição
{15,5,13,11,0,4,8,10,1,12,9,14,6,3,2,7},
{2,8,0,14,3,7,11,13,4,15,12,1,9,6,5,10},
{7,13,5,3,8,12,0,2,9,4,1,6,14,11,10,15},
{14,4,12,10,15,3,7,9,0,11,8,13,5,2,1,6},
{6,12,4,2,7,11,15,1,8,3,0,5,13,10,9,14},
{10,0,8,6,11,15,3,5,12,7,4,9,1,14,13,2},
{13,3,11,9,14,2,6,8,15,10,7,12,4,1,0,5},
{5,11,3,1,6,10,14,0,7,2,15,4,12,9,8,13},
{11,1,9,7,12,0,4,6,13,8,5,10,2,15,14,3},
{12,2,10,8,13,1,5,7,14,9,6,11,3,0,15,4},
{4,10,2,0,5,9,13,15,6,1,14,3,11,8,7,12},
{1,7,15,13,2,6,10,12,3,14,11,0,8,5,4,9},
{3,9,1,15,4,8,12,14,5,0,13,2,10,7,6,11},
{8,14,6,4,9,13,1,3,10,5,2,7,15,12,11,0},
{9,15,7,5,10,14,2,4,11,6,3,8,0,13,12,1},
{0,6,14,12,1,5,9,11,2,13,10,15,7,4,3,8},
};
```

```
Matriz de distribuicao linear
00000000(0):  {256,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00000001(1):  {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00000010(2):  {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00000011(3):  {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00000100(4):  {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00000101(5):  {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00000110(6):  {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00000111(7):  {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00001000(8):  {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00001001(9):  {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00001010(10): {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00001011(11): {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00001100(12): {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00001101(13): {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00001110(14): {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00001111(15): {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00010000(16): {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00010001(17): {128,128,136,120,116,132,132,132,132,148,124,140,120,136,136,120},
00010010(18): {128,128,136,136,132,124,124,132,116,132,132,132,120,112,136,128},
00010011(19): {128,128,112,128,120,128,128,136,128,128,136,120,120,128,120,128},
00010100(20): {128,96, 136,120,132,124,124,132,136,128,120,128,124,124,124,140},
00010101(21): {128,96, 128,128,128,120,136,128,124,132,132,124,124,124,124,108},
00010110(22): {128,128,128,128,120,120,136,120,124,132,124,116,132,108,132,140},
00010111(23): {128,128,120,136,132,132,132,132,120,144,112,120,124,132,124,132},
00011000(24): {128,128,136,136,124,132,124,116,128,144,128,128,124,132,116,124},
00011001(25): {128,128,128,144,120,128,120,128,132,116,124,124,124,116,132,124},
00011010(26): {128,128,128,128,120,120,128,128,124,108,124,140,132,148,124,140},
00011011(27): {128,128,120,136,132,132,140,124,136,120,128,144,124,124,132,132},
00011100(28): {128,96, 112,128,128,128,120,120,128,136,128,136,128,136,136,128},
00011101(29): {128,160,120,120,132,116,124,124,132,140,124,132,120,128,128,120},
00011110(30): {128,128,120,120,124,132,124,132,124,132,124,132,136,104,128,128},
00011111(31): {128,128,128,144,128,120,128,136,136,144,128,136,136,136,128,128},
00100000(32): {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00100001(33): {128,128,144,120,128,120,112,128,120,144,120,120,128,112,128,136},
00100010(34): {128,128,136,144,132,132,132,124,120,120,120,128,140,124,132,140},
00100011(35): {128,128,104,120,132,124,116,124,128,120,128,136,124,124,132,116},
00100100(36): {128,144,144,120,132,132,132,124,124,124,124,116,120,136,120,128},
00100101(37): {128,144,128,128,124,132,124,132,132,124,132,140,128,112,144,128},
```

```
00100110(38): {128,128,128,128,128,128,120,136,124,108,132,116,132,132,116,132},
00100111(39): {128,128,112,136,120,128,128,128,116,124,140,108,124,124,124,116},
00101000(40): {128,128,136,144,132,124,132,132,128,128,128,120,124,116,132,116},
00101001(41): {128,128,120,152,140,124,124,124,128,120,128,136,124,132,132,140},
00101010(42): {128,128,128,128,136,128,120,128,128,144,128,128,128,152,128,120},
00101011(43): {128,128,112,136,112,128,128,136,128,136,112,128,128,136,128,144},
00101100(44): {128,144,104,120,136,128,136,128,124,140,124,124,132,124,124,132},
00101101(45): {128,112,120,112,136,136,136,128,124,132,124,124,124,116,132,132},
00101110(46): {128,128,120,112,132,124,124,124,132,116,124,116,136,128,128,144},
00101111(47): {128,128,120,152,132,132,124,124,132,140,124,116,128,120,120,128},
00110000(48): {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
00110001(49): {128,128,136,128,124,132,140,124,132,124,124,124,136,120,120,128},
00110010(50): {128,128,128,136,128,136,120,136,156,124,124,116,124,132,132,132},
00110011(51): {128,128,120,120,124,140,132,132,136,128,128,120,140,132,148,124},
00110100(52): {128,112,136,128,128,136,120,136,124,132,140,124,148,132,132,124},
00110101(53): {128,112,128,128,140,124,132,148,128,128,136,136,124,124,124,140},
00110110(54): {128,128,128,128,136,104,144,144,128,120,120,128,128,136,128,120},
00110111(55): {128,128,120,128,132,140,124,124,140,124,140,132,128,120,112,128},
00111000(56): {128,128,128,136,120,104,136,112,128,128,144,120,128,128,144,136},
00111001(57): {128,128,120,136,116,124,132,124,124,132,132,140,144,128,112,128},
00111010(58): {128,128,128,128,128,120,136,144,140,140,140,124,116,124,124,132},
00111011(59): {128,128,120,128,140,124,132,124,128,136,136,120,140,132,132,132},
00111100(60): {128,112,120,120,120,112,128,120,132,124,116,124,124,124,124,124},
00111101(61): {128,144,128,120,132,116,124,148,128,128,136,128,140,124,124,132},
00111110(62): {128,128,128,120,120,136,128,120,120,128,144,128,112,136,128,144},
00111111(63): {128,128,120,136,132,140,124,148,108,124,124,124,120,128,136,128},
01000000(64): {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
01000001(65): {128,128,136,120,124,132,124,132,128,128,136,120,132,140,116,124},
01000010(66): {128,128,136,136,132,116,132,132,140,132,124,116,136,128,112,120},
01000011(67): {128,128,112,128,128,120,128,136,124,132,116,140,140,124,132,132},
01000100(68): {128,128,136,120,132,116,132,132,116,124,140,132,136,128,136,112},
01000101(69): {128,128,128,128,120,112,136,112,132,140,116,124,132,132,132,148},
01000110(70): {128,128,128,128,112,128,128,96, 128,128,136,136,120,120,128,112},
01000111(71): {128,128,120,136,132,124,132,140,144,128,144,144,132,124,140,116},
01001000(72): {128,128,136,136,124,156,116,116,128,128,120,136,132,132,140,124},
01001001(73): {128,128,128,144,128,136,120,128,120,136,136,120,136,128,128,136},
01001010(74): {128,128,128,128,120,120,128,112,140,116,132,124,132,124,132,124},
01001011(75): {128,128,120,136,124,132,132,124,116,124,132,108,136,136,120,120},
01001100(76): {128,128,112,128,128,144,120,120,132,124,132,124,124,132,116,124},
```

```
01001101(77): {128,128,120,120,124,116,132,108,124,132,132,124,136,136,128,128},
01001110(78): {128,128,120,120,132,132,124,140,128,128,128,112,116,132,132,116},
01001111(79): {128,128,128,144,128,104,136,128,120,120,128,128,112,136,128,120},
01010000(80): {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
01010001(81): {128,128,128,112,136,128,136,128,124,124,124,124,124,132,124,116},
01010010(82): {128,128,144,128,120,128,128,120,120,128,136,112,120,136,144,128},
01010011(83): {128,128,112,144,128,128,136,120,140,132,124,116,124,132,148,140},
01010100(84): {128,128,128,112,120,128,128,120,140,124,132,132,116,124,132,124},
01010101(85): {128,128,128,128,128,144,120,120,120,136,128,128,128,128,112,144},
01010110(86): {128,128,128,128,136,152,120,136,132,124,124,132,132,124,140,116},
01010111(87): {128,128,128,144,128,120,128,120,120,128,128,152,120,120,128,128},
01011000(88): {128,128,144,128,136,128,128,152,136,136,144,128,120,128,136,144},
01011001(89): {128,128,144,144,128,128,136,136,124,124,116,132,124,124,108,124},
01011010(90): {128,128,128,128,128,144,128,128,120,128,144,136,120,128,128,136},
01011011(91): {128,128,128,144,136,128,120,128,132,124,140,116,116,132,124,124},
01011100(92): {128,128,112,144,128,128,128,144,124,124,124,124,132,132,132,116},
01011101(93): {128,128,112,128,120,144,120,128,128,128,128,128,120,128,120,128},
01011110(94): {128,128,112,128,128,120,136,128,124,132,156,132,124,124,116,132},
01011111(95): {128,128,144,144,120,136,128,112,136,128,136,128,136,128,128,120},
01100000(96): {128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
01100001(97): {128,128,120,128,124,124,124,132,128,136,120,136,140,148,124,140},
01100010(98): {128,128,128,120,128,128,128,120,140,148,124,124,148,124,132,132},
01100011(99): {128,128,136,136,124,124,124,124,124,124,132,132,128,128,144,112},
01100100(100):{128,112,120,128,128,128,128,120,104,128,128,112,128,136,128,128},
01100101(101):{128,112,128,128,124,140,124,124,136,136,136,152,124,124,140,124},
01100110(102):{128,128,128,128,128,144,120,136,124,140,124,108,124,124,116,116},
01100111(103):{128,128,136,128,124,124,132,124,124,132,148,116,136,144,112,112},
01101000(104):{128,128,128,120,136,128,128,144,128,128,136,128,120,144,136,120},
01101001(105):{128,128,136,120,132,124,124,132,112,136,128,136,132,116,132,132},
01101010(106):{128,128,128,128,128,136,120,128,132,108,140,116,140,124,124,140},
01101011(107):{128,128,136,128,124,132,132,132,116,132,116,124,136,128,136,152},
01101100(108):{128,112,136,136,136,128,128,136,136,128,120,128,136,136,120,136},
01101101(109):{128,144,128,136,132,140,124,124,120,136,128,136,132,140,132,132},
01101110(110):{128,128,128,136,128,120,128,128,132,148,124,116,116,124,124,140},
01101111(111):{128,128,136,120,124,132,124,116,132,124,116,124,112,144,120,136},
01110000(112):{128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
01110001(113):{128,128,128,136,144,128,128,120,116,140,132,116,132,124,132,116},
01110010(114):{128,128,120,128,124,132,132,132,120,128,152,120,156,124,124,132},
01110011(115):{128,128,136,120,140,132,132,124,116,132,132,132,120,128,136,144},
```

```
01110100(116):{128,112,128,136,124,132,132,132,104,120,128,136,108,132,140,124},
01110101(117):{128,112,128,128,132,124,124,132,124,132,116,124,136,120,120,136},
01110110(118):{128,128,128,128,136,120,128,128,136,112,136,128,128,120,136,128},
01110111(119):{128,128,128,120,128,128,120,128,116,132,132,140,148,116,140,116},
01111000(120):{128,128,120,128,124,124,132,124,120,136,128,136,116,116,124,132},
01111001(121):{128,128,120,120,132,132,140,124,108,116,116,124,128,136,136,128},
01111010(122):{128,128,128,128,136,128,136,128,128,136,136,144,144,144,120,120},
01111011(123):{128,128,128,120,128,120,112,128,108,124,132,124,116,140,124,124},
01111100(124):{128,112,136,120,120,128,136,128,136,136,136,120,136,128,128,120},
01111101(125):{128,144,136,128,120,128,136,136,108,132,124,124,124,124,132,124},
01111110(126):{128,128,136,128,132,132,132,124,136,112,128,128,124,116,108,124},
01111111(127):{128,128,120,120,132,132,132,132,148,132,124,124,104,120,120,120},
10000000(128):{128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
10000001(129):{128,128,144,144,120,136,152,136,128,128,128,128,128,128,128,128},
10000010(130):{128,128,112,144,120,120,120,120,128,128,128,128,128,128,128,128},
10000011(131):{128,128,128,96, 112,128,144,128,128,128,128,128,128,128,128,128},
10000100(132):{128,128,144,144,120,120,120,120,128,128,128,128,128,128,128,128},
10000101(133):{128,128,128,128,128,144,128,112,128,128,128,128,128,128,128,128},
10000110(134):{128,128,128,128,128,160,128,128,128,128,128,128,128,128,128,128},
10000111(135):{128,128,112,112,136,120,136,120,128,128,128,128,128,128,128,128},
10001000(136):{128,128,112,144,136,136,120,152,128,128,128,128,128,128,128,128},
10001001(137):{128,128,96, 128,112,128,128,144,128,128,128,128,128,128,128,128},
10001010(138):{128,128,128,128,112,144,128,128,128,128,128,128,128,128,128,128},
10001011(139):{128,128,112,112,152,136,136,120,128,128,128,128,128,128,128,128},
10001100(140):{128,128,128,96, 128,128,112,144,128,128,128,128,128,128,128,128},
10001101(141):{128,128,144,112,120,136,104,120,128,128,128,128,128,128,128,128},
10001110(142):{128,128,144,112,120,120,136,136,128,128,128,128,128,128,128,128},
10001111(143):{128,128,96, 128,112,128,128,112,128,128,128,128,128,128,128,128},
10010000(144):{128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
10010001(145):{128,128,120,120,140,140,156,124,124,140,132,132,128,112,128,128},
10010010(146):{128,128,136,120,116,124,124,132,132,116,116,132,144,120,128,120},
10010011(147):{128,128,128,144,128,136,152,128,120,120,112,128,136,128,120,112},
10010100(148):{128,128,120,120,116,124,124,132,120,128,120,128,132,132,132,132},
10010101(149):{128,128,128,128,136,128,128,120,132,124,140,116,124,124,140,124},
10010110(150):{128,128,128,128,136,136,136,120,124,116,124,132,132,124,116,140},
10010111(151):{128,128,136,136,140,124,124,124,128,136,136,112,132,124,132,140},
10011000(152):{128,128,136,120,124,132,124,132,128,128,112,112,132,124,140,116},
10011001(153):{128,128,144,128,112,136,144,136,124,124,148,132,124,132,132,140},
10011010(154):{128,128,128,128,120,136,144,128,140,140,124,124,132,148,140,124},
```

```
10011011(155):{128,128,136,136,156,124,116,116,128,128,120,136,132,132,140,124},
10011100(156):{128,128,128,144,112,128,120,136,128,136,128,136,128,120,120,128},
10011101(157):{128,128,120,136,108,124,116,132,124,132,132,124,128,120,136,128},
10011110(158):{128,128,120,136,124,132,140,132,140,116,124,116,128,128,152,136},
10011111(159):{128,128,144,128,120,128,136,128,128,136,120,128,120,120,128,128},
10100000(160):{128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
10100001(161):{128,128,128,136,152,128,120,120,136,144,120,136,128,128,128,120},
10100010(162):{128,128,120,128,124,124,140,132,144,128,128,120,116,116,140,132},
10100011(163):{128,128,136,120,148,124,132,124,136,128,136,112,132,132,140,124},
10100100(164):{128,144,128,136,124,124,140,132,132,132,132,124,144,128,128,136},
10100101(165):{128,144,128,128,124,116,124,116,124,132,140,132,120,120,120,120},
10100110(166):{128,128,128,128,128,128,120,104,124,124,116,116,132,116,132,132},
10100111(167):{128,128,128,120,128,120,120,136,132,140,124,124,124,124,108,132},
10101000(168):{128,128,120,128,124,148,124,124,128,144,144,120,124,132,132,132},
10101001(169):{128,128,120,120,140,140,140,124,128,120,128,136,140,116,116,124},
10101010(170):{128,128,128,128,136,128,136,112,136,120,136,136,120,144,120,144},
10101011(171):{128,128,128,120,120,120,104,128,136,128,136,136,136,128,136,136},
10101100(172):{128,144,136,120,120,144,136,128,132,132,116,132,124,132,132,124},
10101101(173):{128,112,136,128,112,128,144,120,132,140,132,132,132,124,124,124},
10101110(174):{128,128,136,128,140,132,132,132,116,132,140,132,120,112,128,144},
10101111(175):{128,128,120,120,132,116,140,124,116,140,124,132,128,136,136,128},
10110000(176):{128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
10110001(177):{128,128,120,128,116,140,148,132,124,132,116,116,128,128,128,136},
10110010(178):{128,128,128,120,128,120,120,136,116,132,132,124,124,132,132,148},
10110011(179):{128,128,136,136,116,132,140,140,136,128,144,136,116,124,140,132},
10110100(180):{128,144,120,128,128,120,120,136,132,124,132,116,116,132,116,124},
10110101(181):{128,144,128,128,132,116,140,124,128,128,120,152,132,116,132,132},
10110110(182):{128,128,128,128,120,120,144,112,128,120,136,112,128,136,128,120},
10110111(183):{128,128,136,128,140,132,132,132,116,116,132,124,120,128,120,104},
10111000(184):{128,128,128,120,120,136,120,112,128,128,144,136,120,120,120,128},
10111001(185):{128,128,136,120,108,132,124,132,132,124,108,132,128,128,128,128},
10111010(186):{128,128,128,128,112,120,136,128,116,132,132,132,124,132,116,124},
10111011(187):{128,128,136,128,148,132,140,116,128,136,120,120,124,132,116,148},
10111100(188):{128,144,136,136,120,128,112,120,124,132,124,116,132,132,132,132},
10111101(189):{128,112,128,136,124,108,116,124,128,128,120,128,124,124,124,132},
10111110(190):{128,128,128,136,120,136,128,136,120,128,128,128,136,128,104,136},
10111111(191):{128,128,136,120,124,116,132,140,132,132,132,116,136,128,120,128},
11000000(192):{128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
11000001(193):{128,128,136,120,156,132,140,116,136,120,128,128,132,140,132,140},
```

```
11000010(194):{128,128,136,136,116,132,132,132,132,140,132,140,120,128,112,136},
11000011(195):{128,128,112,128,144,136,144,120,124,132,148,140,124,124,116,132},
11000100(196):{128,128,136,120,116,132,132,132,124,132,132,124,136,128,120,128},
11000101(197):{128,128,128,128,136,128,120,128,132,124,116,140,132,132,132,116},
11000110(198):{128,128,128,128,144,128,128,128,128,144,136,120,120,136,128,128},
11000111(199):{128,128,120,136,132,124,116,124,136,120,120,120,132,140,124,116},
11001000(200):{128,128,136,136,124,124,132,132,120,120,128,144,132,132,108,124},
11001001(201):{128,128,128,144,128,136,152,128,136,136,120,120,136,128,144,120},
11001010(202):{128,128,128,128,136,136,144,128,124,116,116,124,132,108,116,124},
11001011(203):{128,128,120,136,140,116,100,124,124,132,124,132,136,120,120,136},
11001100(204):{128,128,112,128,112,128,136,136,132,124,132,124,124,132,132,140},
11001101(205):{128,128,120,120,108,132,132,140,132,124,124,132,136,136,128,128},
11001110(206):{128,128,120,120,132,132,140,124,120,136,104,136,132,132,116,116},
11001111(207):{128,128,128,144,128,136,136,128,120,120,128,128,128,136,128,136},
11010000(208):{128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
11010001(209):{128,128,128,128,136,128,120,128,124,108,124,124,132,108,132,140},
11010010(210):{128,128,128,128,128,120,136,128,128,120,128,136,128,144,136,136},
11010011(211):{128,128,128,128,136,120,128,128,132,124,132,124,124,132,132,124},
11010100(212):{128,96, 128,128,128,120,136,128,132,132,124,124,124,132,124,132},
11010101(213):{128,96, 128,128,120,120,128,112,128,112,136,136,128,128,128,128},
11010110(214):{128,128,128,128,120,136,120,104,132,124,124,132,132,156,124,132},
11010111(215):{128,128,128,128,128,120,128,136,120,112,128,120,128,128,120,136},
11011000(216):{128,128,128,128,128,152,120,128,128,112,136,120,128,120,128,136},
11011001(217):{128,128,128,128,136,136,128,128,132,132,124,140,124,140,124,124},
11011010(218):{128,128,128,128,128,128,128,112,120,160,128,120,120,112,128,120},
11011011(219):{128,128,128,128,120,128,120,128,132,140,124,132,124,124,132,132},
11011100(220):{128,96, 128,128,128,144,128,128,124,124,124,124,132,116,132,132},
11011101(221):{128,160,128,128,120,128,136,112,128,112,128,128,128,120,128,136},
11011110(222):{128,128,128,128,136,128,128,136,132,124,132,140,132,148,124,140},
11011111(223):{128,128,128,128,128,112,136,120,128,120,128,120,136,112,128,136},
11100000(224):{128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
11100001(225):{128,128,120,128,124,124,124,132,136,144,128,112,124,116,124,124},
11100010(226):{128,128,128,120,128,128,128,120,140,116,124,124,108,116,108,140},
11100011(227):{128,128,136,136,124,124,124,124,132,132,140,140,136,120,120,136},
11100100(228):{128,112,120,128,128,128,128,120,136,128,144,128,152,128,120,120},
11100101(229):{128,112,128,128,124,140,124,124,128,128,112,128,132,116,132,132},
11100110(230):{128,128,128,128,128,144,120,136,124,108,140,124,124,124,132,132},
11100111(231):{128,128,136,128,124,124,132,124,148,124,124,124,120,112,128,112},
11101000(232):{128,128,128,120,136,128,128,144,120,136,128,136,136,112,120,120},
```

```
11101001(233):{128,128,136,120,132,124,124,132,144,120,128,120,148,132,132,132},
11101010(234):{128,128,128,128,128,136,120,128,140,132,116,140,116,148,116,116},
11101011(235):{128,128,136,128,124,132,132,132,132,132,132,124,144,136,112,128},
11101100(236):{128,112,136,136,136,128,128,136,128,136,128,120,112,128,128,128},
11101101(237):{128,144,128,136,132,140,124,124,136,136,128,120,140,116,124,124},
11101110(238):{128,128,128,136,128,120,128,128,108,108,116,124,132,124,124,124},
11101111(239):{128,128,136,120,124,132,124,116,100,140,132,124,128,128,136,120},
11110000(240):{128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128},
11110001(241):{128,128,128,120,112,128,128,136,132,124,132,132,140,116,140,124},
11110010(242):{128,128,136,128,132,124,124,124,152,128,152,136,140,124,124,132},
11110011(243):{128,128,120,136,116,124,124,132,116,132,148,116,128,136,128,136},
11110100(244):{128,144,128,120,132,124,124,124,104,136,128,136,140,132,140,140},
11110101(245):{128,144,128,128,124,132,132,124,124,116,132,124,128,128,112,112},
11110110(246):{128,128,128,128,120,136,128,128,136,128,120,128,128,136,136,144},
11110111(247):{128,128,128,136,128,128,136,128,132,132,116,124,156,124,116,140},
11111000(248):{128,128,136,128,132,132,124,132,112,128,136,128,124,124,116,140},
11111001(249):{128,128,136,136,124,124,116,132,116,124,124,132,144,136,136,112},
11111010(250):{128,128,128,128,120,128,120,128,136,144,128,136,136,120,112,128},
11111011(251):{128,128,128,136,128,136,144,128,116,132,140,148,132,124,140,124},
11111100(252):{128,144,120,136,136,128,120,128,144,128,128,128,128,120,136,128},
11111101(253):{128,112,120,128,136,128,120,120,116,124,132,132,140,124,132,124},
11111110(254):{128,128,120,128,124,124,124,132,128,120,120,152,116,124,116,132},
11111111(255):{128,128,136,136,124,124,124,124,124,124,116,132,104,120,136,136},
};
Indicador Linear : 0.125 Ocorre na linha 20


Matriz de distribuicao de diferenças
00000000(  0):{256,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
00000001(  1):{0, 0, 32, 8, 16, 32, 24, 16, 0, 0, 16, 24, 16, 32, 24, 16},
00000010(  2):{0, 0, 32, 0, 32, 0, 32, 0, 64, 0, 32, 0, 32, 0, 32, 0},
00000011(  3):{0, 32, 0, 32, 32, 16, 8, 16, 32, 0, 16, 0, 32, 16, 8, 16},
00000100(  4):{0, 48, 0, 24, 0, 16, 0, 32, 0, 48, 0, 40, 0, 16, 0, 32},
00000101(  5):{0, 0, 32, 16, 16, 32, 24, 16, 0, 0, 16, 16, 16, 32, 24, 16},
00000110(  6):{0, 16, 0, 24, 0, 40, 0, 32, 0, 32, 0, 40, 0, 40, 0, 32},
00000111(  7):{0, 0, 0, 8, 0, 24, 24, 16, 32, 16, 48, 24, 0, 24, 24, 16},
00001000(  8):{0, 16, 16, 24, 16, 16, 24, 16, 0, 16, 32, 8, 16, 16, 24, 16},
00001001(  9):{0, 32, 32, 32, 16, 16, 16, 16, 32, 0, 0, 0, 16, 16, 16, 16},
00001010(  10):{0, 16, 32, 16, 16, 8, 24, 16, 0, 32, 16, 16, 16, 8, 24, 16},
```

```
00001011(  11):{0, 32, 16, 16, 32, 8, 16, 16, 0, 16, 16, 16, 32, 8, 16, 16},
00001100(  12):{0, 32, 16, 16, 32, 8, 8, 16, 32, 16, 0, 16, 32, 8, 8, 16},
00001101(  13):{0, 32, 0, 24, 32, 8, 16, 16, 0, 16, 32, 8, 32, 8, 16, 16},
00001110(  14):{0, 0, 32, 8, 0, 8, 24, 16, 32, 48, 16, 24, 0, 8, 24, 16},
00001111(  15):{0, 0, 16, 8, 16, 24, 16, 16, 32, 16, 16, 24, 16, 24, 16, 16},
00010000(  16):{0, 16, 0, 24, 16, 24, 0, 24, 32, 32, 0, 24, 16, 24, 0, 24},
00010001(  17):{12, 12, 18, 24, 14, 8, 20, 8, 16, 12, 18, 16, 14, 24, 16, 24},
00010010(  18):{12, 26, 12, 24, 4, 22, 0, 28, 4, 22, 4, 24, 12, 26, 16, 20},
00010011(  19):{20, 20, 10, 6, 16, 22, 14, 20, 16, 20, 18, 18, 20, 10, 14,12},
00010100(  20):{22, 6, 26, 10, 12, 4, 24, 12, 26, 10, 22, 6, 36, 12, 24, 4},
00010101(  21):{14, 16, 16, 8, 8, 16, 18, 32, 14, 16, 20, 24, 20, 8, 18, 8},
00010110(  22):{24, 6, 28, 14, 34, 12, 30, 8, 24, 6, 20, 6, 14, 8, 18, 4},
00010111(  23):{16, 14, 22, 22, 20, 12, 14, 12, 12, 14, 14, 14, 8, 16, 22, 24},
00011000(  24):{16, 18, 16, 12, 18, 14, 14, 12, 12, 14, 24, 24, 10, 26, 18, 8},
00011001(  25):{18, 14, 12, 12, 14, 6, 24, 16, 14, 18, 16, 16, 18, 26, 12, 20},
00011010(  26):{16, 20, 18, 14, 16, 30, 14, 12, 12, 16, 22, 10, 12, 6, 18, 20},
00011011(  27):{14, 14, 14, 20, 18, 22, 14, 16, 18, 22, 14, 12, 14, 6, 22, 16},
00011100(  28):{18, 14, 18, 20, 20, 6, 16, 12, 18, 22, 14, 20, 16, 14, 8, 20},
00011101(  29):{16, 16, 18, 14, 14, 18, 24, 16, 16, 12, 10, 10, 18, 18, 12, 24},
00011110(  30):{20, 26, 10, 14, 12, 24, 18, 12, 8, 10, 22, 18, 16, 12, 22, 12},
00011111(  31):{18, 18, 18, 18, 20, 16, 12, 16, 14, 10, 18, 14, 12, 20, 16, 16},
00100000(  32):{0, 0, 0, 0, 16, 16, 8, 16, 64, 32, 16, 32, 16, 16, 8, 16},
00100001(  33):{16, 12, 18, 16, 8, 24, 14, 20, 10, 12, 24, 20, 22, 16, 16, 8},
00100010(  34):{24, 20, 8, 20, 12, 16, 32, 8, 8, 12, 16, 12, 20, 16, 8, 24},
00100011(  35):{14, 8, 14, 22, 28, 6, 12, 20, 20, 16, 16, 18, 10, 18, 14, 20},
00100100(  36):{20, 20, 16, 12, 20, 4, 20, 24, 12, 16, 12, 12, 12, 24, 16, 16},
00100101(  37):{12, 12, 20, 14, 22, 14, 22, 8, 14, 24, 22, 10, 8, 30, 8, 16},
00100110(  38):{20, 22, 12, 12, 16, 26, 12, 16, 12, 14, 16, 8, 16, 18, 24, 12},
00100111(  39):{26, 30, 24, 28, 10, 18, 16, 12, 4, 2, 10, 8, 16, 14, 22, 16},
00101000(  40):{10, 10, 22, 22, 12, 16, 20, 20, 16, 14, 16, 18, 18, 8, 14, 20},
00101001(  41):{10, 14, 16, 10, 8, 16, 6, 12, 22, 30, 28, 22, 24, 20, 14, 4},
00101010(  42):{12, 20, 24, 12, 16, 20, 16, 12, 14, 16, 18, 16, 14, 8, 14, 24},
00101011(  43):{14, 20, 10, 10, 28, 18, 16, 16, 14, 16, 14, 18, 8, 10, 24, 20},
00101100(  44):{20, 20, 8, 12, 12, 12, 12, 20, 14, 20, 14, 16, 26, 12, 22, 16},
00101101(  45):{12, 16, 12, 18, 22, 14, 14, 20, 16, 16, 16, 18, 14, 18, 22, 8},
00101110(  46):{22, 16, 30, 30, 16, 10, 16, 20, 8, 8, 8, 18, 10, 14, 18, 12},
00101111(  47):{24, 16, 22, 18, 10, 26, 20, 12, 8, 8, 10, 10, 22, 14, 12, 24},
00110000(  48):{0, 48, 0, 40, 0, 24, 8, 24, 32, 0, 16, 8, 0, 24, 8, 24},
00110001(  49):{12, 16, 12, 16, 22, 12, 18, 24, 18, 12, 26, 16, 12, 16, 8, 16},
```

```
00110010(  50):{12, 18, 4, 20, 8, 26, 16, 28, 4, 30, 4, 28, 8, 22, 8, 20},
00110011(  51):{26, 8, 28, 8, 12, 12, 14, 24, 4, 20, 6, 20, 22, 16, 16, 20},
00110100(  52):{22, 14, 22, 6, 32, 8, 20, 16, 26, 6, 30, 10, 16, 4, 24, 0},
00110101(  53):{14, 20, 16, 18, 14, 22, 16, 12, 16, 12, 14, 10, 20, 18, 18, 16},
00110110(  54):{20, 16, 24, 6, 14, 6, 22, 0, 28, 8, 28, 2, 34, 18, 22, 8},
00110111(  55):{18, 4, 18, 14, 18, 14, 14, 20, 16, 20, 12, 22, 12, 18, 20, 16},
00111000(  56):{18, 12, 22, 18, 18, 22, 14, 8, 12, 16, 16, 22, 16, 6, 12, 24},
00111001(  57):{24, 12, 20, 10, 22, 26, 18, 12, 8, 24, 12, 10, 10, 18, 14, 16},
00111010(  58):{12, 16, 14, 18, 12, 10, 14, 16, 18, 20, 20, 14, 22, 26, 16, 8},
00111011(  59):{16, 18, 16, 14, 14, 12, 18, 12, 12, 14, 12, 22, 22, 20, 18, 16},
00111100(  60):{18, 14, 10, 20, 20, 18, 12, 16, 12, 10, 20, 20, 14, 14, 22, 16},
00111101(  61):{20, 8, 14, 12, 20, 16, 22, 16, 8, 28, 10, 16, 16, 12, 18, 20},
00111110(  62):{10, 18, 16, 20, 12, 14, 18, 20, 24, 14, 14, 16, 18, 10, 16, 16},
00111111(  63):{14, 14, 20, 16, 18, 14, 12, 8, 18, 22, 16, 20, 14, 14, 16, 20},
01000000(  64):{0, 16, 0, 8, 16, 8, 8, 16, 64, 32, 16, 24, 16, 8, 8, 16},
01000001(  65):{10, 8, 20, 22, 12, 22, 18, 28, 8, 8, 14, 10, 26, 10, 20, 20},
01000010(  66):{24, 18, 8, 8, 20, 10, 8, 12, 8, 14, 16, 16, 12, 22, 32, 28},
01000011(  67):{18, 12, 10, 12, 26, 8, 14, 4, 16, 20, 20, 32, 12, 24, 12, 16},
01000100(  68):{22, 20, 22, 24, 14, 20, 14, 8, 18, 16, 14, 16, 10, 8, 14, 16},
01000101(  69):{8, 12, 16, 20, 24, 24, 24, 20, 10, 16, 18, 16, 14, 12, 14, 8},
01000110(  70):{16, 22, 12, 8, 24, 18, 20, 12, 12, 14, 20, 12, 12, 26, 12, 16},
01000111(  71):{22, 20, 24, 26, 8, 10, 14, 20, 4, 4, 14, 14, 22, 14, 20, 20},
01001000(  72):{12, 14, 18, 10, 18, 16, 24, 28, 14, 14, 8, 2, 12, 20, 22, 24},
01001001(  73):{14, 12, 14, 12, 18, 20, 10, 4, 18, 28, 26, 20, 14, 20, 14, 12},
01001010(  74):{14, 20, 18, 16, 6, 20, 18, 12, 16, 20, 16, 16, 20, 20, 20, 4},
01001011(  75):{16, 16, 20, 16, 4, 16, 12, 12, 16, 16, 20, 20, 28, 16, 12, 16},
01001100(  76):{22, 14, 12, 16, 18, 10, 28, 32, 16, 14, 2, 8, 16, 10, 14, 24},
01001101(  77):{16, 12, 26, 28, 12, 24, 14, 8, 16, 16, 18, 24, 20, 12, 6, 4},
01001110(  78):{22, 24, 18, 10, 16, 14, 20, 20, 12, 12, 20, 10, 6, 14, 14, 24},
01001111(  79):{20, 16, 18, 20, 20, 16, 10, 20, 8, 12, 14, 16, 16, 20, 22, 8},
01010000(  80):{0, 16, 16, 16, 32, 16, 8, 16, 32, 16, 0, 16, 32, 16, 8, 16},
01010001(  81):{16, 12, 16, 14, 16, 26, 12, 16, 14, 16, 22, 14, 10, 10, 22, 20},
01010010(  82):{20, 16, 16, 16, 16, 20, 36, 16, 12, 16, 8, 16, 16, 12, 4, 16},
01010011(  83):{20, 12, 18, 26, 20, 6, 14, 20, 18, 16, 8, 14, 14, 14, 16, 20},
01010100(  84):{16, 10, 16, 22, 22, 24, 14, 8, 16, 18, 20, 18, 10, 12, 14, 16},
01010101(  85):{16, 28, 14, 12, 10, 8, 16, 4, 14, 20, 16, 12, 16, 24, 26, 20},
01010110(  86):{16, 16, 16, 18, 10, 18, 10, 8, 16, 20, 20, 10, 22, 26, 18, 12},
01010111(  87):{12, 18, 24, 12, 18, 22, 22, 20, 14, 18, 14, 16, 12, 6, 12, 16},
01011000(  88):{14, 16, 24, 18, 10, 10, 16, 24, 16, 4, 18, 22, 16, 18, 14, 16},
```

```
01011001(  89):{20, 26, 10, 4, 16, 22, 14, 20, 12, 18, 18, 12, 16, 14, 22, 12},
01011010(  90):{16, 16, 18, 14, 12, 14, 18, 12, 14, 16, 20, 22, 14, 18, 16, 16},
01011011(  91):{18, 10, 14, 24, 14, 6, 18, 20, 18, 26, 18, 12, 14, 22, 14, 8},
01011100(  92):{22, 20, 4, 20, 20, 16, 10, 12, 16, 24, 22, 16, 14, 4, 20, 16},
01011101(  93):{16, 16, 12, 10, 14, 18, 10, 20, 20, 8, 16, 18, 14, 22, 26, 16},
01011110(  94):{16, 14, 18, 16, 18, 10, 20, 24, 10, 6, 16, 16, 12, 18, 18, 24},
01011111(  95):{18, 10, 20, 14, 8, 20, 18, 16, 14, 14, 20, 22, 24, 20, 6, 12},
01100000(  96):{0, 32, 32, 24, 32, 8, 16, 16, 0, 16, 0, 8, 32, 8, 16, 16},
01100001(  97):{14, 12, 18, 18, 20, 2, 16, 24, 14, 16, 18, 22, 16, 18, 12, 16},
01100010(  98):{16, 14, 8, 24, 20, 22, 28, 20, 16, 18, 8, 16, 12, 10, 20, 4},
01100011(  99):{20, 16, 12, 34, 10, 18, 22, 8, 16, 12, 8, 30, 18, 2, 22, 8},
01100100(  100):{18, 8, 10, 20, 10, 24, 14, 8, 22, 16, 14, 20, 14, 16, 26, 16},
01100101(  101):{16, 32, 12, 14, 10, 14, 6, 0, 12, 20, 16, 18, 26, 14, 30, 16},
01100110(  102):{12, 20, 20, 8, 16, 20, 20, 8, 16, 28, 8, 8, 20, 28, 16, 8},
01100111(  103):{4, 6, 16, 14, 22, 8, 22, 32, 20, 22, 24, 18, 18, 12, 2, 16},
01101000(  104):{18, 12, 16, 8, 14, 20, 8, 16, 18, 4, 24, 16, 14, 12, 16, 40},
01101001(  105):{24, 30, 22, 10, 22, 32, 12, 4, 8, 14, 22, 6, 10, 20, 8, 12},
01101010(  106):{22, 16, 14, 12, 14, 16, 18, 24, 18, 16, 22, 12, 10, 32, 10, 0},
01101011(  107):{22, 8, 18, 14, 12, 22, 20, 24, 22, 20, 14, 10, 8, 14, 12, 16},
01101100(  108):{22, 10, 12, 16, 14, 6, 20, 16, 18, 14, 8, 24, 10, 18, 24, 24},
01101101(  109):{20, 12, 22, 14, 10, 18, 12, 8, 24, 8, 10, 26, 10, 26, 20, 16},
01101110(  110):{16, 12, 12, 12, 12, 8, 8, 32, 16, 12, 32, 12, 20, 16, 12, 24},
01101111(  111):{12, 16, 12, 14, 18, 18, 14, 16, 16, 20, 28, 10, 18, 10, 10, 24},
01110000(  112):{0, 0, 16, 16, 16, 32, 16, 16, 32, 0, 16, 16, 16, 32, 16, 16},
01110001(  113):{24, 16, 18, 18, 12, 10, 14, 16, 24, 12, 14, 14, 4, 10, 18, 32},
01110010(  114):{20, 16, 8, 16, 16, 12, 8, 16, 12, 16, 8, 16, 16, 20, 40, 16},
01110011(  115):{18, 12, 12, 28, 8, 16, 22, 8, 14, 16, 12, 36, 24, 4, 18, 8},
01110100(  116):{8, 18, 16, 26, 18, 12, 22, 4, 8, 14, 16, 22, 30, 20, 10, 12},
01110101(  117):{24, 16, 10, 18, 16, 22, 26, 12, 24, 28, 14, 14, 0, 14, 14, 4},
01110110(  118):{20, 26, 16, 6, 14, 36, 14, 12, 20, 22, 16, 10, 10, 12, 18, 4},
01110111(  119):{22, 12, 24, 12, 8, 12, 10, 20, 18, 8, 16, 20, 16, 16, 14, 28},
01111000(  120):{16, 6, 18, 8, 14, 6, 16, 44, 16, 22, 14, 8, 18, 14, 16, 20},
01111001(  121):{18, 20, 18, 6, 12, 22, 12, 12, 14, 28, 22, 10, 20, 26, 12, 4},
01111010(  122):{12, 28, 14, 18, 28, 18, 14, 0, 12, 16, 18, 14, 12, 18, 18, 16},
01111011(  123):{12, 26, 20, 14, 22, 8, 14, 12, 12, 14, 20, 18, 18, 16, 10, 20},
01111100(  124):{14, 12, 8, 16, 16, 12, 30, 28, 10, 16, 8, 16, 24, 8, 18, 20},
01111101(  125):{12, 16, 24, 24, 20, 12, 16, 8, 12, 24, 16, 24, 20, 12, 8, 8},
01111110(  126):{14, 10, 16, 10, 18, 16, 12, 24, 10, 10, 24, 6, 22, 12, 12, 40},
01111111(  127):{22, 22, 18, 20, 18, 10, 10, 24, 18, 10, 22, 12, 6, 22, 14, 8},
```

```
10000000(  128):{0, 0, 16, 24, 32, 32, 16, 16, 0, 0, 16, 8, 32, 32, 16, 16},
10000001(  129):{22, 28, 18, 12, 8, 12, 12, 4, 30, 28, 14, 12, 4, 12, 20, 20},
10000010(  130):{16, 16, 16, 12, 12, 20, 24, 20, 16, 16, 16, 20, 20, 12, 8, 12},
10000011(  131):{22, 20, 16, 10, 14, 22, 16, 16, 14, 20, 16, 14, 14, 18, 16, 8},
10000100(  132):{4, 12, 20, 18, 22, 18, 18, 16, 12, 12, 12, 14, 26, 22, 14, 16},
10000101(  133):{26, 20, 14, 20, 8, 8, 8, 20, 26, 12, 18, 20, 4, 8, 24, 20},
10000110(  134):{18, 12, 14, 18, 12, 2, 12, 24, 22, 12, 18, 30, 12, 6, 20, 24},
10000111(  135):{16, 16, 18, 14, 4, 22, 22, 8, 20, 24, 14, 10, 24, 18, 10, 16},
10001000(  136):{22, 26, 12, 14, 14, 12, 20, 20, 14, 22, 20, 10, 14, 20, 12, 4},
10001001(  137):{20, 10, 20, 26, 22, 8, 18, 28, 12, 6, 12, 22, 10, 8, 14, 20},
10001010(  138):{16, 10, 18, 24, 18, 18, 12, 12, 12, 14, 14, 16, 18, 6, 20, 28},
10001011(  139):{14, 14, 16, 14, 20, 16, 14, 20, 18, 18, 16, 18, 12, 16, 18, 12},
10001100(  140):{12, 18, 20, 10, 12, 20, 16, 8, 16, 14, 12, 14, 24, 28, 16, 16},
10001101(  141):{18, 18, 16, 12, 22, 14, 12, 16, 14, 22, 16, 20, 10, 10, 20, 16},
10001110(  142):{12, 18, 8, 12, 30, 26, 10, 12, 8, 14, 24, 12, 14, 22, 22, 12},
10001111(  143):{18, 18, 14, 16, 6, 6, 26, 16, 22, 22, 18, 16, 18, 18, 6, 16},
10010000(  144):{0, 0, 16, 16, 16, 16, 24, 16, 0, 32, 32, 16, 16, 16, 24, 16},
10010001(  145):{14, 20, 16, 22, 20, 6, 30, 24, 24, 20, 6, 14, 14, 18, 4, 4},
10010010(  146):{16, 20, 24, 4, 20, 24, 8, 36, 16, 12, 16, 12, 12, 8, 16, 12},
10010011(  147):{16, 24, 20, 8, 12, 28, 8, 12, 14, 16, 30, 8, 14, 12, 14, 20},
10010100(  148):{12, 16, 24, 8, 20, 32, 16, 8, 20, 12, 12, 16, 12, 4, 12, 32},
10010101(  149):{18, 16, 22, 12, 18, 4, 6, 16, 20, 12, 16, 20, 16, 16, 12, 32},
10010110(  150):{16, 16, 10, 24, 4, 8, 6, 20, 16, 12, 18, 12, 28, 12, 30, 24},
10010111(  151):{20, 16, 8, 30, 18, 10, 18, 16, 14, 16, 14, 14, 20, 22, 16, 4},
10011000(  152):{24, 22, 6, 12, 20, 26, 22, 12, 14, 18, 4, 12, 14, 14, 24, 12},
10011001(  153):{14, 6, 10, 26, 16, 24, 12, 8, 18, 14, 26, 30, 16, 4, 16, 16},
10011010(  154):{22, 10, 14, 10, 16, 4, 16, 28, 16, 18, 16, 18, 18, 32, 10, 8},
10011011(  155):{14, 14, 18, 22, 20, 16, 24, 12, 22, 14, 14, 14, 8, 20, 8, 16},
10011100(  156):{10, 14, 22, 2, 4, 16, 12, 24, 20, 10, 12, 10, 22, 24, 26, 28},
10011101(  157):{20, 14, 16, 30, 18, 12, 14, 8, 16, 18, 28, 14, 10, 20, 6, 12},
10011110(  158):{24, 26, 12, 16, 16, 22, 24, 4, 10, 14, 6, 24, 22, 18, 14, 4},
10011111(  159):{16, 22, 18, 14, 18, 8, 16, 12, 16, 18, 6, 22, 14, 16, 24, 16},
10100000(  160):{0, 16, 16, 24, 16, 24, 24, 16, 0, 0, 32, 8, 16, 24, 24, 16},
10100001(  161):{20, 26, 6, 16, 10, 26, 20, 12, 26, 22, 12, 16, 16, 6, 18, 4},
10100010(  162):{16, 14, 24, 8, 12, 14, 16, 8, 16, 18, 16, 16, 20, 18, 8, 32},
10100011(  163):{20, 14, 22, 12, 18, 10, 20, 16, 10, 18, 8, 16, 8, 22, 22, 20},
10100100(  164):{10, 16, 14, 18, 24, 14, 12, 20, 14, 12, 22, 22, 16, 22, 16, 4},
10100101(  165):{22, 18, 18, 10, 10, 20, 22, 20, 24, 18, 16, 10, 16, 8, 0, 24},
10100110(  166):{16, 16, 22, 26, 18, 10, 16, 8, 20, 12, 10, 26, 10, 10, 16, 20},
```

```
10100111(  167):{20, 16, 14, 10, 20, 22, 10, 12, 18, 24, 8, 14, 14, 18, 24, 12},
10101000(  168):{22, 16, 10, 22, 14, 18, 18, 8, 16, 20, 16, 22, 20, 10, 12, 12},
10101001(  169):{18, 8, 10, 14, 12, 10, 20, 24, 14, 16, 14, 18, 20, 14, 20, 24},
10101010(  170):{16, 12, 12, 18, 20, 10, 12, 24, 18, 12, 22, 14, 18, 14, 10, 24},
10101011(  171):{16, 18, 24, 18, 14, 16, 6, 12, 16, 14, 16, 18, 18, 16, 18, 16},
10101100(  172):{12, 18, 24, 8, 22, 18, 10, 20, 14, 18, 22, 16, 8, 26, 16, 4},
10101101(  173):{20, 14, 18, 8, 12, 14, 18, 28, 12, 22, 10, 12, 20, 14, 18, 16},
10101110(  174):{12, 16, 6, 24, 22, 12, 16, 16, 18, 12, 16, 12, 20, 24, 18, 12},
10101111(  175):{16, 18, 16, 20, 12, 18, 16, 12, 20, 18, 16, 16, 16, 10, 16, 16},
10110000(  176):{0, 16, 48, 16, 0, 24, 32, 16, 0, 0, 16, 16, 0, 24, 32, 16},
10110001(  177):{28, 14, 8, 14, 10, 16, 6, 12, 20, 18, 16, 14, 22, 16, 18, 24},
10110010(  178):{16, 14, 16, 8, 20, 14, 0, 8, 16, 18, 32, 16, 12, 18, 16, 32},
10110011(  179):{10, 18, 30, 2, 12, 12, 8, 32, 14, 14, 34, 2, 12, 20, 8, 28},
10110100(  180):{14, 12, 18, 12, 22, 8, 6, 28, 10, 20, 22, 4, 18, 24, 18, 20},
10110101(  181):{26, 18, 10, 14, 20, 20, 16, 20, 22, 14, 14, 14, 12, 12, 8, 16},
10110110(  182):{18, 12, 18, 12, 18, 20, 22, 28, 18, 20, 18, 12, 10, 12, 6, 12},
10110111(  183):{16, 12, 12, 18, 18, 18, 14, 8, 28, 20, 8, 26, 18, 14, 14, 12},
10111000(  184):{16, 20, 4, 16, 16, 16, 16, 4, 24, 16, 8, 28, 24, 12, 20, 16},
10111001(  185):{20, 24, 12, 22, 14, 14, 18, 12, 12, 12, 8, 18, 18, 14, 26, 12},
10111010(  186):{18, 20, 12, 20, 22, 16, 20, 12, 18, 16, 4, 16, 22, 12, 12, 16},
10111011(  187):{16, 14, 10, 26, 14, 12, 24, 12, 12, 22, 14, 14, 22, 16, 16, 12},
10111100(  188):{14, 14, 22, 20, 14, 14, 22, 24, 6, 22, 26, 8, 14, 14, 10, 12},
10111101(  189):{10, 22, 22, 14, 16, 12, 12, 20, 18, 14, 18, 18, 20, 16, 12, 12},
10111110(  190):{16, 12, 2, 20, 24, 16, 22, 12, 20, 16, 6, 32, 20, 20, 18, 0},
10111111(  191):{18, 14, 12, 22, 16, 24, 18, 8, 18, 14, 12, 18, 12, 12, 22, 16},
11000000(  192):{0, 0, 48, 0, 32, 8, 32, 8, 0, 16, 16, 16, 32, 8, 32, 8},
11000001(  193):{24, 20, 16, 10, 16, 18, 12, 20, 16, 28, 16, 14, 16, 6, 12, 12},
11000010(  194):{24, 10, 22, 18, 14, 0, 24, 0, 24, 6, 34, 6, 34, 16, 16, 8},
11000011(  195):{10, 28, 14, 4, 16, 20, 16, 16, 22, 12, 26, 4, 8, 28, 16, 16},
11000100(  196):{10, 18, 6, 18, 8, 20, 12, 36, 6, 22, 10, 22, 8, 36, 4, 20},
11000101(  197):{22, 14, 12, 22, 18, 12, 28, 24, 18, 10, 12, 26, 14, 4, 4, 16},
11000110(  198):{10, 18, 14, 38, 8, 12, 8, 36, 6, 22, 6, 22, 8, 12, 4, 32},
11000111(  199):{12, 20, 12, 14, 20, 18, 8, 12, 20, 20, 16, 14, 20, 14, 20, 16},
11001000(  200):{16, 30, 16, 8, 16, 26, 12, 4, 24, 18, 16, 20, 16, 14, 12, 8},
11001001(  201):{16, 10, 6, 24, 14, 6, 20, 28, 16, 6, 18, 20, 18, 10, 20, 24},
11001010(  202):{22, 8, 8, 16, 18, 16, 16, 12, 18, 16, 12, 24, 14, 16, 20, 20},
11001011(  203):{20, 10, 14, 20, 10, 18, 8, 8, 20, 14, 14, 12, 14, 22, 28, 24},
11001100(  204):{18, 12, 18, 26, 20, 30, 12, 12, 14, 12, 26, 14, 4, 18, 16, 4},
11001101(  205):{14, 20, 14, 20, 12, 20, 16, 20, 26, 12, 14, 4, 12, 12, 20, 20},
```

```
11001110(  206):{20, 20, 20, 4, 20, 20, 12, 4, 12, 20, 8, 20, 20, 28, 16, 12},
11001111(  207):{18, 18, 16, 14, 14, 12, 20, 16, 14, 22, 12, 18, 18, 12, 16, 16},
11010000(  208):{0, 48, 0, 24, 16, 8, 8, 24, 0, 32, 16, 24, 16, 8, 8, 24},
11010001(  209):{10, 24, 12, 12, 18, 4, 16, 4, 8, 16, 18, 24, 28, 28, 18, 16},
11010010(  210):{8, 22, 10, 30, 10, 24, 0, 24, 8, 26, 6, 26, 6, 24, 16, 16},
11010011(  211):{18, 4, 16, 34, 12, 22, 22, 4, 16, 28, 2, 14, 18, 18, 24, 4},
11010100(  212):{34, 6, 14, 8, 6, 18, 22, 4, 30, 6, 22, 8, 26, 2, 38, 12},
11010101(  213):{8, 6, 18, 30, 14, 20, 0, 28, 10, 22, 20, 10, 32, 8, 26, 4},
11010110(  214):{20, 6, 18, 16, 44, 2, 42, 12, 20, 6, 22, 8, 12, 2, 14, 12},
11010111(  215):{14, 28, 20, 2, 24, 14, 22, 16, 8, 12, 14, 22, 18, 18, 8, 16},
11011000(  216):{16, 6, 20, 14, 16, 12, 12, 12, 18, 26, 14, 6, 14, 28, 18, 24},
11011001(  217):{14, 14, 24, 16, 16, 6, 14, 8, 18, 6, 20, 28, 16, 22, 6, 28},
11011010(  218):{18, 20, 18, 6, 10, 22, 14, 32, 24, 0, 16, 22, 12, 14, 16, 12},
11011011(  219):{24, 14, 16, 8, 10, 30, 14, 28, 20, 6, 20, 12, 10, 14, 14, 16},
11011100(  220):{22, 12, 10, 14, 14, 22, 26, 8, 20, 12, 20, 14, 8, 26, 8, 20},
11011101(  221):{24, 12, 14, 14, 8, 18, 18, 8, 20, 12, 18, 22, 12, 22, 14, 20},
11011110(  222):{14, 12, 30, 20, 12, 20, 8, 20, 24, 28, 16, 4, 14, 12, 10, 12},
11011111(  223):{12, 22, 16, 8, 26, 14, 18, 24, 12, 18, 12, 12, 14, 10, 18, 20},
11100000(  224):{0, 0, 48, 0, 16, 0, 40, 8, 0, 32, 32, 16, 16, 0, 40, 8},
11100001(  225):{18, 18, 12, 14, 18, 12, 8, 16, 16, 22, 6, 18, 28, 20, 22, 8},
11100010(  226):{24, 12, 30, 14, 38, 6, 16, 12, 24, 4, 34, 2, 10, 10, 16, 4},
11100011(  227):{8, 22, 16, 6, 8, 20, 28, 20, 18, 10, 22, 6, 14, 20, 14, 24},
11100100(  228):{12, 22, 12, 26, 6, 24, 10, 16, 12, 22, 8, 22, 2, 28, 2, 32},
11100101(  229):{18, 12, 12, 12, 24, 4, 10, 12, 16, 16, 14, 16, 22, 24, 12, 32},
11100110(  230):{8, 22, 14, 22, 6, 20, 4, 28, 4, 22, 6, 42, 14, 16, 8, 20},
11100111(  231):{16, 24, 12, 18, 24, 18, 12, 20, 18, 16, 6, 10, 22, 14, 18, 8},
11101000(  232):{20, 20, 10, 28, 20, 16, 6, 4, 22, 16, 16, 20, 18, 20, 16, 4},
11101001(  233):{14, 12, 8, 16, 16, 16, 22, 32, 18, 12, 8, 12, 16, 8, 26, 20},
11101010(  234):{26, 10, 10, 22, 16, 16, 16, 20, 20, 14, 12, 10, 18, 16, 10, 20},
11101011(  235):{18, 14, 18, 12, 12, 22, 24, 16, 22, 10, 18, 24, 12, 18, 4, 12},
11101100(  236):{14, 16, 26, 20, 2, 24, 10, 4, 16, 12, 32, 20, 16, 20, 12, 12},
11101101(  237):{16, 16, 12, 4, 6, 24, 22, 24, 24, 12, 12, 8, 18, 12, 18, 28},
11101110(  238):{28, 18, 6, 20, 20, 22, 22, 4, 14, 18, 12, 16, 18, 14, 8, 16},
11101111(  239):{16, 18, 10, 22, 24, 12, 6, 20, 12, 18, 18, 14, 12, 16, 30, 8},
11110000(  240):{0, 48, 0, 24, 0, 16, 16, 24, 0, 16, 32, 24, 0, 16, 16, 24},
11110001(  241):{16, 18, 16, 20, 36, 26, 16, 12, 12, 14, 16, 8, 8, 14, 8, 16},
11110010(  242):{8, 20, 18, 34, 2, 26, 8, 20, 8, 28, 6, 30, 14, 22, 0, 12},
11110011(  243):{16, 6, 18, 12, 12, 18, 18, 20, 12, 18, 14, 24, 16, 14, 22, 16},
11110100(  244):{32, 10, 20, 4, 20, 10, 32, 16, 24, 6, 20, 4, 20, 6, 24, 8},
```

```
11110101(  245):{12, 16, 14, 16, 24, 16, 14, 12, 16, 16, 10, 20, 20, 24, 18, 8},
11110110(  246):{22, 10, 18, 4, 18, 6, 18, 4, 22, 6, 30, 8, 34, 10, 30, 16},
11110111(  247):{22, 20, 8, 14, 24, 14, 14, 12, 10, 20, 24, 10, 16, 18, 10, 20},
11111000(  248):{16, 12, 26, 22, 20, 10, 14, 24, 20, 16, 10, 18, 16, 18, 6, 8},
11111001(  249):{12, 12, 22, 16, 18, 12, 20, 20, 20, 24, 6, 12, 14, 16, 16, 16},
11111010(  250):{14, 14, 12, 20, 16, 18, 14, 12, 26, 14, 8, 16, 16, 10, 22, 24},
11111011(  251):{22, 14, 12, 8, 12, 14, 14, 20, 14, 14, 16, 16, 16, 22, 22, 20},
11111100(  252):{18, 16, 26, 20, 16, 24, 12, 4, 14, 20, 18, 24, 8, 12, 16, 8},
11111101(  253):{22, 12, 16, 10, 18, 14, 16, 20, 14, 16, 12, 14, 10, 22, 20, 20},
11111110(  254):{10, 14, 20, 20, 8, 14, 6, 16, 30, 14, 16, 16, 24, 14, 14, 20},
11111111(  255):{14, 14, 10, 12, 12, 18, 24, 20, 14, 14, 18, 12, 24, 18, 12, 20},
};
Indicador Diferencial : 0.25 Ocorre na linha 2

Ind Linear & Ind Diferencial para cada tabela
Tabela 0 : |p-0.5|=0.125 na linha 20.0. Carac Difer:  0.25 na linha 2.0
Tabela 1 : |p-0.5|=0.125 na linha 23.0. Carac Difer:  0.25 na linha 3.0
Tabela 2 : |p-0.5|=0.125 na linha 27.0. Carac Difer:  0.25 na linha 7.0
Tabela 3 : |p-0.5|=0.125 na linha 44.0. Carac Difer:  0.25 na linha 5.0
Tabela 4 : |p-0.5|=0.125 na linha 136.0. Carac Difer:  0.25 na linha 32.0
Tabela 5 : |p-0.5|=0.125 na linha 81.0. Carac Difer:  0.25 na linha 8.0
Tabela 6 : |p-0.5|=0.125 na linha 102.0. Carac Difer:  0.25 na linha 16.0
Tabela 7 : |p-0.5|=0.125 na linha 49.0. Carac Difer:  0.25 na linha 3.0
Tabela 8 : |p-0.5|=0.125 na linha 54.0. Carac Difer:  0.25 na linha 1.0
Tabela 9 : |p-0.5|=0.125 na linha 75.0. Carac Difer:  0.25 na linha 15.0
```

Obs: na matriz estão as ocorrências e no indicador a probabilidade (divide pelo somatorio na linha). A RS/R = 0.25/(1/16)=2.

12.11.1.2 RESUMO DO RESULTADO DAS 10 TABELAS (QL)

```
Ind Linear & Ind Diferencial para cada tabela
Tabela 0 : |p-0.5|=0.125 na linha 20.0. Carac Difer:  0.25 na linha 2.0
Tabela 1 : |p-0.5|=0.125 na linha 23.0. Carac Difer:  0.25 na linha 3.0
Tabela 2 : |p-0.5|=0.125 na linha 27.0. Carac Difer:  0.25 na linha 7.0
```

```
Tabela 3 : |p-0.5|=0.125 na linha 44.0. Carac Difer:  0.25 na linha 5.0
Tabela 4 : |p-0.5|=0.125 na linha 136.0. Carac Difer: 0.25 na linha 32.0
Tabela 5 : |p-0.5|=0.125 na linha 81.0. Carac Difer:  0.25 na linha 8.0
Tabela 6 : |p-0.5|=0.125 na linha 102.0. Carac Difer: 0.25 na linha 16.0
Tabela 7 : |p-0.5|=0.125 na linha 49.0. Carac Difer:  0.25 na linha 10.0
Tabela 8 : |p-0.5|=0.125 na linha 54.0. Carac Difer:  0.25 na linha 1.0
Tabela 9 : |p-0.5|=0.125 na linha 75.0. Carac Difer:  0.25 na linha 15.0
```

Pode-se notar que, em geral, os valores críticos ocorrem em linhas diferentes para diferentes tabelas. Da mesma forma, esses valores também ocorrem em colunas diferentes. Foram registradas apenas as linhas, pois é suficiente que os valores críticos ocoorram em linhas (ou colunas) distintas para que a associação de mais de um QL tenha melhores valores de RS/R que cada um.

12.11.2 CÓDIGO FONTE QUE GEROU O ITEM 12.11.1

Os dez QL que podem ser vistos no código fonte foram obtidos por tentativa testando 9730 QL gerados aleatoriamente. O número 9730 não tem qualquer significado especial e é apresentado apenas para dar uma idéia da frequência de ocorrência de tais QL.

```
 /*************************************************************************
* CRIADO POR JORGE LAMBERT P/ TESTAR S-BOXES TIPO QUADRADO   LATINO GERADAS*
* PELO GERA_TNL.JAVA QUANTO A RESISTENCIA ÀS CRIPTOANÁLISES E              *
* LINEAR. Foram testados apenas conjuntos de tabelas com ind.linear = 32   *
* (o menor encontrado)     (significa |p -.5| = 32/256)                    *
* as  tabelas para formação dos conjuntos encontram-se na variavel boxes   *
* e diferencial = 24 (o menor encontrado)                                  *
* As tabelas testadas aqui foram selecionadas de um grupo de 9730          *
* tabelas geradas pelo geraTNL.java                                        *
*************************************************************************/
// modificado 6/ 12/ 03 a 1 h
// com base no resultado obtido com este mesmo codigo foram escolhidas as tabelas
// que aqui se apresentam. Mais tabelas com bons resultados encontram-se em
// sotabelas.doc e "melhor tabela 28_11_aprimorado.txt",
// juntamente com os codigos fonte
// as quatro primeiras tabelas serao usadas combinadas para apresentarem melhores
// uma vez que os indicadores aparecem em linhas diferentes. Vide
// 4_tabelas_combinadas.java

public class seleciona_conj_4_tabelas {
public static void main(String[] args) {

int boxes[][][]= {
{
{15,5,13,11,0,4,8,10,1,12,9,14,6,3,2,7},
{2,8,0,14,3,7,11,13,4,15,12,1,9,6,5,10},
{7,13,5,3,8,12,0,2,9,4,1,6,14,11,10,15},
{14,4,12,10,15,3,7,9,0,11,8,13,5,2,1,6},
{6,12,4,2,7,11,15,1,8,3,0,5,13,10,9,14},
{10,0,8,6,11,15,3,5,12,7,4,9,1,14,13,2},
{13,3,11,9,14,2,6,8,15,10,7,12,4,1,0,5},
{5,11,3,1,6,10,14,0,7,2,15,4,12,9,8,13},
{11,1,9,7,12,0,4,6,13,8,5,10,2,15,14,3},
```

```
{12,2,10,8,13,1,5,7,14,9,6,11,3,0,15,4},
{4,10,2,0,5,9,13,15,6,1,14,3,11,8,7,12},
{1,7,15,13,2,6,10,12,3,14,11,0,8,5,4,9},
{3,9,1,15,4,8,12,14,5,0,13,2,10,7,6,11},
{8,14,6,4,9,13,1,3,10,5,2,7,15,12,11,0},
{9,15,7,5,10,14,2,4,11,6,3,8,0,13,12,1},
{0,6,14,12,1,5,9,11,2,13,10,15,7,4,3,8}},
 {
{13,9,10,7,4,3,5,1,14,0,2,12,8,15,11,6},
{9,5,6,3,0,15,1,13,10,12,14,8,4,11,7,2},
{15,11,12,9,6,5,7,3,0,2,4,14,10,1,13,8},
{0,12,13,10,7,6,8,4,1,3,5,15,11,2,14,9},
{5,1,2,15,12,11,13,9,6,8,10,4,0,7,3,14},
{7,3,4,1,14,13,15,11,8,10,12,6,2,9,5,0},
{11,7,8,5,2,1,3,15,12,14,0,10,6,13,9,4},
{10,6,7,4,1,0,2,14,11,13,15,9,5,12,8,3},
{14,10,11,8,5,4,6,2,15,1,3,13,9,0,12,7},
{2,14,15,12,9,8,10,6,3,5,7,1,13,4,0,11},
{6,2,3,0,13,12,14,10,7,9,11,5,1,8,4,15},
{1,13,14,11,8,7,9,5,2,4,6,0,12,3,15,10},
{8,4,5,2,15,14,0,12,9,11,13,7,3,10,6,1},
{4,0,1,14,11,10,12,8,5,7,9,3,15,6,2,13},
{12,8,9,6,3,2,4,0,13,15,1,11,7,14,10,5},
{3,15,0,13,10,9,11,7,4,6,8,2,14,5,1,12}},
{
{14,10,1,3,11,8,5,6,2,15,7,9,12,4,13,0},
{15,11,2,4,12,9,6,7,3,0,8,10,13,5,14,1},
{2,14,5,7,15,12,9,10,6,3,11,13,0,8,1,4},
{1,13,4,6,14,11,8,9,5,2,10,12,15,7,0,3},
{4,0,7,9,1,14,11,12,8,5,13,15,2,10,3,6},
{6,2,9,11,3,0,13,14,10,7,15,1,4,12,5,8},
{12,8,15,1,9,6,3,4,0,13,5,7,10,2,11,14},
{8,4,11,13,5,2,15,0,12,9,1,3,6,14,7,10},
{3,15,6,8,0,13,10,11,7,4,12,14,1,9,2,5},
{0,12,3,5,13,10,7,8,4,1,9,11,14,6,15,2},
{5,1,8,10,2,15,12,13,9,6,14,0,3,11,4,7},
{10,6,13,15,7,4,1,2,14,11,3,5,8,0,9,12},
{7,3,10,12,4,1,14,15,11,8,0,2,5,13,6,9},
{11,7,14,0,8,5,2,3,15,12,4,6,9,1,10,13},
```

```
{13,9,0,2,10,7,4,5,1,14,6,8,11,3,12,15},
{9,5,12,14,6,3,0,1,13,10,2,4,7,15,8,11}},
{
{8,6,11,1,14,3,15,2,0,5,7,12,13,10,9,4},
{7,5,10,0,13,2,14,1,15,4,6,11,12,9,8,3},
{6,4,9,15,12,1,13,0,14,3,5,10,11,8,7,2},
{10,8,13,3,0,5,1,4,2,7,9,14,15,12,11,6},
{15,13,2,8,5,10,6,9,7,12,14,3,4,1,0,11},
{11,9,14,4,1,6,2,5,3,8,10,15,0,13,12,7},
{5,3,8,14,11,0,12,15,13,2,4,9,10,7,6,1},
{3,1,6,12,9,14,10,13,11,0,2,7,8,5,4,15},
{9,7,12,2,15,4,0,3,1,6,8,13,14,11,10,5},
{2,0,5,11,8,13,9,12,10,15,1,6,7,4,3,14},
{4,2,7,13,10,15,11,14,12,1,3,8,9,6,5,0},
{14,12,1,7,4,9,5,8,6,11,13,2,3,0,15,10},
{0,14,3,9,6,11,7,10,8,13,15,4,5,2,1,12},
{13,11,0,6,3,8,4,7,5,10,12,1,2,15,14,9},
{12,10,15,5,2,7,3,6,4,9,11,0,1,14,13,8},
{1,15,4,10,7,12,8,11,9,14,0,5,6,3,2,13}},
{
{14,12,13,10,9,8,6,2,7,3,5,15,4,1,11,0},
{8,6,7,4,3,2,0,12,1,13,15,9,14,11,5,10},
{4,2,3,0,15,14,12,8,13,9,11,5,10,7,1,6},
{0,14,15,12,11,10,8,4,9,5,7,1,6,3,13,2},
{6,4,5,2,1,0,14,10,15,11,13,7,12,9,3,8},
{3,1,2,15,14,13,11,7,12,8,10,4,9,6,0,5},
{10,8,9,6,5,4,2,14,3,15,1,11,0,13,7,12},
{1,15,0,13,12,11,9,5,10,6,8,2,7,4,14,3},
{5,3,4,1,0,15,13,9,14,10,12,6,11,8,2,7},
{2,0,1,14,13,12,10,6,11,7,9,3,8,5,15,4},
{13,11,12,9,8,7,5,1,6,2,4,14,3,0,10,15},
{11,9,10,7,6,5,3,15,4,0,2,12,1,14,8,13},
{7,5,6,3,2,1,15,11,0,12,14,8,13,10,4,9},
{15,13,14,11,10,9,7,3,8,4,6,0,5,2,12,1},
{9,7,8,5,4,3,1,13,2,14,0,10,15,12,6,11},
{12,10,11,8,7,6,4,0,5,1,3,13,2,15,9,14}},
{
{14,11,13,10,9,8,6,2,7,3,5,15,4,1,12,0},
{8,5,7,4,3,2,0,12,1,13,15,9,14,11,6,10},
```

```
{4,1,3,0,15,14,12,8,13,9,11,5,10,7,2,6},
{0,13,15,12,11,10,8,4,9,5,7,1,6,3,14,2},
{6,3,5,2,1,0,14,10,15,11,13,7,12,9,4,8},
{3,0,2,15,14,13,11,7,12,8,10,4,9,6,1,5},
{10,7,9,6,5,4,2,14,3,15,1,11,0,13,8,12},
{1,14,0,13,12,11,9,5,10,6,8,2,7,4,15,3},
{5,2,4,1,0,15,13,9,14,10,12,6,11,8,3,7},
{2,15,1,14,13,12,10,6,11,7,9,3,8,5,0,4},
{13,10,12,9,8,7,5,1,6,2,4,14,3,0,11,15},
{11,8,10,7,6,5,3,15,4,0,2,12,1,14,9,13},
{7,4,6,3,2,1,15,11,0,12,14,8,13,10,5,9},
{15,12,14,11,10,9,7,3,8,4,6,0,5,2,13,1},
{9,6,8,5,4,3,1,13,2,14,0,10,15,12,7,11},
{12,9,11,8,7,6,4,0,5,1,3,13,2,15,10,14}
},
// 21
{
{2,11,0,6,8,10,13,9,12,5,15,4,3,14,7,1},
{0,9,14,4,6,8,11,7,10,3,13,2,1,12,5,15},
{13,6,11,1,3,5,8,4,7,0,10,15,14,9,2,12},
{5,14,3,9,11,13,0,12,15,8,2,7,6,1,10,4},
{4,13,2,8,10,12,15,11,14,7,1,6,5,0,9,3},
{15,8,13,3,5,7,10,6,9,2,12,1,0,11,4,14},
{3,12,1,7,9,11,14,10,13,6,0,5,4,15,8,2},
{8,1,6,12,14,0,3,15,2,11,5,10,9,4,13,7},
{6,15,4,10,12,14,1,13,0,9,3,8,7,2,11,5},
{14,7,12,2,4,6,9,5,8,1,11,0,15,10,3,13},
{1,10,15,5,7,9,12,8,11,4,14,3,2,13,6,0},
{12,5,10,0,2,4,7,3,6,15,9,14,13,8,1,11},
{11,4,9,15,1,3,6,2,5,14,8,13,12,7,0,10},
{7,0,5,11,13,15,2,14,1,10,4,9,8,3,12,6},
{10,3,8,14,0,2,5,1,4,13,7,12,11,6,15,9},
{9,2,7,13,15,1,4,0,3,12,6,11,10,5,14,8}
},



//25
{
```

```
{0,9,2,4,6,10,11,13,8,3,12,1,5,15,14,7},
{12,5,14,0,2,6,7,9,4,15,8,13,1,11,10,3},
{5,14,7,9,11,15,0,2,13,8,1,6,10,4,3,12},
{14,7,0,2,4,8,9,11,6,1,10,15,3,13,12,5},
{4,13,6,8,10,14,15,1,12,7,0,5,9,3,2,11},
{11,4,13,15,1,5,6,8,3,14,7,12,0,10,9,2},
{3,12,5,7,9,13,14,0,11,6,15,4,8,2,1,10},
{8,1,10,12,14,2,3,5,0,11,4,9,13,7,6,15},
{6,15,8,10,12,0,1,3,14,9,2,7,11,5,4,13},
{9,2,11,13,15,3,4,6,1,12,5,10,14,8,7,0},
{2,11,4,6,8,12,13,15,10,5,14,3,7,1,0,9},
{1,10,3,5,7,11,12,14,9,4,13,2,6,0,15,8},
{10,3,12,14,0,4,5,7,2,13,6,11,15,9,8,1},
{15,8,1,3,5,9,10,12,7,2,11,0,4,14,13,6},
{7,0,9,11,13,1,2,4,15,10,3,8,12,6,5,14},
{13,6,15,1,3,7,8,10,5,0,9,14,2,12,11,4}
},
//26
{
{1,9,2,4,6,10,11,13,8,3,12,0,7,15,14,5},
{13,5,14,0,2,6,7,9,4,15,8,12,3,11,10,1},
{2,10,3,5,7,11,12,14,9,4,13,1,8,0,15,6},
{15,7,0,2,4,8,9,11,6,1,10,14,5,13,12,3},
{5,13,6,8,10,14,15,1,12,7,0,4,11,3,2,9},
{12,4,13,15,1,5,6,8,3,14,7,11,2,10,9,0},
{4,12,5,7,9,13,14,0,11,6,15,3,10,2,1,8},
{9,1,10,12,14,2,3,5,0,11,4,8,15,7,6,13},
{11,3,12,14,0,4,5,7,2,13,6,10,1,9,8,15},
{10,2,11,13,15,3,4,6,1,12,5,9,0,8,7,14},
{3,11,4,6,8,12,13,15,10,5,14,2,9,1,0,7},
{6,14,7,9,11,15,0,2,13,8,1,5,12,4,3,10},
{7,15,8,10,12,0,1,3,14,9,2,6,13,5,4,11},
{0,8,1,3,5,9,10,12,7,2,11,15,6,14,13,4},
{8,0,9,11,13,1,2,4,15,10,3,7,14,6,5,12},
{14,6,15,1,3,7,8,10,5,0,9,13,4,12,11,2}
},
//27

//28
```

```
{
{13,14,9,11,0,4,15,12,6,2,7,8,3,10,5,1},
{3,4,15,1,6,10,5,2,12,8,13,14,9,0,11,7},
{2,3,14,0,5,9,4,1,11,7,12,13,8,15,10,6},
{7,8,3,5,10,14,9,6,0,12,1,2,13,4,15,11},
{15,0,11,13,2,6,1,14,8,4,9,10,5,12,7,3},
{4,5,0,2,7,11,6,3,13,9,14,15,10,1,12,8},
{9,10,5,7,12,0,11,8,2,14,3,4,15,6,1,13},
{11,12,7,9,14,2,13,10,4,0,5,6,1,8,3,15},
{8,9,4,6,11,15,10,7,1,13,2,3,14,5,0,12},
{1,2,13,15,4,8,3,0,10,6,11,12,7,14,9,5},
{0,1,12,14,3,7,2,15,9,5,10,11,6,13,8,4},
{10,11,6,8,13,1,12,9,3,15,4,5,0,7,2,14},
{14,15,10,12,1,5,0,13,7,3,8,9,4,11,6,2},
{5,6,1,3,8,12,7,4,14,10,15,0,11,2,13,9},
{12,13,8,10,15,3,14,11,5,1,6,7,2,9,4,0},
{6,7,2,4,9,13,8,5,15,11,0,1,12,3,14,10}
},
 };

int i,j,k,indice, max_na_linha, soma_linha, linha;
int desvio,indicador_linear;
int tabela_16x16 [][] = new int [16][16];
int matriz_teste256 [][] = new int [256][16];
int acumulador_diferencial [][] = new int [256][16];
int acumulador_linear [][] = new int [256][16];
float auxiliar, indicador_diferencial, caracteristica_linha ;
float resumo [][] = new float [25][4];//variavel que vai acumular os lineares
                                      //e dif das 22 tabelas e linhas em que ocorrem

for (indice=0;indice<10;indice++){
    for (i=0;i<16;i++){// atribuição
        for (j=0;j<16;j++){
            tabela_16x16 [i][j] = boxes[indice][i][j];
        }
    }
    System.out.println ("Tabela número " + indice);
    k= imprime_16 (tabela_16x16);// exibe tabela que esta sendo testada
```

```
    // calculo do indicador linear
    matriz_teste256 = teste_linear(tabela_16x16,0);
    indicador_linear=0;
    linha=0;
    desvio=0;
    for (i=0;i<256;i++){//calculo do indicador linear
        max_na_linha=0;
        soma_linha=0;
        for (j=0;j<16;j++){
            if ((i+j)==0) j=1;
            if (Math.abs(matriz_teste256[i][j]-128) > max_na_linha) max_na_linha = Math.abs(matriz_teste256[i][j]-
128);
            }
        desvio = max_na_linha;
    //    System.out.print (desvio + "  "+i);
        if (desvio > indicador_linear){
            indicador_linear = desvio;
            linha=i;
            }// fim do if
            }//fim do for j

    k= imprime_256x16 (matriz_teste256, "distribuicao linear");
    auxiliar = (float) indicador_linear;
    auxiliar = auxiliar/256;
    System.out.println ("Indicador Linear : " + auxiliar + " Ocorre na linha " + linha);
    resumo [indice][0] =  auxiliar ;
    resumo [indice][1] =  linha ;
    // calculo do indicador diferencial
    matriz_teste256 = teste_diferencial(tabela_16x16,0);
    indicador_diferencial = 0;
    for (i=0;i<256;i++){//calculo do indicador diferencial
        max_na_linha=0;
        soma_linha=0;
        for (j=0;j<16;j++){
            if ((i+j)==0) j=1;
            if (matriz_teste256 [i][j] > max_na_linha) max_na_linha = matriz_teste256 [i][j];
            soma_linha = soma_linha + matriz_teste256 [i][j];
            }//fim do for j
        auxiliar = (float) max_na_linha;
```

```
        caracteristica_linha = auxiliar/soma_linha;
//      System.out.println ("linha " + i + "  soma_linha " + soma_linha + "  max " + max_na_linha + "  carac " +
caracteristica_linha);
        if (caracteristica_linha > indicador_diferencial){
            indicador_diferencial = caracteristica_linha;
            linha=i;
            }// fim do if
        }// fim do for i
    resumo [indice][2] = indicador_diferencial;
    resumo [indice][3] = linha;
    k= imprime_256x16 (matriz_teste256, "distribuicao de diferenças");
    System.out.println ("Indicador Diferencial : " + indicador_diferencial+ " Ocorre na linha " + linha);
    }// fim do for indice


//imprime resumo

System.out.println("Ind Linear & Ind Diferencial para cada tabela");
for (i=0;i<10;i++){
System.out.println("Tabela "+i+" : |p-0.5|=" +resumo [i][0]+ " na linha "+
    resumo [i][1]+". Carac Difer:  "+resumo [i][2]+" na linha "+resumo [i][3]);
}// fim do for j

}// FIM DE MAIN

/*********************************************************************
 *****               metodo substituicao -   S Box               ******
 *****                   argumento = int[8] binario              ******
 *****                   retorna = int[4] binario                ******
 *********************************************************************/
 public static int SUB(int TABS[][], int argumento){//byte xy e o que sera substituido por zw
 int lin,col, substituto;
 int[] xy=new int[8];
 xy = CG2a8(argumento);
 lin=xy[0]*8+xy[1]*4+xy[2]*2+xy[3];//clacula linha de entrada na caixa (caso hexa)
 col=xy[4]*8+xy[5]*4+xy[6]*2+xy[7];//calcula coluna de entrada na caixa  (caso hexa)
 substituto=TABS[lin][col];
 int polinomio[] = CG2a8(argumento);
// System.out.println(" lin "+lin + " col " + col+ " substituto  " + substituto);
```

```
 return substituto;
}// ********************fim do metodo SUBSTITUICAO  ************************


/*********************************************************************
 ***** metodo CG2a8- converte inteiro ate 255 em polinomio        ******
 *****                            argumento = int                 ******
 *****          retorna = um polinomio do CG(2^8)                 ******
 *********************************************************************/
public static int [] CG2a8(int inteiro){
int polinomio[]={0,0,0,0,0,0,0,0};
int p2=128;
for (int i=0;i<8;i++){
    if (inteiro>(p2-1)){
        polinomio[i]=1;
        inteiro=inteiro-p2;
    }
    p2=p2/2;
}
return polinomio;
}// fim do metodo CG2a8

/*********************************************************************
 *****         metodo impressão - imprime tabela de substituicao  ******
 *****                            argumento = int [16][16]        ******
 *********************************************************************/
public static int imprime_16(int [][]TAB){
int lixo=0;
System.out.println(" ");
System.out.println(" ");
System.out.println("Tabela de Substituição ");
for (int j=0;j<16;j++){
    System.out.print("{");
    for(int i=0;i<16;i++){
        System.out.print(TAB [j][i]);
        if (i<15)System.out.print(",");
        else System.out.println("},");
    }// fim do for i
}// fim do for j
```

```
System.out.println("};");
return lixo;
}// fim do metodo imprime_16


/*********************************************************************
 *****            metodo imprime_256x16 - imprime matriz          ******
 *****                 rgumento = int [256][16], string           ******
 *********************************************************************/
public static int imprime_256x16(int [][]TAB, String tipo){
int lixo=0;
int linha_polinomio [] = new int [8];
System.out.println(" ");
System.out.println(" ");
System.out.println("Matriz de "+tipo);
for (int i=0;i<256;i++){
        linha_polinomio = CG2a8(i);//a variavel entrada foi usada apenas por conveniencia
        for(int k=0;k<8;k++){
               System.out.print( linha_polinomio [k]);
              }// fim do for j
         System.out.print("(Linha  "+i+"): ");
         if (i<10) System.out.print(" ");
         if (i<100) System.out.print(" ");
         System.out.print("{");
         for(int j=0;j<16;j++){
            System.out.print(TAB [i][j]);
            if (j<15)System.out.print(",");
            if (TAB [i][j]<100)System.out.print(" ");
            if (TAB [i][j]<10)System.out.print(" ");
            if (j==15) System.out.println("},");
         }// fim do for j
}// fim do for i
System.out.println("};");
return lixo;
}// fim do metodo imprime_256x16

/*********************************************************************
 ***** metodo CG2a4- converte inteiro ate 15  em polinomio        ******
 *****                           argumento = int                  ******
```

```
 *****          retorna = um polinomio do CG(2^4)                  ******
 ***************************************************************/
public static int [] CG2a4 (int inteiro){
int polinomio[]={0,0,0,0};
String temp=inteiro + "   ";
int p2=8;
for (int i=0;i<4;i++){
    if (inteiro>(p2-1)){//bit mais significativo à esquerda
        polinomio[i]=1;
        inteiro=inteiro-p2;
        }
    temp=temp+polinomio[i];
    p2=p2/2;
    }
//System.out.println(temp);
return polinomio;
}// fim do metodo CG2a8

/*************************************************************************
 ** metodo teste linear - monta todas as equacoes lineares possiveis para  *
 ** a tabela de substituicao e testa em quatas das 256 substituicoes       *
 ** possiveis cada a equacao vale                                      ****
 ***                   retorna tabela 256x16 = int [][]                 ****
 ***                    argumento  tabela 16x16                         ****
 *************************************************************************/
public static int [][] teste_linear (int [][] TAB, int imprime){
int vezes_que_equacao_vale [][] = new int [256][16];//esta matriz armazena o número de vezes que cada equação vale
                                        // o valor armazenado em 0,0 nao deve ser considerado, pois é uma eq. que
nao existe
int lixo=0;
int entrada [] = new int[8];
int saida [] = new int [4];
int [] primeiro_membro = new int[8];//vetor quer armazena quai os elementos do lado esquerdo da equacao
int [] segundo_membro = new int[4];//vetor quer armazena quai os elementos do lado direito da equacao
int esquerdo, direito, numero_da_equacao;
if (imprime==1){
    System.out.println(" ");
    System.out.println("TESTE DE CARACTERÍSTICAS LINEARES DA TABELA DE SUBSTITUIÇÃO");
    System.out.println("Entrada: e0 e1 e2 e3 e4 e5 e6 e7        Saida: s0 s1 s2 s3");
```

```
    }
int max=0;//numero de sucessos da equacao com mais sucessos
int min=256;//numero de sucessos da equacao com menos sucessos
int eqmax=0;
int eqmin=0;
int maior_distancia=0;

for (int i=0;i<256;i++){//porque a entrada tem 8 bites
    for (int j=0;j<16;j++){//porque a saida tem 4 bites
        numero_da_equacao = 16*i+j;
        String equacao = "0 ";//vai acumular a expressao algebrica da equacao
        primeiro_membro = CG2a8(i);// gera todas as equacoes possíveis(lado esq)
        segundo_membro = CG2a4(j);// gera todas as equacoes possíveis(lado dir eq)
        for (int k=0;k<8;k++){
            if (primeiro_membro[k]==1) equacao = equacao+"+ e"+k+" ";
        }// fim do for k
        equacao = equacao + "   =   0 ";
        for (int k=0;k<4;k++){
             if (segundo_membro[k]==1) equacao = equacao+"+ s"+k+" ";
        }// fim do for k
        for (int teste=0;teste<256;teste++){// testa todas as substituicoes
            entrada = CG2a8(teste);
            int auxiliar = SUB(TAB, teste);
            saida = CG2a4( auxiliar);
            esquerdo=0;//variavel que vai acumular o XOR do primeiro membro da equacao (i,j)
            direito=0;//variavel que vai acumular o XOR do segundo membro da equacao (i,j)
                      //OBS: Eq (i,j) é a equacao 16*i+j
            for (int k=0;k<8;k++){
                esquerdo = esquerdo + (primeiro_membro[k]*entrada [k]);
            }
            for (int k=0;k<4;k++){
                direito = direito + (segundo_membro[k]*saida [k]);
            }
            esquerdo=esquerdo%2;
            direito=direito%2;
            if (esquerdo==direito) vezes_que_equacao_vale [i][j] = vezes_que_equacao_vale [i][j] + 1;
        }// fim do for teste

        if (vezes_que_equacao_vale [i][j] > max){
```

```
            max = vezes_que_equacao_vale [i][j];
            if(max==256)max=0;
            eqmax = numero_da_equacao;
            }
        if (vezes_que_equacao_vale [i][j]<min){
            min = vezes_que_equacao_vale [i][j];
            eqmin=numero_da_equacao;
            }
 //       if (vezes_que_equacao_vale [i][j]!=128){
 //           if (imprime==1){
 //               System.out.print("  Eq "+numero_da_equacao+": "+equacao+"  ");
 //               System.out.println("Probabilidade p="+vezes_que_equacao_vale[i][j]+"/256.");
 //               }//fim do if
 //           }//fim do if
        }// fim do for j
}//fim do for i
//if (imprime==1)System.out.println("Maximo= "+max+"/256(Eq:"+eqmax+")  e   Mínimo="+min+"/256 (Eq:"+eqmin+")");
//maior_distancia = max-128;
//if (maior_distancia < 128-min) maior_distancia = 128-min;
return vezes_que_equacao_vale;

}// fim do metodo teste_linear

/*************************************************************************
 ***              metodo teste diferencial - matriz de diferenças       ****
 ***            retorna matriz 256x16 de distribuicao de diferencas     ****
 *************************************************************************/

public static int [][] teste_diferencial (int [][] TAB, int imprime){

/*********************CONSTRUCAO DA MATRIZ DE DIFERENCAS******************
 *SERAO MOTADOS TODOS OS PARES POSSÍVEIS (I,J) I DE 0 A 255 E J DE 0 A 255*
 * PARA CADA UM SERÁ CALCULADA A DIFERENCA NA ENTRADA "DE" E A DIFERENCA  *
 * NA SAIDA "DS" E É INCREMENTADO O ACUMULADOR NA LINHA 'DE' E COLUNA "DS"*
 *************************************************************************/
int max = 0;
int  i,j,k, entrada2_decimal, DS_decimal;
int dif_ent_x_dif_sda [][] = new int [256][16];//matriz que vai ocorrencias de diferencas
int entrada1 [] = new int [8];
```

```
int entrada2 [] = new int [8];
int DE  [] = new int [8];
int saida1 [] = new int [4];
int saida2 [] = new int [4];
int DS [] = new int [4];
    for(i=0;i<256;i++){//i e a diferenca decmal na entrada, todas as diferencas possiveis
        for(j=0;j<256;j++){
            entrada1 = CG2a8(j);
            DE = CG2a8(i);
            for (k=0;k<8;k++) entrada2[k]=(entrada1[k]+DE[k])%2;

entrada2_decimal=128*entrada2[0]+64*entrada2[1]+32*entrada2[2]+16*entrada2[3]+8*entrada2[4]+4*entrada2[5]+2*entrada2
[6]+entrada2[7];
            saida1 = CG2a4(SUB(TAB,j));
            saida2 = CG2a4(SUB(TAB,entrada2_decimal));
            for (k=0;k<4;k++) DS[k]=(saida1[k]+saida2[k])%2;
            DS_decimal=8*DS[0]+4*DS[1]+2*DS[2]+DS[3];
            dif_ent_x_dif_sda [i][DS_decimal]=dif_ent_x_dif_sda [i][DS_decimal]+1;
            if (dif_ent_x_dif_sda [i][DS_decimal] > max )  max = dif_ent_x_dif_sda [i][DS_decimal];
          }// fim do for j
    }// fim do for i
 /*************************************************************************
                  IMPRIME MATRIZ    dif_ent_x_dif_sda                    **
*************************************************************************/
if (imprime==1){
    System.out.println(" ");
    System.out.println("Matriz de Distribuição de Diferenças ");
    for(i=0;i<256;i++){
        entrada1 = CG2a8(i);//a variavel entrada foi usada apenas por conveniencia
        System.out.print( "Delta Ent: ");
        for(j=0;j<8;j++){
               System.out.print( entrada1[j]);
             }// fim do for j
         System.out.print(" (linha  "+i+"): ");
         if (i<10) System.out.print(" ");
         if (i<100) System.out.print(" ");
         for(j=0;j<16;j++){
            System.out.print(dif_ent_x_dif_sda [i][j]+"  ");
            if (dif_ent_x_dif_sda [i][j]<10) System.out.print(" ");
```

```
            }// fim do for j
        System.out.println(" ");
        }// fim do for i

     /*************************************************************************                exibe somas linhas
e colunas de dif_ent_x_dif_sda         *
*************************************************************************/
        System.out.println(" ");
        System.out.println("Somas das linhas= 256     Somas das colunas = 4096 ");
} //fim do if imprime

return dif_ent_x_dif_sda;

}// fim do metodo teste_diferencial


}// FIM DE testa_conj_4_boxes
```

12.11.3 QUATRO QL REFERIDO NO ITEM 7.3.1.2

```
Para a associação,

RS/R=0.15625/(1/16)
|p-0.5|= 0.0625.

{15,5,13,11,0,4,8,10,1,12,9,14,6,3,2,7},
{2,8,0,14,3,7,11,13,4,15,12,1,9,6,5,10},
{7,13,5,3,8,12,0,2,9,4,1,6,14,11,10,15},
{14,4,12,10,15,3,7,9,0,11,8,13,5,2,1,6},
{6,12,4,2,7,11,15,1,8,3,0,5,13,10,9,14},
{10,0,8,6,11,15,3,5,12,7,4,9,1,14,13,2},
{13,3,11,9,14,2,6,8,15,10,7,12,4,1,0,5},
{5,11,3,1,6,10,14,0,7,2,15,4,12,9,8,13},
{11,1,9,7,12,0,4,6,13,8,5,10,2,15,14,3},
{12,2,10,8,13,1,5,7,14,9,6,11,3,0,15,4},
{4,10,2,0,5,9,13,15,6,1,14,3,11,8,7,12},
{1,7,15,13,2,6,10,12,3,14,11,0,8,5,4,9},
{3,9,1,15,4,8,12,14,5,0,13,2,10,7,6,11},
```

```
{8,14,6,4,9,13,1,3,10,5,2,7,15,12,11,0},
{9,15,7,5,10,14,2,4,11,6,3,8,0,13,12,1},
{0,6,14,12,1,5,9,11,2,13,10,15,7,4,3,8}

{14,12,13,10,9,8,6,2,7,3,5,15,4,1,11,0},
{8,6,7,4,3,2,0,12,1,13,15,9,14,11,5,10},
{4,2,3,0,15,14,12,8,13,9,11,5,10,7,1,6},
{0,14,15,12,11,10,8,4,9,5,7,1,6,3,13,2},
{6,4,5,2,1,0,14,10,15,11,13,7,12,9,3,8},
{3,1,2,15,14,13,11,7,12,8,10,4,9,6,0,5},
{10,8,9,6,5,4,2,14,3,15,1,11,0,13,7,12},
{1,15,0,13,12,11,9,5,10,6,8,2,7,4,14,3},
{5,3,4,1,0,15,13,9,14,10,12,6,11,8,2,7},
{2,0,1,14,13,12,10,6,11,7,9,3,8,5,15,4},
{13,11,12,9,8,7,5,1,6,2,4,14,3,0,10,15},
{11,9,10,7,6,5,3,15,4,0,2,12,1,14,8,13},
{7,5,6,3,2,1,15,11,0,12,14,8,13,10,4,9},
{15,13,14,11,10,9,7,3,8,4,6,0,5,2,12,1},
{9,7,8,5,4,3,1,13,2,14,0,10,15,12,6,11},
{12,10,11,8,7,6,4,0,5,1,3,13,2,15,9,14}

{2,11,0,6,8,10,13,9,12,5,15,4,3,14,7,1},
{0,9,14,4,6,8,11,7,10,3,13,2,1,12,5,15},
{13,6,11,1,3,5,8,4,7,0,10,15,14,9,2,12},
{5,14,3,9,11,13,0,12,15,8,2,7,6,1,10,4},
{4,13,2,8,10,12,15,11,14,7,1,6,5,0,9,3},
{15,8,13,3,5,7,10,6,9,2,12,1,0,11,4,14},
{3,12,1,7,9,11,14,10,13,6,0,5,4,15,8,2},
{8,1,6,12,14,0,3,15,2,11,5,10,9,4,13,7},
{6,15,4,10,12,14,1,13,0,9,3,8,7,2,11,5},
{14,7,12,2,4,6,9,5,8,1,11,0,15,10,3,13},
{1,10,15,5,7,9,12,8,11,4,14,3,2,13,6,0},
{12,5,10,0,2,4,7,3,6,15,9,14,13,8,1,11},
{11,4,9,15,1,3,6,2,5,14,8,13,12,7,0,10},
{7,0,5,11,13,15,2,14,1,10,4,9,8,3,12,6},
{10,3,8,14,0,2,5,1,4,13,7,12,11,6,15,9},
{9,2,7,13,15,1,4,0,3,12,6,11,10,5,14,8}

{13,14,9,11,0,4,15,12,6,2,7,8,3,10,5,1},
```

```
{3,4,15,1,6,10,5,2,12,8,13,14,9,0,11,7},
{2,3,14,0,5,9,4,1,11,7,12,13,8,15,10,6},
{7,8,3,5,10,14,9,6,0,12,1,2,13,4,15,11},
{15,0,11,13,2,6,1,14,8,4,9,10,5,12,7,3},
{4,5,0,2,7,11,6,3,13,9,14,15,10,1,12,8},
{9,10,5,7,12,0,11,8,2,14,3,4,15,6,1,13},
{11,12,7,9,14,2,13,10,4,0,5,6,1,8,3,15},
{8,9,4,6,11,15,10,7,1,13,2,3,14,5,0,12},
{1,2,13,15,4,8,3,0,10,6,11,12,7,14,9,5},
{0,1,12,14,3,7,2,15,9,5,10,11,6,13,8,4},
{10,11,6,8,13,1,12,9,3,15,4,5,0,7,2,14},
{14,15,10,12,1,5,0,13,7,3,8,9,4,11,6,2},
{5,6,1,3,8,12,7,4,14,10,15,0,11,2,13,9},
{12,13,8,10,15,3,14,11,5,1,6,7,2,9,4,0},
{6,7,2,4,9,13,8,5,15,11,0,1,12,3,14,10}
```