INSTITUTO TECNOLÓGICO DE AERONÁUTICA

Daniel Miranda Barros Moreira

Estudo do stream cipher de Turing

Trabalho de Graduação

2006

INFRA

007:51

Daniel Miranda Barros Moreira

Estudo do stream cipher de Turing

Orientadora
Prof[a]. Dr[a] Tânia Nunes Rabello – ITA

Divisão de Engenharia de Infra-Estrutura Aeronáutica

SÃO JOSÉ DOS CAMPOS
COMANDO-GERAL DE TECNOLOGIA AEROESPACIAL
INSTITUTO TECNOLÓGICO DE AERONÁUTICA

2006

Dados Internacionais de Catalogação-na-Publicação (CIP)
Divisão Biblioteca Central do ITA/CTA

Moreira, Daniel Miranda Barros
Estudo do stream cipher de Turing
São José dos Campos, 2006.
81f.

Trabalho de Graduação – Divisão de Engenharia de Infra-Estrutura Aeronáutica – Instituto Tecnológico de Aeronáutica, 2006. Orientadores: Prof[a].Dr[a] Tânia Nunes Rabello.

1. Criptografia. 2. Seqüências pseudo-aleatórias. 3. Complexidade computacional. I. Centro Técnico Aeroespacial. Instituto Tecnológico de Aeronáutica. Divisão de Engenharia de Infra-Estrutura Aeronáutica. II. Título

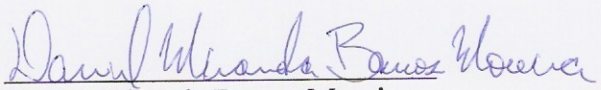
REFERÊNCIA BIBLIOGRÁFICA

MOREIRA, Daniel Miranda Barros. **Estudo do stream cipher de Turing**. 2006. 81f. Trabalho de Conclusão de Curso. (Graduação) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSÃO DE DIREITOS

NOME DO AUTOR: Daniel Miranda Barros Moreira
TÍTULO DO TRABALHO: Estudo do stream cipher de Turing
TIPO DO TRABALHO/ANO: Graduação / 2006

É concedida ao Instituto Tecnológico de Aeronáutica permissão para reproduzir cópias deste trabalho de graduação e para emprestar ou vender cópias somente para propósitos acadêmicos e científicos. O autor reserva outros direitos de publicação e nenhuma parte desta monografia de graduação pode ser reproduzida sem a autorização do autor.



Daniel Miranda Barros Moreira
Rua São Francisco de Assis, 93, Santo Antônio.
36204-634 – Barbacena - MG

ESTUDO DO STREAM CIPHER DE TURING

Essa publicação foi aceita como Relatório Final de Trabalho de Graduação

Daniel Miranda Barros Moreira, Asp.Of.
Autor

Profª. Drª Tânia Nunes Rabello – ITA
Orientadora

Prof. Dr. Flávio Mendes Neto
Coordenador do Curso de Engenharia de Infra-Estrutura Aeronáutica

São José dos Campos, 09 de novembro de 2006

Agradecimentos

Agradeço, antes de todos, a Deus, que me ajudou e continua me ajudando em todos os momentos da minha vida.

Aos meus pais, que estiveram presentes em todos os momentos da minha vida, sempre dispostos a me ajudar.

Não podia deixar de agradecer a minha orientadora, Prof. Tânia, por ter me ajudado e, mesmo sabendo que era leigo em criptografia, sempre esteve disposta a me ajudar na hora que fosse necessária.

Ao meu conselheiro no ITA, Prof. Botelho, pelo apoio educacional prestado.

Glossário

As seguintes definições são usadas nesse trabalho de graduação.

AES	Advanced Encryption Standard. O sistema de codificação/decodificação acontece por meio de cifras de bloco (block ciphers). O tamanho de cada bloco é de 128 bits.
Bit	Dígito binário que apresenta os valores 0 ou 1.
Bloco	Seqüência de bits binários. O comprimento da seqüência corresponde ao número de bits que ela contém.
Byte	Grupo de 8 bits que pode ser tratado ou como uma simples entidade ou como um conjunto individual de 8 bits.
Chave	Corresponde a uma informação (seqüência de bits) que controla a operação de um algoritmo de criptografia.
DES	Data Encryption Standard. Foi o sistema de codificação/decodificação (mecanismo de cifras de bloco, cujo tamanho é de 64 bits) padrão nos EUA entre 1976 e 2001, sendo substituído pelo AES.
Hash	Função que possui a característica de transformar um texto de qualquer tamanho em um texto ininteligível de tamanho fixo.
Keystream	Seqüência de bits gerada por meio da chave.
Know-plaintext	Situação na qual a mensagem a ser codificada é conhecida.
LFSR	Linear Feedback Shift Register. É um gerador de números aleatórios que faz a realimentação por uma recursão linear.
M-sequência	Seqüência que apresenta máximo período.
Palavra	Grupo de 32 bits que pode ser tratado tanto como uma simples entidade quanto como uma seqüência de 4 bytes.
Q-box	Substituição não-linear que transforma 8 em 32 bits. Na criptografia, tal expressão é utilizada na expansão da chave e do vetor de inicialização.
Rijndael	Esse algoritmo, cujo nome se refere aos seus inventores Rijmen e Daemen, foi escolhido para se tornar o AES, que veio substituir o DES por necessidade de segurança.
S-box	Também é uma substituição não-linear, mas que mantém as mesmas dimensões da cadeia de entrada, ou seja, pega 8 bits e leva em 8 bits.

Resumo

Nos dias atuais, a criptografia é uma área estratégica para a comunicação segura, pois as informações são, na maioria das vezes, transmitidas por meios eletrônicos e, portanto, expostas abertamente. Os sistemas criptográficos fornecem a segurança na transmissão por uso de transformações. Os streams ciphers são sistemas criptográficos que dividem o texto claro em caracteres e codificam cada um deles com uma função que depende, dentre outras coisas, da chave e do vetor de inicialização. São utilizados, por exemplo, na comunicação entre aeronaves e/ou entre aeronave-base, por serem facilmente implementáveis em hardware. O stream cipher de Turing foi projetado para que, além das características do stream cipher, seja extremamente eficiente nas implementações em software.

Nesta trabalho há o estudo dos mecanismos de funcionamento do stream cipher de Turing, compreendendo processos lineares, como a pseudo-transformação de Hadamard (PHT), e não-lineares, como as S-box e Q-box. No final há o código-fonte documentado das quatro implementações disponíveis pelo fabricante desse stream cipher, assim como o estudo dos resultados baseados numa das implementações, conhecida como *Turing Reference*.

Abstract

Lately, the cryptography is a strategic area for safe communications, since the information are transmitted using a manner electronic and, thus, displayed openly. The cryptography systems supply security in transmission by using transformations. Stream cipher is a cryptography system that divide the clear text in characters and codify each one with a function that depends, amongst other things, on the key and on the initialization vector. Stream cipher is used, for example, in the communication between aircrafts and/or aircraft-base, for being easily implemented in hardware. The stream cipher of Turing was projected so that, beyond characteristics of the stream cipher, be extremely efficient in software's implementations.

In this work, there is a study of the mechanisms of functioning of the stream cipher of Turing, including linear processes, as the pseudo-transformation of Hadamard (PHT), and nonlinear, as the S-box and Q-box. In the end, there is the source code documented of four available implementations for the manufacturer of this stream cipher, as well as the study of the results based on one of the implementations, known as *Turing Reference.*

Sumário

1 Introdução 1

2 Conceitos básicos 2

2.1 Algoritmos simétricos e assimétricos 2

2.1.1 Algoritmos modernos 2

2.2 Assinatura digital 3

2.3 Gerador de números pseudo-aleatórios 4

2.4 M-sequências 6

2.4.1 Análise da sequência obtida por meio da chave 6

2.5 Algoritmo de Berlekamp-Massey 7

2.6 Teoria não-linear de sequências periódicas 8

2.7 Contribuição de Alan Turing à criptologia 9

3 Stream cipher 12

4 Stream cipher de Turing 16

4.1 Introdução 16

4.2 LFSR utilizado no stream cipher de Turing 17

4.3 O filtro não-linear 22

4.3.1 A dependência da chave com a transformação S-box 23

4.3.2 A transformação Q-box 25

4.3.3 A pseudo-transformação de Hadamard (PHT) 28

4.3.4 O estágio final das adições 28

4.3.5 Saída 29

4.4 Chave utilizada no stream cipher de Turing 29

4.4.1 Leitura da chave 31

4.4.2 Leitura do vetor de inicialização 32

4.5 Performance 33

4.6 Segurança 34

4.6.1 Conhecimento da mensagem original 35

4.6.2 Relações com a chave e escolha do vetor de inicialização 35

4.6.3 Ataques baseados em estatísticas 36

4.6.4 Ataques usando correlações 36

5 Implementações 38

5.1 Turing Reference 38

5.2 Turing Table-lookup 43
5.3 Turing Fast 48
5.4 Turing Table-lookup/Lazy 55
6 Resultados 61
6.1 Resultados finais 61
7 Conclusão 65
8 Referências bibliográficas 66
Apêndice A – Tabela de multiplicação do stream cipher de Turing 68
Apêndice B – S-box do stream cipher de Turing 70
Apêndice C – Q-box do stream cipher de Turing 71
Apêndice D – O polinômio binário equivalente do LFSR 73

1 Introdução

A criptografia é a ciência de escrever mensagens que ninguém, exceto o remetente e o destinatário, com o conhecimento de uma chave, possam ler. Criptoanálise é a ciência de decifrar e ler estas mensagens cifradas, sem o conhecimento de nenhuma chave. As palavras, caracteres ou letras da mensagem original inteligível constituem o Texto ou Mensagem Original, como também Texto ou Mensagem Clara. As palavras, caracteres ou letras da mensagem cifrada são chamados de Texto Cifrado, Mensagem Cifrada ou Criptograma. O processo de converter Texto Original em Texto Cifrado é chamado de composição de cifra ou encriptação e o inverso é chamado de decifração ou decriptação. Na prática, qualquer mensagem cifrada é o resultado da aplicação de um sistema geral (ou algoritmo), que é invariável, associado a uma chave específica, que pode ser variável. Porém, é necessário que tanto o remetente quanto o destinatário conheçam o sistema e a chave.

De acordo com o que consta no site "Aldeia NumaBoa"[20], o primeiro exemplo documentado de escrita cifrada ocorre por volta de 1900 a.C., numa vila egípcia situada na proximidade do rio Nilo. Um arquiteto do faraó Amenemhet II construiu alguns monumentos para esse último. Porém, tais obras precisavam ser documentadas, sendo que o público em geral não poderia ter acesso às informações contidas nesse documento. A solução foi fazer a escrita em tabletes de argila e assim alterar a ordem das palavras e das sílabas das palavras, de forma que se o documento fosse extraviado, o ladrão não conseguiria chegar às obras, que possuíam grande valor, sendo conduzido para as catacumbas da pirâmide, fazendo com que tal pessoa morresse de fome.

Avançando um pouco mais na história, chega-se à Idade Moderna, mais especificamente em 1518, data na qual *Johannes Trithemius* publica o primeiro livro impresso de criptologia que se tem notícia, introduzindo a noção de troca de alfabetos a cada letra. A partir daí, o desenvolvimento dessa técnica continua e a existência de duas grandes guerras mundiais (se há uma vantagem no surgimento de uma guerra, com certeza será nos grandes avanços da ciência e tecnologia) alimenta ainda mais tal desenvolvimento, uma vez que a criptologia tornou-se essencial no envio/recebimento de mensagens, pois as comunicações não poderiam ser interceptadas. Nessa época, acentuam-se tanto os estudos da criptografia quanto da criptoanálise. Surge, então, o computador, que revoluciona praticamente todos os afazeres humanos, inclusive a criptologia. Hoje, a criptografia é essencial para muitas atividades, sendo que algumas fazem parte das *commodities* criadas pelos homens.

2 Conceitos básicos

A seguir, serão discutidos conceitos importantes dentro da criptografia. Tais conceitos são úteis na tentativa de melhor compreender o mecanismo de funcionamento do stream cipher de Turing, que é o objetivo desse trabalho.

2.1 Algoritmos simétricos e assimétricos

Existem duas classes de algoritmos baseados em chaves: os simétricos (ou de chave secreta) e os assimétricos (ou de chave pública). O que os diferencia é que os algoritmos simétricos utilizam a mesma chave, tanto para cifrar quanto para decifrar, enquanto que os algoritmos assimétricos usam chaves diferentes. É importante acrescentar que a chave de decifragem não pode ser derivada diretamente da chave de cifragem. Se fosse, a segurança da informação estaria em risco.

Os algoritmos simétricos podem ser divididos em cifras de fluxo (stream cipher) e em cifras de bloco (block cipher). As cifras de fluxo encriptam um texto claro bit a bit, ao passo que as cifras de bloco usam conjuntos com um número fixo de bits (geralmente 64 bits nas cifras modernas) como unidades de cifragem.

Segundo Meneses [3], as cifras assimétricas, também chamadas de algoritmos de chave pública ou, genericamente, de criptografia de chave pública, permitem que a chave de cifragem seja de domínio público - pode até ser publicada em jornais ou revistas. Qualquer pessoa pode, então, cifrar mensagens utilizando a chave, mas apenas o destinatário e real proprietário da chave de decifração, que deve ser mantida secreta, serão capazes de decifrar o texto, pois é o único que a conhece. A chave de cifragem também é denominada de chave pública e a chave de decifragem de chave privada ou chave secreta.

2.1.1 Algoritmos modernos

Os algoritmos criptográficos modernos estão muito distantes das chamadas cifras de lápis e papel. Algoritmos criptográficos fortes são projetados para serem executados por computadores ou por dispositivos especializados de hardware. De acordo com o site “Numaboa” [22], na maioria das aplicações, a criptografia é realizada através de software de computador. De modo geral, os algoritmos simétricos são executados muito mais rapidamente

que os assimétricos. Na prática, muitas vezes são utilizados em conjunto. Por exemplo, um algoritmo de chave pública é utilizado para cifrar uma chave gerada aleatoriamente e enviá-la aos seus usuários e então esta chave é usada para cifrar a mensagem com um algoritmo simétrico. Este processo é denominado cifragem híbrida.

2.2 Assinatura digital

Alguns algoritmos de chave pública podem ser utilizados para gerar assinaturas digitais. Uma assinatura digital é uma pequena quantidade de dados que foi criada usando-se alguma chave secreta. Através de uma chave pública é possível verificar se a assinatura realmente foi gerada com o uso da chave privada correspondente. O algoritmo usado para gerar a assinatura precisa garantir que, se a chave secreta não for conhecida, não seja possível criar uma assinatura válida.

As assinaturas digitais são usadas para verificar se uma mensagem realmente foi enviada pelo remetente declarado. Também podem ser usadas para datar documentos: uma pessoa de confiança assina o documento e sua data e hora com sua chave secreta atestando que o documento existia no momento indicado.

Assinaturas digitais também podem ser usadas para testemunhar (ou certificar) que determinada chave pública pertence a uma determinada pessoa. Isto é feito assinando-se a combinação da chave e a informação sobre o proprietário através de chave de confiança. A assinatura digital de terceiros de confiança (proprietários de chaves de confiança), a chave pública e a informação a respeito de seu proprietário, são geralmente chamadas de certificados.

Segundo o que consta na obra de Schneier [6], a confiança numa chave de terceiros pode ser obtida através de uma nova assinatura por outra chave de confiança, e assim por diante, formando uma hierarquia de confiança. Independentemente do número de assinaturas, é preciso existir uma raiz nesta hierarquia, isto é, a confiança não se deve a alguém, mas sim porque se acredita a priori que determinada chave seja de confiança. Numa infra-estrutura centralizada de chaves existem poucas raízes na rede de confiança, por exemplo, as agências governamentais de confiança. Estas raízes também são chamadas de autoridades de certificação. Numa infra-estrutura distribuída não existe a necessidade da existência de quaisquer raízes universalmente aceitas e cada pessoa pode ter raízes de confiança diferentes. Este é o conceito de teia de confiança usado, por exemplo, no PGP.

A assinatura digital de um documento qualquer é tipicamente criada computando-se um resumo do texto do documento, concatenado as informações sobre o assinante, data e hora etc. A string resultante é então cifrada usando-se um algoritmo apropriado e este bloco de bits cifrados é a assinatura. Geralmente essa assinatura é distribuída juntamente com a chave pública utilizada. Para conferir a assinatura, o destinatário precisa determinar inicialmente se acredita que a chave pertence à pessoa à qual se imagina que deve pertencer (usando a teia de confiança ou um conhecimento a priori) e depois decifrar a assinatura usando a chave pública da referida pessoa. Se a assinatura puder ser decifrada corretamente e a informação corresponder à mensagem (resumo apropriado etc.), a assinatura é aceita como válida. Maiores informações podem ser conseguidas em Cbeji [21].

2.3 Gerador de números pseudo-aleatórios

Geradores de números aleatórios são usados em aplicações criptográficas como, por exemplo, chaves. Geradores de números aleatórios convencionais, encontrados na maioria das linguagens de programação ou ambientes de programação, não são apropriados para aplicações criptográficas porque foram planejados para usar aleatoriedade estatística e não resistem à criptoanálise.

Numa situação ótima, os números aleatórios baseiam-se em fontes físicas verdadeiras de aleatoriedade que não possam ser previstas. Tais fontes podem incluir ruídos de um dispositivo semicondutor, bits menos significativos de uma entrada de áudio ou os intervalos entre interrupções de dispositivos ou da digitação de um usuário. O ruído obtido de uma fonte física é então "destilado" por uma função hash para que cada bit dependa de todos os outros bits.

Quando não for possível contar com uma aleatoriedade física, é preciso usar números pseudo-aleatórios. Esta não é uma situação desejável, mas, com freqüência, ocorre em computadores de uso geral. Sempre é desejável obter algum ruído ambiental - mesmo sendo os de latências de dispositivos, estatísticas de utilização de recursos, estatísticas de rede, interrupções de teclado ou qualquer outro. O importante é que os dados precisam ser imprevisíveis para qualquer observador externo e, para atingir este objetivo, o conjunto aleatório precisa ter no mínimo 128 bits de entropia verdadeira.

A característica dos geradores de números pseudo-aleatórios é que possuem um grande reservatório ("valor da semente" ou "seed value") com aleatoriedade. Obtêm-se bits deste

reservatório tirando dados do reservatório e opcionalmente passando estes dados por uma função hash para evitar que o conteúdo do reservatório seja revelado. Quando houver necessidade de mais bits, o reservatório é misturado cifrando seu conteúdo com uma cifra adequada de chave aleatória (que pode ser retirada de uma parte não utilizada do reservatório) de modo que cada bit do reservatório fique dependente de cada um dos outros bits. É aconselhável inserir novos ruídos ambientais no reservatório antes de misturá-lo para fazer com que a previsão de valores passados ou futuros se torne ainda mais impossível.

Apesar de não ser muito difícil criar geradores de números aleatórios criptograficamente fortes, com freqüência eles não são devidamente valorizados. A importância dos geradores de números aleatórios precisa ser enfatizada - se mal projetados, podem se tornar o ponto mais fraco de todo um sistema.

Um exemplo de geradores de números aleatórios são aqueles que fazem a realimentação por uma recursão linear. Vamos supor que o grau dessa recursão seja M. Assim, tem-se:

$$b_n = -\sum_{i=1}^{M} a_i b_{n-i}$$

Vale a pena ressaltar que essa igualdade é válida para n > M.

Nesse gerador, se houver um dispositivo de deslocamento, a estrutura será denominada de Linear-Feedback Shift Register (LFSR). Na figura abaixo, constante no trabalho de Yamamoto [12], há uma estrutura conhecida como gerador de Fibonacci, pode ser modelada com a teoria da álgebra dos corpos finitos, sendo que sua implementação em hardware é considerada muito fácil.

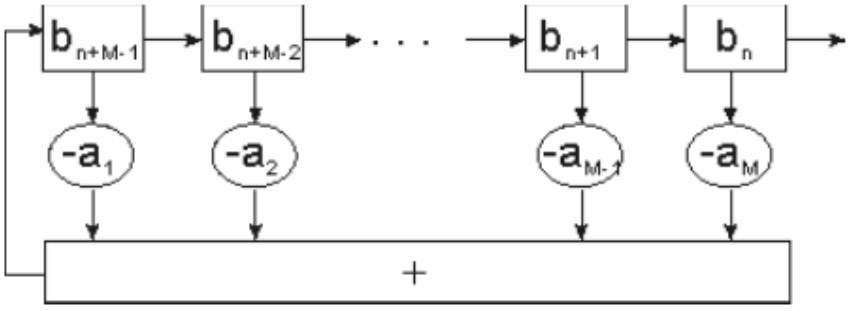


Figura 01 – Gerador de Fibonacci

Vale a pena ressaltar que nessa estrutura o coeficiente a_M não é nulo e os demais coeficientes a_i e b_i pertencem ao mesmo corpo.

2.4 M- Seqüências

O maior interesse da criptografia é encontrar LFSR que gere seqüência de máximo período. Vamos supor que essa seqüência possua **n** bits, sendo que cada bit deve possuir o valor 0 ou 1. Assim, o limitante superior do período da seqüência é 2^n, a princípio. Porém, a seqüência nula não será considerada, pois caso o LFSR atingisse o estado nulo ele não geraria outros estados uma vez que a recursão é linear. Assim, o período da seqüência é majorado por 2^n-1. As seqüências que atingem esse período máximo são chamadas de m-seqüências.

Pode-se definir um polinômio de conexão do LFSR como sendo $A(z) = a_0 z^M + a_1 z^{M-1} + ... + a_{M-1} z + a_M$. Para que tal polinômio produza uma seqüência de período máximo (m-seqüência) é necessário que ele seja primitivo. Uma das conseqüências de o polinômio ser primitivo é que a seqüência produzida vai passar por todos os estados não nulos antes de se repetir. Segundo consta no artigo de Yamamoto [12], outra característica importante é que tal seqüência goza de propriedades estatísticas muito boas.

2.4.1 Análise da seqüência obtida por meio da chave

É de extrema importância que a chave possua imprevisibilidade. Para tanto, serão discutidos alguns fatores que influenciam esse aspecto. Dentre eles, pode-se citar o período, as propriedades estatísticas e a complexidade linear.

Período: a formação da keystream normalmente é feita de forma determinística, em que o elemento atual depende apenas do anterior. Essa característica carrega uma conseqüência: a formação de seqüências periódicas. Se o período for muito pequeno, fatalmente a criptoanálise conseguirá atacar de forma eficiente, pois a mesma seqüência será utilizada na cifragem de pedaços diferentes da mensagem, caracterizando uma fraqueza conhecida como Know-plaintext. Para atingir imprevisibilidade razoável, ou seja, fazer com que uma parte da seqüência não seja usada mais de uma vez, a keystream deve possuir período muito grande.

Propriedades estatísticas: o gerador de seqüências deve ser verdadeiramente aleatório. Porém cada teste de aleatoriedade garante apenas que uma seqüência não é aleatória segundo algum critério. Para aumentar a segurança, as seqüências devem ser submetidas a vários testes, antes de serem usadas na prática. Exemplos de conjuntos de testes bastante usados são os de *Diehard*, proposto por Marsaglia [17], e NIST [18].

Complexidade linear: maneira mais usada para medir a imprevisibilidade, baseada na teoria do LFSR. A complexidade linear de uma seqüência finita periódica nada mais é do que o grau da menor recorrência linear que a gera, para algum estado inicial. Há um algoritmo eficiente que calcula a complexidade linear, o algoritmo de Berlekamp-Massey, que será apresentado a seguir.

É provável que qualquer seqüência periódica possua complexidade linear muito próxima de seu período. Vale a pena ressaltar que uma complexidade linear grande não implica necessariamente imprevisibilidade alta. Rueppel [5] analisou a forma como a complexidade linear cresce quando se consideram os n primeiros termos da seqüência. A partir de verificações experimentais, chegou-se a conclusão de que o perfil típico de complexidade linear de uma seqüência aleatória é aquele que segue de forma aproximada a reta n/2, onde n representa o número de bits. Na figura seguinte, há uma ilustração, retirada do livro de Rueppel [5], do que se está dizendo.

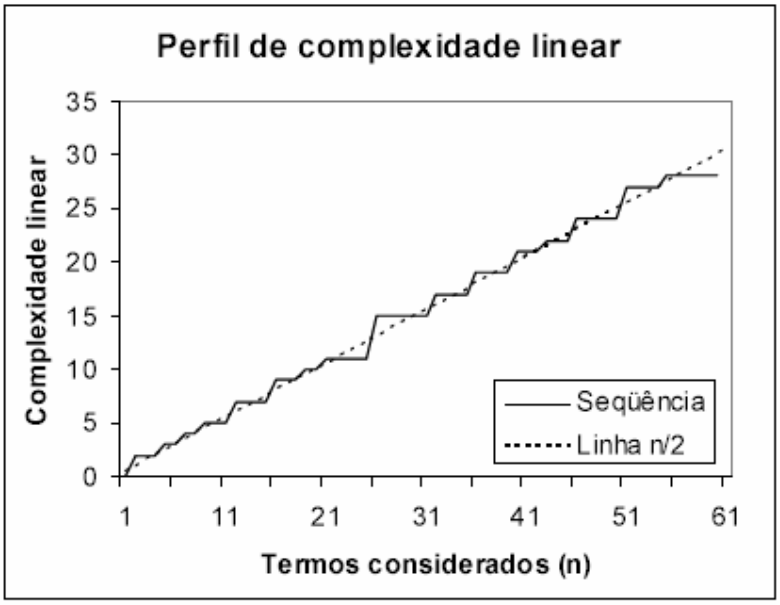


Figura 02 – Perfil de complexidade linear: (Rueppel [5])

2.5 Algoritmo de Berlekamp-Massey

Como já dito, o algoritmo de Berlekamp-Massey é utilizado para se calcular a complexidade linear de uma seqüência finita (s_0, s_1, ... , s_n), retornando, também, um polinômio característico de um LFSR com esse grau que gera os n primeiros termos desta seqüência. A implementação é feita usando-se apenas duas operações, que são as de XOR e

AND, em um mesmo corpo finito GF(q). Essa notação para tal corpo é padronizada, podendo-se encontrá-la em diversos livros, como no de Adamson [1] e no de Dean [2]. A seguir é mostrado, em pseudocódigo, o algoritmo de Berlekamp-Massey, constante no trabalho de Yamamoto [12], adaptado para GF(2):

```
INPUT: a binary sequence s^n = s_0,s_1,s_2,...,s_{n-1} of length n.
OUTPUT: the linear complexity L(s^n) of s^n, 0 ≤ L(s^n) ≤ N.

1. Initialization: C(D)←1, L←0 m←-1, B(D)←1, N←0.
2. While (N < n) do the following:
        2.1 Compute the next discrepancy d.
            d←(s_N + Σ_{i=1}^{L} c_i s_{N-i}) mod 2.
        2.2 If d = 1 then do the following:
            T(D)←C(D), C(D)←C(D)+B(D)*D^{N-m}.
            If L ≤ N/2 then L←N+1-L, m←N, B(B)←T(D).
        2.3 N←N+1.
3. Return(L).
```

Figura 03 – Pseudocódigo do algoritmo de Berlekamp-Massey

Como se pode observar, para que o algoritmo funcione, deve-se entrar com uma seqüência binária de n bits. Dentro da cadeia de comandos, há um *"while"* que faz com que ocorram n repetições, sendo que ao término da repetição *i*, a complexidade linear da seqüência (s_0, s_1, s_2, ..., s_{i-1}) é armazenada na variável L e o polinômio de grau mínimo que gera a seqüência é armazenado no vetor C(D).

Como ainda não se chegou ao final das repetições (N<n), mais um termo s_i da seqüência de entrada é considerado. Calcula-se a discrepância *d*, que é a diferença entre s_i e o termo seguinte do LFSR obtido no passo anterior. Se $d = 0$, não é necessário fazer nada e passa-se para o próximo passo. Caso contrário, é calculado o novo LFSR que gera a seqüência (s_0, s_1, s_2, ..., s_i), havendo o aumento da complexidade linear apenas no caso $L \leq N/2$. A partir dessas observações, obtém-se a seguinte lei de recorrência:

$$\begin{cases} d = 0 \Rightarrow L(s^n) = L(s^{n-1}) \\ d = 1 \Rightarrow L(s^n) = \begin{cases} L(s^{n-1}) & , se\, L(s^{n-1}) \geq n/2 \\ n - L(s^{n-1}) & , se\, L(s^{n-1}) < n/2 \end{cases} \end{cases}$$

2.6 Teoria não-linear de seqüências periódicas

A fim de aumentar a imprevisibilidade de uma seqüência, transformações não-lineares devem ser utilizadas. Tais transformações alteram a complexidade linear de uma ou mais seqüências obtidas pelos LFSR's. A função não-linear mais simples é a multiplicação binária, que corresponde à operação booleana AND. A partir dela, consegue-se expressar qualquer função $f : \{0,1\}^n \longrightarrow \{0,1\}$ em termos das operações básicas, que são as de multiplicação e soma. Esta forma de descrever a função f é denominada forma normal algébrica, representada na equação:

$$f(x_1, x_2, ..., x_n) = a_0 + a_1x_1 + a_2x_2 + ... + a_nx_n + (a_{12}x_1x_2 + a_{13}x_1x_3 + ... + a_{n-1,n}x_{n-1}x_n) + ... + a_{123...n}x_1x_2...x_n$$

A partir dessa equação, pode-se perceber que o grau da função não linear é n. Essa função pode ser aplicada tanto aos estados internos de um único LFSR quanto às saídas de LFSR's distintas. No primeiro caso, a nomenclatura será filtro não-linear. Já no segundo será denominado combinador não-linear. A figura a seguir foi retirada da obra de Stinson [8].

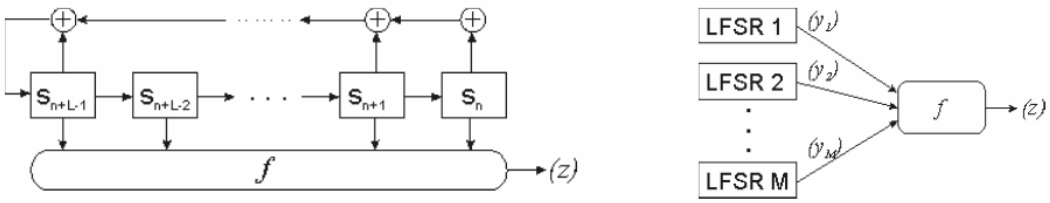


Figura 04 – Ilustração de filtro não-linear e de combinador não-linear: (Stinson [8]).

Para o caso dos filtros não-lineares, há uma relação, de difícil determinação, entre a ordem da função utilizada e a complexidade linear da seqüência resultante. Rueppel relata em seu livro que há garantias de complexidade linear mínima para a maioria das funções não lineares quando é utilizado LFSR de período máximo.

Os combinadores não-lineares também podem ser utilizados para obter uma seqüência com alta complexidade linear. Uma multiplicação das seqüências pode fazer com que a complexidade linear da seqüência resultante seja multiplicada. Em se tratando das m-sequências, a condição necessária é a de que as seqüências geradoras sejam de tamanhos diferentes, permitindo assim que as seqüências resultantes possuam alto grau de complexidade linear.

2.7 Contribuição de Alan Turing à criptologia

A proposta para realização deste trabalho é a de estudar e implementar o stream cipher de Turing. Muito embora possua tal nome, esse algoritmo criptográfico não foi desenvolvido

por Alan Turing, levando esse nome apenas em homenagem a esse ilustre colaborador. Foram várias as suas contribuições, porém as mais relevantes na área de criptologia serão agora citadas. Alan Mathison Turing nasceu em Londres, no Reino Unido, em 1912. Em 1930, licenciou-se em matemática no King's College de Cambridge, destacando-se pela suas extraordinárias capacidades intelectual e criativa. Aos 25 anos de idade, inicia a publicação de diversos artigos referentes à concepção de uma máquina autônoma, que seria um escopo do que hoje é o computador atual. Em um de seus trabalhos, Turing estabelece alguns conceitos chave, ou seja, essenciais, que são utilizados na computação moderna, resolvendo um problema da existência de um algoritmo de resposta universal.

Com o surgimento da guerra, houve grande desenvolvimento no ramo da criptologia. O governo alemão desenvolveu uma máquina denominada de *Enigma*, usada na codificação de mensagens entre as unidades do exército nazista. Porém, graças principalmente a Turing, essa máquina foi decifrada, fazendo com que a guerra se transformasse num jogo de cartas marcadas, com os ingleses podendo prever grande parte das iniciativas do exército inimigo. Desde 1939, Alan vinha trabalhando no governo britânico, nos mais avançados projetos de criptografia. Tal decifração foi possível devido ao desenvolvimento de uma máquina decifradora 'Bomb' e vários computadores 'Colossus'. Tais computadores utilizavam símbolos perfurados em fitas de papel que processavam a uma velocidade de 25.000 caracteres por segundo, cuja missão era a de decifrar os códigos alemães produzidos pela máquina alemã Enigma. Posteriormente, dirigiu o projeto Automatic Computing Engine (ACE), que chegou a desenvolver a máquina de computação mais avançada do momento e lhe permitiu conceber a idéia de 'rede neural'. Foi professor da Universidade de Manchester (1948).

O pensamento de Turing compreende um amplo campo de reflexão, já que sua concepção das máquinas inteligentes, exposta no trabalho 'Computing Machinery & Inteligence', aparecido na revista *Mind* (1950), supõe uma visão científica e filosófica que está na origem das grandes transformações que darão lugar à idéia de uma sociedade tecnológica da informação. Segundo informações contidas em [23], ele foi um dos primeiros a prever que no século XXI o termo inteligência artificial ganharia força na comunidade científica, sendo alvo de constantes pesquisas nessa área. Para Turing, não há nada humano, incluído o pensamento, que não possa ser reproduzido por uma máquina pensante, ou seja, programada para pensar. Dentre suas contribuições, há uma outra importante: desenvolveu um teste em que um operador desenvolvia algumas perguntas, que eram respondidas. Porém, essa pessoa analisaria se tal resposta vinha ou não de um computador. Se sim, o computador

poderia ser dotado de inteligência artificial, podendo ser programado para imitar qualquer sistema formal.

Em homenagem a sua pessoa, a Association for Computing Machinery confere o Prêmio Turing a uma pessoa que tenha contribuído para o desenvolvimento da ciência da computação, com apenas uma ressalva: tal contribuição deve ser duradoura. Tal prêmio é considerado como sendo o Nobel da computação.

Alan Turing suicidou-se em junho de 1954, comendo uma maçã envenenada com cianeto. Em homenagem a ele, a Apple criou a logomarca de uma maçã, com a indicação de que alguém tinha comido parte dela.

3 Stream cipher

Um algoritmo que transforma uma mensagem qualquer em uma mensagem cifrada, ou seja, em código, um bit de cada vez é chamado de stream cipher. Sua estrutura é mostrada na seguinte figura:

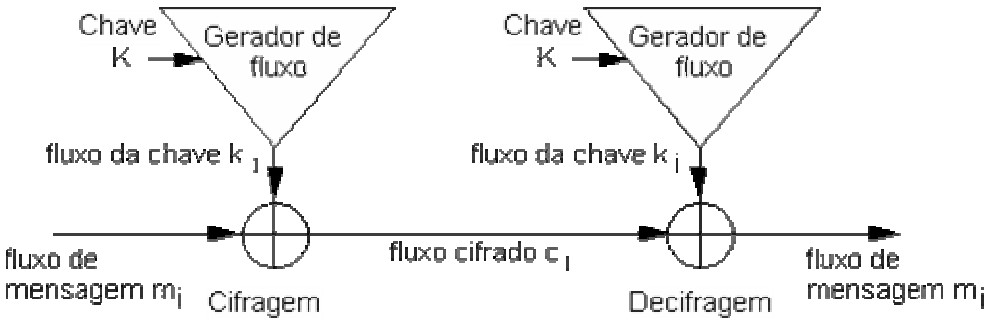


Figura 05 – Ilustração de stream cipher

Como se pode perceber, tal sistema é um algoritmo de chave simétrica. A chave K gera um fluxo de bits (fluxo esse aleatório), que na figura 05 foi chamado de fluxo da chave k_i. Esses bits encontram-se com os bits da mensagem a ser cifrada (fluxo de mensagem mi) e através da operação ou-exclusivo, produz um fluxo de bits cifrados (c_i). Processo semelhante é feito na decodificação da mensagem. A mensagem decodificada é idêntica à mensagem original, pois a chave é a mesma. Para isso, porém, os geradores de fluxo devem ser determinísticos e a seqüência gerada deve ser pseudo-aleatória, com período grande, condição necessária, mas não suficiente, para garantir boa segurança ao sistema. Isso é importante, pois há a premissa de que esses números devem ser imprevisíveis, ou seja, de que um bit não seja determinado pelos anteriores. Se o período da seqüência for pequeno, a chance de decifrar tal mensagem é alta. Idealmente o gerador deveria produzir uma seqüência aleatória infinita, fato que não acontece na prática, pois como já foi dito, os geradores de números pseudo-aleatórios são determinísticos, gerando, portanto uma seqüência periódica.

Há uma grande diferença entre stream cipher e block cipher (cifradores de fluxo e de bloco, respectivamente). Os stream ciphers trabalham com um bit de cada vez ou com pequenos blocos de bits, o que os torna difíceis de serem implementados em software, mas mais adequados para implementação em hardware sendo, porém, mais fáceis de serem analisados matematicamente. Já os cifrários de bloco podem ser mais facilmente implementáveis, porém correspondem a operações mais complexas e, portanto, mais

demoradas. Quanto à aplicação, os block ciphers são os mais utilizados, justamente por suas operações mais complicadas, o que os torna mais seguros. A utilização desses dois tipos deve depender do objetivo da aplicação e sofre influências de diversos fatores, como tempo de resposta, grau de segurança, dentre outros.

A idéia principal do stream cipher é fazer com que as funções de codificação e de decodificação usem um gerador de números pseudo-aleatórios. Assim, a seqüência de bits (inicialmente aleatória) da chave de decodificação/codificação apresenta características pseudo-aleatórias e sua representação é PRNG (K). Essa PRNG (K) apresenta grau de aleatoriedade satisfatório mesmo quando possuir comprimento menor que o tamanho original da chave. Tal característica é muito importante, pois chaves menores tornam os algoritmos criptográficos mais rápidos; e, devido a isso, muitos sistemas criptográficos utilizam essa PRNG (K), inclusive o stream cipher. Para se ter idéia de grandeza, uma chave de 128 bits com características pseudo-aleatórias é, na maioria dos casos, suficiente para garantir segurança e rapidez no algoritmo, isso se levado em conta a velocidade de processamento dos computadores atuais.

Podem-se citar diversos exemplos de stream cipher. Será citado um deles, conhecido como RC4, muito utilizado nos dias atuais e que foi desenvolvido por Ronald Rivest (personagem importante na criptologia, com várias contribuições na área). O seu algoritmo é simples e rápido, sendo, por isso usado em WEP (*Wireless Equivalent Privacy*), sistema que visa tornar as redes sem-fio tão seguras quanto às redes com fio. Este algoritmo foi mantido em segredo até o ano de 1994, ano em que uma mensagem anônima foi enviada a uma lista de discussão contendo o tal código fonte. O RC4 é utilizado para garantir a confidencialidade na troca de informações de computadores que usam a rede sem-fio. Devido a sua estrutura pequena, tal sistema possui várias vulnerabilidades, apresentando problemas de sincronização, em especial em ambientes com uma taxa de perda elevada. Ataques ao algoritmo de geração da seqüência pseudo-aleatória são considerados difíceis. Porém, a forma como é feita a inicialização a partir da chave inicial é considerada insegura.

A seqüência de bytes gerada por RC4 satisfaz todos os testes de aleatoriedade usuais, mesmo quando inicializada com uma chave de um único byte. O princípio desse sistema é o de modificar o vetor de inteiros à medida que gera a seqüência pseudo-aleatória, ou seja, ocorre uma permutação dos números de 0 a 255, gerados pela chave inicial. Logo abaixo segue a estrutura do RC4, que pode ser implementada em C++:

```
typedef unsigned char uns8;
```

```
typedef unsigned short uns16;

static uns8 state[256], x, y;

uns8 rc4step() {
uns8 t;
t = state[ y += state[++x]];
state[y] = state[x];
state[x] = t;
return (state[state[x] + state[y]]);
}
```

Um outro stream cipher conhecido é o A5. A principal aplicação é no sistema de telefonia celular GSM. É um algoritmo fraco, ou seja, não apresenta dificuldade alguma para a criptoanálise, principalmente por causa do baixo grau do polinômio de conexão e porque este polinômio é esparso. Sua implementação é eficiente em hardware, possuindo boas características estatísticas.

Como exemplo de stream cipher implementado em software, pode-se citar o SEAL (Software-optimized Encryption Algorithm), proposto por Don Coppersmith [9]. Trata-se de um sistema muito rápido. O que o diferencia dos demais é que o SEAL é uma família de funções pseudo-aleatórias, ou seja, a partir de uma chave k, ocorre a expansão, em quantidades de bits, de um número. Um número de 16 bits, por exemplo, pode ser expandido para até 32 kbytes, com objetivo de fazer com que essa chave expandida se torne indistinguível. A grande vantagem desse sistema é que, ao contrário dos stream ciphers usuais, pode-se obter um bit numa posição qualquer da chave expandida sem precisar gerar todos os bits anteriores, simplificando o problema de sincronização.

Foram citados stream ciphers considerados não muito recentes, isso se colocados na escala da evolução da criptologia. Baseado nisso, será citado um exemplo de stream cipher recente, ou seja, surgido de 2000 até os dias atuais. Um exemplo pode ser o Rabbit (RFC 4503), um stream cipher com gerador pseudo-aleatório, de chave de 128 bits e um vetor de inicialização de 64 bits, gerando uma seqüência de 128 bits, obtida por meio da operação ou-exclusivo. O método foi publicado em 2003 e é amplamente utilizado em implementações de software, apresentando codificação muito rápida. Por isso, seu uso está ocorrendo em protocolos de internet, principalmente em locais de grande quantidade de dados a serem

processados. É um algoritmo considerado muito seguro, até mesmo porque desde 2003, ano de sua divulgação, não foi constatada nenhuma fraqueza, nem pelos projetistas nem pelos usuários dele. O Rabbit foi criado com a capacidade de codificar blocos de mensagem de 2 a até 128 bits com a mesma chave, porém, os dados podem possuir blocos de maior comprimento, sendo que, nesses casos, a chave encriptadora deve ser trocada.

4 Stream cipher de Turing

4.1 Introdução

O stream cipher de Turing, cuja chave possui comprimento de 256 bits, é extremamente eficiente para implementações em software, tanto nos supercomputadores quanto nos computadores pessoais. Quando o assunto é hardware, esse stream cipher apresenta boa performance em processadores embarcados, mesmo que possuam baixíssima memória RAM. Uma de suas principais características é que há a combinação de um gerador LFSR com uma função que mistura a chave, remanescente da estrutura do block cipher. A estrutura do LFSR é baseada no SOBER-t32 [10] e no SNOW 2.0 [11].

Esse programa foi desenhado, segundo a enciclopédia Wikipedia [16], para atender as necessidades das aplicações embarcadas, como na codificação da voz em telefones sem-fio, onde ocorrem esforços no sentido de aumentar o poder dos processadores, diminuir o tamanho do espaço ocupado pelo programa e maximizar a memória disponível utilizada pelo algoritmo de codificação. Em telefones celulares, que possuem processador e memória pequenos, o uso do stream cipher de Turing é altamente recomendado, uma vez que tal software, mesmo nessas condições, é rápido. Esse stream cipher é capaz de superar a ineficiência do LFSR binário utilizando um LFSR definido num corpo $GF((2^8)^4)$, que é uma representação isomórfica do corpo $GF(2^{32})$, e uma quantidade de técnicas que aumentam a velocidade de geração de seqüências pseudo-aleatórias. A transformação de $GF(2^{32})$ para $GF((2^8)^4)$ permite-nos concluir que em Turing a análise é feita sobre os bytes e não mais sobre os bits, como é feita nos casos em geral.

Os quatro principais componentes do stream cipher de Turing são:

1 Leitura da chave;
2 Leitura do vetor de inicialização;
3 Uso de um LFSR (linear feedback shift register);
4 Filtro não linear fechado à chave;

Os dois últimos passos são mostrados detalhadamente na figura seguinte, disponível no trabalho de Rose e Hawkes [13]:

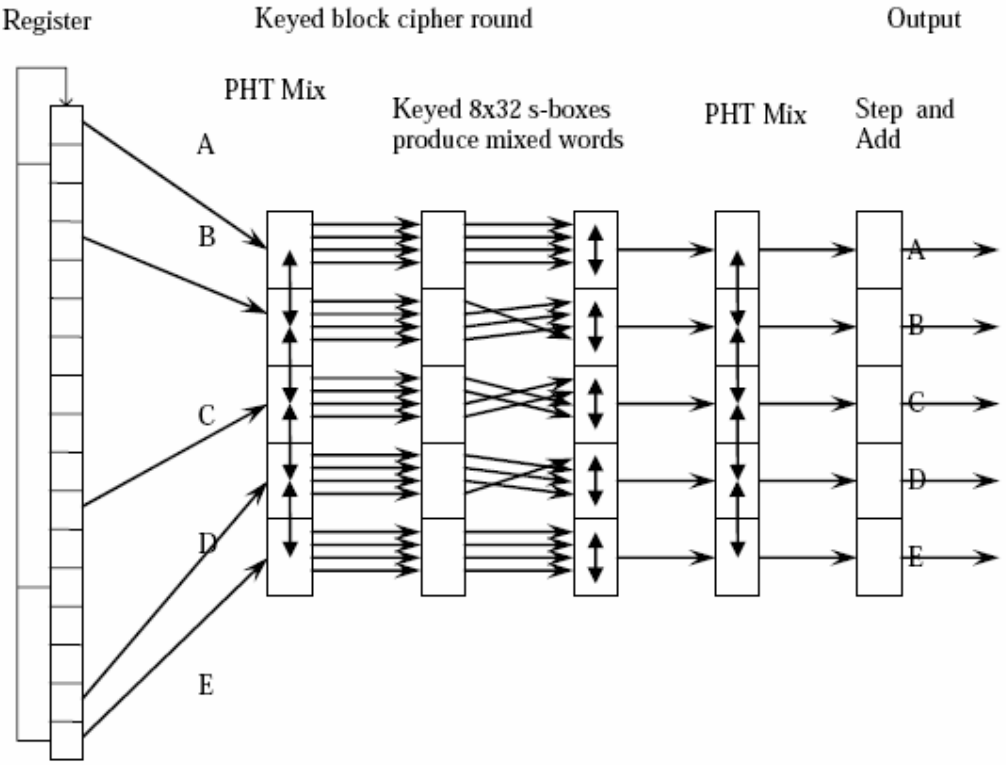


Figura 06 – Componentes do stream cipher de Turing: (Rose e Hawkes [13])

Para se obter a seqüência da chave (keystream), inicialmente são misturadas cinco palavras, selecionadas do LFSR. Vale a pena ressaltar que cada palavra possui exatamente 32 bits, ou seja, 4 bytes. A seguir, há uma transformação altamente não-linear, denominada S-box. Logo após, as palavras resultantes são misturadas novamente e a seguir combinadas com 4 novas palavras, também retiradas do LFSR. O tamanho da keystream resultante é de 160 bits.

4.2 LFSR utilizado no stream cipher de Turing

O LFSR binário pode ser extremamente ineficiente quando faz parte de uma implementação em software, por diferentes motivos ligados ao microprocessador. O linear feedback shift register é capaz de operar em diversos corpos finitos. Porém, esse será tão mais eficiente quanto mais ajustado o corpo finito estiver com o processador. Experiências mostram que o Corpo de Galois, com 2^w elementos é um corpo afim com os mais variados tipos de processadores. Vale a pena ressaltar que o número w representa o tamanho dos itens no processador, que geralmente são bytes ou palavras de 32 bits. Os elementos desse corpo e

os coeficientes da relação de recorrência ocupam exatamente uma unidade de armazenamento e podem ser manipulados em software de forma eficiente.

Dizer que um corpo é GF(2 ^{w}) significa dizer que se um elemento "a" qualquer pertence a esse corpo, então ele pode ser representado por um polinômio de grau menor ou igual a (w-1), ou seja $a_{w-1}X^{w-1} + a_{w-2}X^{w-2} + ... + a_1X + a_0$, onde cada a_i pertence a GF(2). Nesse caso, os coeficientes do polinômio podem assumir dois possíveis valores, que são 0 e 1. Podem-se também representar os elementos deste corpo como uma palavra de w bits ($a_{w-1},a_{w-2},...,a_1,a_0$), onde os a_i são os coeficientes do polinômio. Assim, os elementos do corpo GF(2^8) são representados por bytes, onde cada um de seus bits representa o coeficiente em GF(2) do polinômio de grau menor ou igual a 7. Já cada elemento do corpo GF((2^8)^4) é representado por uma palavra de 32 bits, isto é, 4 bytes, sendo cada byte o coeficiente do polinômio de grau menor ou igual a 3 que representa este elemento.

O estado de informação do LFSR é de 17 palavras de 32 bits cada uma. Como já foi definido, seja $A(z) = a_0 z^M + a_1 z^{M-1} + ... + a_{M-1} z + a_M$ (o valor de M será visto e justificado a seguir, correspondendo ao número 544) o polinômio que representa o LFSR sobre o corpo GF(2), configuração presente no apêndice D. A configuração ideal é aquela em que as atualizações do LFSR são de grande eficiência. Para tanto, as seguintes condições devem ser satisfeitas:

- LFSR deve produzir seqüências de período máximo. Quando esse polinômio for primitivo e de grau 17, o período possuirá comprimento máximo de (2^{17w} - 1), pois as unidades de armazenamento do LFSR são palavras de w bits. Como w = 32 temos que o comprimento será de (2^{544} - 1), o que aparentemente nos mostra que tal stream cipher produz uma cadeia suficientemente longa antes de se repetir, uma vez que seqüências com período acima de (2^{256} - 1) já são consideradas como seqüências longas.
- Para máxima difusão e com o objetivo de proteger contra a criptoanálise, é necessário que aproximadamente metade dos coeficientes seja 1. A explicação para isso está no fato de que a análise bruta inicia-se em um dos casos extremos: ou com todos os elementos iguais a 1 ou com todos iguais a 0.

Antes de citar o mecanismo de funcionamento do LFSR do stream cipher de Turing, convém apresentar como são feitas as operações nos corpos GF(2), GF(2^8) e GF((2^8)^4), correspondendo às operações em bits, bytes e palavras, respectivamente. As operações definidas em GF(2) são duas, a multiplicação módulo 2 e a adição módulo 2, definidas,

respectivamente, por:

- $a \odot_2 b$ é o resto da divisão de (a . b) por 2.
- $a \oplus_2 b$ é o resto da divisão de (a + b) por 2.

Em GF(2), supõe-se que "a" e "b" assumam o valor 0. Assim, tanto $a \oplus b$ quanto $a \odot_2 b$ retorna o valor 0. Se "a" assumir o valor 1 e "b", 0, então $a \oplus b$ retorna o valor 1 e $a \odot_2 b$ retorna o valor 0. Por último, se tanto "a" quanto "b" forem 1, $a \oplus b$ retorna 0 e $a \odot_2 b$ retorna 1.

Para definir as operações em GF(2^8) e em GF((2^8)^4) precisa-se da seguinte definição:

Definição: Seja F um corpo. Diz-se que um polinômio p é irredutível em F se e somente se p é um polinômio com coeficientes em F e que não pode ser decomposto como produto de polinômios com coeficientes em F de grau maior que 0 e menor que o grau de p. Assim, por exemplo, p=X^2+X+1 é irredutível em GF(2) pois seus coeficientes estão no corpo em questão e ele não pode ser decomposto no produto de polinômios de grau 1 com coeficientes em GF(2). Já o polinômio $X^2+1=(X+1)^2$ não é irredutível em GF(2).

Seja "w" um elemento qualquer no corpo GF(2^8). Como já fora dito, "w" é um byte, ou seja, possui 8 bits. Assim, a forma geral desse elemento pode ser escrita como:

w = ($a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0$), onde a_i é bit, ou seja, pertence ao corpo GF(2).

Vale a pena ressaltar que "w" também pode ser representado por um polinômio, de grau menor ou igual a 7, com coeficientes em GF(2), isto é,

$$w = a_7 x^7 + a_6 x^6 + a_5 x^5 + a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0$$

Seja "y" outro elemento de GF(2^8), representado por:

$$Y = (b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0) = b_7 x^7 + b_6 x^6 + b_5 x^5 + b_4 x^4 + b_3 x^3 + b_2 x^2 + b_1 x + b_0$$

Para definir as operações em GF(2^8) necessita-se antes de mais nada escolher um polinômio p irredutível em GF(2) de grau igual a 8. Assim as operações de multiplicação e adição módulo 2^8 são definidas da seguinte maneira:

- $a \odot_8 b$ é o resto da divisão do polinômio (a . b) por p.
- $a \oplus_8 b$ é o resto da divisão do polinômio (a + b) por p.

No caso do stream cipher de Turing o polinômio p escolhido é $z^8 + z^6 + z^3 + z^2 + 1$. A seguir será dado um exemplo destas operações.

Sejam w = (10100001) = $z^7 + z^5 + 1$ e y = (01100000) = $z^6 + z^5$ elementos de GF(2^8). Então:

- $(w \oplus_8 y) = z^7 + z^6 + 1 = (11000001)$. Percebe-se então que o coeficiente do termo de grau 5 é zero, pois os coeficientes estão em GF(2). Como o grau do polinômio resultante é menor que 8, o resto da divisão deste pelo polinômio p será o próprio resultado da soma, o que justifica esse resultado obtido.
- $(w \odot_8 y) = z^6 + z^4 + z^3 + z^2 + 1 = (01011101)$. A simples multiplicação destes polinômios resulta o polinômio $z^{13} + z^{12} + z^{11} + z^{10} + z^6 + z^5$. Esse polinômio, dividido pelo polinômio irredutível $z^8 + z^6 + z^3 + z^2 + 1$, gera como resto o polinômio $z^6 + z^4 + z^3 + z^2 + 1$, resultado da operação.

O stream cipher de Turing trabalha no corpo $GF((2^8)^4)$, no qual a forma geral de seus elementos é um polinômio de grau menor ou igual a 3, com coeficientes em $GF(2^8)$, que também pode ser interpretado como uma palavra de 4 bytes, isto é, se w é um elemento de $GF((2^8)^4)$ então

$w = (A_3A_2A_1A_0) = A_3Y^3+A_2Y^2+A_1Y+A_0$, onde A_i é byte, ou seja, pertence ao corpo $GF(2^8)$.

Na literatura computacional, a representação dos polinômios quando os coeficientes são bytes segue a representação da base hexadecimal, exemplificada a seguir:

$$W = Y^4 + 0xD0.Y^3 + 0x2B.Y^2 + 0x43.Y + 0x67$$

Para correta identificação desses coeficientes, basta coletar os elementos à direita de 0*x* e transformá-los para a base binária. Por exemplo, na base hexadecimal o elemento "D0" equivale aos algarismos 13 e 0, respectivamente, na base decimal. Transformando para a base binária, temos que 13 nada mais é do que 1101 e 0 é 0000. Juntando-se esses elementos, temos um conjunto de 8 bits (1 byte), dado por 11010000, que é o coeficiente do termo Y^3. Assim, por exemplo, o elemento $W=Y^3+0xA3Y^2+0xF1$ de $GF((2^8)^4)$ pode ser representado como sendo a palavra de 4 bytes dada por W=(0x01,0xA3,0x00,0xF1).

Novamente para se definir as operações em $GF((2^8)^4)$ precisa-se escolher um polinômio q irredutível em $GF(2^8)$. As operações de adição e multiplicação módulo 2^{32} são definidas da seguinte forma:

Sejam w_1 e w_2 palavras, isto é, elementos do corpo $GF((2^8)^4)$, ou seja, $w_1 = B_3Y^3 + B_2Y^2 + B_1Y + B_0 = (B_3B_2B_1B_0)$ e $w_2 = C_3Y^3 + C_2Y^2 + C_1Y + C_0 = (C_3C_2C_1C_0)$, então

- $(w_1 \oplus_{32} w_2) = (B_3 \oplus_8 C_3)Y^3 + (B_2 \oplus_8 C_2)Y^2 + (B_1 \oplus_8 C_1)Y + B_0 \oplus C_0$
- $(w_1 \odot_{32} w_2)$ é o resto da divisão do polinômio $(B_3 \odot_8 C_3)Y^6 + (B_3 \odot_8 C_2 \oplus_8 B_2 \odot_8 C_3)Y^5 + ... + B_0 \odot_8 C_0$ pelo polinômio

irredutível q, que no caso do Stream Cipher de Turing é q=$W = Y^4 + 0xD0.Y^3 + 0x2B.Y^2 + 0x43.Y + 0x67$.

Uma vez explicado como são feitas as operações nos diversos corpos, o próximo passo consiste no estudo do mecanismo de funcionamento do LFSR utilizado no stream cipher de Turing.

O LFSR utilizado é constituído por 17 "caixas", em geral denominadas registradores, sendo que cada "caixa" contém uma palavra de 32 bits. Ao definir o que seria o LFSR, foram citadas as principais propriedades, que é a do deslocamento e a da realimentação. A realimentação acontece de acordo com o seguinte polinômio característico: $x^{17} + x^{15} + x^4 + \alpha$, onde α pertence a GF($(2^8)^4$) e corresponde ao polinômio Y, isto é, à palavra (0x00,0x00,0x01,0x00).

Após essas definições, pode-se perceber que a implementação do LFSR pode ser feita de forma eficiente e rápida, pois a constante α é simples e bem definida. Como fora dito, cada "caixa" é uma palavra, ou seja, conjunto de 32 bits, que pode ser representado sob a forma polinomial $A_3Y^3 + A_2Y^2 + A_1Y + A_0$. Ao se multiplicar este polinômio pelo polinômio Y, obtém-se $A_3Y^4 + A_2Y^3 + A_1Y^2 + A_0Y$. Porém, o resultado da operação de multiplicação em GF($(2^8)^4$) é o resto da divisão desse polinômio pelo polinômio irredutível $W = Y^4 + 0xD0.Y^3 + 0x2B.Y^2 + 0x43.Y + 0x67$. O resultado dessa operação é $(A_2 \oplus_8 (-(A_3 \odot_8 0xD0)))Y^3 + (A_1 \oplus_8 (-(A_3 \odot_8 0x2B)))Y^2 + (A_0 \oplus_8 (-(A_3 \odot_8 0x43)))Y - (A_3 \odot_8 (0x67))$
Assim, ao multiplicar-se um elemento do GF($(2^8)^4$), que pode ser representado como uma palavra de 4 bytes, pelo elemento Y=(0x00,0x00,0x01,0x00) de GF($(2^8)^4$), faz-se primeiramente uma translação à esquerda de 8 bits, ou seja, um byte e em seguida uma operação de ou-exclusivo (XOR) desse resultado com os valores de uma tabela pré-calculada, valores esses que devem se encontrar em um arquivo em anexo. É importante citar que a operação é iniciada pelos 8 bits mais significativos, ou seja, pelo coeficiente do termo de maior grau do polinômio.

A atualização do LFSR é feita da seguinte forma: a palavra que sai do registrador R_1, que ocupa a posição 1, será chamada R_0 e é multiplicada por α. Após essa multiplicação (módulo 2^{32}) por α, acontece a adição (módulo 2^{32}) desse resultado com as palavras que estão nos registradores R_{15} e R_4, que ocupam as posições 15 e 4. Esse resultado final vai para o registrador 17, enquanto que a palavra que ocupava a posição 17 vai para a posição 16, a 16 vai para a 15 e assim sucessivamente até que a palavra que ocupava a posição 1 sai e diz-se que ela ocupa a posição 0. Para melhor compreensão, a seguir encontra-se uma figura

mostrando essa atualização.

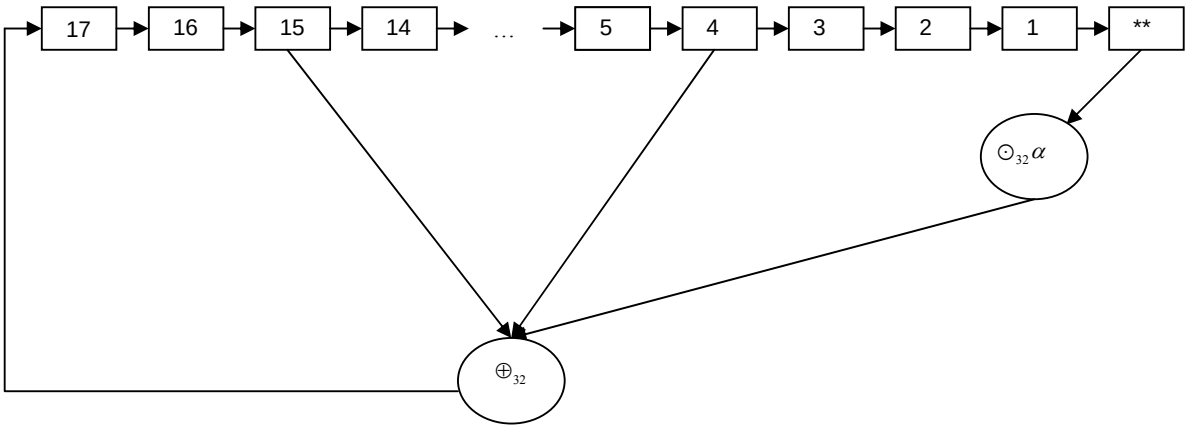


Figura 07 – Atualização do LFSR do stream cipher de Turing

Em linguagem C, que é a utilizada neste trabalho para implementar o stream cipher de Turing, para obter a nova palavra utiliza-se a seguinte linha de comando:

new_ = R[15]_ ^_ R[4]_ ^_ (R[0]_ <<_ 8)_ ^_ Multab[R[0]_ >>_ 24];

Vale a pena ressaltar que a função "Multab" corresponde à tabela de multiplicação, exemplificada no apêndice A.

Já para o deslocamento, é utilizado o comando:

R[0]_ =_ R[1];_ R[1]_ =_ R[2];_ ..._ ;_ R[16]_ =_ R[17];_ R[17]_ =_ new;

Após atualização do LFSR por esse processo estudado, 5 palavras são escolhidas como entrada no filtro não-linear, que corresponde à próxima etapa na produção do keystream. As 5 palavras escolhidas ocupam as posições dos registradores R_{16}, R_{13}, R_6, R_1 e R_0, sendo R_0 aquela que ocupava a posição 1 no instante anterior à última atualização e que foi definida como a palavra que ocupa a posição 0. Daqui para frente, essas palavras escolhidas serão tratadas como A, B, C, D e E, respectivamente. Antes de escolher mais 5 palavras para que o processo continue, o LFSR avança duas vezes. Assim das 17 palavras de um determinado estado do LFSR, exatamente 9 palavras distintas são usadas. Vale a pena ressaltar que nenhum par de palavras será usado mais de uma vez, pois caso contrário, isso produziria uma vulnerabilidade estatística, que facilitaria o trabalho da criptoanálise.

4.3 O filtro não-linear

Basicamente, nessa etapa estão previstos três componentes: S-box, Q-box e a pseudo-transformação de Hadamard. Dentre esses, tem-se que os únicos componentes altamente não lineares do stream cipher de Turing são as S-box e as Q-box. Porém, outras não-linearidades

existem, advindas da combinação da adição módulo 2^{32} com a operação XOR. Enquanto cada uma destas operações é linear em seu respectivo grupo matemático, elas podem ser consideradas não-lineares em outros grupos que não os seus.

O filtro não-linear no stream cipher de Turing consiste em:

1 Seleção de cinco palavras para a entrada no filtro não-linear advindas do LFSR
2 Misturar as palavras usando a pseudo-transformação de Hadamard
3 Transformação dos bytes e mistura das palavras, usando quatro transformações não-lineares S-box, dependentes da chave, levando 8 bits em 32 bits (8 → 32)
4 Uma nova mistura das palavras usando a pseudo-transformação de Hadamard
5 Seleção de novas palavras, adicionando-as àquelas que saíram das S-box, seguindo a operação de adição mod 2^{32}.

4.3.1 A dependência da chave para com a transformação S-box

Cinco palavras são selecionadas para dar início a um processo não linear, passando pela S-box e depois pela Q-box. A S-box possui um mecanismo responsável por criar permutações fixas em cada byte de cada uma das palavras selecionadas. Comumente essa operação é representada por 8 → 8. Após esse processo, acontece a expansão de cada byte em quatro novos bytes, representada por 8 → 32, utilizando uma transformação não-linear fixa. Esse processo é conhecido como Q-box. Assim, as S-box e Q-box transformam cada palavra em 4 palavras. Para que o resultado seja uma única palavra é feito um XOR entre estas palavras, ou seja, considerando que a palavra A foi transformada pelas S-box e Q-box nas palavras A_1,A_2,A_3,A_4 então o resultado é $A' = A_1 \oplus A_2 \oplus A_3 \oplus A_4$. Porém, há de se acrescentar que esses novos bytes formados devem ser independentes, pois caso contrário esse algoritmo seria tecnicamente fraco perante a análise estatística. O processo não-linear é não inversível. Essa conclusão pode ser constatada com facilidade, porque o processo que leva 8 bits em 32 bits não é sobrejetor. Portanto, não é considerado uma bijeção. Se não temos uma bijeção, esse processo não é inversível. Essa informação é muito importante, pois mesmo que se descubra o código cifrado, não se conseguiria chegar à mensagem original.

Ainda para evitar ataques potenciais, as palavras B, C e D sofrem rotações à esquerda internamente de, respectivamente, 8, 16 e 24 bits. Essas rotações acontecem antes do início do processo S-box, ou seja, antes da permutação interna de cada byte, que, conforme citado, é representada como 8 → 8.

A intenção é estudar a fundo tanto o S-box quanto o Q-box. Na nomenclatura da linguagem C, quando for feita alguma referência à transformação S-box, será utilizado o formato Sbox[.]. Essa transformação é fixa, acontecendo uma permutação interna em cada byte das palavras, sendo que essa permutação deve possuir uma não-linearidade mínima de 104. Vale a pena ressaltar que o mecanismo utilizado e responsável por garantir essas características é o mesmo que está presente no RC4.

No stream cipher de Turing, o algoritmo do RC4 foi chaveado com "Alan Turing", totalizando 11 caracteres, presentes num arquivo texto para leitura no estilo ASCII. Esse algoritmo utilizado no RC4 e responsável pela permutação S-box, foi testado pela QUALCOMM [19], empresa que criou o stream cipher de Turing, obedecendo alguns critérios, como não-linearidade e relação entre os bytes gerados, considerando-se se há relação de dependência entre eles. Essa empresa fabricante divulgou, para fins de constatação, que foi gerada uma seqüência de 10000 bytes para análises, sendo que a menor não-linearidade foi conseguida com um polinômio de grau 104, ocorrendo depois que 736 bytes haviam sido gerados. O valor 10000 bytes não foi um valor escolhido aleatoriamente. Para cadeias maiores, a análise se torna mais complicada, pois a complexidade linear aumenta sensivelmente, o que é de se esperar para geradores que devem ter características de imprevisibilidade.

Há um ponto forte nesse processo e que foi gerado por pura coincidência: ao longo de toda a cadeia, Sbox[x] $\neq$ x, ou seja, não há pontos fixos nesta transformação.

Esse stream cipher é considerado pelos analistas como sendo rápido. Um dos fatores que ajuda nessa rotulação é o fato de que esses processos não-lineares (S-box e Q-box) são desencadeados com o auxílio de tabelas que devem ser pré-carregadas no início do programa, no formato ASCII. As tabelas usuais para S-box e Q-box encontram-se no apêndice B e C, respectivamente.

Na tabela do S-box, há 8 colunas e 32 linhas de valores, totalizando 256 possíveis valores. Esse número (256) se justifica pelo fato de que a S-box transforma 1 byte em outro, sendo que 1 byte admite $2^8 = 256$ possíveis combinações. Para ilustrar, vamos supor que um dos bytes de entrada na S-box seja (0001001). Esse valor equivale ao número 9 na base decimal. Assim, como cada linha apresenta 8 valores, a S-box retornará o valor contido na segunda linha e na primeira coluna, isto é o nono valor que aparece na tabela quando se percorre a tabela pelas linhas, que, de acordo com o apêndice B, corresponde a 0x7c, que na representação binária equivale a (01111100). O que foi explicado corresponde ao mecanismo interno de funcionamento da S-box, sendo que os valores de entrada na S-box serão

explicados quando já tiver sido ilustrado o mecanismo da Q-box.

4.3.2 A transformação Q-box

A Q-box é uma transformação fixada que pega 8 bits e leva em 32 bits. Quando do desenvolvimento desse stream cipher, a QUALCOMM solicitou que a Universidade Tecnológica de Queensland desenvolvesse um algoritmo que utilizasse 32 funções "booleanas" independentes para 8 bits de entrada. Matematicamente:

$$f_i(b_0b_1b_2b_3b_4b_5b_6b_7) = c_i \text{, onde i varia entre 0 e 31}$$

Porém, essas funções deveriam obedecer a alguns critérios:

- Devem ser altamente não lineares (foi observado que cada uma possuía não-linearidade de 114).
- Devem ser balanceadas, ou seja, apresentar aproximadamente metade dos termos sendo 0 e a outra metade 1, condição muito próxima da máxima difusão. Essa característica é importante, pois aumenta a resistência à criptoanálise, principalmente aquela que utiliza a força bruta.
- Não devem ser correlacionadas aos pares. Se elas fossem, a partir da descoberta de uma função, conseguir-se-ia encontrar outras, fato que constituiria uma fraqueza frente à criptoanálise.

O resultado da operação dessas funções encontra-se pré-calculado no apêndice C. Novamente, há 256 possíveis valores, dado que os valores que entram na Q-box são aqueles que saem da S-box, possuindo, portanto, o tamanho de 1 byte. No stream cipher de Turing, a tabela Q-box é carregada no formato ASCII, apresentando em cada linha 4 valores, necessitando, para completar os 256 valores possíveis, de 64 linhas. Para exemplificar, será retomado o exemplo da S-box, em que o valor de entrada na S-box era (0001001) e o valor de saída 0x7c, que na representação binária equivale a (01111100). Esse último valor equivale, na representação decimal, ao número 124. Ao entrar na Q-box o valor de saída da S-box, que nesse exemplo é 0x7c, sairá da Q-box o valor constante na linha 31 e na coluna 4 da tabela do apêndice C, que é 0xe4afb226. Esse valor corresponde a 1 palavra, ou seja, a 32 bits. Nesse caso, os bytes são:

- 26, cuja representação binária é (00100110).
- b2, cuja representação binária é (10110010).
- af, cuja representação binária é (10101111).
- e4, cuja representação binária é (11100100).

Quatro bytes (uma palavra) de entrada na S-box geram 4 bytes de entrada na Q-box, gerando 16 bytes de saída, ou seja, 4 palavras. O exemplo feito na página anterior tomava 1 byte de uma palavra e levava a 4 novos bytes, relativos à nova palavra que é formada, representada por A_1. Os 3 próximos bytes são levados nas palavras A_2, A_3 e A_4, respectivamente. Essas quatro palavras devem se transformar em apenas uma, processo esse que é conseguido por meio da operação XOR nessas quatro palavras.

Foram explicados os mecanismos internos tanto da S-box quanto da Q-box. Agora será dada mais atenção aos valores que entram nesses dispositivos não-lineares. O byte de entrada, antes de iniciar o processo na S-box, é combinado com os bytes da chave. É importante dizer que o material da chave é acessado como palavras K_i, $0 \leq i < N$, sendo que N é o número de palavras da chave. Como cada palavra possui exatamente 4 bytes, conclui-se que o comprimento da chave deve ser de um múltiplo de 4 bytes.

O resultado da combinação do byte de entrada com um byte da chave passa pela função S-box e gera uma saída. Essa saída é combinada com outro byte da chave (byte esse que deve estar em outra palavra e ocupando a mesma posição relativa que aquele inicial) passando novamente pela S-box e assim sucessivamente até que haja N combinações. Computacionalmente:

$$t_i(x) = Sbox[K_{i,N-1} \oplus Sbox[K_{i,N-2} \oplus \ldots Sbox[K_{i,0} \oplus x]\ldots]]$$

Os bytes $K_{i,j}$ mencionados são bytes das palavras da chave. O índice j ($0 \leq j < N$, em que N é o número de palavras da chave) indica a localização da palavra na posição j da chave. Já o índice i ($0 \leq i \leq 3$) indica a posição do byte na palavra, sendo que a numeração 0 indica o byte que ocupa a posição mais significativa dentro da palavra.

Por exemplo, supõe-se que a combinação ocorra com o byte 1 da palavra. Na figura 08 há a exemplificação do mecanismo utilizado no stream cipher de Turing e que foi representado pela fórmula computacional representada por t_i (x).

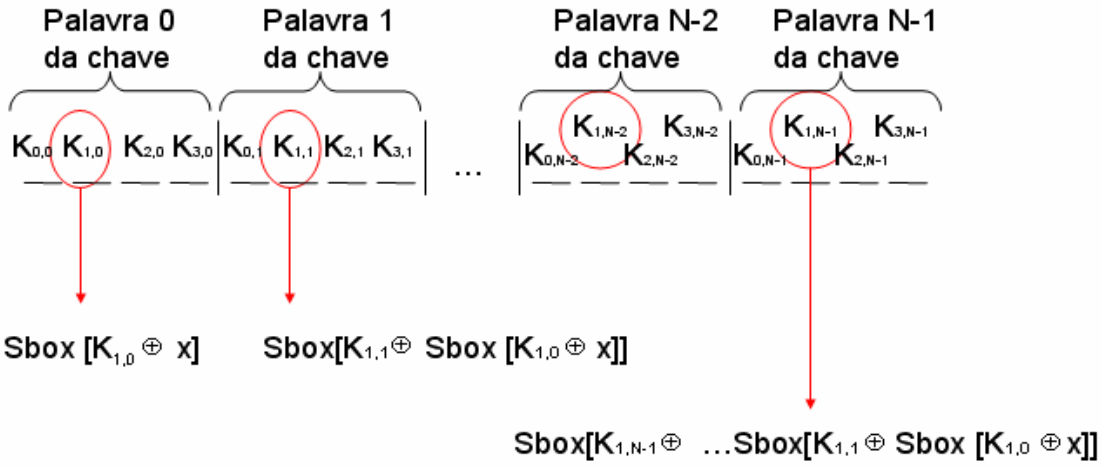


Figura 08 – Exemplificação da S-box do stream cipher de Turing

Em paralelo, acontece um processo que utiliza a função Q-box. A cada mexida, ou seja, a cada vez que é solicitada a operação S-box, uma palavra rotacionada que vem da função Q-box é acumulada em uma palavra temporária. Vale a pena ressaltar que a rotação dos bytes na Q-box depende da posição do byte na palavra e de qual palavra dentro da chave que se está trabalhando. Além disso, essa rotação garante que nenhuma entrada na Q-box possa cancelar resultados vindos de outros bytes de entrada em momentos anteriores ao atual.

Essa palavra acumulada é altamente não-linear com relação aos bytes de entrada, pois além da função Q-box , que apresenta não linearidade acentuada, tem-se ainda a rotação dos bytes, juntamente com a operação XOR. A palavra em questão apresenta também grande dependência dos bytes que estão na chave. Apesar dessas características, é importante acrescentar que as posições dos bits dessa palavra temporária provavelmente não são balanceadas, isto é, há posições que apresentam, estatisticamente, mais zeros ou uns, fato que caracteriza uma fraqueza. Já o byte t, representado anteriormente como sendo o byte que contém as combinações da chave com o byte de entrada, aplicadas à função S-box, é balanceado, pois a função S-box, além de apresentar alta não linearidade, apresenta, também, resultados balanceados.

Quando o byte t já houver sido formado, o programa do stream cipher de Turing retorna a palavra final relativa àquele byte de entrada. Essa palavra final estava sendo acumulada numa posição temporária de tamanho 32 bytes. A seguir tem-se a representação em linguagem C desses processos. Como já foi dito, a rotação da palavra depende de dois fatores, dentre eles da posição do byte dentro da palavra. Nesse exemplo, a posição do byte dentro da palavra é a posição 0, que corresponde ao byte mais significativo.

WORD_ S0(BYTE_ b)_

```
{int i;
WORD ws;
ws = 0;
for (i = 0; I < keylen; ++i) {              /* keylen é o comprimento da chave */
b = Sbox[B(K[i], 0) ^ b];
ws ^= ROTL(Qbox[b], i + 0); /* "+0" para o byte mais significativo */
}
ws = (ws & 0x00FFFFFFUL) | (b << 24);
return ws;
}
```

4.3.3 A pseudo-transformação de Hadamard (PHT)

Esta é uma forma muito simples de alterar os valores de dois bytes. Sua fórmula é: (a,b) = (2a + b, a + b). É importante lembrar que cada byte é considerado um elemento do corpo $GF(2^8)$ e portanto a adição entre eles é a definida neste corpo ou seja a adição denominada módulo 2^8. No stream cipher de Turing, a operação é estendida para as palavras, que são consideradas elementos do corpo $GF((2^8)^4)$. Portanto, a operação é a adição módulo 2^{32} (já definida anteriormente), uma vez que quando o código chega nessa etapa, o byte já passou pela S-box e pela Q-box, ou seja, já houve a transformação 8 → 32 bits. Além disso, o PHT mistura um número arbitrário de palavras. No caso desse stream cipher, essa pseudo-transformação toma as palavras resultantes A', B', C', D' e E' que são misturadas segundo a matriz representada a seguir:

$$\begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix}$$

É importante ressaltar que todos os valores da diagonal principal possuem o valor 2, com exceção daquele último, que assume o valor 1.

4.3.4 O estágio final de adições

As primeiras quatro operações do filtro não-linear não são inversíveis. A explicação para isso é simples: esse filtro não-linear nada mais é do que uma função. Para que a função seja inversível, é necessário que a mesma seja bijetora. Porém, ao fazer a transformação no S-box 8 → 32, temos que no domínio existem 2^8 e no contra-domínio 2^{32} possíveis combinações. Ou seja, no contra-domínio existe um número muito maior de elementos que no domínio. Portanto, a imagem nunca poderá ser igual ao contra-domínio, não sendo, por isso, sobrejetora, o que justifica o fato das quatro primeiras operações não serem inversíveis. Dessas afirmações, surge uma implicação: dos 160 bits resultantes, nem todas as combinações resultantes serão possíveis, e os resultados produzidos não são igualmente prováveis, caracterizando uma fraqueza, pois através de uma análise estatística a criptoanálise pode obter pistas sobre a informação codificada. A adição de mais palavras retiradas do LFSR ao resultado da última aplicação da pseudo-transformação de Hadamard apresenta três efeitos principais:

- Isso faz com que todos os valores de $GF((2^8)^4)$ sejam possíveis e igualmente prováveis (será verdadeira essa afirmação desde que as novas palavras sejam completamente independentes das palavras de entrada originalmente).
- Essas operações asseguram a mistura do último estágio do PHT, uma vez que quem estiver atacando precisa remover os efeitos destas palavras antes de estar apto a reverter as etapas na mistura dessas palavras.
- Adicionar quatro novas palavras faz com que aproximadamente metade das palavras existentes num determinado estado do LFSR esteja envolvida na função de filtro não-linear.

4.3.5 Saída

As cinco palavras produzidas por esse processo descrito acima são usadas na seqüência de saída da chave, obedecendo à seguinte ordem: A, B, C, D e E. Há apenas uma ressalva: o byte mais significante de cada palavra deve vir primeiro. Vale a pena ressaltar que problemas surgidos no armazenamento de bytes para o arquivo codificado que não é múltiplo de 20 bytes não estão dentro do escopo de estudo deste trabalho.

4.4 Chave utilizada no stream cipher de Turing

No stream cipher de Turing, tanto a chave quanto o vetor de inicialização são vistos como uma cadeia de bytes, isto é, são convertidos em palavras de 32 bits – 4 bytes, sendo que os mais significativos devem vir primeiro nas representações (característica conhecida como big-endian). Logo, conclui-se que tanto a chave quanto o vetor de inicialização devem apresentar cadeias de comprimento que são múltiplos de 4 bytes.

O tamanho mínimo para o comprimento de uma chave é de 32 bits. Porém, esse tamanho é pouco usado, uma vez que o código cifrado depende da chave e, se essa for considerada frágil pela criptoanálise, poder-se-á chegar com facilidade à mensagem original. Já chaves com cadeias de comprimento muito longo tornam o processo de codificação/decodificação muito lento, sendo, por isso, inviáveis. O ideal é que a chave possua comprimento não muito longo, nem muito curto, e que apresente grau de aleatoriedade satisfatório. Na tentativa de otimizar esse balanceamento, há no stream cipher de Turing um bom gerador de números pseudo-aleatórios, fato que permite diminuir o tamanho da chave, mantendo-lhe o grau de aleatoriedade. Para que um algoritmo de codificação/decodificação seja considerado seguro, a chave deve possuir comprimento de no mínimo 256 bits. Graças a esse gerador de números pseudo-aleatórios, considerado bom estatisticamente, a chave no stream cipher de Turing apresenta tamanho de 256 bits.

Usualmente, há uma etapa em que acontece a leitura da chave. Porém, nas aplicações em sistemas embarcados que possuam um dispositivo que permita apenas a leitura da chave (e não sua modificação), é aceitável que a etapa de leitura dessa chave seja descartada, após a primeira leitura. Assim, a chave deve permanecer na memória armazenada. Esse mecanismo permite que o processo de codificação/decodificação seja mais rápido, pois, uma vez lida a chave, não será necessário lê-la novamente. Vale a pena ressaltar que o procedimento de leitura da chave só poderá ser abandonado caso a chave apresente ótimas qualidades criptográficas, ou seja, não apresente fraquezas frente à criptoanálise.

O vetor de inicialização deve apresentar tamanho mínimo de 0 byte. Porém, uma etapa de leitura do vetor de inicialização é necessária, porque o LFSR só é inicializado quando a leitura do vetor de inicialização for concluída. O máximo comprimento do vetor de inicialização depende do tamanho da chave, pois a soma do comprimento desses dois termos não deve exceder 384 bits, ou seja, 12 palavras. Outra informação importante é a de que o tamanho do vetor de inicialização não precisa necessariamente ser constante.

A estrutura do stream cipher de Turing garante que vetor de inicialização e chave diferentes gerem cadeias distintas. Porém, essas combinações não geram mais de 2^{160} (160-bit) blocos de saída, dado que cinco palavras (cinco conjuntos de 32 bits) são selecionadas

para dar entrada ao processo não-linear.

4.4.1 A leitura da chave

O processo de leitura da chave no stream cipher de Turing mistura os bytes da palavra por meio das transformações fixas já definidas como S-box e Q-box. Essa combinação tem como objetivo garantir que todos os bytes da chave afetam todos os quatro bytes resultantes do processo que passa pela S-box e pela Q-box. Após o processo de transformação de 8 em 32 bits e do posterior processo de redução por meio da operação XOR, as palavras são misturadas usando a pseudo-transformação de Hadamard. Nos processos S-box e Q-box, há a garantia de que diferentes bytes de entrada geram diferentes bytes de saída. Porém, deve-se mostrar que a garantia da diferenciação seja válida não só durante os processos não lineares, mas também durante toda a etapa de codificação/decodificação. Para garantir essa diferenciação, basta mostrar que tal propriedade é válida para a pseudo-transformação de Hadamard. De fato, essa pseudo-transformação foi definida por uma transformação linear, a qual apresentava uma matriz de transformação com todos os elementos sendo 1, exceto os quatro primeiros valores da diagonal principal que apresentavam o valor 2. Essa matriz é claramente inversível. Dessa forma, a transformação também o é, garantindo que dois bytes distintos da chave geram resultados diferentes.

As palavras resultantes são armazenadas para uso futuro. Vale a pena ressaltar que essas palavras resultantes ocupam exatamente o mesmo espaço que a chave original ocupa, isto é, essas bases de dados possuem o mesmo comprimento.

É importante explicar os detalhes, porém deve-se também fornecer uma visão sistêmica do processo. A principal pergunta que surge é por quais motivos são feitos esses processos com a chave. A única razão para que ocorram essas transformações é a tentativa de impedir ataques potenciais à chave criptográfica. Sem algum tipo de mistura na chave inicial, chaves parecidas produziriam resultados parecidos nas saídas das S-box e Q-box. A partir dessa combinação, a criptoanálise precisaria conhecer a chave final para ser capaz de prognosticar e/ou minimizar os efeitos de uma chave conhecida. Esse é um dos motivos pelos quais é permitida a possibilidade de carregar a chave diretamente do hardware.

Se uma implementação rápida do stream cipher de Turing for desejada, há a solução de não calcular os valores tanto da S-box quanto da Q-box, bastando, nesse caso, carregar esses conjuntos de valores em tabelas pré-calculadas, tabelas essas como aquelas

presentes nos apêndices B e C, respectivamente. Para cada palavra, as operações combinadas constituem-se de 4 índices de bytes e de 4 operações XOR entre palavras. Vale a pena ressaltar que uma otimização parecida é utilizada em implementações rápidas no algoritmo de Rijndael.

4.4.2 A leitura do vetor de inicialização

O carregamento do vetor de inicialização faz com que seja iniciado o LFSR, dispositivo que apresenta, inicialmente, valores derivados de uma combinação não linear entre a chave e o vetor de inicialização. O LFSR é inicializado com palavras da seguinte maneira:

- As palavras do vetor de inicialização são copiadas para um determinado local e processadas usando a mesma transformação inversível S-box já mencionada.
- As palavras da chave são anexadas sem necessidade de processos adicionais.
- Seja L o comprimento da chave (a unidade de comprimento são as palavras) e I o comprimento do vetor de inicialização. Uma única palavra, 0x010203LI, é anexada depois do vetor de inicialização e da chave. Essa inclusão garante que diferentes comprimentos de chave e de vetor de inicialização não conseguem criar o mesmo estado inicial do LFSR.
- A soma dos comprimentos da chave e do vetor de inicialização não deve exceder 12 palavras. Entre essas duas partes, é adicionada uma palavra, que foi definida no marcador que antecede a esse, totalizando, então, no máximo 13 palavras. Com esse tamanho, o LFSR não é inicializado. Para tal, devem ser adicionadas outras palavras. Essas palavras são preenchidas pela adição da palavra imediatamente anterior à palavra que ocupa a posição (L+I). Porém, antes de colocar tal palavra na posição (L+I), essa palavra passa pela S-box e pela Q-box. Só assim acontece o preenchimento de todo o registrador. Isto é, a k-ésima palavra K_k ($L+I+1 \leq k < 17$) é ajustada para $S[K_{k-1}+K_{k-L-I-1}]$.
- Finalmente, quando o registrador do LFSR tiver sido preenchido com as palavras, esse conteúdo é misturado com uma pseudo-transformação de Hadamard de dimensão 17, ou seja, a matriz de transformação é uma

matriz quadrada de 17 linhas, sendo que todos os valores dela são 1, exceto os 16 primeiros valores da diagonal principal, que valem 2. A geração do fluxo da chave pode agora ser iniciada.

4.5 Performance

Estando disponível um acesso suficiente à memória randômica, as operações das quatro S-boxes podem ser pré-calculadas no momento de ajuste (setup) da chave, resultando quatro tabelas, uma para cada byte da palavra de entrada.

Muitos microprocessadores atuais permitem a execução simultânea de tarefas, desde que essas tarefas sejam suficientemente independentes. Observam-se que as operações mencionadas acima são altamente paralelas, permitindo performances muito boas nesses processadores.

Nos casos em que a chave é fornecida no hardware, é possível, no processo como um todo das atividades com a chave, incluindo o cálculo dessas tabelas, que esse processo esteja pronto dentro de um tempo previamente desejável. Deste modo, ao invés de 4 kbytes de RAM e 1280 bytes de ROM, 4 kbytes de ROM são suficientes, proporcionando uma implementação muito rápida. É importante acrescentar que uma quantidade adicional de 1024 bytes de ROM é requerida para a construção da tabela de multiplicação.

O stream cipher de Turing proporciona flexibilidade na eficiência de implementação. A fabricante desse stream cipher, a empresa Qualcomm, fornece quatro tipos de implementação, que são:

- TuringRef.c, uma referência de implementação não otimizada, que usa pequena quantidade de memória RAM (Random Access Memory). Embora a utilização de memória RAM seja pequena, não há nenhuma tabela pré-calculada, ou seja, todas são calculadas junto com a execução do programa.
- TuringTAB.c utiliza tabelas pré-calculadas necessárias na S-box e na Q-box a partir do momento em que a chave foi escolhida. Nesse modo, são utilizados 4 kbytes de memória RAM além do que é utilizado no LFSR.
- TuringLazy.c é uma implementação rápida da chave, em que as tabelas da S-box e da Q-box são carregadas apenas quando são necessárias. Dessa maneira, a configuração da chave e do vetor de inicialização é

relativamente rápida. Além disso, em testes realizados pelo fabricante, foi detectado que para esse tipo de configuração a velocidade de codificação é adequada, se comparada com as velocidades de codificação de outros sistemas criptográfcos.

- TuringFast.c usa tabelas da S-box computadas no instante da configuração da chave. Seu desempenho depende da quantidade de cálculos que são feitos em série. Assim, quanto maior a quantidade de cálculos em série, melhor será o desempenho.

Na Tabela 1, apresenta-se o resultado dos desempenhos desses quatro tipos de implementação citados. Essa experimentação foi realizada pela Qualcomm e encontra-se disponível nas referências [13] e [14]. Os tempos foram medidos quando a implementação ocorreu em um laptop da IBM, com um processador Intel Pentium III, 900 MHz. O software utilizado foi o Microsoft Visual C++ V6.0. Há ainda na Tabela 1 uma comparação de tempo entre essas quatro configurações, a implementação do AES e uma implementação em um codificador compatível com o RC4. Os dados estão registrados a seguir:

Tabela 1 – Dados relativos ao desempenho das quatro implementações

Cipher	cycles/B	Key	IV setup	tables	RAM	MByte/s
TuringRef	134.92	477.00	4272.31	2304	68	6.67
TuringLazy	20.73	1802.70	991.80	2304	4164	43.42
TuringTab	17.26	72457.93	900.90	2304	4164	52.15
TuringFast	5.45	72417.12	882.90	2304	4164	165.15
arrsyfor	37.49	0.00	10347.42	0	258	24.00
AES enc.	26.85	239.00	0.00	20480	176	33.53
MHz	900.00					

Observação: Em todas as implementações, foram utilizadas chaves de comprimento de 128 bits. No RC4, a operação segura consiste na configuração do vetor de inicialização, não incluindo o tempo dos bytes gerados que são descartados nem o tempo da função hash entre a chave o vetor de inicialização, procedimentos esses que poderiam ser necessários para aumentar a segurança do algoritmo.

4.6 Segurança

A segurança é tema importantíssimo dentro da criptografia. Por isso, há diversos

estudos, tanto nacionais, como o realizado por Puttini [15], quanto estudos realizados por estrangeiros, como o realizado por Stallings [7]. Nessa parte to trabalho, serão discutidos aspectos de segurança frente a potenciais formas de ataques e que são conhecidas. O stream cipher de Turing apresenta duas forças criptográficas, vindas de mecanismos independentes, de modo a permitir que cada um proteja o outro contra ataques padrões. Os dois mecanismos usados são a estrutura de geração de filtros não-lineares, em combinação com o processo que depende da chave e que é altamente não linear e não-inversível, definido como transformação S-box (nesse contexto, também se insere a transformação Q-box). A segurança desse stream cipher é confiada a esses dois componentes. Há informações fornecidas pelo fabricante do stream cipher de Turing a respeito de potenciais ataques que esse sistema criptográfico pode sofrer. O estudo dessas potencialidades é algo bastante complexo e, por isso, não será alvo desse trabalho de graduação.

4.6.1 Conhecimento da mensagem original

O stream cipher de Turing é considerado síncrono, isto é, a seqüência gerada por meio da chave (keystream) é independente da mensagem original. O mau uso de qualquer stream cipher, tal como a reutilização de seqüências geradas por meio da chave, pode comprometer a mensagem original, porém não revela qualquer informação sobre o mecanismo que faz a codificação das mensagens. Há, no stream cipher de Turing, aumento na segurança devido ao fato de existir um mecanismo integrado que protege os vetores de inicialização, fazendo com que seja mais fácil de usá-lo corretamente.

Para atacar o mecanismo que faz a codificação das mensagens, é necessário muito conhecimento sobre a mensagem original: ou se conhece completamente essa mensagem não-codificada ou pelo menos algumas repetições significativas nas mensagens originais. Na discussão seguinte, assume-se ou que a mensagem original seja conhecida, ou que se tenha um acesso direto à saída do gerador de seqüências.

4.6.2 Relações com a chave e escolha do vetor de inicialização

No stream cipher a chave passa por um gerador de seqüências pseudo-aleatórias. O fluxo de saída desse gerador entra em contato com a mensagem original/cifrada e assim acontece a codificação/decodificação, respectivamente, da mensagem. Em um determinado

instante, uma pessoa mal intencionada pode “roubar” a seqüência que sai desse gerador pseudo-aleatório. Porém, esse ataque não oferece riscos à segurança do programa, pois para se iniciar o LSFR, acontece o carregamento da chave. Conforme visto, o mecanismo não-linear das S-box e Q-box garante que qualquer alteração na seqüência inicial do LSFR, mesmo que seja mínima, altera de forma significativa o fluxo da seqüência que é utilizada na codificação/decodificação.

Já os vetores de inicialização, esses são normalmente conhecidos e podem, inclusive, ser escolhidos por uma pessoa que deseja testar esse sistema criptográfico. O vetor de inicialização é utilizado para iniciar o funcionamento do LFSR. Portanto, para se preencher o estado inicial do LFSR, são necessários cuidados especiais para que essa seqüência dependa fortemente da chave e que essa dependência seja altamente não-linear, para que qualquer modificação nesse vetor de inicialização gere grandes alterações na seqüência que codifica/decodifica a mensagem.

4.6.3 Ataque baseado em estatísticas

A análise estatística normalmente obtém sucesso quando o gerador de seqüências vindas da chave é tendencioso ou então quando esse gerador for considerado fraco frente à criptoanálise. O LFSR usado nesse stream cipher apresenta características estatísticas muito boas, características essas que se refletem nos dados de saída do LFSR. Para encobrir esse processo linear, há os altamente não-lineares, que foram estudados nesse trabalho, como a S-box e a Q-box.

Os fabricantes desse stream cipher testaram exaustivamente o mecanismo de codificação frente às análises estatísticas e não detectaram possíveis fraquezas em sua estrutura. Para esse teste, foi usado o pacote Crypt-X, que pertence à IBM, pacote esse que contém grande quantidade de testes estatísticos usados na mensuração das diferentes propriedades estatísticas que uma seqüência verdadeiramente aleatória deve ter. A idéia foi criar um gerador pseudo-aleatório que seja utilizado na produção das chaves aleatórias. Um gerador pseudo-aleatório é um algoritmo determinístico que recebe como entrada uma pequena seqüência aleatória (semente) e gera uma longa seqüência de *bits*, que se comporta como uma seqüência verdadeiramente aleatória.

4.6.4 Ataques usando correlações

As correlações constituem uma etapa na criptoanálise, principalmente quando se estiver diante de processos não-lineares. Coppersmith e outros [9] desenvolveram um modelo geral para distintos ataques criptográficos nos geradores de filtros não-lineares. Esse modelo assume que muitas correlações significantes podem ser identificadas na função do filtro não-linear e que essa correlação permanece mesmo depois que as saídas tiverem sido combinadas como forma de eliminar a parte linear do processo.

O filtro não-linear do stream cipher de Turing foi projetado para usar quantia significante dos dados de entrada e realizar uma profunda transformação nesses. Por definição, continuam existindo muitas correlações. Porém, há grandes chances de que essas correlações sejam suficientemente fracas, garantindo que esses ataques sejam impedidos. É importante dizer que, mesmo sendo altas essas chances, não há confiança plena na segurança até então. Para torná-lo confiável e seguro, ocorre que esse processo não linear é dependente da chave, pois para se iniciar o LFSR é preciso tanto da chave quanto do vetor de inicialização. De fato, valores desconhecidos sendo gerados no LFSR protegem da criptoanálise a função do filtro não-linear.

5 Implementações

Há quatro possíveis implementações para o stream cipher de Turing, conhecidas como Turing Reference, Turing Table-lookup, Turing Fast e Turing Table-lookup/Lazy. A seguir, estão representadas essas quatro implementações, disponíveis na página eletrônica da fabricante [19], devidamente comentadas.

5.1 Turing Reference

```
#include <stdlib.h>            /* Definição da biblioteca padrão */
#include "Turing.h"            /* Carregar o arquivo do polinômio binomial equivalente */
#include "TuringSbox.h"        /* Carregar o arquivo da função S-box */
#include "QUTsbox.h"           /* Carregar o arquivo da função Q-box */
#include "TuringMultab.h"      /* Carregar o arquivo com a tabela de multiplicação */

int     keylen;                /* Comprimento da chave – valor inteiro */
WORD  K[MAXKEY/4];             /* Vetor de palavras, para uso de estoque */
WORD  R[LFSRLEN];              /* Vetor de comprimento do LFSR */

Void                           /* Essa função é o mecanismo do LFSR */
step(void)
{
   register int   i;
   register WORD       w;
                               /* Cálculo da palavra que atualizará o LFSR */
   w = R[15] ^ R[4] ^ (R[0] << 8) ^ Multab[(R[0] >> 24) & 0xFF];
   for (i = 1; i < LFSRLEN; ++i)
         R[i-1] = R[i];        /* Deslocamento das palavras */
   R[LFSRLEN-1] = w;           /* Atualização do LFSR */
}

static WORD
fixedS(WORD w)
{
   WORD    b;                  /* Designação de uma palavra que será usada apenas nessa função
*/
```

/* A partir da palavra de entrada “w”, essa passa pela S-box, resultando na palavra “b”, e depois pela Q-box, retornando a palavra “w”. Essa operação acontece byte a byte. Dependendo da posição do byte dentro da palavra (0, 1, 2 ou 3) acontece um deslocamento à esquerda em 24, 16, 8 e 0 bytes, respectivamente. Essa operação dificulta os ataques da criptoanálise, pois chaves/vetores de inicialização distintos geram resultados ainda mais distintos. */

```
   b = Sbox[B(w, 0)]; w = ((w ^      Qbox[b])     & 0x00FFFFFF) | (b << 24);
   b = Sbox[B(w, 1)]; w = ((w ^ ROTL(Qbox[b],8))  & 0xFF00FFFF) | (b << 16);
   b = Sbox[B(w, 2)]; w = ((w ^ ROTL(Qbox[b],16)) & 0xFFFF00FF) | (b << 8);
   b = Sbox[B(w, 3)]; w = ((w ^ ROTL(Qbox[b],24)) & 0xFFFFFF00) | b;
   return w;
}

static WORD
S(WORD w, int r)
{
   register int   i;
   BYTE                b[4];
   WORD                ws[4];

   w = ROTL(w, r);              /* A palavra de entrada “w” é rotacionada à esquerda em “r” bytes */

   WORD2BYTE(w, b);
   ws[0] = ws[1] = ws[2] = ws[3] = 0;    /* É importante anular esse vetor de palavras para que o lixo de memória que lá estava depositado não interfira no resultado */
```

/* A palavra de entrada é dividida em bytes (que ocupam as posições 0,1,2 e 3). Os bytes dessa palavra de entrada combinam-se com bytes das palavras da chave que ocupam a mesma posição interna. Logo após, a palavra é rotacionada à esquerda em 0, 8, 16 e 24 bits, para as posições 0,1,2 e 3, respectivamente.*/

```
   for (i = 0; i < keylen; ++i) {
         b[0] = Sbox[B(K[i], 0) ^ b[0]]; ws[0] ^= ROTL(Qbox[b[0]], i + 0);
         b[1] = Sbox[B(K[i], 1) ^ b[1]]; ws[1] ^= ROTL(Qbox[b[1]], i + 8);
         b[2] = Sbox[B(K[i], 2) ^ b[2]]; ws[2] ^= ROTL(Qbox[b[2]], i + 16);
         b[3] = Sbox[B(K[i], 3) ^ b[3]]; ws[3] ^= ROTL(Qbox[b[3]], i + 24);
   }
   w  = (ws[0] & 0x00FFFFFFUL) | (b[0] << 24);/* A palavra que ocupa a posição 0 é deslocada de 24 bits. */
```

```
   w ^= (ws[1] & 0xFF00FFFFUL) | (b[1] << 16);          /* A palavra que ocupa a posição 1 é deslocada de 16 bits. Logo após acontece a combinação daquela que ocupava a posição 0 com a da posição 1. */
   w ^= (ws[2] & 0xFFFF00FFUL) | (b[2] << 8); /* A palavra que ocupa a posição 2 é deslocada de 8 bits. Logo após acontece a combinação daquela que ocupava a posição 0 com a da posição 1 e da posição 2. */
   w ^= (ws[3] & 0xFFFFFF00UL) | b[3];          /* Há a soma do resultado anterior com a palavra que ocupava a posição 3. */

   return w;                          /* Como resultado dessa função, a palavra "w" retorna */
}

                                              /* Pseudo-transformação de Hadamard */

#define PHT(A,B,C,D,E) { \                    /* Essa função tem como objetivo misturar as 5 palavras que são retiradas do LFSR */
        (E) += (A) + (B) + (C) + (D); \
        (A) += (E); \
        (B) += (E); \
        (C) += (E); \
        (D) += (E); \
}

/* Generalizando a pseudo-transformação de Hadamard para n palavras */
void
mixwords(WORD w[], int n)
{
   register WORD        sum;
   register int   i;

   for (sum = i = 0; i < n-1; ++i)
        sum += w[i];
   w[n-1] += sum;
   sum = w[n-1];
   for (i = 0; i < n-1; ++i)
        w[i] += sum;
}

/* O procedimento abaixo é para aumentar a segurança da cifra. Para isso, há nova combinação das palavras da chave */
```

```
void
TuringKey(const BYTE key[], const int keylength)
{
   register int   i;
```

/* Se o comprimento da chave for menor que 32 bytes ou maior que o tamanho máximo da chave, o procedimento será abortado */

```
   if ((keylength & 0x03) != 0 || keylength > MAXKEY)
         abort();
   keylen = 0;
```

/* Caso contrário, acontecerá a mistura. A variável “i” é atualizada em 4 unidades, dado que as palavras (conjunto de 4 bytes) é que são realmente combinadas */

```
   for (i = 0; i < keylength; i += 4)
         K[keylen++] = fixedS(BYTE2WORD(&key[i]));
   mixwords(K, keylen);
}
```

/* Leitura do vetor de inicialização */

/* Para preencher o estado inicial do LFSR, será preciso o vetor de inicialização, a chave, o comprimento da vetor de inicialização e da chave, e mais outros componentes. O vetor de inicialização passa pelas transformações S-box e Q-box. A chave já foi pré-combinada na função acima. O restante é preenchido por uma combinação definida e que passa pela S-box e pela Q-box. */

```
void
TuringIV(const BYTE iv[], const int ivlength)
{
   register int   i, j;

/* Verificando se o tamanho do vetor de inicialização é válido */
   if ((ivlength & 0x03) != 0 || (ivlength + 4*keylen) > MAXKIV)
         abort();
   /* Copiando o vetor de inicialização e passando esse vetor pelas funções S-box e Q-box */
   for (i = j = 0; j < ivlength; j +=4)
         R[i++] = fixedS(BYTE2WORD(&iv[j]));
```

```
   /* Copiando as informações contidas no vetor da chave */
   for (j = 0; j < keylen; ++j)
         R[i++] = K[j];
   /* Acréscimo de uma palavra contendo os comprimentos do vetor de inicialização e da chave. Esse
mecanismo garante que diferentes comprimentos de chave/vetor de inicialização gerem estados
iniciais distintos */
   R[i++] = (keylen << 4) | (ivlength >> 2) | 0x01020300UL;
   /* Para completar o preenchimento, acontece a adição da palavra imediatamente anterior à
palavra que ocupa a posição (L+I), sendo que essa adição deve passar pelas funções S-box e Q-
box */
   for (j = 0; i < LFSRLEN; ++i, ++j)
         R[i] = S(R[j] + R[i-1], 0);
   /* Finalmente, acontece uma mistura de todas as palavras */
   mixwords(R, LFSRLEN);
}

/* Gerando uma saída contendo 5 palavras */
int
TuringGen(BYTE *buf)
{
   register WORD        A, B, C, D, E;
/* As palavras A,B,C,D e E são retiradas das posições 16, 13, 6, 1 e 0 do LFSR */
   step();
   A = R[16];    B = R[13];       C = R[6];        D = R[1];        E = R[0];
/* Acontece uma mistura dessas palavras */
   PHT(A,        B,               C,               D,               E);
/* O processo não linear que compreende as S-box e Q-box é iniciado por esta linha de comando */
   A = S(A,0);   B = S(B,8);      C = S(C,16);     D = S(D,24);     E = S(E,0);
/* Acontece uma nova mistura */
   PHT(A,        B,               C,               D,               E);
/* O LFSR é atualizado três vezes */
   step(); step(); step();
/* É feita uma nova combinação só que agora com as palavras que ocupam as posições 14, 12, 8,1 e
0 do LFSR */
   A += R[14];   B += R[12];      C += R[8];       D += R[1];       E += R[0];
         WORD2BYTE(A, buf);
               WORD2BYTE(B, buf+4);
                     WORD2BYTE(C, buf+8);
                           WORD2BYTE(D, buf+12);
                                 WORD2BYTE(E, buf+16);
```

```
    step();
    return 20;
}
```

5.2 Turing Table-lookup

```
#include <stdlib.h>                 /* Inclusão da biblioteca padrão */
#include "Turing.h"                 /* Carregar o arquivo do polinômio binomial equivalente */
#include "TuringSbox.h"             /* Carregar o arquivo da função S-box */
#include "QUTsbox.h"                /* Carregar o arquivo da função Q-box */
#include "TuringMultab.h"           /* Carregar o arquivo com a tabela de multiplicação */

int     keylen;                     /* Comprimento da chave – valor inteiro */
WORD  K[MAXKEY/4];                  /* Vetor de palavras, para uso de estoque */
WORD  R[LFSRLEN];                   /* Vetor de comprimento do LFSR */

WORD  S0[256], S1[256], S2[256], S3[256];     /* Quatro tabelas com 256 palavras cada, referentes
a quatro S-boxes */

Void                                /* Essa função é o mecanismo do LFSR */
step(void)
{
    register int   i;
    register WORD       w;
                                    /* Cálculo da palavra que atualizará o LFSR */
    w = R[15] ^ R[4] ^ (R[0] << 8) ^ Multab[(R[0] >> 24) & 0xFF];
    for (i = 1; i < LFSRLEN; ++i)
        R[i-1] = R[i];              /* Deslocamento das palavras */
    R[LFSRLEN-1] = w;               /* Atualização do LFSR */
}

static WORD
fixedS(WORD w)
{
    WORD    b;                      /* Designação de uma palavra que será usada apenas nessa função
*/
```

/* A partir da palavra de entrada “w”, essa passa pela S-box, resultando na palavra “b”, e depois pela Q-box, retornando a palavra “w”. Essa operação acontece byte a byte. Dependendo da posição do byte dentro da palavra (0, 1, 2 ou 3) acontece um deslocamento à esquerda em 24, 16, 8 e 0 bytes, respectivamente. Essa operação dificulta os ataques da criptoanálise, pois chaves/vetores de inicialização distintos geram resultados ainda mais distintos. */

```
   b = Sbox[B(w, 0)]; w = ((w ^      Qbox[b])     & 0x00FFFFFF) | (b << 24);
   b = Sbox[B(w, 1)]; w = ((w ^ ROTL(Qbox[b],8))  & 0xFF00FFFF) | (b << 16);
   b = Sbox[B(w, 2)]; w = ((w ^ ROTL(Qbox[b],16)) & 0xFFFF00FF) | (b << 8);
   b = Sbox[B(w, 3)]; w = ((w ^ ROTL(Qbox[b],24)) & 0xFFFFFF00) | b;
   return w;
}
```

/* Mais um mecanismo de mistura das palavras. Inicialmente a palavra é rotacionada à esquerda e depois essa palavra rotacionada passa em quatro S-box distintas. Logo após, acontece uma combinação dessas novas palavras, gerando como resultado uma outra palavra. Esse estrutura precede a entrada da palavra na Q-box. */

```
static WORD
S(WORD w, int r)
{
   w = ROTL(w, r);
   return S0[B(w, 0)] ^ S1[B(w, 1)] ^ S2[B(w, 2)] ^ S3[B(w, 3)];
}
```

/* Pseudo-transformação de Hadamard */

```
#define PHT(A,B,C,D,E) { \
```

/* Essa função tem como objetivo misturar as 5 palavras que são retiradas do LFSR */

```
        (E) += (A) + (B) + (C) + (D); \
        (A) += (E); \
        (B) += (E); \
        (C) += (E); \
        (D) += (E); \
}
```

/* Generalizando a pseudo-transformação de Hadamard para n palavras */

```
void
mixwords(WORD w[], int n)
{
   register WORD        sum;
   register int   i;

   for (sum = i = 0; i < n-1; ++i)
         sum += w[i];
   w[n-1] += sum;
   sum = w[n-1];
   for (i = 0; i < n-1; ++i)
         w[i] += sum;
}
```

/* O procedimento abaixo é para aumentar a segurança da cifra. Para isso, há nova combinação das palavras da chave */

```
void
TuringKey(const BYTE key[], const int keylength)
{
   register int   i, j, k;
   register WORD        w;
```

/* Se o comprimento da chave for menor que 32 bytes ou maior que o tamanho máximo da chave, o procedimento será abortado */

```
   if ((keylength & 0x03) != 0 || keylength > MAXKEY)
         abort();
   keylen = 0;
```

/* Caso contrário, acontecerá a mistura. A variável “i” é atualizada em 4 unidades, dado que as palavras (conjunto de 4 bytes) é que são realmente combinadas */

```
   for (i = 0; i < keylength; i += 4)
         K[keylen++] = fixedS(BYTE2WORD(&key[i]));
   mixwords(K, keylen);
```

/* Construção de novas tabelas para as 4 funções S-box dessa implementação. */

/* Procedimento para construção da S-box “S0”. */

```
   for (j = 0; j < 256; ++j) {
         w = 0;
```

```
            k = j;
            for (i = 0; i < keylen; ++i) {
               k = Sbox[B(K[i], 0) ^ k];
               w ^= ROTL(Qbox[k], i + 0);
            }
            S0[j] = (w & 0x00FFFFFFUL) | (k << 24);
      }
```

/* Procedimento para construção da S-box “S1”. */

```
      for (j = 0; j < 256; ++j) {
            w = 0;
            k = j;
            for (i = 0; i < keylen; ++i) {
               k = Sbox[B(K[i], 1) ^ k];
               w ^= ROTL(Qbox[k], i + 8);
            }
            S1[j] = (w & 0xFF00FFFFUL) | (k << 16);
      }
```

/* Procedimento para construção da S-box “S2”. */

```
      for (j = 0; j < 256; ++j) {
            w = 0;
            k = j;
            for (i = 0; i < keylen; ++i) {
               k = Sbox[B(K[i], 2) ^ k];
               w ^= ROTL(Qbox[k], i + 16);
            }
            S2[j] = (w & 0xFFFF00FFUL) | (k << 8);
      }
```

/* Procedimento para construção da S-box “S3”. */

```
      for (j = 0; j < 256; ++j) {
            w = 0;
            k = j;
            for (i = 0; i < keylen; ++i) {
               k = Sbox[B(K[i], 3) ^ k];
               w ^= ROTL(Qbox[k], i + 24);
```

```
        }
        S3[j] = (w & 0xFFFFFF00UL) | k;
    }
}
```

/* Leitura do vetor de inicialização */

/* Para preencher o estado inicial do LFSR, será preciso o vetor de inicialização, a chave, o comprimento da vetor de inicialização e da chave, e mais outros componentes. O vetor de inicialização passa pelas transformações S-box e Q-box. A chave já foi pré-combinada na função acima. O restante é preenchido por uma combinação definida e que passa pela S-box e pela Q-box. */

```
void
TuringIV(const BYTE iv[], const int ivlength)
{
    register int   i, j;

/* Verificando se o tamanho do vetor de inicialização é válido */
    if ((ivlength & 0x03) != 0 || (ivlength + 4*keylen) > MAXKIV)
        abort();
    /* Copiando o vetor de inicialização e passando esse vetor pelas funções S-box e Q-box */
    for (i = j = 0; j < ivlength; j +=4)
        R[i++] = fixedS(BYTE2WORD(&iv[j]));
    /* Copiando as informações contidas no vetor da chave */
    for (j = 0 /* i continues */; j < keylen; ++j)
        R[i++] = K[j];
    /* Acréscimo de uma palavra contendo os comprimentos do vetor de inicialização e da chave. Esse mecanismo garante que diferentes comprimentos de chave/vetor de inicialização gerem estados iniciais distintos */
    R[i++] = (keylen << 4) | (ivlength >> 2) | 0x01020300UL;
    /* Para completar o preenchimento, acontece a adição da palavra imediatamente anterior à palavra que ocupa a posição (L+I), sendo que essa adição deve passar pelas funções S-box e Q-box */
    for (j = 0 /* i continues */; i < LFSRLEN; ++i, ++j)
        R[i] = S(R[j] + R[i-1], 0);
    /* Finalmente, acontece uma mistura de todas as palavras */
    mixwords(R, LFSRLEN);
}
```

/* Gerando uma saída contendo 5 palavras */

```
int
TuringGen(BYTE *buf)
{
   register WORD        A, B, C, D, E;

   step();
/* As palavras A,B,C,D e E são retiradas das posições 16, 13, 6, 1 e 0 do LFSR */
   A = R[16];     B = R[13];        C = R[6];         D = R[1];         E = R[0];
/* Acontece uma mistura dessas palavras */
   PHT(A,         B,                C,                D,                E);
/* O processo não linear que compreende as S-box e Q-box é iniciado por esta linha de comando */
   A = S(A,0);    B = S(B,8);       C = S(C,16);      D = S(D,24);      E = S(E,0);
/* Acontece uma nova mistura */
   PHT(A,         B,                C,                D,                E);
/* O LFSR é atualizado três vezes */
   step(); step(); step();
/* É feita uma nova combinação só que agora com as palavras que ocupam as posições 14, 12, 8,1 e
0 do LFSR */
   A += R[14];    B += R[12];       C += R[8];        D += R[1];        E += R[0];
   WORD2BYTE(A, buf);
                  WORD2BYTE(B, buf+4);
                                    WORD2BYTE(C, buf+8);
                                                      WORD2BYTE(D, buf+12);
                                                                        WORD2BYTE(E, buf+16);
   step();
   return 20;
}
```

5.3 Turing Fast

```
#include <stdlib.h>          /* Inclusão da biblioteca padrão */
#include "Turing.h"          /* Carregar o arquivo do polinômio binomial equivalente */
#include "TuringSbox.h"      /* Carregar o arquivo da função S-box */
#include "QUTsbox.h"         /* Carregar o arquivo da função Q-box */
#include "TuringMultab.h"    /* Carregar o arquivo com a tabela de multiplicação */

int       keylen;            /* Comprimento da chave – valor inteiro */
WORD  K[MAXKEY/4];           /* Vetor de palavras, para uso de estoque */
WORD  R[LFSRLEN];            /* Vetor de comprimento do LFSR */
```

```
WORD  S0[256], S1[256], S2[256], S3[256];      /* Quatro tabelas com 256 palavras cada, referentes a quatro S-boxes */

                                    /* A seguir, há a compensação da posição corrente do registrador */
#define OFF(zero, i) (((zero)+(i)) % LFSRLEN)

#define STEP(z) \                   /* Esse processo nada mais é do que o cálculo em um instante qualquer do elemento que entrará no LFSR */
   R[OFF(z,0)] = R[OFF(z,15)] ^ R[OFF(z,4)] ^ \
         (R[OFF(z,0)] << 8) ^ Multab[(R[OFF(z,0)] >> 24) & 0xFF]


static WORD
fixedS(WORD w)
{
   WORD    b;                       /* Designação de uma palavra que será usada apenas nessa função */


/* A partir da palavra de entrada "w", essa passa pela S-box, resultando na palavra "b", e depois pela Q-box, retornando a palavra "w". Essa operação acontece byte a byte. Dependendo da posição do byte dentro da palavra (0, 1, 2 ou 3) acontece um deslocamento à esquerda em 24, 16, 8 e 0 bytes, respectivamente. Essa operação dificulta os ataques da criptoanálise, pois chaves/vetores de inicialização distintos geram resultados ainda mais distintos. */


   b = Sbox[B(w, 0)]; w = ((w ^      Qbox[b])     & 0x00FFFFFF) | (b << 24);
   b = Sbox[B(w, 1)]; w = ((w ^ ROTL(Qbox[b],8))  & 0xFF00FFFF) | (b << 16);
   b = Sbox[B(w, 2)]; w = ((w ^ ROTL(Qbox[b],16)) & 0xFFFF00FF) | (b << 8);
   b = Sbox[B(w, 3)]; w = ((w ^ ROTL(Qbox[b],24)) & 0xFFFFFF00) | b;
   return w;
}

/*Definição de uma função que faz com que uma palavra seja combinada em quatro S-boxes distintas, resultando uma única palavra */

#define S(w,b) (S0[B((w), ((0+b)&0x3))] \
                  ^ S1[B((w), ((1+b)&0x3))] \
                  ^ S2[B((w), ((2+b)&0x3))] \
                  ^ S3[B((w), ((3+b)&0x3))])
```

```
                                        /* Pseudo-transformação de Hadamard */

#define PHT(A,B,C,D,E) { \              /* Essa função tem como objetivo misturar as 5 palavras que
são retiradas do LFSR */

        (E) += (A) + (B) + (C) + (D); \
        (A) += (E); \
        (B) += (E); \
        (C) += (E); \
        (D) += (E); \
}

                /* Generalizando a pseudo-transformação de Hadamard para n palavras */
void
mixwords(WORD w[], int n)
{
   register WORD        sum;
   register int   i;

   for (sum = i = 0; i < n-1; ++i)
        sum += w[i];
   w[n-1] += sum;
   sum = w[n-1];
   for (i = 0; i < n-1; ++i)
        w[i] += sum;
}

/* O procedimento abaixo é para aumentar a segurança da cifra. Para isso, há nova combinação das
palavras da chave */
void
TuringKey(const BYTE key[], const int keylength)
{
   register int   i, j, k;
   register WORD        w;

/* Se o comprimento da chave for menor que 32 bytes ou maior que o tamanho máximo da chave, o
procedimento será abortado */
   if ((keylength & 0x03) != 0 || keylength > MAXKEY)
```

```
        abort();
    keylen = 0;
```

/* Caso contrário, acontecerá a mistura. A variável “i” é atualizada em 4 unidades, dado que as palavras (conjunto de 4 bytes) é que são realmente combinadas */

```
    for (i = 0; i < keylength; i += 4)
        K[keylen++] = fixedS(BYTE2WORD(&key[i]));
    mixwords(K, keylen);
```

/* Construção de novas tabelas para as 4 funções S-box dessa implementação. */

/* Procedimento para construção da S-box “S0”. */

```
    for (j = 0; j < 256; ++j) {
        w = 0;
        k = j;
        for (i = 0; i < keylen; ++i) {
            k = Sbox[B(K[i], 0) ^ k];
            w ^= ROTL(Qbox[k], i + 0);
        }
        S0[j] = (w & 0x00FFFFFFUL) | (k << 24);
    }
```

/* Procedimento para construção da S-box “S1”. */

```
    for (j = 0; j < 256; ++j) {
        w = 0;
        k = j;
        for (i = 0; i < keylen; ++i) {
            k = Sbox[B(K[i], 1) ^ k];
            w ^= ROTL(Qbox[k], i + 8);
        }
        S1[j] = (w & 0xFF00FFFFUL) | (k << 16);
    }
```

/* Procedimento para construção da S-box “S2”. */

```
    for (j = 0; j < 256; ++j) {
        w = 0;
        k = j;
```

```
        for (i = 0; i < keylen; ++i) {
            k = Sbox[B(K[i], 2) ^ k];
            w ^= ROTL(Qbox[k], i + 16);
        }
        S2[j] = (w & 0xFFFF00FFUL) | (k << 8);
    }
```

/* Procedimento para construção da S-box “S3”. */

```
    for (j = 0; j < 256; ++j) {
        w = 0;
        k = j;
        for (i = 0; i < keylen; ++i) {
            k = Sbox[B(K[i], 3) ^ k];
            w ^= ROTL(Qbox[k], i + 24);
        }
        S3[j] = (w & 0xFFFFFF00UL) | k;
    }
}
```

/* Leitura do vetor de inicialização */

/* Para preencher o estado inicial do LFSR, será preciso o vetor de inicialização, a chave, o comprimento da vetor de inicialização e da chave, e mais outros componentes. O vetor de inicialização passa pelas transformações S-box e Q-box. A chave já foi pré-combinada na função acima. O restante é preenchido por uma combinação definida e que passa pela S-box e pela Q-box. */

```
void
TuringIV(const BYTE iv[], const int ivlength)
{
    register int    i, j;
```

/* Verificando se o tamanho do vetor de inicialização é válido */

```
    if ((ivlength & 0x03) != 0 || (ivlength + 4*keylen) > MAXKIV)
        abort();
    /* Copiando o vetor de inicialização e passando esse vetor pelas funções S-box e Q-box */
    for (i = j = 0; j < ivlength; j +=4)
        R[i++] = fixedS(BYTE2WORD(&iv[j]));
    /* Copiando as informações contidas no vetor da chave */
```

```
   for (j = 0 /* i continues */; j < keylen; ++j)
         R[i++] = K[j];
   /* Acréscimo de uma palavra contendo os comprimentos do vetor de inicialização e da chave. Esse
mecanismo garante que diferentes comprimentos de chave/vetor de inicialização gerem estados
iniciais distintos */
   R[i++] = (keylen << 4) | (ivlength >> 2) | 0x01020300UL;
   /* Para completar o preenchimento, acontece a adição da palavra imediatamente anterior à
palavra que ocupa a posição (L+I), sendo que essa adição deve passar pelas funções S-box e Q-
box */
   for (j = 0 /* i continues */; i < LFSRLEN; ++i, ++j)
         R[i] = S(R[j] + R[i-1], 0);
   /* Finalmente, acontece uma mistura de todas as palavras */
   mixwords(R, LFSRLEN);
}

                              /* Gerando uma saída contendo 5 palavras */
#define ROUND(z,b) \
{ \
   STEP(z); \       /* Há uma atualização sem que haja elementos indo para os filtros não-lineares */

/* A palavra “A” recebe o conteúdo do elemento que ocupa a posição definida pelo resto da soma de
(Z+1+16) pelo comprimento do vetor do LFSR */
         A = R[OFF(z+1,16)]; \

/* A palavra “B” recebe o conteúdo do elemento que ocupa a posição definida pelo resto da soma de
(Z+1+13) pelo comprimento do vetor do LFSR */
                  B = R[OFF(z+1,13)]; \
/* A palavra “C” recebe o conteúdo do elemento que ocupa a posição definida pelo resto da soma de
(Z+1+6) pelo comprimento do vetor do LFSR */
                           C = R[OFF(z+1,6)]; \
/* A palavra “A” recebe o conteúdo do elemento que ocupa a posição definida pelo resto da soma de
(Z+1+1) pelo comprimento do vetor do LFSR */
                                    D = R[OFF(z+1,1)]; \
/* A palavra “A” recebe o conteúdo do elemento que ocupa a posição definida pelo resto da soma de
(Z+1+0) pelo comprimento do vetor do LFSR */
                                             E = R[OFF(z+1,0)]; \
/* Acontece uma mistura dessas palavras */
   PHT(A,      B,      C,      D,      E); \
/* O processo não linear que compreende as S-box e Q-box é iniciado por esta linha de comando */
   A = S(A,0);   B = S(B,1); C = S(C,2); D = S(D,3); E = S(E,0); \
```

```
/* Acontece uma nova mistura dessas palavras */
   PHT(A,      B,      C,          D,      E); \
/* O LFSR é atualizado três vezes */
   STEP(z+1); \
   STEP(z+2); \
   STEP(z+3); \
/* A palavra "A" recebe o conteúdo do elemento que ocupa a posição definida pelo resto da soma de
(Z+4+14) pelo comprimento do vetor do LFSR */
   A += R[OFF(z+4,14)]; \
/* A palavra "B" recebe o conteúdo do elemento que ocupa a posição definida pelo resto da soma de
(Z+4+12) pelo comprimento do vetor do LFSR */
            B += R[OFF(z+4,12)]; \
/* A palavra "C" recebe o conteúdo do elemento que ocupa a posição definida pelo resto da soma de
(Z+4+8) pelo comprimento do vetor do LFSR */
                     C += R[OFF(z+4,8)]; \
/* A palavra "D" recebe o conteúdo do elemento que ocupa a posição definida pelo resto da soma de
(Z+4+1) pelo comprimento do vetor do LFSR */
                              D += R[OFF(z+4,1)]; \
/* A palavra "E" recebe o conteúdo do elemento que ocupa a posição definida pelo resto da soma de
(Z+4+0) pelo comprimento do vetor do LFSR */
                                       E += R[OFF(z+4,0)]; \

   WORD2BYTE(A, b); \
            WORD2BYTE(B, b+4); \
                     WORD2BYTE(C, b+8); \
                              WORD2BYTE(D, b+12); \
                                       WORD2BYTE(E, b+16); \
   STEP(z+4); \
}

/* Aqui são gerados 17 blocos de saída, sendo que cada bloco contém 5 palavras. O que retorna
dessa função é o número de bytes da cadeia gerada */

int
TuringGen(BYTE *buf)
{
   WORD        A, B, C, D, E;

   ROUND(0,buf);
   ROUND(5,buf+20);
```

```
   ROUND(10,buf+40);
   ROUND(15,buf+60);
   ROUND(3,buf+80);
   ROUND(8,buf+100);
   ROUND(13,buf+120);
   ROUND(1,buf+140);
   ROUND(6,buf+160);
   ROUND(11,buf+180);
   ROUND(16,buf+200);
   ROUND(4,buf+220);
   ROUND(9,buf+240);
   ROUND(14,buf+260);
   ROUND(2,buf+280);
   ROUND(7,buf+300);
   ROUND(12,buf+320);
   return 17*20;
}
```

5.4 Turing Table-lookup/Lazy

```
#include <stdlib.h>          /* Inclusão da biblioteca padrão */
#include "Turing.h"          /* Carregar o arquivo do polinômio binomial equivalente */
#include "TuringSbox.h"      /* Carregar o arquivo da função S-box */
#include "QUTsbox.h"         /* Carregar o arquivo da função Q-box */
#include "TuringMultab.h"    /* Carregar o arquivo com a tabela de multiplicação */

int      keylen;             /* Comprimento da chave – valor inteiro */
WORD  K[MAXKEY/4];           /* Vetor de palavras, para uso de estoque */
WORD  R[LFSRLEN];            /* Vetor de comprimento do LFSR */

WORD  S0[256], S1[256], S2[256], S3[256];    /* Quatro tabelas com 256 palavras cada, referentes
a quatro S-boxes */

int      bingo;

Void                                 /* Essa função é o mecanismo do LFSR */
step(void)
{
   register int   i;
   register WORD      w;
```

```
                                    /* Cálculo da palavra que atualizará o LFSR */
   w = R[15] ^ R[4] ^ (R[0] << 8) ^ Multab[(R[0] >> 24) & 0xFF];
   for (i = 1; i < LFSRLEN; ++i)
         R[i-1] = R[i];                    /* Deslocamento das palavras */
   R[LFSRLEN-1] = w;                       /* Atualização do LFSR */
}

static WORD
fixedS(WORD w)
{
   WORD    b;                    /* Designação de uma palavra que será usada apenas nessa função
*/
```

/* A partir da palavra de entrada “w”, essa passa pela S-box, resultando na palavra “b”, e depois pela Q-box, retornando a palavra “w”. Essa operação acontece byte a byte. Dependendo da posição do byte dentro da palavra (0, 1, 2 ou 3) acontece um deslocamento à esquerda em 24, 16, 8 e 0 bytes, respectivamente. Essa operação dificulta os ataques da criptoanálise, pois chaves/vetores de inicialização distintos geram resultados ainda mais distintos. */

```
   b = Sbox[B(w, 0)]; w = ((w ^      Qbox[b])     & 0x00FFFFFF) | (b << 24);
   b = Sbox[B(w, 1)]; w = ((w ^ ROTL(Qbox[b],8))  & 0xFF00FFFF) | (b << 16);
   b = Sbox[B(w, 2)]; w = ((w ^ ROTL(Qbox[b],16)) & 0xFFFF00FF) | (b << 8);
   b = Sbox[B(w, 3)]; w = ((w ^ ROTL(Qbox[b],24)) & 0xFFFFFF00) | b;
   return w;
}
```

/* Essa função calcula a matriz de uma S-box */

```
#define BINGO 0
static WORD
lazyS(BYTE b, int pos)
{
   static WORD *tab[4] = {S0, S1, S2, S3};
   int          i;
   WORD         w;
   BYTE         k;
```

```
    w = 0;
    k = b;
    for (i = 0; i < keylen; ++i) {
        k = Sbox[B(K[i], pos) ^ k];
        w ^= ROTL(Qbox[k], i + 8*pos);
    }
    w = (w & ~(0xFFUL << (24 - 8*pos))) | (k << (24 - 8*pos));
    tab[pos][b] = w;
#if BINGO
    if (w == 0) {
        printf("Bingo! S%d[0x%02x] == 0\n", pos, b);
    }
#endif
    return w;
}

#define LS0(x) (S0[x] != 0 ? S0[x] : lazyS((x), 0))
#define LS1(x) (S1[x] != 0 ? S1[x] : lazyS((x), 1))
#define LS2(x) (S2[x] != 0 ? S2[x] : lazyS((x), 2))
#define LS3(x) (S3[x] != 0 ? S3[x] : lazyS((x), 3))
```

/* Mais um mecanismo de mistura das palavras. Inicialmente a palavra é rotacionada à esquerda e depois essa palavra rotacionada passa em quatro S-box distintas. Logo após, acontece uma combinação dessas novas palavras, gerando como resultado uma outra palavra. Esse estrutura precede a entrada da palavra na Q-box. */

```
static WORD
S(WORD w, int r)
{
    w = ROTL(w, r);
    return LS0(B(w, 0)) ^ LS1(B(w, 1)) ^ LS2(B(w, 2)) ^ LS3(B(w, 3));
}
```

/* Pseudo-transformação de Hadamard */

```
#define PHT(A,B,C,D,E) { \
```

/* Essa função tem como objetivo misturar as 5 palavras que são retiradas do LFSR */

```
        (E) += (A) + (B) + (C) + (D); \
        (A) += (E); \
        (B) += (E); \
```

```
        (C) += (E); \
        (D) += (E); \
}

                /* Generalizando a pseudo-transformação de Hadamard para n palavras */
void
mixwords(WORD w[], int n)
{
   register WORD        sum;
   register int   i;

   for (sum = i = 0; i < n-1; ++i)
        sum += w[i];
   w[n-1] += sum;
   sum = w[n-1];
   for (i = 0; i < n-1; ++i)
        w[i] += sum;
}
```

/* O procedimento abaixo é para aumentar a segurança da cifra. Para isso, há nova combinação das palavras da chave */

```
void
TuringKey(const BYTE key[], const int keylength)
{
   register int   i, j, k;
   register WORD        w;
```

/* Se o comprimento da chave for menor que 32 bytes ou maior que o tamanho máximo da chave, o procedimento será abortado */

```
   if ((keylength & 0x03) != 0 || keylength > MAXKEY)
        abort();
   keylen = 0;
```

/* Caso contrário, acontecerá a mistura. A variável “i” é atualizada em 4 unidades, dado que as palavras (conjunto de 4 bytes) é que são realmente combinadas */

```
   for (i = 0; i < keylength; i += 4)
        K[keylen++] = fixedS(BYTE2WORD(&key[i]));
   mixwords(K, keylen);
```

```
/* As tabelas da S-box são iniciadas contendo o valor 0. Posteriormente, se necessário a função
lazyS preencherá essas tabelas com outros valores */
   for (j = 0; j < 256; ++j)
         S0[j] = S1[j] = S2[j] = S3[j] = 0;
}

                              /* Leitura do vetor de inicialização */

/* Para preencher o estado inicial do LFSR, será preciso o vetor de inicialização, a chave, o
comprimento da vetor de inicialização e da chave, e mais outros componentes. O vetor de
inicialização passa pelas transformações S-box e Q-box. A chave já foi pré-combinada na função
acima. O restante é preenchido por uma combinação definida e que passa pela S-box e pela Q-box.
*/
void
TuringIV(const BYTE iv[], const int ivlength)
{
   register int   i, j;

/* Verificando se o tamanho do vetor de inicialização é válido */
   if ((ivlength & 0x03) != 0 || (ivlength + 4*keylen) > MAXKIV)
         abort();
   /* Copiando o vetor de inicialização e passando esse vetor pelas funções S-box e Q-box */
   for (i = j = 0; j < ivlength; j +=4)
         R[i++] = fixedS(BYTE2WORD(&iv[j]));
   /* Copiando as informações contidas no vetor da chave */
   for (j = 0 /* i continues */; j < keylen; ++j)
         R[i++] = K[j];
   /* Acréscimo de uma palavra contendo os comprimentos do vetor de inicialização e da chave. Esse
mecanismo garante que diferentes comprimentos de chave/vetor de inicialização gerem estados
iniciais distintos */
   R[i++] = (keylen << 4) | (ivlength >> 2) | 0x01020300UL;
   /* Para completar o preenchimento, acontece a adição da palavra imediatamente anterior à
palavra que ocupa a posição (L+I), sendo que essa adição deve passar pelas funções S-box e Q-
box */
   for (j = 0 /* i continues */; i < LFSRLEN; ++i, ++j)
         R[i] = S(R[j] + R[i-1], 0);
   /* Finalmente, acontece uma mistura de todas as palavras */
   mixwords(R, LFSRLEN);
}
```

```
/* Gerando uma saída contendo 5 palavras */
int
TuringGen(BYTE *buf)
{
   register WORD        A, B, C, D, E;

   step();
/* As palavras A,B,C,D e E são retiradas das posições 16, 13, 6, 1 e 0 do LFSR */
   A = R[16];     B = R[13];        C = R[6];         D = R[1];         E = R[0];
/* Acontece uma mistura dessas palavras */
   PHT(A,         B,                C,                D,                E);
/* O processo não linear que compreende as S-box e Q-box é iniciado por esta linha de comando */
   A = S(A,0);    B = S(B,8);       C = S(C,16);      D = S(D,24);      E = S(E,0);
/* Acontece uma nova mistura */
   PHT(A,         B,                C,                D,                E);
/* O LFSR é atualizado três vezes */
   step(); step(); step();
/* É feita uma nova combinação só que agora com as palavras que ocupam as posições 14, 12, 8,1 e
0 do LFSR */
   A += R[14];    B += R[12];       C += R[8];        D += R[1];        E += R[0];
   WORD2BYTE(A, buf);
                  WORD2BYTE(B, buf+4);
                                    WORD2BYTE(C, buf+8);
                                                      WORD2BYTE(D, buf+12);
                                                                        WORD2BYTE(E, buf+16);
   step();
   return 20;
}
```

6 Resultados

Para testar o stream cipher de Turing, foi implementada a versão conhecida como *Turing Reference*. Inicialmente será feita uma análise com respeito aos resultados finais, para depois passar aos resultados intermediários, isto é, aqueles que saem de cada um dos processos lineares e não-lineares estudados.

6.1 Resultados finais

O algoritmo foi configurado para que primeiro fosse digitada a função "-cipher", para codificar, ou "-decipher" para decodificar. A partir disso, deveriam ser digitados, na base hexadecimal, a chave e o vetor de inicialização, obedecendo a essa ordem. Por fim, deve-se digitar a estrutura que deverá ser codificada ou decodificada, dependendo da função a ser escolhida. Há testes no próprio programa que verificam se os comprimentos do vetor de inicialização e da chave são valores válidos.

Para codificar, o conjunto de palavras escolhidas será "Daniel_Miranda_Barros_Moreira". A chave utilizada nessa primeira etapa será "6f4dab30" e o vetor de inicialização, "28b4c1d9". Com esses valores, encontrou-se a seguinte codificação:

Tabela 2 – Codificação da mensagem

Mensagem	**Chave**	**Vetor de inicialização**
Daniel_Miranda_Barros_Moreira	6f4dab30	28b4c1d9
Resultado		
a112846b6305ef5ef8aa76375a94b8d09bafe88103a9a7d5dbcb1f9ca3		

Para testar a confiabilidade desse stream cipher, foi modificado apenas um dígito da chave. A nova chave será "6f4dab3**f**". Com essa chave, e mantendo-se todos os demais termos, obteve-se:

Tabela 3 – Codificação da mensagem

Mensagem	**Chave**	**Vetor de inicialização**
Daniel_Miranda_Barros_Moreira	6f4dab3f	28b4c1d9
Resultado		
206711b3fc1e6f5ca750eb0e574cc62a056a6c9ba1f027aaf7ce8a7910		

Pode-se perceber que modificando-se apenas um dígito, que corresponde a 4 bits, a resultado final muda drasticamente. O próximo passo consiste na modificação de um dígito do vetor de inicialização. O novo vetor de inicialização será “28b4**d**1d9”. O resultado dessa modificação encontra-se a seguir:

Tabela 4 – Codificação da mensagem

Mensagem	**Chave**	**Vetor de inicialização**
Daniel_Miranda_Barros_Moreira	6f4dab3f	28b4d1d9
Resultado		
76b550de014cbf06ad7bbf023a112184f1ebaf17bca647aa364bdf2b8b		

Houve, novamente, grande modificação em relação ao resultado anterior. A próxima etapa consiste na modificação da mensagem a ser cifrada. Agora, o termo que será cifrado é “Daniel_Miranda_Ba**st**os_Moreira”. O resultado foi:

Tabela 5 – Codificação da mensagem

Mensagem	**Chave**	**Vetor de inicialização**
Daniel_Miranda_Bastos_Moreira	6f4dab3f	28b4d1d9
Resultado		
76b550de014cbf06ad7bbf023a112184f1eaa917bca647aa364bdf2b8b		

Esse resultado não é muito diferente do anterior. Porém, isso já era esperado, pois a partir de mesmo vetor de inicialização e chave, a seqüência gerada é a mesma. A justificativa encontra-se no fato de que a codificação em si corresponde à última etapa (etapa em que acontece a operação ou-exclusivo), após terem sido executados todos os processos existentes.

A próxima etapa consiste na decodificação da mensagem. Para tanto, serão realizados alguns testes para constatar que, se não houver conhecimento total do vetor de inicialização, da chave e do fluxo cifrado, não se conseguirá ter acesso à mensagem original. Por exemplo, pode-se supor que uma terceira pessoa saiba a chave, o vetor de inicialização e parte da seqüência cifrada do caso inicialmente representado, ou seja, saiba a chave “6f4dab30”,o vetor de inicialização “28b4c1d9” e parte da mensagem cifrada “a112846b6305ef5ef8aa76375a94**a**8d09bafe88103a9a7d5dbcb1f9ca3” (o dígito em negrito e sublinhado foi alterado de “b” – valor original - para “a”, ou seja, a seqüência cifrada foi alterada em 1 bit). A função agora a ser digitada é “-decipher”. Ao colocar esses dados no programa, obteve-se o seguinte resultado:

Tabela 6 – Decodificação da mensagem

Mensagem codificada	**Chave**	**Vet. Inic.**
a112846b6305ef5ef8aa76375a94a8d09bafe88103a9a7d5dbcb1f9ca3	6f4dab30	28b4c1d9
Resultado		
Daniel_MirandaOBarros_Moreira		

O resultado foi bem parecido com a mensagem original. Porem, isso já era esperado, pois com mesmo vetor de inicialização e chave, a seqüência gerada é a mesma. A partir dessa seqüência gerada, basta realizar a operação ou-exclusivo duas vezes que assim resultará a mensagem original. Portanto, o foco da segurança deve-se constituir no armazenamento seguro da chave e do vetor de inicialização.

Agora, será testado o caso em que são conhecidas tanto a mensagem cifrada quanto a chave. Porém, o vetor de inicialização é parcialmente conhecido ("28**a**4c1d9"), ou seja, é próximo ao valor original ("28b4c1d9"), mas não idêntico. De acordo com o programa, obteve-se:

Tabela 7 – Decodificação da mensagem

Mensagem codificada	**Chave**	**Vet. Inic.**
a112846b6305ef5ef8aa76375a94b8d09bafe88103a9a7d5dbcb1f9ca3	6f4dab30	28a4c1d9
Resultado		
???jCâ╖ñ¿bY????╛ tH6? ?»?x╡ ╞??		

Ou seja, a mensagem obtida não é nem próxima àquela original.

O próximo passo consiste no teste em que são conhecidos tanto a mensagem cifrada quanto o vetor de inicialização. Porém, a chave não é idêntica àquela utilizada na codificação. Será testado um caso crítico, ou seja, um caso em que há mínima diferença entre a chave utilizada ("6f4d**b**b30") e a original ("6f4dab30"). O resultado encontra-se a seguir:

Tabela 8 – Decodificação da mensagem

Mensagem codificada	**Chave**	**Vet. Inic.**
a112846b6305ef5ef8aa76375a94b8d09bafe88103a9a7d5dbcb1f9ca3	6f4dbb30	28b4c1d9
Resultado		
?▒??╖i_ÇN??UôvC=`╖╟WZ0??R*??╖		

Com base nesse resultado, pode-se constatar que uma mínima alteração na chave – alteração essa de 1 bit, pois da letra “a”, que equivale a 10 (1010), foi trocado por “b”, que equivale a 11 (1011) – foi capaz de mudar completamente o resultado, tornando-o muito diferente da mensagem original.

Por fim, foi testado o caso em que eram conhecidos todos os elementos, isto é, a chave, o vetor de inicialização e a mensagem cifrada. Esse teste está representado logo abaixo.

Tabela 9 – Decodificação da mensagem

Mensagem codificada	**Chave**	**Vet. Inic.**
a112846b6305ef5ef8aa76375a94b8d09bafe88103a9a7d5dbcb1f9ca3	6f4dab30	28b4c1d9
Resultado		
Daniel_Miranda_Barros_Moreira		

Dessa forma, consegue-se obter a mensagem original.

7 Conclusão

A comunicação sempre foi muito importante na vida do ser humano, desde os primórdios até os tempos atuais. Porém, os meios utilizados para se comunicar sofreram alterações. Os noticiários, por exemplo, deixam de estar vinculados ao jornal impresso e à TV, e passam a figurar na internet, aumentando a amplitude das trocas de informações. Em paralelo, surgem algumas comodidades, como o acesso em casa à conta bancária, dentre outras. Inicia-se, então, a preocupação com relação à segurança nas trocas de informações.

Nesse contexto, aumenta-se a importância da criptografia, na tentativa de evitar que outras pessoas tenham acesso à informação que determinado usuário emitiu. Diversos sistemas criptográficos existem, sendo que alguns desses são considerados lentos computacionalmente quando a codificação/decodificação acontece para grandes mensagens, e até mesmo inseguros frente a usuários mal-intencionados.

Por isso, grandes empresas de software investem vultosas quantias em pesquisa e desenvolvimento de sistemas criptográficos. Dentre estes recentes softwares desenvolvidos encontra-se o stream cipher de Turing. Esse software apresenta mecanismos internos avançados que fazem com que a seqüência gerada apresente elevado grau de imprevisibilidade e um período muito grande, evidenciando ser um software bastante seguro.

Com relação ao desempenho, as informações que constam nesse trabalho de graduação vêm do fabricante e relatam que é um programa extremamente rápido. Em relação aos principais programas criptográficos usados atualmente, a codificação/decodificação acontece mais rapidamente no stream cipher de Turing.

Dessa forma, o stream cipher de Turing mostrou, com base nas tecnologias disponíveis atualmente, ser um programa extremamente eficiente e que atende às principais exigências desse segmento de mercado da informática. Porém, novas tecnologias surgirão e poderão fazer com que esse stream cipher caia em desuso, quer pela necessidade de melhores desempenhos, quer pela necessidade de maior segurança.

8 Referências bibliográficas

Livros

1. Adamson, Iain T. *Introduction to the field theory*. 2nd ed. University Press, Cambridge, 1982.

2. Dean, Richard A. *Elementos de álgebra abstrata*; tradução de Carlos Alberto A. de Carvalho. Rio de Janeiro, Livros Técnicos e Científicos, 1974.

3. Menezes, Alfred J., Van Oorschot, Paul e Vanstone, Scott. *Handbook of applied cryptography*. Boca Raton: CRC Press, 1996.

4. Robshaw, M.J.B. *Stream Ciphers* Technical Report TR-701, version 2.0, RSA Laboratories, 1995.

5. Rueppel, R. *Analysis and Design of Stream Ciphers*. Berlin, Heidelberg: Springer-Verlag, 1986.

6. Schneier, B. *Applied Cryptography Second Edition: protocols, algorithms, and source code in C*. John Wiley & Sons, 1996.

7. Stallings, W. *Cryptography and Network Security: Principles and Practice*. Prentice Hall.

8. Stinson, D. *Cryptography: Theory and Practice*. Boca Raton, FL: CRC Press, 1995.

9. Coppersmith et al, "Cryptanalysis of Stream Ciphers with Linear Masking", proc. Crypto 2002, LNCS 2442, Springer 2002.

10. E. Dawson, W. Millan, L. Burnett, G. Carter, "On the Design of 8*32 S-boxes". Unpublished report, by the Information Systems Research Centre, Queensland University of Technology, 1999.

11. Patrick Ekdahl, Thomas Johansson, "A new version of the stream cipher SNOW", Proc. Selected Areas in Cryptography, LNCS 2002.

Artigo de evento

12. Yamamoto, Daniel Massaki; Rabello, Tânia Nunes. Análise de "stream ciphers". In: 9° Encontro de Iniciação Científica e Pós-Graduação do ITA, 2003, São José dos Campos.

Artigos na Internet

13. Rose, Greg; Hawkes, Philip. Turing: a fast stream cipher, p.1-12, nov.,2002. Disponível em: < http://eprint.iacr.org/2002/185.pdf>. Acesso em 26 abr. 2006.

14. Rose, Greg; Hawkes, Philip; Paddon, Michael. The mundja streaming MAC, p.1-15, abr.,2001. Disponível em: < http://www.ecrypt.eu.org/stvl/sasc/slides5.pdf >. Acesso em 08 out. 2006.

15. Puttini, Ricardo. Criptografia. p. 13-30, mai.,2004. Universidade de Brasília – Departamento de Engenharia Elétrica. Disponível em: <http://www.redes.unb.br/security/criptografia.pdf>. Acesso em 23 ago. 2006.

Internet

16. WIKIPEDIA: the free encyclopedia. Definição do termo Turing (cipher). Disponível em: <http://en.wikipedia.org/wiki/Turing_(cipher)> . Acesso: 27 abr. 2006.
17. MARSAGLIA, G. Diehard Statistical tests. Disponível em: <http://stat.fsu.edu/~geo/diehard.html> . Acesso: 25 abr. 2006
18. NIST, Random Number Generation and Testing. Disponível em: <http://csrc.nist.gov/mg/> . Acesso: 25 abr. 2006
19. QUALCOMM, Austrália. Código-fonte das implementações, artigo e apresentação sobre stream cipher de Turing. Disponível em: <http://www.qualcomm.com.au/Turing.html> . Acesso: 20 abr. 2006.
20. NUMABOA. Introdução à criptologia, conceitos básicos. Disponível em: <http://www.numaboa.com.br/criptologia/intro.php>. Acesso: 04 mai. 2006
21. CBEJI, Centro de Excelência em Direito da Tecnologia da Informação. Definição de assinatura digital. Disponível em: <http://www.cbeji.com.br/br/novidades/artigos/main.asp?id=1713> . Acesso: 18 mai. 2006.
22. NUMABOA. Definição de algoritmos modernos e assinaturas digitais. Disponível em: <http://www.numaboa.com.br/criptologia/cifras/algoritmos.php> . Acesso: 31 mai. 2006.
23. UFRJ, Instituto de Matemática. Contribuição de Alan Turing. Disponível em: <www.dmm.im.ufrj.br/~risk/Site_AI/turing.htm>. Acesso: 24 jun. 2006.

Apêndice A: Tabela de multiplicação do stream cipher de Turing

Essa tabela de multiplicação foi obtida utilizando como dado inicial a palavra `0x72C688E8.`

```
unsigned_ long_ Multab[256]_ =_ {_
_ _ _ _ 0x00000000,_ 0x72C688E8,_ 0xE4C15D9D,_ 0x9607D575,_
_ _ _ _ 0x85CFBA77,_ 0xF709329F,_ 0x610EE7EA,_ 0x13C86F02,_
_ _ _ _ 0x47D339EE,_ 0x3515B106,_ 0xA3126473,_ 0xD1D4EC9B,_
_ _ _ _ 0xC21C8399,_ 0xB0DA0B71,_ 0x26DDDE04,_ 0x541B56EC,_
_ _ _ _ 0x8EEB7291,_ 0xFC2DFA79,_ 0x6A2A2F0C,_ 0x18ECA7E4,_
_ _ _ _ 0x0B24C8E6,_ 0x79E2400E,_ 0xEFE5957B,_ 0x9D231D93,_
_ _ _ _ 0xC9384B7F,_ 0xBBFEC397,_ 0x2DF916E2,_ 0x5F3F9E0A,_
_ _ _ _ 0x4CF7F108,_ 0x3E3179E0,_ 0xA836AC95,_ 0xDAF0247D,_
_ _ _ _ 0x519BE46F,_ 0x235D6C87,_ 0xB55AB9F2,_ 0xC79C311A,_
_ _ _ _ 0xD4545E18,_ 0xA692D6F0,_ 0x30950385,_ 0x42538B6D,_
_ _ _ _ 0x1648DD81,_ 0x648E5569,_ 0xF289801C,_ 0x804F08F4,_
_ _ _ _ 0x938767F6,_ 0xE141EF1E,_ 0x77463A6B,_ 0x0580B283,_
_ _ _ _ 0xDF7096FE,_ 0xADB61E16,_ 0x3BB1CB63,_ 0x4977438B,_
_ _ _ _ 0x5ABF2C89,_ 0x2879A461,_ 0xBE7E7114,_ 0xCCB8F9FC,_
_ _ _ _ 0x98A3AF10,_ 0xEA6527F8,_ 0x7C62F28D,_ 0x0EA47A65,_
_ _ _ _ 0x1D6C1567,_ 0x6FAA9D8F,_ 0xF9AD48FA,_ 0x8B6BC012,_
_ _ _ _ 0xA27B85DE,_ 0xD0BD0D36,_ 0x46BAD843,_ 0x347C50AB,_
_ _ _ _ 0x27B43FA9,_ 0x5572B741,_ 0xC3756234,_ 0xB1B3EADC,_
_ _ _ _ 0xE5A8BC30,_ 0x976E34D8,_ 0x0169E1AD,_ 0x73AF6945,_
_ _ _ _ 0x60670647,_ 0x12A18EAF,_ 0x84A65BDA,_ 0xF660D332,_
_ _ _ _ 0x2C90F74F,_ 0x5E567FA7,_ 0xC851AAD2,_ 0xBA97223A,_
_ _ _ _ 0xA95F4D38,_ 0xDB99C5D0,_ 0x4D9E10A5,_ 0x3F58984D,_
_ _ _ _ 0x6B43CEA1,_ 0x19854649,_ 0x8F82933C,_ 0xFD441BD4,_
_ _ _ _ 0xEE8C74D6,_ 0x9C4AFC3E,_ 0x0A4D294B,_ 0x788BA1A3,_
_ _ _ _ 0xF3E061B1,_ 0x8126E959,_ 0x17213C2C,_ 0x65E7B4C4,_
_ _ _ _ 0x762FDBC6,_ 0x04E9532E,_ 0x92EE865B,_ 0xE0280EB3,_
_ _ _ _ 0xB433585F,_ 0xC6F5D0B7,_ 0x50F205C2,_ 0x22348D2A,_
_ _ _ _ 0x31FCE228,_ 0x433A6AC0,_ 0xD53DBFB5,_ 0xA7FB375D,_
_ _ _ _ 0x7D0B1320,_ 0x0FCD9BC8,_ 0x99CA4EBD,_ 0xEB0CC655,_
_ _ _ _ 0xF8C4A957,_ 0x8A0221BF,_ 0x1C05F4CA,_ 0x6EC37C22,_
_ _ _ _ 0x3AD82ACE,_ 0x481EA226,_ 0xDE197753,_ 0xACDFFFBB,_
_ _ _ _ 0xBF1790B9,_ 0xCDD11851,_ 0x5BD6CD24,_ 0x291045CC,_
_ _ _ _ 0x09F647F1,_ 0x7B30CF19,_ 0xED371A6C,_ 0x9FF19284,_
_ _ _ _ 0x8C39FD86,_ 0xFEFF756E,_ 0x68F8A01B,_ 0x1A3E28F3,_
_ _ _ _ 0x4E257E1F,_ 0x3CE3F6F7,_ 0xAAE42382,_ 0xD822AB6A,_
_ _ _ _ 0xCBEAC468,_ 0xB92C4C80,_ 0x2F2B99F5,_ 0x5DED111D,_
_ _ _ _ 0x871D3560,_ 0xF5DBBD88,_ 0x63DC68FD,_ 0x111AE015,_
_ _ _ _ 0x02D28F17,_ 0x701407FF,_ 0xE613D28A,_ 0x94D55A62,_
_ _ _ _ 0xC0CE0C8E,_ 0xB2088466,_ 0x240F5113,_ 0x56C9D9FB,_
_ _ _ _ 0x4501B6F9,_ 0x37C73E11,_ 0xA1C0EB64,_ 0xD306638C,_
_ _ _ _ 0x586DA39E,_ 0x2AAB2B76,_ 0xBCACFE03,_ 0xCE6A76EB,_
_ _ _ _ 0xDDA219E9,_ 0xAF649101,_ 0x39634474,_ 0x4BA5CC9C,_
_ _ _ _ 0x1FBE9A70,_ 0x6D781298,_ 0xFB7FC7ED,_ 0x89B94F05,_
_ _ _ _ 0x9A712007,_ 0xE8B7A8EF,_ 0x7EB07D9A,_ 0x0C76F572,_
_ _ _ _ 0xD686D10F,_ 0xA44059E7,_ 0x32478C92,_ 0x4081047A,_
_ _ _ _ 0x53496B78,_ 0x218FE390,_ 0xB78836E5,_ 0xC54EBE0D,_
_ _ _ _ 0x9155E8E1,_ 0xE3936009,_ 0x7594B57C,_ 0x07523D94,_
_ _ _ _ 0x149A5296,_ 0x665CDA7E,_ 0xF05B0F0B,_ 0x829D87E3,_
_ _ _ _ 0xAB8DC22F,_ 0xD94B4AC7,_ 0x4F4C9FB2,_ 0x3D8A175A,_
_ _ _ _ 0x2E427858,_ 0x5C84F0B0,_ 0xCA8325C5,_ 0xB845AD2D,_
_ _ _ _ 0xEC5EFBC1,_ 0x9E987329,_ 0x089FA65C,_ 0x7A592EB4,_
_ _ _ _ 0x699141B6,_ 0x1B57C95E,_ 0x8D501C2B,_ 0xFF9694C3,_
_ _ _ _ 0x2566B0BE,_ 0x57A03856,_ 0xC1A7ED23,_ 0xB36165CB,_
_ _ _ _ 0xA0A90AC9,_ 0xD26F8221,_ 0x44685754,_ 0x36AEDFBC,_
_ _ _ _ 0x62B58950,_ 0x107301B8,_ 0x8674D4CD,_ 0xF4B25C25,_
```

```
_ _ _ _ 0xE77A3327,_ 0x95BCBBCF,_ 0x03BB6EBA,_ 0x717DE652,_
_ _ _ _ 0xFA162640,_ 0x88D0AEA8,_ 0x1ED77BDD,_ 0x6C11F335,_
_ _ _ _ 0x7FD99C37,_ 0x0D1F14DF,_ 0x9B18C1AA,_ 0xE9DE4942,_
_ _ _ _ 0xBDC51FAE,_ 0xCF039746,_ 0x59044233,_ 0x2BC2CADB,_
_ _ _ _ 0x380AA5D9,_ 0x4ACC2D31,_ 0xDCCBF844,_ 0xAE0D70AC,_
_ _ _ _ 0x74FD54D1,_ 0x063BDC39,_ 0x903C094C,_ 0xE2FA81A4,_
_ _ _ _ 0xF132EEA6,_ 0x83F4664E,_ 0x15F3B33B,_ 0x67353BD3,_
_ _ _ _ 0x332E6D3F,_ 0x41E8E5D7,_ 0xD7EF30A2,_ 0xA529B84A,_
_ _ _ _ 0xB6E1D748,_ 0xC4275FA0,_ 0x52208AD5,_ 0x20E6023D,_
};
```

Apêndice B: A S-box do stream cipher de Turing

O uso desses valores encontra-se explicado no capítulo 4 deste trabalho de graduação.

```
unsigned_ char_ Sbox[256]_ =_ {_
_ _ _ _ 0x61,_ 0x51,_ 0xeb,_ 0x19,_ 0xb9,_ 0x5d,_ 0x60,_ 0x38,_
_ _ _ _ 0x7c,_ 0xb2,_ 0x06,_ 0x12,_ 0xc4,_ 0x5b,_ 0x16,_ 0x3b,_
_ _ _ _ 0x2b,_ 0x18,_ 0x83,_ 0xb0,_ 0x7f,_ 0x75,_ 0xfa,_ 0xa0,_
_ _ _ _ 0xe9,_ 0xdd,_ 0x6d,_ 0x7a,_ 0x6b,_ 0x68,_ 0x2d,_ 0x49,_
_ _ _ _ 0xb5,_ 0x1c,_ 0x90,_ 0xf7,_ 0xed,_ 0x9f,_ 0xe8,_ 0xce,_
_ _ _ _ 0xae,_ 0x77,_ 0xc2,_ 0x13,_ 0xfd,_ 0xcd,_ 0x3e,_ 0xcf,_
_ _ _ _ 0x37,_ 0x6a,_ 0xd4,_ 0xdb,_ 0x8e,_ 0x65,_ 0x1f,_ 0x1a,_
_ _ _ _ 0x87,_ 0xcb,_ 0x40,_ 0x15,_ 0x88,_ 0x0d,_ 0x35,_ 0xb3,_
_ _ _ _ 0x11,_ 0x0f,_ 0xd0,_ 0x30,_ 0x48,_ 0xf9,_ 0xa8,_ 0xac,_
_ _ _ _ 0x85,_ 0x27,_ 0x0e,_ 0x8a,_ 0xe0,_ 0x50,_ 0x64,_ 0xa7,_
_ _ _ _ 0xcc,_ 0xe4,_ 0xf1,_ 0x98,_ 0xff,_ 0xa1,_ 0x04,_ 0xda,_
_ _ _ _ 0xd5,_ 0xbc,_ 0x1b,_ 0xbb,_ 0xd1,_ 0xfe,_ 0x31,_ 0xca,_
_ _ _ _ 0xba,_ 0xd9,_ 0x2e,_ 0xf3,_ 0x1d,_ 0x47,_ 0x4a,_ 0x3d,_
_ _ _ _ 0x71,_ 0x4c,_ 0xab,_ 0x7d,_ 0x8d,_ 0xc7,_ 0x59,_ 0xb8,_
_ _ _ _ 0xc1,_ 0x96,_ 0x1e,_ 0xfc,_ 0x44,_ 0xc8,_ 0x7b,_ 0xdc,_
_ _ _ _ 0x5c,_ 0x78,_ 0x2a,_ 0x9d,_ 0xa5,_ 0xf0,_ 0x73,_ 0x22,_
_ _ _ _ 0x89,_ 0x05,_ 0xf4,_ 0x07,_ 0x21,_ 0x52,_ 0xa6,_ 0x28,_
_ _ _ _ 0x9a,_ 0x92,_ 0x69,_ 0x8f,_ 0xc5,_ 0xc3,_ 0xf5,_ 0xe1,_
_ _ _ _ 0xde,_ 0xec,_ 0x09,_ 0xf2,_ 0xd3,_ 0xaf,_ 0x34,_ 0x23,_
_ _ _ _ 0xaa,_ 0xdf,_ 0x7e,_ 0x82,_ 0x29,_ 0xc0,_ 0x24,_ 0x14,_
_ _ _ _ 0x03,_ 0x32,_ 0x4e,_ 0x39,_ 0x6f,_ 0xc6,_ 0xb1,_ 0x9b,_
_ _ _ _ 0xea,_ 0x72,_ 0x79,_ 0x41,_ 0xd8,_ 0x26,_ 0x6c,_ 0x5e,_
_ _ _ _ 0x2c,_ 0xb4,_ 0xa2,_ 0x53,_ 0x57,_ 0xe2,_ 0x9c,_ 0x86,_
_ _ _ _ 0x54,_ 0x95,_ 0xb6,_ 0x80,_ 0x8c,_ 0x36,_ 0x67,_ 0xbd,_
_ _ _ _ 0x08,_ 0x93,_ 0x2f,_ 0x99,_ 0x5a,_ 0xf8,_ 0x3a,_ 0xd7,_
_ _ _ _ 0x56,_ 0x84,_ 0xd2,_ 0x01,_ 0xf6,_ 0x66,_ 0x4d,_ 0x55,_
_ _ _ _ 0x8b,_ 0x0c,_ 0x0b,_ 0x46,_ 0xb7,_ 0x3c,_ 0x45,_ 0x91,_
_ _ _ _ 0xa4,_ 0xe3,_ 0x70,_ 0xd6,_ 0xfb,_ 0xe6,_ 0x10,_ 0xa9,_
_ _ _ _ 0xc9,_ 0x00,_ 0x9e,_ 0xe7,_ 0x4f,_ 0x76,_ 0x25,_ 0x3f,_
_ _ _ _ 0x5f,_ 0xa3,_ 0x33,_ 0x20,_ 0x02,_ 0xef,_ 0x62,_ 0x74,_
_ _ _ _ 0xee,_ 0x17,_ 0x81,_ 0x42,_ 0x58,_ 0x0a,_ 0x4b,_ 0x63,_
_ _ _ _ 0xe5,_ 0xbe,_ 0x6e,_ 0xad,_ 0xbf,_ 0x43,_ 0x94,_ 0x97,_
};_
```

Apêndice C: A Q-box do stream cipher de Turing

O uso desses valores encontra-se explicado no capítulo 4 deste trabalho de graduação.

```
WORD_ Qbox[256]_ =_ {_
_ 0x1faa1887,_ _ _ 0x4e5e435c,_ _ _ 0x9165c042,_ _ _ 0x250e6ef4,_ _ _ _
_ 0x5957ee20,_ _ _ 0xd484fed3,_ _ _ 0xa666c502,_ _ _ 0x7e54e8ae,_ _ _ _
_ 0xd12ee9d9,_ _ _ 0xfc1f38d4,_ _ _ 0x49829b5d,_ _ _ 0x1b5cdf3c,_ _ _ _
_ 0x74864249,_ _ _ 0xda2e3963,_ _ _ 0x28f4429f,_ _ _ 0xc8432c35,_ _ _ _
_ 0x4af40325,_ _ _ 0x9fc0dd70,_ _ _ 0xd8973ded,_ _ _ 0x1a02dc5e,_ _ _ _
_ 0xcd175b42,_ _ _ 0xf10012bf,_ _ _ 0x6694d78c,_ _ _ 0xacaab26b,_ _ _ _
_ 0x4ec11b9a,_ _ _ 0x3f168146,_ _ _ 0xc0ea8ec5,_ _ _ 0xb38ac28f,_ _ _ _
_ 0x1fed5c0f,_ _ _ 0xaab4101c,_ _ _ 0xea2db082,_ _ _ 0x470929e1,_ _ _ _
_ 0xe71843de,_ _ _ 0x508299fc,_ _ _ 0xe72fbc4b,_ _ _ 0x2e3915dd,_ _ _ _
_ 0x9fa803fa,_ _ _ 0x9546b2de,_ _ _ 0x3c233342,_ _ _ 0x0fcee7c3,_ _ _ _
_ 0x24d607ef,_ _ _ 0x8f97ebab,_ _ _ 0xf37f859b,_ _ _ 0xcd1f2e2f,_ _ _ _
_ 0xc25b71da,_ _ _ 0x75e2269a,_ _ _ 0x1e39c3d1,_ _ _ 0xeda56b36,_ _ _ _
_ 0xf8c9def2,_ _ _ 0x46c9fc5f,_ _ _ 0x1827b3a3,_ _ _ 0x70a56ddf,_ _ _ _
_ 0x0d25b510,_ _ _ 0x000f85a7,_ _ _ 0xb2e82e71,_ _ _ 0x68cb8816,_ _ _ _
_ 0x8f951e2a,_ _ _ 0x72f5f6af,_ _ _ 0xe4cbc2b3,_ _ _ 0xd34ff55d,_ _ _ _
_ 0x2e6b6214,_ _ _ 0x220b83e3,_ _ _ 0xd39ea6f5,_ _ _ 0x6fe041af,_ _ _ _
_ 0x6b2f1f17,_ _ _ 0xad3b99ee,_ _ _ 0x16a65ec0,_ _ _ 0x757016c6,_ _ _ _
_ 0xba7709a4,_ _ _ 0xb0326e01,_ _ _ 0xf4b280d9,_ _ _ 0x4bfb1418,_ _ _ _
_ 0xd6aff227,_ _ _ 0xfd548203,_ _ _ 0xf56b9d96,_ _ _ 0x6717a8c0,_ _ _ _
_ 0x00d5bf6e,_ _ _ 0x10ee7888,_ _ _ 0xedfcfe64,_ _ _ 0x1ba193cd,_ _ _ _
_ 0x4b0d0184,_ _ _ 0x89ae4930,_ _ _ 0x1c014f36,_ _ _ 0x82a87088,_ _ _ _
_ 0x5ead6c2a,_ _ _ 0xef22c678,_ _ _ 0x31204de7,_ _ _ 0xc9c2e759,_ _ _ _
_ 0xd200248e,_ _ _ 0x303b446b,_ _ _ 0xb00d9fc2,_ _ _ 0x9914a895,_ _ _ _
_ 0x906cc3a1,_ _ _ 0x54fef170,_ _ _ 0x34c19155,_ _ _ 0xe27b8a66,_ _ _ _
_ 0x131b5e69,_ _ _ 0xc3a8623e,_ _ _ 0x27bdfa35,_ _ _ 0x97f068cc,_ _ _ _
_ 0xca3a6acd,_ _ _ 0x4b55e936,_ _ _ 0x86602db9,_ _ _ 0x51df13c1,_ _ _ _
_ 0x390bb16d,_ _ _ 0x5a80b83c,_ _ _ 0x22b23763,_ _ _ 0x39d8a911,_ _ _ _
_ 0x2cb6bc13,_ _ _ 0xbf5579d7,_ _ _ 0x6c5c2fa8,_ _ _ 0xa8f4196e,_ _ _ _
_ 0xbcdb5476,_ _ _ 0x6864a866,_ _ _ 0x416e16ad,_ _ _ 0x897fc515,_ _ _ _
_ 0x956feb3c,_ _ _ 0xf6c8a306,_ _ _ 0x216799d9,_ _ _ 0x171a9133,_ _ _ _
_ 0x6c2466dd,_ _ _ 0x75eb5dcd,_ _ _ 0xdf118f50,_ _ _ 0xe4afb226,_ _ _ _
_ 0x26b9cef3,_ _ _ 0xadb36189,_ _ _ 0x8a7a19b1,_ _ _ 0xe2c73084,_ _ _ _
_ 0xf77ded5c,_ _ _ 0x8b8bc58f,_ _ _ 0x06dde421,_ _ _ 0xb41e47fb,_ _ _ _
_ 0xb1cc715e,_ _ _ 0x68c0ff99,_ _ _ 0x5d122f0f,_ _ _ 0xa4d25184,_ _ _ _
_ 0x097a5e6c,_ _ _ 0x0cbf18bc,_ _ _ 0xc2d7c6e0,_ _ _ 0x8bb7e420,_ _ _ _
_ 0xa11f523f,_ _ _ 0x35d9b8a2,_ _ _ 0x03da1a6b,_ _ _ 0x06888c02,_ _ _ _
_ 0x7dd1e354,_ _ _ 0x6bba7d79,_ _ _ 0x32cc7753,_ _ _ 0xe52d9655,_ _ _ _
_ 0xa9829da1,_ _ _ 0x301590a7,_ _ _ 0x9bc1c149,_ _ _ 0x13537f1c,_ _ _ _
_ 0xd3779b69,_ _ _ 0x2d71f2b7,_ _ _ 0x183c58fa,_ _ _ 0xacdc4418,_ _ _ _
_ 0x8d8c8c76,_ _ _ 0x2620d9f0,_ _ _ 0x71a80d4d,_ _ _ 0x7a74c473,_ _ _ _
_ 0x449410e9,_ _ _ 0xa20e4211,_ _ _ 0xf9c8082b,_ _ _ 0x0a6b334a,_ _ _ _
_ 0xb5f68ed2,_ _ _ 0x8243cc1b,_ _ _ 0x453c0ff3,_ _ _ 0x9be564a0,_ _ _ _
_ 0x4ff55a4f,_ _ _ 0x8740f8e7,_ _ _ 0xcca7f15f,_ _ _ 0xe300fe21,_ _ _ _
_ 0x786d37d6,_ _ _ 0xdfd506f1,_ _ _ 0x8ee00973,_ _ _ 0x17bbde36,_ _ _ _
_ 0x7a670fa8,_ _ _ 0x5c31ab9e,_ _ _ 0xd4dab618,_ _ _ 0xcc1f52f5,_ _ _ _
_ 0xe358eb4f,_ _ _ 0x19b9e343,_ _ _ 0x3a8d77dd,_ _ _ 0xcdb93da6,_ _ _ _
_ 0x140fd52d,_ _ _ 0x395412f8,_ _ _ 0x2ba63360,_ _ _ 0x37e53ad0,_ _ _ _
_ 0x80700f1c,_ _ _ 0x7624ed0b,_ _ _ 0x703dc1ec,_ _ _ 0xb7366795,_ _ _ _
_ 0xd6549d15,_ _ _ 0x66ce46d7,_ _ _ 0xd17abe76,_ _ _ 0xa448e0a0,_ _ _ _
_ 0x28f07c02,_ _ _ 0xc31249b7,_ _ _ 0x6e9ed6ba,_ _ _ 0xeaa47f78,_ _ _ _
_ 0xbbcfffbd,_ _ _ 0xc507ca84,_ _ _ 0xe965f4da,_ _ _ 0x8e9f35da,_ _ _ _
_ 0x6ad2aa44,_ _ _ 0x577452ac,_ _ _ 0xb5d674a7,_ _ _ 0x5461a46a,_ _ _ _
_ 0x6763152a,_ _ _ 0x9c12b7aa,_ _ _ 0x12615927,_ _ _ 0x7b4fb118,_ _ _ _
_ 0xc351758d,_ _ _ 0x7e81687b,_ _ _ 0x5f52f0b3,_ _ _ 0x2d4254ed,_ _ _ _
```

```
_ 0xd4c77271,_ _ _ 0x0431acab,_ _ _ 0xbef94aec,_ _ _ 0xfee994cd,_ _ _ _
_ 0x9c4d9e81,_ _ _ 0xed623730,_ _ _ 0xcf8a21e8,_ _ _ 0x51917f0b,_ _ _ _
_ 0xa7a9b5d6,_ _ _ 0xb297adf8,_ _ _ 0xeed30431,_ _ _ 0x68cac921,_ _ _ _
_ 0xf1b35d46,_ _ _ 0x7a430a36,_ _ _ 0x51194022,_ _ _ 0x9abca65e,_ _ _ _
_ 0x85ec70ba,_ _ _ 0x39aea8cc,_ _ _ 0x737bae8b,_ _ _ 0x582924d5,_ _ _ _
_ 0x03098a5a,_ _ _ 0x92396b81,_ _ _ 0x18de2522,_ _ _ 0x745c1cb8,_ _ _ _
_ 0xa1b8fe1d,_ _ _ 0x5db3c697,_ _ _ 0x29164f83,_ _ _ 0x97c16376,_ _ _ _
_ 0x8419224c,_ _ _ 0x21203b35,_ _ _ 0x833ac0fe,_ _ _ 0xd966a19a,_ _ _ _
_ 0xaaf0b24f,_ _ _ 0x40fda998,_ _ _ 0xe7d52d71,_ _ _ 0x390896a8,_ _ _ _
_ 0xcee6053f,_ _ _ 0xd0b0d300,_ _ _ 0xff99cbcc,_ _ _ 0x065e3d40,_ _ _ _
};_
```

Apêndice D: O polinômio binário equivalente do LFSR

O LFSR do stream cipher de Turing é equivalente a 32 LFSR's binários em paralelo, apresentando uma característica polinomial. Logo abaixo há a representação desses valores binários. É importante acrescentar que o primeiro bit mostrado é relativo ao termo constante do polinômio (grau 0) e que o último termo se refere ao coeficiente do termo de maior grau (grau 544). Assim:

```
10000000000000000101010101011101110111011000110110000111110001101
10111010111011100110011110110111111001111100011101101101001001011
01111000101111110110101111000000000000101111111110100110001101001 0
10110010101011111000000010011010011101000111001111110010100101011 1
01111101110100010101000100111010000001100011010101101000010101100 1
10001101001110011110100001100100110000101101011000010010001100010 1
11111100101001000111111001100110000101111110010011000000111010111 1
01011010011100111101110000101000000010000111000010001010001000000 0
10000000100000001
```

Ou seja, com base nesses bits acima, obtém-se o polinômio $p_1(x) = 1 + x^{17} + x^{19} + x^{21} + x^{23} + x^{25} + x^{27} + x^{29} + x^{30} + \ldots + x^{520} + x^{528} + x^{536} + x^{544}$.

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. DOCUMENTO N°	4. N° DE PÁGINAS
TC	09 de novembro de 2006	CTA/ITA-IEI/TC-001/2006	81

5. TÍTULO E SUBTÍTULO:
Estudo do stream cipher de Turing.

6. AUTOR(ES):

Daniel Miranda Barros Moreira

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):
Instituto Tecnológico de Aeronáutica. Divisão de Engenharia de Infra-estrutura Aeronáutica – ITA/IEI

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

1. Criptografia; 2. Stream cipher; 3. Implementação do stream cipher de Turing.

9.PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Criptografia; Seqüências pseudo-aleatórias; Complexidade computacional; Programação matemática; Algoritmos; Matemática; Comunicações

10. APRESENTAÇÃO: **X Nacional** **Internacional**

Trabalho de Graduação, ITA, São José dos Campos, 2006. 81 páginas.

11. RESUMO:

Nos dias atuais, a criptografia é uma área estratégica para a comunicação segura, pois as informações são, na maioria das vezes, transmitidas por meios eletrônicos e, portanto, expostas abertamente. Os sistemas criptográficos fornecem a segurança na transmissão por uso de transformações. Os streams ciphers são sistemas criptográficos que dividem o texto claro em caracteres e codificam cada um deles com uma função que depende, dentre outras coisas, da chave e do vetor de inicialização. São utilizados, por exemplo, na comunicação entre aeronaves e/ou entre aeronave-base, por serem facilmente implementáveis em hardware. Já o stream cipher de Turing, esse foi projetado para que, além das características do stream cipher, seja extremamente eficiente nas implementações em software. Nesta tese há o estudo dos mecanismos de funcionamento do stream cipher de Turing, compreendendo processos lineares, como a pseudo-transformação de Hadamard (PHT), e não-lineares, como as S-box e Q-box. No final desse trabalho, há o código-fonte documentado das quatro implementações disponíveis pelo fabricante desse stream cipher, assim como o estudo dos resultados baseados numa das implementações, conhecida como *Turing Reference*.

12. GRAU DE SIGILO:

(X) OSTENSIVO **( ) RESERVADO** **( ) CONFIDENCIAL** **( ) SECRETO**