VERIFICAÇÃO DA ROBUSTEZ DA ALEATORIEDADE DE ALGORITMOS USADOS EM CRIPTOGRAFIA

Jaime Ribeiro Júnior*

Anderson Cavalcante Gonçalves**

RESUMO— Ataques de análise diferencial e análise linear são utilizados para reduzir o espaço de busca por uma chave criptográfica, muitas vezes de forma satisfatória. Uma vez que a sequência binária é distribuída de maneira fortemente aleatória análises podem ser frustradas. O uso de números aleatórios se destaca na área de Segurança Computacional, algoritmos aleatórios robustos são desejados na área. O NIST estipulou uma série de testes durante a seleção do AES. Os testes de aleatoriedade foram tão importantes que eles foram padronizados, revisados e publicados em 2008 pelo NIST na publicação especial 800-22. Este trabalho apresenta um protótipo da implementação de 2 dos 16 testes de aleatoriedade estipulados.
PALAVRAS CHAVE -- Criptologia, Aleatoriedade, Estatística, NIST, TACP.
ABSTRACT - Attacks differential analysis and linear analysis are used to reduce the search space by a cryptographic key, often satisfactorily. Since the binary sequence is random distributed manner analyzes can be greatly frustrated. The use of random numbers excels in the area of Computer Security, randomized algorithms are robust in the desired area. NIST stipulated a series of tests during the selection of the AES. Tests of randomness were so important that they were standardized, revised and published in 2008 by the NIST Special Publication 800-22. This paper presents a prototype implementation of 2 of the 16 tests of randomness stipulated.
KEYWORDS -- Cryptology, Randomness, Statistics, NIST, TACP.

Data de recepção:17/08/2013
Data de aprovação do trabalho:22/09/2013

INTRODUÇÃO

Algoritmos Criptográficos geradores de cifras aleatórias são sequências de passos matemáticos que geram valores diferentes de saída toda vez que são passados o mesmo valor de entrada e que consigam fazer os cálculos de volta gerando o valor inicial como saída para todos os valores gerados anteriormente. Já um algoritmo criptográfico com saídas não aleatórias gera sempre um mesmo valor de saída para um valor de entrada.

J. Ribeiro Júnior, Professor do Curso Superior de Tecnologia em Redes de Computadores, Universidade Estadual de Goiás – UnU de Pires do Rio. (jaime.ribeirojunior@ueg.br).
**A. Cavalcante Gonçalves, Professor do Curso Superior de Tecnologia em Redes de Computadores, Universidade Estadual de Goiás – UnU de Pires do Rio. (andersontwoand@gmail.com).*

Mesmo algoritmos que consigam gerar números aleatórios devem passar por uma série de testes para garantir que exista um bom nível de aleatoriedade. Pois com recursos computacionais adequados pode-se realizar a criptoanálise. Com o algoritmo gerador e os testes matemáticos adequados, é possível prever os bits originários de uma sequência de bits. Com um algoritmo que gere saídas aleatórias, esse tipo de criptoanálise se torna impraticável.

Segundo Stallings (2008) Quanto mais próximas forem às saídas dos algoritmos criptográficos das fontes verdadeiramente aleatórias maior será a dificuldade na previsão de uma próxima saída.

Existem testes e funções matemáticas que são usados para checar o nível da aleatoriedade que uma função pode gerar. O NIST (*National Institute of Standars and Technology*) no processo de seleção do substituto do DES (*Data Encryption Standard*), o AES (*Advanced Encryption Standard*), impôs que os algoritmos candidatos deveriam passar nos testes de aleatoriedade. Isto é, os algoritmos criptográficos apresentados deveriam se comportar como geradores de números pseudoaleatórios.

Uma bateria de 16 testes foi proposta para avaliar o grau da aleatoriedade gerada na saída dos algoritmos candidatos, esta foi publicada no documento SP800-22 do NIST (Rukhin, 2008). Um candidato aprovado nos testes, segundo o NIST, seria imune a ataques de criptoanálise diferencial e linear que foram empregados na quebra do DES.

A bateria contém os seguintes testes: Teste de frequência(*monobit*); Teste de frequência dentro de blocos; Teste de corridas; Teste para a mais longa corrida de uns em um bloco; Teste de posto para matrizes binárias; Teste espectral para transformação discreta de Fourier; Teste de não sobreposição de padrão; Teste de sobreposição de padrão; Teste de estatística universal de Maurer; Teste de compressão de Lempel-Ziv; Teste de complexidade linear; Teste serial; Teste de entropia aproximado; Teste de somas cumulativas; Teste de excursões aleatórias; Teste variante de excursões aleatórias.

Este artigo irá apresentar os resultados obtidos pelo protótipo do software TACP (Teste de Aleatoriedade Criptográfica em Prompt de Comandos) que põe em prática os dois primeiros testes da bateria o de frequência (*monobit*) e o de frequência dentro de blocos.

Os Principais Conceitos da Criptografia

A Criptologia não é uma ciência nova. Enviar mensagens secretas é uma necessidade que surgiu nos tempos antigos. Desde o início das civilizações a humanidade travou grandes guerras. As mensagens enviadas pela frente de batalha não poderiam depender exclusivamente da destreza do mensageiro, pois sua captura pelo inimigo poderia tornar-se algo desastroso (Videira, 2011).

A necessidade de se transportar informações que mesmo interceptadas não pudessem ser compreendidas tornou-se um diferencial bélico de peso para os povos antigos, dessa necessidade surgiu à criptografia.

Criptografar nada mais é do que transformar um texto original em um código incompreensível pelos demais. Onde apenas as pessoas que possuem o segredo do código poderão interpretar a mensagem descobrindo o texto original (Videira, 2011).

Com o tempo a criptografia evoluiu junto com as descobertas matemáticas e o desenvolvimento de dispositivos que facilitassem a sua implementação, tanto no processo de se criar métodos de cifra quanto em métodos para se descobrir a mensagem sem conhecer o método (Videira, 2011).

Destes dispositivos, hoje se destaca, o computador. Devido a sua capacidade crescente de processamento.

A criptografia computacional é baseada em algoritmos criptográficos, alguns dos mais conhecidos são DES, AES, MD5 e RSA (Stamp, 2006).

Um algoritmo criptográfico é um conjunto de passos matemáticos que devem ser seguidos de forma sistemática para que os cálculos possam produzir um texto cifrado a partir de um texto claro. Este processo de cifrar textos claros faz uso de uma chave, de modo que, sem a chave correta, não deve ser possível recuperar o texto claro a partir do cifrado.

Quando se conhece o algoritmo criptográfico, a cifra gerada através do mesmo pode ser quebrada utilizando o método de força bruta, isto é, tenta-se todas as chaves possíveis até se descobrir qual fornece o texto claro correto.

Em um algoritmo criptográfico eficiente, deve se ter a certeza de que o espaço de chaves seja grande o suficiente para que seja inviável, mesmo com os recursos computacionais conhecidos e estimados, se testar todas as possíveis chaves em um tempo humanamente viável.

Conforme Stamp (2006), a criptoanálise é a técnica que tenta encontrar o texto claro sem conhecer a chave criptográfica. Quando um texto claro é cifrado por um algoritmo criptográfico com uma chave especifica um texto cifrado é gerado. Como tudo isso é feito de forma digital no final o texto cifrado se resume em um bloco de bits da mesma forma que o texto claro e a chave criptográfica. Comparando os blocos de bits se presume quais bits foram gerados a partir do cálculo e assim é possível fazer um algoritmo que poderá ser utilizado para decifrar mensagens interceptadas, mas não esta não é uma tarefa trivial.

Uma análise nos bits da mensagem necessita de diversos conhecimentos sobre operações binárias e técnicas matemáticas de criptoanálise. Quando se sabe os detalhes de como bits foram gerados é possível prever quais bits serão gerados a partir da análise de uma sequência conhecida. Com essa técnica se poderá gerar supostos textos claros a partir do texto cifrado, com um bom percentual de acerto. Com certas técnicas de criptoanálise também se pode definir um conjunto de chaves candidatas tornando a força bruta viável (Stamp, 2006).

Para dificultar o processo de criptoanálise existem os processos de difusão e de confusão. A

difusão acontece quando se faz com que cada dígito do texto claro afete vários dígitos do texto cifrado. Assim as estatísticas do texto claro serão dissipadas no texto cifrado. Já a confusão busca tornar a relação entre o texto cifrado e a chave criptográfica o mais complexa possível. Uma chave grande é usada contra a força bruta uma chave com um nível forte de operações de confusão e difusão dificultam o uso de técnicas de criptoanálise (Feistel, 1973).

Criptoanálise Diferencial e Linear

O uso de chaves grandes inviabilizam a força bruta e técnicas como a difusão e a confusão dificultam a criptoanálise. Como cada vez mais o tamanho das chaves vem crescendo, cifras atualizadas têm em média 128 bits, o ataque por força bruta se tornou impraticável (Stallings, 2008). A criptoanálise, apesar de mais complexa, é a melhor opção a ser explorada para se quebrar criptossistemas que fazem uso de chaves grandes, como as maiores ou iguais a 128 bits (Stallings, 2008).

Uma Análise nos bits da mensagem necessita de diversos conhecimentos sobre operações binárias e técnicas de criptoanálise. Quando se sabe os detalhes de como os bits foram gerados se pode prever quais bits serão gerados a partir da análise de outra sequência. Ou seja, com essa técnica poderá gerar supostos textos claros a partir do texto cifrado, com um bom percentual de acerto. Com a técnica de criptoanálise também se pode definir um conjunto de chaves candidatas tornando a força bruta viável.

Com isso a ênfase em cima do desenvolvimento e aprimoramentos de técnicas de criptoanálise está em alta. Para Stallings (2008) as mais promissoras são as técnicas de criptoanálise diferencial e linear. Estes tipos de técnicas são capazes de explorar as saídas não aleatórias geradas por um criptossistema ou uma função que faça parte dele.

As técnicas de ataque diferencial e linear podem ser usadas em qualquer ferramenta que tenha o objetivo de fazer um ataque a uma cifra de bloco (Stallings, 2008).

A falta de aleatoriedade na saída de criptossistemas permite que equações lineares e análises estatísticas sejam eficazes (Stinson, 2006).

Criptoanálise Diferencial

A criptoanálise diferencial tem como base a comparação de diferenças entre entradas e suas respectivas saídas (Stinson, 2006).

O ataque por criptoanálise diferencial foi o primeiro a conseguir quebrar o DES com menos de 255 criptografias (Stinson, 2006). Biham (1993) afirma que a criptoanálise diferencial é capaz de quebrar o DES com o uso da realização de aproximadamente 247 criptografias utilizando em cada uma um texto claro escolhido.

Criptoanálise Linear

A princípio esta técnica pode ser aplicada a qualquer cifra explorando suas partes não lineares. Os alvos são funções não aleatórias ou com um grau fraco de aleatoriedade. Ela considera como requisitos partes do texto cifrado juntamente com seus pedaços correspondentes em texto claro. Uma análise entre o texto cifrado e texto claro é feita para se tentar descobrir a chave K utilizada no processo de cifragem (Stinson, 2006).

Explorando os subconjuntos de bits do texto cifrado que foram formados a partir do texto claro de forma linear ou não aleatória. Uma aproximação linear é feita para calcular possíveis bits da chave K. De posse do que acredita-se ser pedaços da chave K se cria um conjunto de chaves candidatas que devem ser testadas até que se encontre a chave K (Stinson, 2006).

Se o algoritmo que gerou a cifra se comportar como um gerador de números pseudoaleatórios tais técnicas de criptoanálise serão frustradas.

Números Aleatórios

Segundo Stallings (2008) os algoritmos pseudoaleatórios estão presentes na composição de cifras e desempenham um papel importante nos esforços para dificultar a criptoanálise. Muitos protocolos criptográficos também necessitam de aleatoriedade ou pseudo-aleatoriedade em vários estágios, por exemplo, para gerar assinaturas digitais.

Geradores de Números Aleatórios - RNGs

Uma fonte verdadeira de aleatoriedade é não linear, suas saídas não podem ser representadas por equações.

No caso dos NRG a aleatoriedade é obtida através de um aspecto físico, que para ser usada na computação deve ser convertido em bits (Park, 2006).

Como exemplo pode-se considerar frequências sonoras geradas de forma arbitraria, ou pegar o endereço de memória que está sendo acessado no momento e etc.

Geradores de Números Pseudoaleatórios - PRNGs

Algoritmos que definem funções geradoras de números aleatórios são chamados de PRNGs.

Estas são funções matemáticas que devem gerar saídas diferentes para um mesmo valor de entrada (semente), que não deve ser repetida até que se esgotem todas as prováveis saídas.

Fazendo a análise do algoritmo gerador ou dos bits de saída pode-se prever qual o próximo valor será fornecido na saída. Por esse motivo números aleatórios gerados de forma computacional são chamados de

pseudoaleatórios (Park, 2006).

Quanto mais complexa a previsão maior será seu grau de aleatoriedade. Um bom grau de aleatoriedade torna a Análise de um PRNG impraticável (Stallings, 2008).

Para dificultar à aproximação linear a semente de um PRNG deve ser um número primo ou um produto de números primos. Outra opção é usar valores gerados por uma fonte física como semente, desde que a fonte tenha um bom nível de entropia (Stallings, 2008).

O NIST especificou uma bateria de testes estatísticos com o intuito de verificar se a saída produzida por um PRNG é confiável, mesmo para os fins da segurança da informação.

Testes Estatísticos do NIST

O NIST em publicação no Documento SP800-22 propôs uma bateria de 16 testes que comprovam se uma sequência binária atende aos parâmetros de uma sequência aleatória aceitável (Rukhin, 2008).

Segundo Rukhin (2008), os testes têm como objetivo avaliar as seguintes características presente em uma amostra binária:

Uniformidade: A qualquer ponto na geração de uma sequência de bits aleatórios ou pseudoaleatórios, a ocorrência de um zero ou um é igualmente provável, ou seja, a probabilidade de cada uma delas é exatamente 1/2. Por isso é esperado que a quantidade de zeros e uns fosse bem próximas. Se elas forem consideravelmente distantes a amostra não é aleatória.

Escalabilidade: Qualquer teste aplicável a uma sequência pode também ser aplicado a subsequências extraídas aleatoriamente. Se a sequência é aleatória, então qualquer subsequência extraída deve também ser aleatória. Assim, qualquer subsequência extraída deve passar por qualquer teste de aleatoriedade.

Consistência: O comportamento de um gerador deve ser consistente em valores usados como sementes do algoritmo pseudoaleatório.

Compressão: uma cadeia binária não é considerada aleatória se ela pode ser significativamente comprimida.

Os testes propostos pelo NIST determinam que uma amostra de bits pode ter duas hipóteses, a de que ela representa um sequência aleatória e a de que ela não representa. Para fins de comparação tem-se o chamado valor crítico, este determina o limite do resultado da amostra.

Para os testes descritos por esta bateria pode-se escolher entre 2 valores críticos, o resultado da amostra será chamado de P_valor. Segundo Rukhin (2008), os valores críticos para a comparação das amostras dos referidos testes são:

- O valor crítico de 0,001 indica que se espera que no máximo 1 amostra de cada 1000 amostras seja rejeitada. Para um valor P_valor ≥ 0,001, uma amostra seria considerada aleatória com uma confiança de 99,9%. Para um valor P_valor < 0,001, uma amostra seria considerada não aleatória com uma confiança de 99,9%.
- O valor crítico de 0,01 indica que se espera que no máximo que 1 amostra em 100 amostras poder ser rejeitada. Um valor P_Valor ≥ 0,01 significaria que a amostra seria considerado aleatório com uma confiança de 99%. A P_valor < 0,01 que significaria que a conclusão foi de que a amostra é não aleatória com uma confiança de 99%.

Tem-se que a amostra é um subconjunto de bits de uma sequência. Na aplicação proposta uma sequência será os bits originários de um arquivo gerado por alguma implementação do algoritmo que será testado.

Uma sequência de bits será considerada como não aleatória caso a média de amostras rejeitadas supere 1 em 100 para o valor crítico 0,01 e 1 em 1000 para o valor crítico 0,001. Esta é a probabilidade de erro de tipo 1, denominada α.

A probabilidade de erro de tipo 2 é denominada β. Está define que uma amostra pode parecer aleatória, mas na verdade pode ser um engano. β não possui um valor determinante como α e é muito mais complexo, pois cada amostra pode gerar resultados muito diferentes.

A necessidade da execução de todos os 16 testes é obtida para que possa determinar a inexistência de β, pois a probabilidade de uma sequência de bits aprovada em todos os 16 testes de α também não possuir a anomalia de tipo 2 é matematicamente aceitável (Rukhin, 2008).

A aplicação gerada para este estudo testa apenas a anomalia do tipo α, pois são implementados apenas os 2 primeiros testes. Os 2 testes implementados são descritos resumidamente a seguir e possuem como finalidade de testar as proporções de 1/2 de 0 e 1, que são utilizados para verificar a existência da uniformidade e a escalabilidade.

Frequência (*Monobit*)

O objetivo deste teste é analisar a proporção de zeros e uns em toda a amostra da sequência de bits para verificar se eles são distribuídos de forma uniforme. O propósito do teste é verificar se o número de zeros e uns na sequência ocorrem como seria esperado para uma sequência verdadeiramente aleatória, ou seja, o número de zeros e uns na sequência seriam aproximadamente iguais (Rukhin, 2008).

Se a amostra de bits for reprovada neste teste ela será considerada como não aleatória e não precisara ser submetida aos demais.

Frequência dentro de Blocos

Para a realização deste deve-se dividir a amostra em blocos de M bits onde M deve ser maior ou igual a

20. O objetivo deste teste é analisar a proporção de uns dentro de cada um dos blocos formados por M bits. Este teste verifica a escalabilidade de uma amostra (Rukhin, 2008).

O propósito do teste é determinar se a frequência de uns em cada um dos blocos de M bits é aproximadamente M/2 como seria esperado, caso ocorra aleatoriedade. Para blocos de tamanho M = 1, este teste degenera-se para o Teste 1, ou seja, Teste de Frequência(*Monobit*).

O sistema: TACP - Teste de Aleatoriedade Criptográfica em *Prompt* de Comandos

A implementação foi direcionada para a aplicação dos 2 primeiros testes, o de Frequência (*Monobit*) e o de Frequência dentro de blocos. Seu objetivo é de comprovar a eficiência dos testes através da exemplificação dos testes que testam um requisito fundamental da aleatoriedade, o da uniformidade, que define a necessidade de se existir uma proporção de aproximadamente ½ entre 0 e 1 em uma sequência.

Ela foi batizada com o nome de TACP - Teste de Aleatoriedade Criptográfica em *Prompt* de Comandos. E foi desenvolvido utilizando-se unicamente a linguagem de programação procedural C.

Para esta exemplificação foi construído 2 módulos, um que divide o arquivo em 100 amostras e outro que divide o arquivo em 1000 amostras.

Em ambos, cada amostra deve possuir no mínimo 100 bits. Por esse motivo a aplicação que trabalha com 1000 blocos necessita de um arquivo de no mínimo 10^5 bits o que corresponde a 12,5 Kbytes, e a que trabalha com 100 amostras necessita de um arquivo de no mínimo 1,25Kbyte.

O valor crítico escolhido foi o de 0.01, o que determina que seja aceitável que a reprovação das amostras chegue a 1%. Em 100 se aceita apenas que uma amostra seja rejeitada e em 1000 amostras pode-se ter um total de 10 amostras rejeitadas.

Para fins de ilustração o desvio dos P_valores das amostras vai ser calculado por uma média dos resultados. Um arquivo será considerado aprovado em um teste se ele não ultrapassar o limite de amostras rejeitadas e a média de P_valores das amostras extraídas no arquivo não for menor que o valor crítico.

Entrada de Dados

A entrada de dados é feita através de arquivos. Qualquer tipo arquivo, desde que tenha o tamanho mínimo exigido, pode ser submetido às avaliações.

O arquivo é aberto para leitura binária e todos os dados são gravados em um vetor de *BYTES*, um tipo de dado da linguagem C definido na biblioteca **windows.h**.

A função lerArquivo(**char** *nomeArquivo, BYTE *Cbit) recebe o nome do arquivo que deverá ser aberto e o local onde os bytes serão armazenados. Em caso positivo ela retorna a quantidade de bytes do arquivo se o arquivo não puder ser aberto corretamente ela retorna zero.

Teste de Frequência (Monobit)

Este teste é realizado por intermédio de duas funções:

double frequenciaMonoBit (BYTE *bloco, **double** N) ;

double DiffHamming(BYTE *Cbit, **unsigned long int** tam);

A função frequenciaMonoBit é responsável por realizar os cálculos referentes ao teste e receber os dado para realização do mesmo, que consistem na amostra de bits e no tamanho da amostra.

A função DiffHamming verifica todos os bits contidos nos bytes e calcula o chamado Peso de *Hamming*, que é a diferença entra zeros e uns da amostra de bits.

O P_valor da amostra é calculado a partir do resultado das operações efetuadas pela função frequenciaMonoBit. Para se calcular o P_valor o valor calculado é submetido à função **erfc** (*Complementary error function*) que é definida na biblioteca **math.h** pertencente ao padrão ANSI. Ao final da execução a função retorna o P_valor encontrado.

Teste de Frequência Dentro de Blocos

Este teste é calculado com 3 funções são elas:

double frequenciaDentroDeBloco(**unsigned long int** N, **unsigned long int** M, BYTE *bloco);

double PesoHamming(BYTE *Cbit, **unsigned long int** M, **unsigned long int** n, **long int** *Rb);

double Calc_X2obs(**unsigned long int** *UNS, **double** M, **double** n);

A primeira é responsável por receber os dados utilizados no teste que são a amostra de bits, o tamanho da amostra (N) e o tamanho de cada bloco em bits (M). Este teste divide a amostra em uma quantidade de blocos com a proporção definida por (N / M).

Os blocos são submetidos ao cálculo denominado Peso de *Hamming* que conta a quantidade de uns dentro de cada bloco.

Após a execução da função PesoHamming o cálculo referente ao teste continua. Devido a maior complexidade que envolve o cálculo do P_valor para este teste foi feita uma função específica para esta operação.

A função Calc_X2obs recebe como entrada os valores de uns encontradas em cada bolco, o tamanho de cada bloco em bits e a quantidade de blocos. E retorna o P_valor. Para o cálculo final do P_valor deve ser usada a função de controle **igamc** (*Incomplete Gamma function*).

Definição da realização dos testes

Os argumentos de entrada e saída submetidos às funções citadas anteriormente, a análise dos P_valores

gerados, as saídas de dados e a interatividade com o usuário estão definidas nos arquivos TACP100 e TACP1000, respectivamente uma define que os testes serão realizados com 100 amostras o outro com 1000 amostras.

Os parâmetros de teste estão definidos de forma estática. Foi defini em ambos os arquivos que o valor crítico será de 0,01, o tamanho do bloco em bits usada no teste de frequência dentro de blocos foi definido para trabalhar com 50 bits. O número de amostras rejeitadas não pode superar 1%. Para se usar o programa basta seguir a seguinte sintaxe no *prompt* de comando do MS-Windows:

TACP100 <*nome_do_arquivo*>

ou

TACP1000 <*nome_do_arquivo*>

A figura I apresenta a tele de execução do programa via Prompt de comandos do MS-Windows 7.

Recomenda-se o uso de arquivos com ao menos 1 Mbits para o TACP1000 e de 100 Kbits para o TACP100 pois segundo o documento do NIST SP800-22 em Rukhin (2008) amostras de 1000 bits de comprimento mais são desejáveis e melhoram a precisão do teste estatístico.

Causas dos erros comuns

Alguns erros podem ocorrer em decorrência do programa ou por causa e uma entrada incorreta. Abaixo estão listadas as possíveis mensagens de erro.

Arquivo não pode ser aberto: Neste caso ou o arquivo não existe ou está sendo indicado o um diretório diferente do local do arquivo. Outro motivo são permissões de acesso ao disco ou se o arquivo estiver corrompido.

Amostra com defeito tente outro arquivo: Neste caso a uma das amostras produzidas pelo programa não se encaixaram a função secundaria **gser** do arquivo **gamma.c** utilizada na função **igamc**. Por esse motivo o resultado passa a não ser mais confiável e a execução do programa é interrompida. Se estiver usando o TACP100 uma solução pode ser usar o TACP1000 e vice-versa.

Argumentos Incorretos: Se o usuário digitar mais do que um nome de arquivo ou nenhum nome de arquivo ao chamar TACP100 ou TACP1000 será exibida seguida de instruções de uso.

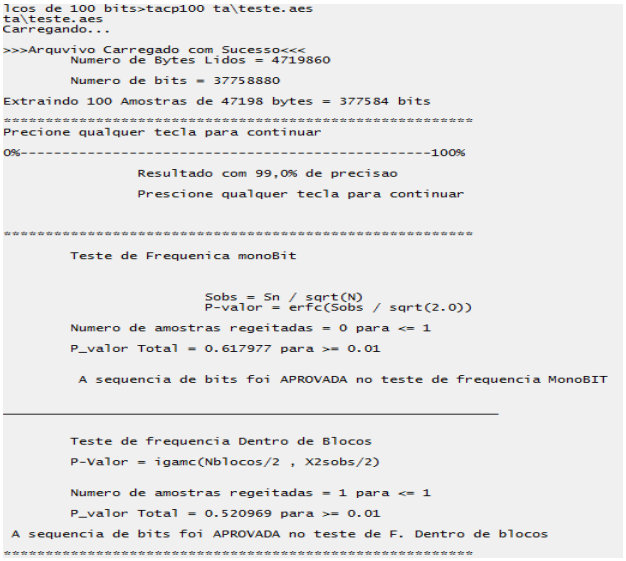

Figura I. Imagem da execução do TACP100.

Testes e Resultados

Foram feitos testes com os arquivos *Teste.txt* e *Avicii.mp3*. Para cada arquivo foi feito o teste com o arquivo original, arquivo compactado com programa *7zip*, Criptografado com *AESfree 2.7* e com um programa menos robusto chamado *Encrypt On Click* o qual não deixa claro qual é o algoritmo de criptografia utilizado.

Para a realização dos testes com o TACP foi utilizado um NoteBook CCE WN98 com chipset SIS 672 da VIA, processador Intel core2duo T5750 2,0 Ghz, 2GB de memoria RAM DDR2 667 Mhz. O Sistema Operacional Utilizado foi o MS-Windows 7 32 bits.

O teste de frequência dentro de blocos só será realizado caso as amostras sejam aprovadas no teste de frequência (*monobit*).

Uma amostra será considerada aprovada apenas se ela obter sucesso nos dois testes. Para que uma sequência de bits seja aprovada a média dos P_valores das amostras devem ser >= 0,01 e as amostras rejeitadas não podem superar 1% no caso do TACP100 as amostras rejeitadas devem ser <= 1 e para o TACP1000 <= 10.

A tabela I mostra os resultados para o TAC1000 e o TACP1000.

Nome do Arquivo	Amostras 100/1000	Tamanho em bits	Resultados Frequência. Monobit		Resultados Frequência dentro de. Blocos		Resultado
			Média P_valor >= 0,01	N. A. Rej.	M. P_valor	Números de Amostras Rejeitadas.	
Teste.txt	TACP100	1.491.560	0,001673	99	N/A	N/A	REP.
Teste.aes	TACP100	1.494.032	0,629803	01	0,484332	01	**APR.**
Teste.7z	TACP100	3760	Tamanho do arquivo insuficiente.				
Teste.EOC	TACP100	12.984	0,529598	14	N/A	N/A	REP.
Avicii.mp3	TACP100	63.423.896	0,003357	95	N/A	N/A	REP.
Avicii.aes	TACP100	63.209.224	Amostra inapta – problema na função gamma.c (gser)				
Avicii.7z	TACP100	62.846.016	Amostra inapta – problema na função gamma.c (gser)				
Avicii.EOC	TACP100	63.186.264	Amostra inapta – problema na função gamma.c (gser)				
Teste.txt	TACP1000	1.491.560	0,294514	19	N/A	N/A	REP.
Teste.aes	TACP1000	1.494.032	0,597069	1	0,499199	9	**APR.**
Teste.7z	TACP1000	3760	Tamanho do arquivo insuficiente.				
Teste.EOC	TACP1000	12.984	Tamanho do arquivo insuficiente.				
Avicii.mp3	TACP1000	63.423.896	0,094965	603	N/A	N/A	REP.
Avicii.aes	TACP1000	63.209.224	0,612949	0	0,498586	9	**APR.**
Avicii.7z	TACP1000	62.846.016	0,597846	0	0,489234	11	REP.
Avicii.EOC	TACP1000	63.186.264	0,611777	1	0,490810	12	REP.

TABELA 1 - RESULTADOS DA EXCUSSÃO DOS APLICATIVOS TACP100 E TACP1000

Trabalhos Futuros

Desde o principio, a proposta do trabalho contemplava a criação de um protótipo de verificador de gerador de números aleatórios, seguindo o proposto pelo NIST. Contudo, o protótipo pode ser melhorado em alguns aspectos:

Incorporar mais testes: programar os 14 testes restantes da suíte do NIST. Pois é necessário confirmar a presença dos quatro quesitos, uniformidade, consistência, escalabilidade e compressão, além de se reduzir ao máximo a probabilidade de uma anomalia do tipo β. E no protótipo apresentado só se verifica a uniformidade e escalabilidade.

Interface gráfica: por se tratar de um protótipo, sem fins lucrativos, não se contemplou a construção de uma interface mais amigável com o usuário. Em uma nova versão isso poderia ser incorporado.

Engenharia do sistema: todo produto de software precisa passar por critérios de construção que vão desde sua elaboração, até os testes de verificação e usabilidade. Numa versão mais incorporada do sistema, a aplicação destas técnicas da engenharia de software deve ser incorporada.

Conclusão

Devido à aleatoriedade comprovadamente trazer uma resistência considerável a algoritmos criptográficos contra a criptoanálise, o NIST definiu como padrão, o de que novos algoritmos devem se comportar de maneira aleatória.

A suíte de testes estatísticos definida no documento SP800-22 em Rukhin (2008) é capaz de verificar a aleatoriedade da saída de algoritmos e consegue garantir um nível de aceitação cientifica. No protótipo foi implementado apenas dois testes, porém a aprovação de apenas dois testes não garante um bom nível de

aleatoriedade.

A realização de todos os 16 testes é importante para se avaliar um algoritmo que seja candidato a se tornar uma cifra comercial. Porém para os propostos deste trabalho a realização dos testes de frequência (*Monobit*) e Frequência dentro de blocos, que testam a uniformidade e escalabilidade da sequência binária, foi suficiente para mostrar como a suíte do NIST é eficiente.

Verificou-se que apenas os arquivos gerados a partir de aplicações que utilização o AES foram aprovados nos dois testes, uma vez que o AES passou pela bateria de testes estatísticos do NIST com excelência. O que deixou claro sua imunidade a todas as técnicas de criptoanálise conhecidas, como a diferencial e a linear.

Os testes realizados com o TACP comprovaram que mesmo nos dois primeiros testes (simples, porém determinantes) já foi possível determinar a falta de aleatoriedade em todos os arquivos não criptografados com AES.

Sabe-se que uma das técnicas utilizadas para melhorar a segurança de uma cifra computacional qualquer é fazer a compressão do texto claro antes do processo de cifragem (Stinson, 2006). O TACP mostra que uma compressão possui, apesar de baixo, certa tendência à aleatoriedade. Um texto claro com tendência aleatoriedade dificulta bastante o trabalho do criptoanalista.

Sendo assim este artigo confirma que a realização dos testes estatísticos para a comprovação de aleatoriedade é muito importante durante o desenvolvimento de uma cifra ou de um gerador de números aleatórios e que a realização dos 16 testes é desejável em testes de robustez de qualquer gerador de números pseudoaleatórios que venham a ser usados na segurança da Informação.

Referências

BIHAM, E. e SHAMIR, A (1993). **Differential cryptanalysis of the data encryption standard**. Springer-Verlag / NewIork

FEISTEL, H. (1973). **Cryptography and computer privacy**. Sientific American, 228(5): 15-23.

PARK, S. e MILLER, K. **Random number generators: good ondes are hard to find**. C. Of the ACM.

RUKHIN, A. e SOTO, J. e NECHVATAL, J. e SMID, M. e BARKER, E. e LEIGH, S. e LEVENSON, M. e VANGEL, M. e BANKS, D. e HECKERT, A. Et. All. (2008). **A Statistical Test Suite for Random and Pseudorandom Number Generatiors for Cryptographic Applications.** NIST / U.S. Department of commerce.

STALLINGS, William (2008). **Criptografia e segurança de redes: princípios e práticas**. Pearson

Prentice Hall / São Paulo.

STAMP, Mark (2006). **Information Security: Principles and practice**. John wiley & Sons / New Jersey.

STINSON, Douglas R(2006). **Cryptography theory and practice**. Tylor & Francis Group / Broken Sound Parkway.

VIDEIRA, Antônio. Criptografia – síntese histórica. Revista da Armada, nº 371 – pág. 10. Disponível em: http://www.marinha.pt/extra/revista/, acessado no dia 28/03/2011.