# Segurança em Computação
# Trabalho Individual II

Gustavo Figueira Olegário

10 de abril de 2019

## 1 Números aleatórios

Para este trabalho, foram implementados os seguintes algoritmos de geração de números aleatórios: o ISAAC (cipher) e o Multiply-with-carry. Todo o trabalho foi implementado utilizando a linguagem Python. Optou-se essa linguagem, por ela, nativamente, não ter limite de número de dígitos para uma operação. Uma vez que esse trabalha solicitava números que utilizassem 2048 e 4096 bits, essa foi a *feature* decisiva para a escolha da linguagem.

Na tabela seguinte é feita uma comparação entre os dois algoritmos, mostrando quanto tempo foi necessário para gerar um número que utiliza uma quantidade X de bits para a sua respectiva representação binária. O tempo apresentado é utilizando como base o comando *time* do Linux na categoria *user*.

| Número de bits do dígito testado | ISAAC | Multiply-with-carry |
|---|---|---|
| 40 | 0.04s | 0.04s |
| 56 | 0.04s | 0.04s |
| 80 | 0.05s | 0.06s |
| 128 | 0.06s | 0.05s |
| 168 | 0.06s | 0.04s |
| 224 | 0.05s | 0.07s |
| 256 | 0.04s | 0.04s |
| 512 | 0.05s | 0.05s |
| 1024 | 0.06s | 0.06s |
| 2048 | 0.05s | 0.0.7s |
| 4096 | 0.06s | 0.12s |

Pela tabela, é possível perceber, que ambos os algoritmos, possuem praticamente o mesmo desempenho para gerar os números com a mesma quantidade de bits. Entretanto, quando compara-se a implementação entre ambos, é muito mais simples implementar o Multiply-with-carry do que o ISAAC.

Já no que se refere a complexidade, o ISAAC é um algoritmo que se baseia em uma seed previamente configurada , no caso a mensagem a ser cifrada, para gerar o número aleatório. Nesse caso, o algoritmo tentende a ter um comportamento próximo ao linear, em que quanto maior a mensagem, maior o tempo para executar o algoritmo, tendo assim um comportamento linear O(n). Já no algoritmo de Multiply-with-carry, não depende de uma entrada previamente configurada. Todos os parâmetros necessários são configurados pelo próprio algoritmo. Ele tem uma complexidade de O(n+k) onde n é a variável CMWC_CYCLE, que é indicada pelo algoritmo para ser usada com o valor de 4096, enquanto que k é número de vezes necessário para gerar o número com valor maior da variável CMWC_C_MAX

## 2 Números primos

Para verificar se um número é primo ou não, além de implementar o algoritmo de Miller Rabin, que foi solicitado, implementou-se o algoritmo de Teste de Primalidade de Fermat. Assim como o primeiro procedimento, ele é um teste estatístico. Caso o programa retorne falso, com certeza o número informado não é primo. Entretanto, caso retorne verdadeiro, há uma grande chance de ele ser primo, mas não garante que seja. Escolheu-se esse algoritmo, visto que era mais trivial de ser implementado do que os demais.

Para a comparação feita abaixo, utilizou-se o algoritmo do Multiply With Carry, para ficar gerando números aleatórios até que um desses fosse primo, e então, o algoritmo de verificação fizesse seu teste. Ou seja, o tempo abaixo exibido considera o tempo para gerar n números, em que pelo menos um deles seja primo.

| Número de bits do dígito testado | Miller Rabin | Fermat |
|---|---|---|
| 40 | 0.06s | 89.82s |
| 56 | 2.11s | 107.14s |
| 80 | 2.89s | 240.60s |
| 128 | 3.57s | 158.84s |
| 168 | 3.91s | 572.13s |
| 224 | 4.75s | 111.54s |
| 256 | 5.52s | 79.18s |
| 512 | 8.56s | 82.69s |
| 1024 | 0.72s | 152.99s |
| 2048 | 0.61s | 797.89s |
| 4096 | 0.15s | 294.01s |

O fato de gerar números primos aleatórios se dá por duas razões: a primeira dela são as operações realizadas. Os operadores de exponenciação e módulo, são muito caros computacionalmente, uma vez que envolvem ponto flutuante, o que torna a computação muito cara, em termos de processamento. Já a outra razão é o fato de ficar gerando número aleatórios indefinidamente até que um deles seja primo. Esse processo, por ser pseudo aleatório, faz com que o processo se torne mais lento.

Já no que se refere a complexidade dos algoritmos, o algoritmo de Fermat tende a ter uma complexidade O(n), onde n representa os n números escolhidos aleatoriamente no intervalo de 2 até k - 2, onde k é o número primo que está sendo avaliado. Já o algoritmo de Miller-Rabin, tende a ter um comportamento O(n+k), onde k é o número usado para testar a primalidade de n.

## 3 Código implementado

Todo o código utilizado neste trabalho está disponível no seguinte repositório do GitHub: https://github.com/olegario96/ine5429/tree/master/t2

## 4 Referências dos algoritmos

Algoritmo ISAAC Cipher: https://rosettacode.org/wiki/The_ISAAC_Cipher

Algoritmo Multiply-with-carry: https://en.wikipedia.org/wiki/Multiply-with-carry_pseudorandom_number_generator

Algoritmo Miller Rabin: https://www.youtube.com/watch?v=qdylJqXCDGs

Algoritmo de Teste de Primalidade de Fermat: https://en.wikipedia.org/wiki/Fermat_primality_test