Instituto Superior Técnico, Tagus Park

**Disciplina de Aplicações e Implementações de Algoritmos Criptográficos**

MERC: 1º ano, 2º semestre, 2012/13

65963 – David Dias Data: 22/03/2013, Sexta-feira

63520 – Fábio Domingos

AES – Implementação em Java

# **Abstract:** Implementação em Java do algoritmo de cifra AES e analise da sua performance face aos vários modos de cifra (ECB e CBC)

# Introdução

O AES(1998) é uma algoritmo de cifra simétrica baseado na cifra Rijandael, que vem para substituir o DES(1977). Os 4 passos fundamentais do AES são a Substituição de Bytes (SubBytes), troca de linhas (ShiftRows), mistura de colunas(MixColumns) e o XOR com a chave(AddRoundKey), especifica por cada Ronda.

Este documento apresenta a implementação desenvolvida em Java para chaves com 128 bits e para os modos de cifra de blocos ECB e CBC.

# Implementação

A implementação desenvolvida apresenta a seguinte interface:

Inicializar o instancia de cifra

***void init( int Cipher. int CMode, byte[] key);***

Cifrar/Decifrar parte da Mensagem

***byte[] update(byte[] plaintext);***

***byte[] update(byte[] plaintext, int inputlen);***

Finalizar a Cifra/Decifra

***byte[] doFinal(byte[] plaintext);***

***byte[] doFinal(byte[] plaintext, int inputlen);***

O mecanismo a ser usado é iniciado pela função *init.*

Os métodos *update()* têm como propósito permitir ao programa começar a cifra/decifra assim que começa a receber informação, não tendo que esperar por receber toda a mensagem para executar o processo. A cifra/decifra é terminada com o método *doFinal()*.

Para testar a aplicação podemos executar o programa da seguinte forma

***AES [–e/d] [–ecb/cbc/ctr] [–f Keyfile] [–k Key] [–i inputfile] [–o outputfile]***

**-e/d:** e(cifra)/d(decifra)

**–ecb/cbc/ctr:** cifra de blocos a usar

**–f Keyfile:** local com a chave

**–k Key:** chave

**–i inputfile:** ficheiro a cifrar/decifrar

**–o outputfile:** onde guardar a cifra/texto

Na implementação desenvolvida, de forma a aumentar o desempenho, foi optado por gerar as chaves de sessão a serem usadas pelas rondas do AES no inicialização da cifra, permitindo que estas sejam pré processadas, reduzindo consideravelmente o tempo de cifra a cada bloco.

# Analise de performance

Analisamos o desempenho da nossa implementação face a disponibilizada pela *API* do *Java* (*javax.crypto.\**) para avaliar a eficiência, podemos observar os resultados na **Figura 1**.

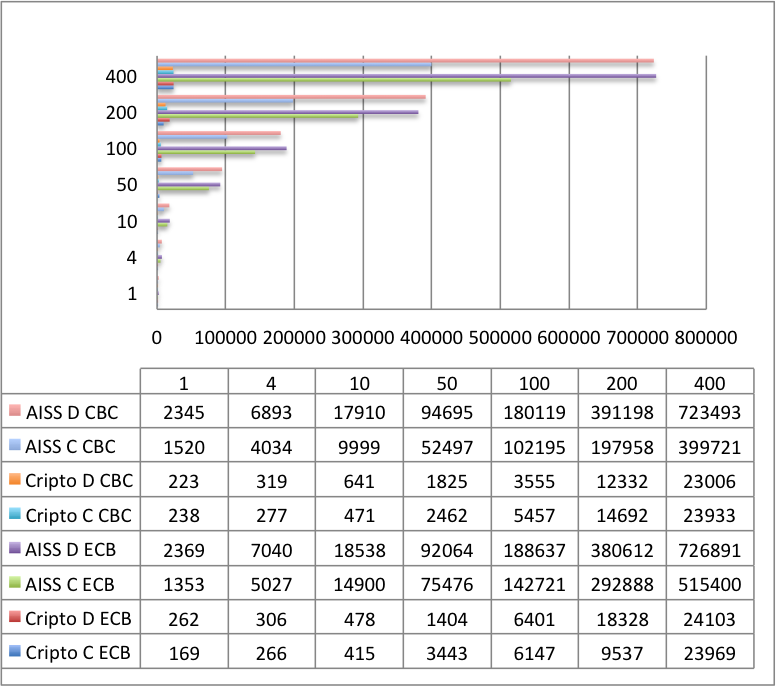


Figure Desempenho do AES AIAC vs AES Javax.crypto

Como podemos observar, há de facto uma grande discrepância na eficiência na cifra *AES* disponibilizada na *API* do *Java*, isto deve-se a ser uma cifra afinada em maior detalhe, ao contrário da nossa implementação oferecida, que tenta ser uma implementação académica para o estudo do funcionamento do *AES*.

# Trabalho futuro

De forma a aumentar o desempenho do nosso algoritmo, propomos 3 soluções

**Linguagem baixo nível -** Utilizar uma linguagem de mais baixo nível com capacidade de manipulação ao bit, como é o caso da linguagem *C*, que permite uma gestão e manipulação da memoria mais eficiência que reduziria o tempo gasto nas operações do *AES* (*SubBytes*, *ShiftRows*, *MixColumns*, *AddRoundKey*) que envolvem muita manipulação de dados.

**Pré Processamento para o *ECB* -** O modo de cifra *ECB* tem um defeito que pode ser usado para o aumento de performance, este é o facto de gerar a mesma cifra para o mesmo texto, ou seja, podemos pré processar muitas das combinações que são comuns surgir e indexar, deixando de ser necessário executar a cifra para os padrões que guardamos em memoria

***T-Box* –** As *T-Box* juntam os passos *SubBytes* e *ShiftRows* num único, tendo em conta que estes passos podem ser pré-processados e indexados de forma a que não seja necessário recalcular a mesma alteração várias vezes. Isto é extremamente útil uma vez que pode ser usado no *ECB* e *CBC*.

# Conclusões

Apesar de um algoritmo de cifra ser um processo pesado, existe uma grande quantidade de factores externos que influenciam muito o tempo de processamento, tal como a leitura e escrita do ficheiro que tem de ser feita em partes devido as limitações de memoria disponível do *runtime* que estamos a usar. O *AES* beneficia muito de pré processamento de alguns dos seus passos e podemos reduzir muito o tempo de cifra/decifra quando mais conseguirmos manipular a memoria a nosso favor, reduzindo o número de leituras e escritas no disco.