Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
SSC0747 - Engenharia de Segurança

# Trabalho 1 - Criptografia

Prof ª Kalinka Castelo Branco

Giovani Ortolani Barbosa - 8936648
Gustavo Lima Lopes - 8910142
Luciano Augusto Campagnoli da Silva - 9313367

São Carlos - SP
Maio de 2018

# 1. Estudo comparativo

## 1.1. Algoritmos Simétricos

A seguir apresentamos os algoritmos assimétricos escolhidos, eles são o RC4 e o DES.

### 1.1.1. RC4

O RC4 é um algoritmo simétrico de criptografia de fluxo com chave de tamanho variado criado por Ronald Rivest, um dos criadores do RSA. Tal criptografia é dita de fluxo pois os caracteres do texto plano são combinados através de um XOR *bitwise* com um caracter de um fluxo criptografado pseudo-aleatório (*keystream*) para gerar o texto cifrado.

Funciona através de permutações e somas de valores inteiros, o que torna este algoritmo muito simples e rápido, e por isso, foi bastante utilizado. Sua maior utilização se deu nos protocolos WEP e TLS.

Sua chave simétrica pode ter de 1 até 2048 bits (ou seja, até 256 caracteres) e deve ser compartilhada através de um canal seguro, utilizando um algoritmo assimétrico, por exemplo.

É um algoritmo considerado inseguro e seu uso não é aconselhado devido ao seu fraco sistema de *key-scheduling*, além disso já existem ataques conhecidos a esse tipo de criptografia.

**Funcionamento**

A permutação é iniciada em uma fase chama *Key-Scheduling Algorithm*:

1) Inicializar um vetor S de 256 posições e preenchê-lo com valores de 0 a 255 ordenadamente.
2) Iterar em cada elemento do vetor S e permutar os valores de acordo com o tamanho e os valores contidos na chave.

```
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap values of S[i] and S[j]
endfor
```

*Figura 1 - Permutação dos elementos de S de acordo com a chave e seu tamanho*

Ao final da operação o vetor S deve estar permutado de maneira aparentemente aleatória.

Após a *KSA* o *Pseudo-Random Generation Algorithm (PRGA)* realiza de fato a criptografia/decriptografia com o texto plano/texto cifrado.

Para cada caractere no texto plano/texto cifrado o *PRGA* calcula índices que irão referenciar o vetor S e a partir de algumas operações sobre os elementos e índices desse vetor utilizando aritmética modular, gera um caractere K que através da operação XOR com o próximo caractere do texto plano/cifrado produzirá o caractere cifrado/decifrado.

A operação *mod 256* utilizada nas 2 fases garante que o índice sempre referencie alguma posição do vetor S.

```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    K := S[(S[i] + S[j]) mod 256]
    output K
endwhile
```

*Figura 2 - Funcionamento geral do PKGA*

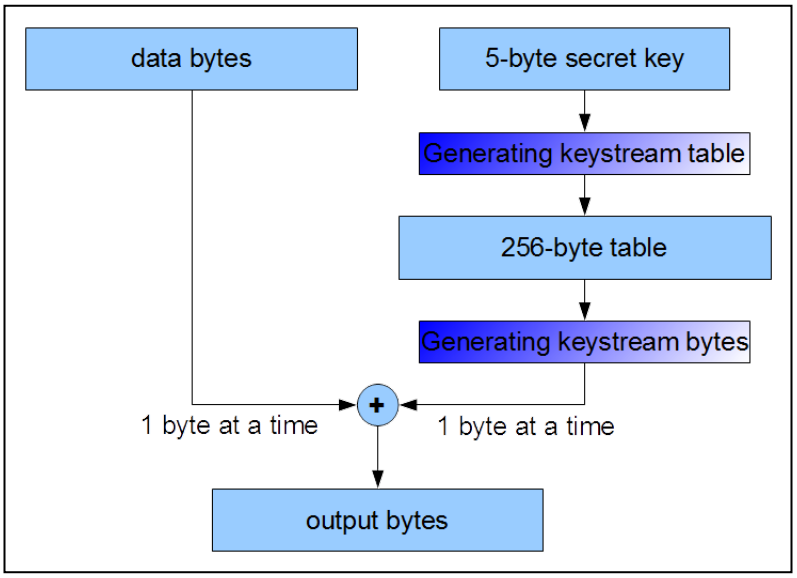



*Figura 3 - representação visual do RC4*
*(fonte: http://www.crypto-it.net/Images/rc4/algorithm_scheme_eng.png)*

### 1.1.2. DES

O Data Encryption Standard é um algoritmo de criptografia simétrica criado por uma equipe da IBM, baseado num algoritmo mais antigo, criado por Horst Feistel. Ele foi criado devido a necessidade por parte da NBS (atualmente conhecida como NIST) de uma criptografia padrão de alta segurança.

O DES criptografa mensagens com tamanho fixo de 64 bits. Para isso, utiliza-se uma chave de 64 bits (dos quais 8 são usados para verificação de paridade e somente os outros 56 são realmente usados para a criptografia).

No início e no fim do processo de criptografia do DES, é aplicada uma permutação dos bits através de uma tabela chamada IP (no início) e FP (no fim), sendo que FP é inversa a IP. Em termos de criptografia, tal permutação não faz sentido em termos de criptografia, sendo que tal processo só foi adicionado devido a problemas relacionados a tempo entre software e hardware nos anos 70 (quando o algoritmo foi criado).

Entre as permutações é onde ocorre o verdadeiro processo criptográfico. Após a permutação IP, os 64 bits da mensagem são divididos em dois blocos de 32 bits. A seguir, o mesmo processo é repetido 16 vezes: aplica-se a função feistel a um dos blocos e em seguida aplica-se um xor entre este ao segundo bloco. O resultado desta operação e o primeiro bloco são passados em frente para uma nova rodada do processo, trocando os blocos entre si. Apenas na última das 16 rodadas, os blocos não são trocados e se concatena os blocos e a permutação FP é feita, tendo como resultado a mensagem criptografada.

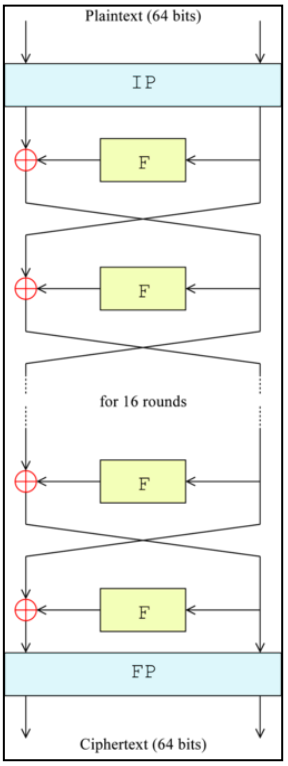



*Figura 4 - funcionamento geral do DES*

A função feistel consiste em pegar o bloco de 32 bits de entrada, fazer uma expansão deste para 48 bits (repetindo-se portanto alguns dos bits) e fazendo um xor deste com uma subchave de 48 bits (criada a partir da chave mencionada anteriormente, de 56 bits válidos). Em seguida, o resultado da operação é dividido em 8 blocos de 6 bits cada, os quais passam por 8 funções S (conhecidas como S-box), as quais utilizam uma transformação não-linear, identificada por uma tabela, para transformar cada um dos blocos de 6 bits em blocos de 4 bits. Vale ressaltar que esta parte é o ponto chave da segurança do DES, visto que todas as outras modificações são lineares, e consequentemente fáceis de quebrar. Para finalizar a função feistel, os 8 blocos de 4 bits são concatenados e seus bits são permutados de acordo com uma função fixa.

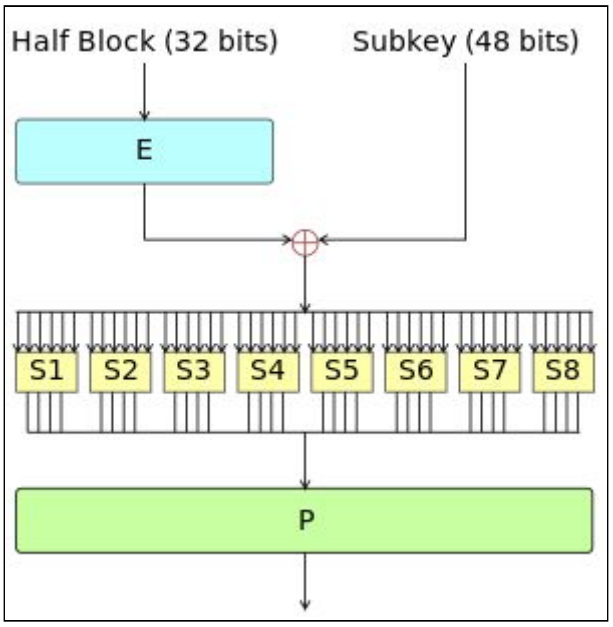



*Figura 5 - funcionamento da função de Feistel*

É importante observar que as subchaves utilizadas na função feistel são diferentes para cada iteração em que a função feistel é aplicada, ou seja, são geradas 16 subchaves distintas. Para gerar cada uma destas, pega-se os 56 bits válidos da chave original e divide-se estes em 2 blocos de 28 bits. Em seguida, durante 16 ciclos é feito um ou dois (quantidade varia dependendo do round) bitshift à esquerda em cada um dos dois blocos, e são selecionados 24 bits de cada um e concatenados, tendo como saída a subchave. Cada rodada de bitshift usa o resultado do bitshift da rodada anterior. A única diferença entre o processo de criptografia e descriptografia do DES é que na descriptografia as subchaves estão em ordem reversa as subchaves da criptografia.

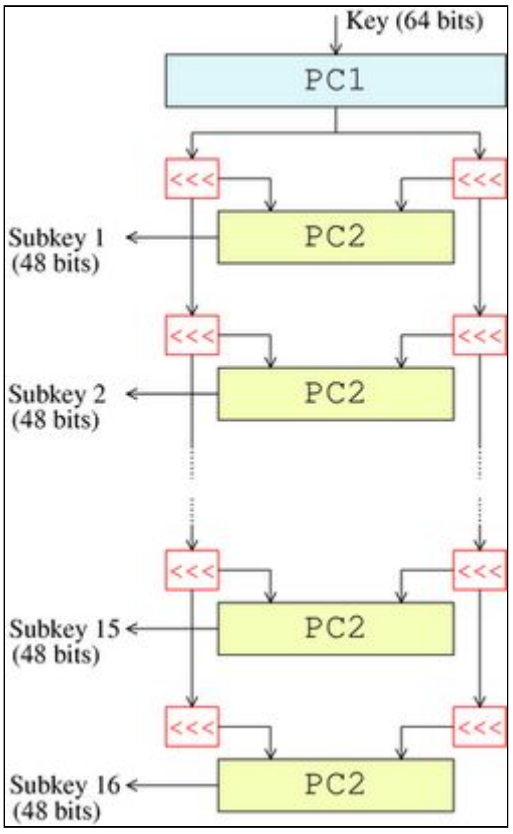



*Figura 6 - algoritmo de geração das subchaves*

Apesar de ser um algoritmo bastante usado no passado, o DES não é muito viável atualmente, visto que o DES é facilmente quebrável devido a evolução da capacidade de processamento dos computadores. Hoje em dia, normalmente são usados em criptografia de chave simétrica ou o triple DES (que é basicamente a repetição do processo DES três vezes com chaves distintas) ou o AES (que hoje é o algoritmo de criptografia simétrica padrão do NIST).

## 1.2. Algoritmos Assimétricos

A seguir apresentamos os algoritmos assimétricos escolhidos, eles são o RSA e o El Gamal.

### 1.2.1. RSA

O algoritmo de criptografia RSA é um algoritmo de criptografia assimétrico desenvolvido pelos pesquisadores Ron Rivest, Adi Shamir e Leonard Adleman, cujas iniciais dão a ele seu nome. Sua criação data de 1978, e sua segurança baseia-se na dificuldade do cálculo do inverso do logaritmo discreto. Nos parágrafos seguintes, segue a sua explanação.

Antes da criptografia propriamente dita, ocorre o processo de pré-codificação, no qual a mensagem é convertida para uma sequência numérica segundo um código (ASCII, *e.g*) e, posteriormente, decomposta em blocos numéricos. Tomando como exemplo a mensagem "Paraty é linda", define-se uma codificação para cada letra do alfabeto, como a seguinte:

| A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 |

Os espaços são codificados com o número 99. Dessa forma, a mensagem, após a codificação, torna-se:

25102710293499149921182313 10

Define-se dois números primos $n$ e $p$. Neste exemplo, utilize-se $n = 11$ e $p = 13$. Decompõe-se a sequência de números em blocos de tamanho menor que $np = 143$. Uma possível decomposição:

25-102-7-102-93-49-91-49-92-118-23-13-10

Feito a primeira etapa, inicia-se a criptografia propriamente dita. Calcula-se, primeiramente, a função totiente $\varphi(n) = (n-1)(p-1)$. Depois, procura-se um número $e$ que é primo com $\varphi(n)$ , ou seja, mdc($e$, $\varphi(n)$ ) = 1. Denomina-se o par ($n$, $e$) de chave de criptografia. Dado um bloco $b$ processo de *encrypting* consiste em fazer a operação $b^e$ mod n. A mensagem criptografada é a concatenação dos blocos criptografados na respectiva sequência inicial. Assim sendo, usando e = 7, temos:

64-119-6-119-102-36-130-36-27-79-23-117-10

Para decriptografar a mensagem, calcula-se *d*, o inverso modular de *e* em $\varphi(n)$, ou seja, *de* mod $\varphi(n) \equiv 1$. O par (n, d) é a chave para decriptografia, e o processo de *decrypting,* dado um bloco criptografado *a* é realizar a operação $a^d$ mod n para cada bloco e concatenar os resultantes para encontrar a mensagem original. A força das chaves está relacionada ao tamnaho dos primo *n* e *p.*

Vale ressaltar que a mensagem inicial deve ser menor ou igual a N, devendo ser quebrada e enviada ao destinatário por partes após a criptografia.

Mostra-se que o RSA funciona se a decriptação da mensagem encriptada resultar na original, ou seja, DC(b) = b. Para prová-lo, prova-se apenas que DC(b) ≡ b mod n. Como DC(b) quanto b estão no intervalo entre 1 e n -1, a única forma de serem congruentes módulo n é sendo iguais entre si.

Por definição:

$$DC(b) \equiv b^{ed} \bmod n \tag{1}$$

**d** é o inverso de **e** módulo $\varphi(n)$. Portanto, $ed = 1 + k\varphi(n)$ (2)

$$b^{ed} \equiv b^{1 + k\varphi(n)} \equiv (b^{\varphi(n)})^k b \bmod n \tag{3}$$

Substituindo a função totiente pela sua definição:

$$ed = 1 + k\varphi(n) = 1 + k(p - 1)(q - 1) \tag{4}$$

Prova-se primeiramente que $b^{ed} \equiv b \bmod p$. Como **p** é grande, pode-se afirmar:

$$b^{ed} \equiv b(b^{p-1})^{k(q-1)} \bmod p \tag{5}$$

Pelo Teorema de Fermat, pode-se afirmar que $b^{p-1} \equiv 1 \bmod p$, pois **p** é primo. Portanto, $b^{ed} \equiv b \bmod p$. Analogamente, pode-se provar também que $b^{ed} \equiv b \bmod q$. Portanto, $\mathbf{b}^{ed} - \mathbf{b}$ é divisível por **p** e por **q** e, consequentemente, por **n = pq**. Portanto, $b^{ed} \equiv b \bmod n$, como se desejava demonstrar.

## 1.2.2. El Gamal

Criado em 1984 pelo egípcio Taher Elgamal, o algoritmo baseia-se no processo de troca de chaves de Diffie-Helmann. Suponha-se dois indivíduos, Alice e Bob, que querem trocar um segredo entre si. Conhece-se, de uma base pública, a dupla (a, N). Para gerar as chaves de encriptação e decriptação, devem escolher, respectivamente, dois números aleatórios **r** e **s**. Alice calcula a chave:

$$K_{iA} = a^r \bmod N \quad (1)$$

, e Bob idem:

$$K_{iB} = a^s \bmod N \quad (2)$$

Para melhor compreensão, segue um exemplo. Suponha que Alice deseja enviar a mensagem **m** = 35 para Bob. Supondo a dupla (a, N) = (5, 89), e eles escolham, respectivamente, **r** = 8 e **s** = 13. Portanto:

$$K_{iA} = 5^8 \bmod 89 = 4$$

$$K_{iB} = 5^{13} \bmod 89 = 40$$

Eles trocam estas chaves intermediárias entre si. Como a descoberta de **r** e **s** recai ao problema do logaritmo discreto, dado N grande, é praticamente impossível um intruso descobrir estes valores privados. Alice obtém a chave de encriptação elevando o número enviado por Bob pelo seu privado:

$$K_A = K_{iB}{}^r \bmod N \quad (3)$$

Substituindo os valores:

$$K_A = 40^8 \bmod 89 = 16$$

A encriptação da mensagem se dá pelo produto modular entre a chave e a mensagem:

$$c = m\, K_A \bmod N \quad (4)$$

Substituindo:

$$c = 16 * 35 \bmod 89 = 26$$

Bob encontra a chave de decriptação realizando:

$$K_B = K_{iA}{}^{s} \bmod N \quad (5)$$

$$(K_{B\text{-}1}) = 40^{13} \bmod 89 = 16$$

Bob descobre que esta é a chave de encriptação usada por Alice, sem que ela fosse transmitida explicitamente. A de decriptação é o inverso modular de $K_B$:

$$K_B = (K_{B\text{-}1})^{-1} \quad (6)$$

$$K_B = 39$$

A decriptação se dá pelo produto:

$$m = c\, K_B \bmod N \quad (7)$$

$$m = 26*39 \bmod 89 = 35$$

Um adendo é feito: as chaves privadas **r** e **s**, por questões de segurança, são descartáveis: depois de usadas, trocam-se por uma nova na próxima comunicação. Por exemplo, se fossem novamente usadas, um intruso poderia realizar um *middle man attack* e interceptar tais informações.

## 1.3. Comparação de tempo entre os algoritmos

Em relação ao tempo de execução, os algoritmos simétricos são mais rápidos que os assimétricos, visto que esses últimos gastam mais processamento na questão do compartilhamento das chaves. Nesta seção, serão comparados os dois algoritmos simétricos descritos no documento (DES e RC4), idem para os assimétricos (RSA e ElGamal).

A seguir, segue um gráfico comparativo das velocidades de famosos algoritmos de criptografia simétricos:

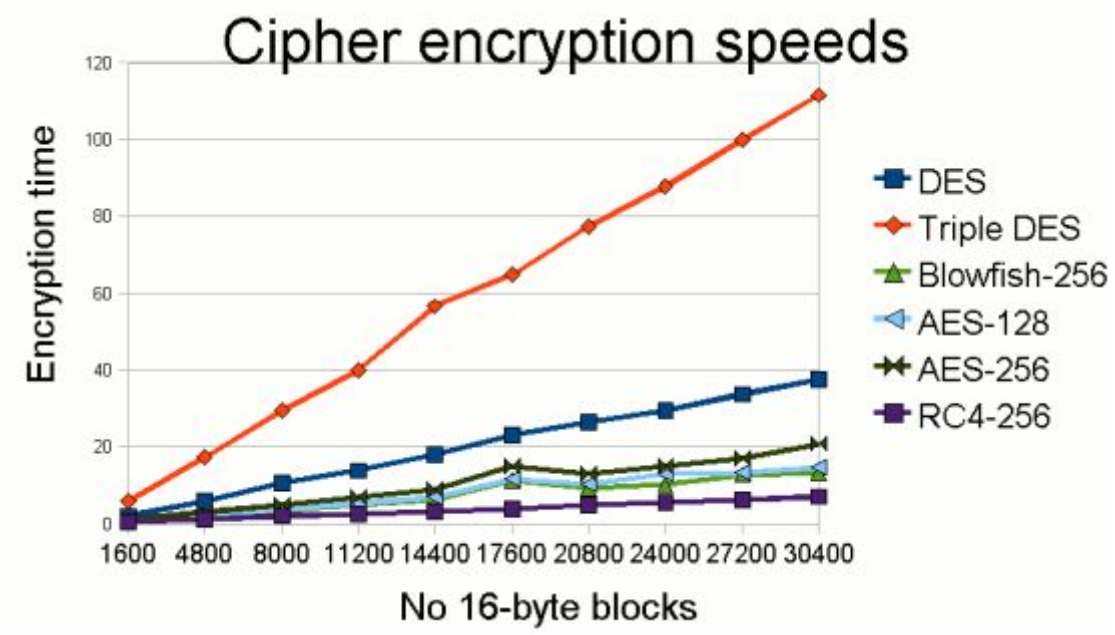



*Gráfico 1: Velocidades de encriptação entre algoritmos simétricos.*

Essa diferença de tempo de criptografia pode ser explicada pelo tempo extra que se demora com operações desnecessárias (em termos de segurança) no DES (por exemplo as permutações iniciais e finais), além da necessidade de geração de 16 diferentes subchaves e utilização destas em consecutivos processos de criptografia, enquanto que o RC4, apesar de ter loops para geração do *KSA* e do *PRGA,* possui menos processamento dentro destes.

Na referência [4], faz-se uma análise comparativa do RSA com o ElGamal. Os resultados são exibidos nas seguintes tabelas:

| Comprimento do texto | ElGamal | RSA |
|---|---|---|
| 18580 | 29083,63 | 3818,86 |
| 9242 | 132312,24 | 641,46 |
| 6095 | 8380,39 | 224,86 |
| 4680 | 6502,88 | 142,94 |
| 3739 | 5199,69 | 108,61 |
| 3209 | 4462,48 | 83,74 |
| 2762 | 3731,93 | 67,61 |
| 2524 | 3490,10 | 62,78 |
| 2247 | 3115,98 | 56,39 |
| 2083 | 2892,63 | 49,94 |

*Tabela 1: Tempos de execução de encriptação (em ms)*

| Comprimento do texto | ElGamal | RSA |
|---|---|---|
| 18580 | 111,95 | 162,42 |
| 9242 | 40,80 | 63,69 |
| 6095 | 23,88 | 40,57 |
| 4680 | 19,49 | 32,78 |
| 3739 | 16,94 | 28,08 |
| 3209 | 13,28 | 23,57 |
| 2762 | 12,11 | 19,25 |
| 2524 | 10,31 | 16,43 |
| 2247 | 9,44 | 14,95 |
| 2083 | 8,52 | 14,34 |

*Tabela 2: Tempo de execução de decriptação (em ms)*

| Comprimento do texto plano | ElGamal | RSA |
|---|---|---|
| 18580 | 3803,32 | 2013,01 |
| 9242 | 1216,92 | 332,15 |
| 6095 | 705,58 | 133,16 |
| 4680 | 526,08 | 94,38 |
| 3739 | 413,85 | 61,70 |
| 3209 | 348,01 | 50,90 |
| 2762 | 302,89 | 39,58 |
| 2524 | 277,07 | 32,60 |
| 2247 | 242,85 | 27,18 |
| 2083 | 224,32 | 27,05 |

*Tabela 3: Tempo de execução de assinaturas (em ms)*

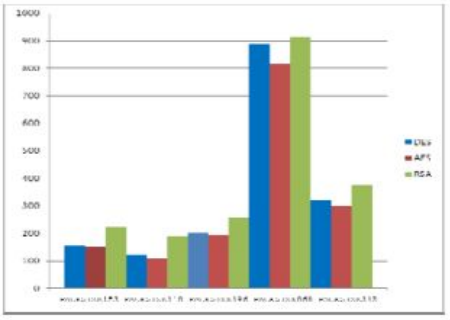
Verifica-se que o ElGamal é mais rápido para decriptação. No entanto, tanto para assinatura digital quanto para encriptação, o RSA é mais rápido (principalmente na encriptação). Isso é um dos motivos que contribui para que o RSA seja o algoritmo de criptografia assimétrica mais utilizado até os dias atuais. No geral, espera-se que o ElGamal seja mais lento, levando em conta as chaves intermediárias que ele cria, além de como mostrado na descrição do algoritmo neste relatório.

Quanto a comparação entre os algoritmos simétricos e os assimétricos, apesar de ser de conhecimento geral que os assimétricos são mais demorados (por causa da geração de chaves), pode-se confirmar tal informação através do gráfico a seguir (o qual compara o RSA, mais rápido entre os dois assimétricos aqui citados, e o DES, mais devagar entre os simétricos discutidos neste trabalho):

| S.NO | Algor | Pack Size (KB) | Encrypt Time (Sec) | Decrypt Time (Sec) | Buff Size |
|---|---|---|---|---|---|
| 1 | DES | 153 | 3.0 | 1 | 157 |
| | AES | | 1.6 | 1.1 | 152 |
| | RSA | | 7.3 | 4.9 | 222 |
| | | | | | |
| 2 | DES | 118 | 3.2 | 1.2 | 121 |
| | AES | | 1.7 | 1.2 | 110 |
| | RSA | | 10.0 | 5.0 | 188 |
| | | | | | |
| 3 | DES | 196 | 2.0 | 1.4 | 201 |
| | AES | | 1.7 | 1.24 | 200 |
| | RSA | | 8.5 | 5.9 | 257 |
| | | | | | |
| 4 | DES | 868 | 4.0 | 1.8 | 888 |
| | AES | | 2.0 | 1.2 | 889 |
| | RSA | | 8.2 | 5.1 | 934 |
| | | | | | |
| 5 | **DES** | 312 | 3.0 | 1.6 | 319 |
| | **AES** | | 1.8 | 1.3 | 300 |
| | **RSA** | | 7.8 | 5.1 | 416 |

algorithm show very minor difference in time taken for encryption and decryption process.

**Figure 5.** Comparative analysis of Buffer Size among DES, AES and RSA algorithm



By analyzing Figure 5 , it shows buffer size usages by AES, DES and RSA algorithm and noticed that RSA algorithm buffer size usages are highest for all sizes of document file.

# 2. Implementação do algoritmo

O algoritmo escolhido para ser implementado foi o RC4. Entretanto, também optou-se por incluir a inversão do texto plano (escrevê-lo de trás para frente) e o algoritmo ROT2 junto ao RC4, como tentativa de incluir um certo obscurecimento do texto plano.

O funcionamento do algoritmo consiste em:

**Encriptação**

1. Leitura dos arquivos com a chave e texto plano. Cada caractere lido é armazenado com sua representação de número inteiro em um lista.
2. Inversão do texto plano.
3. É realizada o obscurecimento pela ROT2.
4. É aplicada a criptografia RC4.
5. Escrita do texto cifrado em formato hexadecimal. Tal escolha deve-se ao fato da manipulação do formato hexadecimal ser mais fácil de ser realizada, pois todos os caracteres serão representado com 2 dígitos.

**Decriptação**

1. Leitura dos arquivos com a chave e texto cifrado. Cada caractere lido é armazenado com sua representação de número inteiro em um lista.
2. É aplicada a decriptografia RC4.

3. É realizado a rotação inversa pela ROT2.
4. Inversão do texto cifrado.
5. Escrita do texto plano em formato legível para o ser humano.

**Restrições**

Uma das grandes limitações do ROT é possibilidade de ser aplicado somente em caracteres alfabéticos. Sendo assim, não é possível realizar o obscurecimento de caracteres especiais e números.

A chave utilizada para o RC4 possui uma limitação máxima de 256 caracteres devido ao modo de operação da função KSA, a qual realiza 256 iterações e manipulação de apenas 256 elementos da chave, sendo quaisquer outros elementos ignorados.

## 2.1. Instruções

O algoritmo foi implementado utilizando o Python 3.5.2.

**Execução**

Os comandos abaixo devem ser executados na linha de comando dentro da pasta do código-fonte *rc4.py*.

- Encriptação:

```
$ python3 rc4.py 1 C <key_file.txt> <plain_text.txt> <cipher_text.txt>
```

- Decriptação:

```
$ python3 rc4.py 1 D <key_file.txt> <cipher_text.txt> <plain_text.txt>
```

# Referências

[1] SUZUKI, Jeff. el gamal encryption. Disponível em: <https://www.youtube.com/watch?v=pyirxbHuvOw>. Acessado em : 30/05/2018.

[2] ELGAMAL, Taher. A PUBLIC KEY CRYPTOSYSTEM AND A SIGNATURE SCHEME BASED ON DISCRETE LOGARITHMS. Disponível em: <https://people.csail.mit.edu/alinush/6.857-spring-2015/papers/elgamal.pdf>. Acessado em: 31/05/2018.

[3] Coutinho, S.C. Números inteiros e criptografia RSA. Rio de Janeiro, IMPA/SBM, 1997.

[4] Okeyinka A. E. Computational Speeds Analysis of RSA and ElGamal Algorithms on Text Data. Proceedings of the World Congress on Engineering and Computer Science 2015 Vol I. Disponível em: <http://www.iaeng.org/publication/WCECS2015/WCECS2015_pp115-118.pdf>. Acessado em: 01/06/2018.

[5]Hybrid Cryptographic Technique Using RSA Algorithm and Scheduling Concepts Disponível em: https://www.researchgate.net/publication/284467708_Hybrid_Cryptographic_Technique_Using_RSA_Algorithm_and_Scheduling_Concepts

[6] Computational Speeds Analysis of RSA and ElGamal Algorithms on Text Data Disponível em: http://www.iaeng.org/publication/WCECS2015/WCECS2015_pp115-118.pdf