UNIVERSIDADE PAULISTA - UNIP

AQUILES ROBERTO COUTINHO JUNIOR – N788CH-6
ARNON DAMASCENO NEVES DE SOUZA – N1047H-2
PATRÍCIA KARENINA ARRAIS MAIA – C975GE-9
RANYELLE EVELIN LISBOA BARBOSA – N10274-8
THOMAS SANTOS DIAS – N861DH-2

AS TÉCNICAS CRIPTOGRÁFICAS:
CONCEITOS, USOS E APLICAÇÕES.

SÃO PAULO - SP
2016

AQUILES ROBERTO COUTINHO JUNIOR – N788CH-6
ARNON DAMASCENO NEVES DE SOUZA – N1047H-2
PATRÍCIA KARENINA ARRAIS MAIA – C975GE-9
RANYELLE EVELIN LISBOA BARBOSA – N10274-8
THOMAS SANTOS DIAS – N861DH-2

AS TÉCNICAS CRIPTOGRÁFICAS:
CONCEITOS, USOS E APLICAÇÕES.

Atividade prática supervisionada e apresentada à graduação de Ciências da Computação, para fins de conhecimento na área.

Orientador: Prof. Ms. Roberto Eduardo Bruzetti Leminski.

SÃO PAULO - SP
2016

DEDICATÓRIA

Os autores do presente trabalho dedicam-no aos três sustentáculos de toda e qualquer aprendizagem humana: ao Criador, à Família e aos Mestres.

AGRADECIMENTOS

Este trabalho é fruto das lições dos Mestres e da colaboração de autores inspirados em aprender, aplicar e contribuir cientificamente.

Destarte, os integrantes desse projeto agradecem os ensinamentos transmitidos pelos três Mestres da turma do 2º semestre de 2016 do curso de Ciências da Computação da Universidade Paulista – UNIP – Unidade Paraíso:

Ao Professor Uanderson Celestino, pelo incentivo à elaboração de uma Atividade Prática Supervisionada que preze pela qualidade de seu conteúdo;

Ao Professor Roberto Eduardo Bruzetti Leminski, pela paciência em prover os instrumentais técnico-científicos necessários para a transformação do aprendizado obtido em conhecimento aplicado;

Ao Professor Celso Aurélio Tassinari, pelo olhar crítico e rígido que permitiu, ante as dificuldades que o trabalho impôs aos autores, o desenvolvimento, entre os integrantes dessa atividade, de uma habilidade fundamental em qualquer situação e trabalho complexos: resiliência.

“Uma criptografia robusta é capaz de resistir a uma aplicação ilimitada de violência. Nenhuma força repressora poderá resolver uma equação matemática”.

(Julian Assange)

RESUMO

Após o advento da escrita, constata-se a manifestação de uma nova necessidade na comunicação pós-ágrafa: a de segurança da privacidade de uma mensagem – no diálogo estabelecido entre emissor-receptor – a ponto de apenas o receptor desejado ter acesso ao conteúdo integral daquela.

Ademais, vislumbra-se que, no decorrer da evolução da tecnologia referente aos meios de comunicação, o sigilo e a integridade das mensagens continuam sendo bastante indispensáveis, visto que qualquer desvio ou modificação de uma comunicação privada podem ser altamente danosos em termos éticos, econômicos e estratégicos, por exemplo.

Nesse cenário, tornou-se imperioso o estudo da Criptografia, que lado a lado com os avanços tecnológicos, utiliza-se de conceitos e de ferramentas matemáticas, objetivando o estabelecimento de uma comunicação, entre grupos privados, livre de espionagens.

Este trabalho se propõe a avaliar os conceitos, os usos e as aplicações das principais técnicas criptográficas e a se ater à aplicação da criptografia *Triplo DES* a um caso específico, mediante a elaboração de um *software*, em linguagem C#, que imprimirá uma relevante mensagem, criptografada em 3 *DES*, para a solução do problema proposto.

ABSTRACT

After the advent of writing, there is a manifestation of a new need in post-graphic communication: for the security of the privacy of a message - in the dialogue established between sender-receiver - to the point that only the intended receiver has access to the integral content of it.

In addition, it is envisaged that, in the course of the evolution of media technology, the secrecy and integrity of the messages remain quite indispensable, since any deviation or modification of a private communication can be highly ethically, economically and strategically harmful, for example.

In this scenario, it became imperative to study Cryptography, which side by side with technological advances, uses concepts and mathematical tools, aiming to establish a communication, between private groups, free of espionage.

This work proposes to evaluate the concepts, uses and applications of the main cryptographic techniques and to stick to the application of Triple *DES* encryption to a specific case, through the elaboration of a software in C # language, that will print a relevant message – encrypted with 3*DES* ciphers – for the solution of the proposed problem.

Keywords: Cryptography; Software; Language C#

LISTA DE ILUSTRAÇÕES

ÍNDICE

1. OBJETIVO DO TRABALHO

O presente trabalho tem por objetivo geral o aprendizado de criptografia, mediante a aplicação de uma das conhecidas técnicas criptográficas em uma simulação de um problema, de viés socioambiental, que será relatado em ocasião futura.

Ademais, ante a compreensão de que, segundo O'CONNOR (2015, p. 28 – 29), há quatro (4) estágios sequenciais do aprendizado (1 – incompetência inconsciente; 2 – incompetência consciente; 3 – competência consciente; 4 – competência inconsciente), a APS (Atividade Prática Supervisionada) em questão apresenta o objetivo específico de proporcionar a cada um de seus integrantes a oportunidade de, gradativamente, aliar conhecimentos teóricos e a aplicação destes, com vista ao desenvolvimento e ao alcance do quarto estágio – supramencionado – do aprendizado, em que se constata o domínio e a excelência no assunto, mediante muita dedicação e muito esforço.

Por fim, cumpre ressaltar que o aprendizado em si, sem o intuito de servir a uma finalidade ética, perde seu teor valorativo. Assim sendo e levando em consideração que o mercado de trabalho brasileiro se encontra carente de profissionais de referência no campo da Criptografia e Segurança em Redes, o presente trabalho cumpre um terceiro objetivo: o de contribuir como material inicial para os interessados em adentrar nessa, ainda pouco explorada, área de especialização.

2. INTRODUÇÃO

A Atividade Prática Supervisionada (APS) do 2º semestre de 2016 estabeleceu a seguinte proposta de trabalho:

> O grupo de alunos deverá, através de fontes formais de informação, aplicar a utilização do conceito de criptografia num caso específico que envolve restrição de acesso a uma área contaminada ambientalmente que contenha riscos à Saúde Pública: um navio foi apreendido pela guarda costeira brasileira por transportar lixo tóxico da Ásia para a região norte do Brasil. O acesso à tripulação, assim como a todo conteúdo tóxico radiativo, deverá ser controlado. Somente inspetores devidamente trajados com roupas especiais poderão adentrar no navio. Por razões legislativas, o navio deve permanecer a uma distância segura: 50 km da costa e todo e qualquer contato deverá ser realizado por meio de helicópteros, para minimizar e restringir o contato. A área do entorno num raio de 10 quilômetros está isolada (UNIP, 2016, p.2).

Depreende-se, do trecho supracitado, que tal *caso* consiste na decodificação das mensagens criptografadas enviadas pela inspeção brasileira para a tripulação do navio asiático que, após acidente ambiental, encontra-se isolada e com contato restrito.

Ante a necessidade de que a logística de resgate seja bem implementada, mister se faz uma comunicação inequívoca, exata e eficaz, que pode ser estabelecida, com uma maior margem de segurança, a partir da utilização de uma técnica criptográfica.

Para a solução do problema de comunicação, a Guarda Costeira Brasileira se valerá do *Triplo DES (3DES)* – padrão criptográfico fundamentado em algoritmo de criptografia simétrica – cumprindo à tripulação do navio estrangeiro a tarefa de

decodificar a mensagem, para que a comunicação se estabeleça de forma plena e segura.

Mediante a utilização do padrão 3*DES*, um bloco de texto – mensagem, em texto puro, a ser transmitida para os tripulantes – é submetido ao processo de cifragem, resultando em um bloco de texto cifrado de igual tamanho (fixo e equivalente a 64 bits). Além disso, para o processamento dessa técnica criptográfica, observa-se a existência de três chaves: uma que criptografa a mensagem, outra que faz o processo de decodificar a mensagem criptografada pela chave anterior, e uma última que volta a criptografar o que foi obtido como resultado na operação antecedente.

A metodologia a qual esse trabalho científico se submete é de natureza dedutiva, visto que, a solução da situação ilustrada será obtida a partir do entendimento geral do *Triplo DES* para sua posterior aplicação no caso em análise.

As linhas a seguir pretendem alcançar os objetivos já mencionados e subdividem o trabalho nos seguintes tópicos:

- Capítulo I - Conceitos gerais de criptografia;
- Capítulo II - Técnicas criptográficas mais utilizadas;
- Capítulo III - Dissertação;
- Capítulo IV - Projeto (estrutura) do programa;
- Capítulo V - Relatório com as linhas de código;
- Referências Bibliográficas.

3. CONCEITOS GERAIS DE CRIPTOGRAFIA

Após o advento da escrita, a comunicação entre os grupos populacionais saiu do nível meramente oral – em que se observava a importância da memória para a retenção de informações – para o nível da grafia – cuja relevância, em comparação aos povos ágrafos, consiste numa maior retenção de informações a longo prazo.

Subsequentemente, com a escrita, nasceram os imperativos de privacidade e de segurança das mensagens escritas trocadas entre pessoas ou grupos. Inclusive, cumpre ressaltar que, ao longo da história da humanidade, a manutenção da escrita, de forma sigilosa, foi fundamental para o emprego de táticas e de estratégias imprevisíveis por parte de grupos político-econômicos, militares, religiosos, dentre outros.

A respeito disso, TANENBAUM e WETHERALL (2003, p. 771) tecem as seguintes considerações:

> Historicamente, quatro grupos de pessoas utilizaram e contribuíram para a arte da criptografia: os militares, os diplomatas, as pessoas que gostam de guardar memórias e os amantes. Dentre eles, os militares tiveram o papel mais importante e definiram as bases para a tecnologia. Dentro das organizações militares, tradicionalmente as mensagens a serem criptografadas eram entregues a auxiliares mal remunerados que se encarregavam de criptografá-las e transmiti-las. O grande volume de mensagens impedia que esse trabalho fosse feito por alguns poucos especialistas.

Esses autores ainda acrescentam que a criptografia deu um salto qualitativo, em termos de segurança e de privacidade, a partir do surgimento dos computadores.

No tocante ao *conceito de criptografia*, BRAGA e DAHAB (2015, p.3) o ilustra da seguinte forma:

> A Criptografia (do grego kryptos, significando oculto) é a ciência que se dedica ao estudo e ao desenvolvimento das técnicas (matemáticas) utilizadas para tornar uma mensagem secreta. Historicamente, o verbo

> criptografar tem sido usado apenas nesse sentido. Entretanto, a criptografia moderna possui funções como assinaturas digitais, resumo (hash) criptográfico e outras, que não se limitam a prover sigilo da informação. De fato, como veremos a seguir, a palavra Criptografia denota hoje um conjunto de técnicas matemáticas das quais uma grande parte dos requisitos, mecanismos e serviços de segurança da informação não podem prescindir.

Ademais, para a melhor compreensão dos estudos sobre criptografia, é imprescindível o aprendizado dos seguintes *termos-chave*:

a) Texto plano/puro: diz respeito aos dados que ainda não passaram pelo processo de encriptação, ou seja, são os dados não encriptados;

b) Chave (Key): refere-se aos dados utilizados para encriptar um texto plano, ou decriptar um texto cifrado;

c) Algoritmo de Criptografia: remete ao método matemático empregado para encriptar/decriptar dados com o uso das chaves de criptografia.

Ressalta-se, também, a importância em se diferenciar *cifras* de *códigos*. Na linguagem leiga, é comum confundir os conceitos. TANENBAUM e WETHERALL (2003, p. 771) apontam que a *cifra* atua no nível sintático, isto é, na representação da mensagem, *bit* a *bit* ou *caractere* por *caractere*. Já o *código* age no nível do significado, manipulando-o geralmente pela simples substituição de frases ou palavras.

Já no que concerne aos *princípios básicos de criptografia*, os principais são:

1) Encriptação (ou cifragem): trata-se da conversão, por pessoas não-autorizadas, de dados legíveis para um formato ilegível (texto cifrado) usando uma chave e um algoritmo criptográfico;

2) Proteção da criptografia: a criptografia tem por objetivo resguardar a privacidade no momento em que uma pessoa armazena dados ou troca informações com outras pessoas;

3) Decriptação: trata-se do processo inverso da encriptação, isto é, a alteração de dados encriptados (ilegíveis) para dados legíveis, valendo-se de uma chave e de um algoritmo criptográfico (cifra). Ressalta-se, aqui, que o receptor de uma mensagem encriptada pode decriptá-la e ler seu comando, no formato original.

Importante, também, se faz mencionar o conceito de *criptoanálise*. Este compreende o processo de transformação de dados cifrados (encriptados) em dados legíveis (decriptados) sem que se conheça a chave de encriptação.

STALLINGS (2015, p.23) aponta a existência de duas técnicas gerais para o ataque a um esquema de encriptação convencional:

> Criptoanálise: os ataques criptoanalíticos utilizam-se da natureza do algoritmo e, talvez de mais algum conhecimento das características comuns ao texto claro, ou ainda de algumas amostras de pares de texto claro-texto-cifrado. Este tipo de ataque explora as características do algoritmo para tentar deduzir um texto claro específico ou a chave utilizada. Ataque por força bruta: o atacante testa todas as chaves possíveis em um trecho do texto cifrado, até obter uma tradução inteligível para o texto claro. Na média, metade de todas as chaves possíveis precisam ser experimentadas para então se obter sucesso.

Acrescenta-se, aqui, que são quatro os *requerimentos de segurança da comunicação*:

a) Autenticação – esse princípio assegura que a mensagem foi realmente originada pelo remetente, e não por outra pessoa;

b) Integridade – diz respeito ao conteúdo das informações trocadas entre transmissor e receptor. Deve-se garantir que o conteúdo da mensagem chegue íntegro a seu destino, ou seja, que não seja alterado de nenhuma forma no meio do caminho;

c) Confidencialidade – certifica que a informação não se encontra disponível para pessoas e processos que não tenham autorização para acessá-la e utilizá-la.

d) Não – repúdio – evita que uma mensagem, após ter sido enviada, seja repudiada pelo transmissor. Em outras palavras, o transmissor não poderá negar que transmitiu a mensagem. Exemplo: assinaturas digitais.

Por fim, quanto aos tipos de criptografia, estes são, fundamentalmente, em número de três e dois deles se classificam em função da chave de cifragem (ou encriptação):

a) Criptografia de Chave Privada ou Simétrica – nesse tipo, há a utilização de uma única chave. O emissor usa esse tipo de chave para encriptar (cifrar) a mensagem, e o receptor utiliza a mesma chave para decriptá-la. Em outras palavras, trata-se de uma chave compartilhada (*shared key*) e, em razão disso, considera-se como uma técnica de criptografia simétrica, muito usada no modo *stream*. Algoritmos conhecidos de criptografia simétrica são: *DES, 3DES, Blowfish, AES, OTP (One-Time-Pad).*

Imagem 01 – Criptografia Simétrica.

Fonte: http://biblioo.info/certificacao-digital/

b) Criptografia de Chave Pública ou Assimétrica – nesse tipo, há o uso de duas chaves distintas, de modo a se obter comunicação segura mediante canais de comunicação duvidosos. É assimétrica, em razão do uso de um par de chaves diferentes: nela, cada participante possui uma chave pública e uma chave privada. A chave privada é secreta, e só o proprietário a conhece, ao passo que a chave pública é compartilhada com todos os interessados na comunicação. Ambas as chaves (pública e privada) têm o

tamanho variando entre 512 e 2048 bits. Esse tipo de criptografia é utilizado em modo de bloco. Algoritmos conhecidos de criptografia assimétrica são: DSA, RSA, GPG.

Imagem 02 – Criptografia Assimétrica.

Fonte: http://biblioo.info/certificacao-digital/

c) Funções de Hash – a técnica de *Hash* não se vale de uma chave como as técnicas descritas acima. Ela emprega um valor de *hash* sobre o texto plano. Esse tipo de criptografia é usado para averiguar a integridade dos dados, objetivando assegurar que estes não tenham sido desprevenidamente modificados (verificações de senha também podem se valer das funções de *Hash*). Atualmente, alguns exemplos de funções de *Hash* são: MD5, SHA1, SHA2.

4. TÉCNICAS CRIPTOGRÁFICAS MAIS UTILIZADAS

No capítulo anterior, constatou-se que as técnicas de criptografia se subdividem em três categorias: chave simétrica, chave assimétrica e funções de *Hash*.

Ademais, acrescenta STALLINGS (2015, p.25) que, de todas as técnicas de encriptação existentes, os seus dois blocos de montagem básicos são: *substituição e transposição*.

Sobre o conceito de *técnica de substituição*, STALLINGS (2015, p.25) tece as seguintes considerações:

> Uma técnica de substituição é aquela em que as letras do texto claro são substituídas por outras letras, números ou símbolos. Se o texto claro for visto como uma sequência de bits, então a substituição envolve trocar padrões de bits de texto claro por padrões de bits de texto cifrado.

O exemplo mais antigo de *cifra de substituição*, que se tem conhecimento, é a chamada *Cifra de César* – cifra esta que se atribui ao governante romano Júlio César. A lógica existente em tal cifra é a de que cada letra do alfabeto seja substituída por outra que se encontre três posições a frente (mas o valor desse salto, isto é, a chave secreta pode ser para mais ou para menos).

A seguir, as representações do alfabeto em texto claro (letra minúscula) e em cifras (letra maiúscula):

texto claro: a b c d e f g h i j k l m n o p q r s t u v w x y z

texto em cifra: D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

Um exemplo prático da aplicação da *Cifra de César* é o que consta abaixo:

texto claro: seja feliz na escolha

texto em cifra: VHMD IHOLC QD HVFROKD

No tocante às *cifras de transposição*, STALLINGS (2015, p.37) compreende que se trata de uma distinta espécie de mapeamento, que se obtém a partir de algum tipo de permutação entre as letras do texto claro.

Um exemplo de uma técnica simples de transposição é a *cerca de trilho*. Nesta, o texto é escrito em uma série de diagonais e lido como uma série de linhas.

Assim, por exemplo, para cifrar, com uma cerca de trilho de profundidade 2, a mensagem “*estudem para as provas finais*”, escreve-se:

e t d m a a s r v s i a s
s u e p r a p o a f n i

Ao final do processo acima, obtém-se a seguinte mensagem encriptada:

ETDMAASRVSIASSUEPRAPOAFNI

Outro exemplo é a *cifra de transposição de colunas*. Sobre esta técnica, TANENBAUM e WETHERALL (2003, p.775) explicam:

> A cifra se baseia em uma chave que é uma palavra ou frase que não contém letras repetidas (...). O objetivo da chave é numerar as colunas de modo que a coluna 1 fique abaixo da letra da chave mais próxima do início do alfabeto e assim por diante. O texto simples é escrito horizontalmente, em linhas. O texto cifrado é lido em colunas, a partir da coluna cuja letra da chave seja a mais baixa.

Assim, supondo que se objetive a encriptação de uma mensagem com a palavra PEDRA, essencial é que essa chave se submeta a três regras:

1) o número de letras deverá ser IGUAL ao número de colunas da mensagem;

2) não poderá haver letras iguais;

3) ordena-se alfabeticamente a chave e numeram-se as letras.

A título de ilustração da explicação acima, o seguinte exemplo:

P	E	D	R	A
4	3	2	5	1
e	s	t	u	d
e	m	p	a	r
a	a	s	p	r
o	v	a	s	f
i	n	a	i	s

Nesse exemplo, conclui-se que o texto simples é estudemparaasprovasfinais e que o texto cifrado é DRRFSTPSAASMAVNEEAOIUAPSI.

A seguir, algumas operações de cifra de blocos, utilizadas em chaves privadas e públicas, apresentadas por STALLINGS (2015, p.136- 150):

a) *Electronic Code Book* (*ECB*) – codifica-se cada bloco de *bits* de texto claro independentemente usando a mesma chave. Sua aplicação típica é na transmissão segura de valores isolados (por exemplo, uma chave de encriptação);

b) *Cipher block chaining (CBC)* – a entrada do algoritmo de encriptação é o XOR dos próximos 64 *bits* de texto claro e os 64 *bits* anteriores de texto cifrado. Sua aplicação é na transmissão de uso geral orientada a bloco e em autenticação;

c) *Cipher feedback* (*CFB*) – a entrada é processada *s bits* de cada vez. O texto cifrado anterior é usado como entrada para o algoritmo de encriptação a fim de produzir saída pseudoaleatória, que é aplicada a um *XOR* com o texto claro para criar a próxima unidade de texto cifrado. Sua aplicação típica é na transmissão de uso geral orientada a fluxo e em autenticação;

d) *Output feedback (OFB)* – semelhante ao CFB, exceto que a entrada do algoritmo de encriptação é a saída *DES* anterior, e são usados blocos completos. Sua aplicação típica é na transmissão orientada a fluxo por canal com ruído (por exemplo, comunicação por satélite);

e) *Counter (CTR)* – Cada bloco de texto claro é aplicado a um *XOR* com um contador encriptado. O contador é incrementado para cada bloco subsequente. Sua aplicação típica é na transmissão orientada a bloco de uso geral. É útil para requisitos de alta velocidade.

No tocante às chaves do tipo simétrica, três se destacam – *Data Encryption Standard, Data Encryption Standard Triplo* e Advanced Encryption Standard:

1) *Data Encryption Standard* (*DES*) – trata-se de uma técnica criptográfica desenvolvida pela gigante da tecnologia IBM, que foi empregada no governo

norteamericano como padrão oficial para as informações de caráter não confidencial. A respeito do *DES*, TANNENBAUM e WETHERALL (2003, p. 555) ensinam:

> A cifra, DES (Data Encryption Standard — padrão de criptografia de dados), foi amplamente adotada pelo setor de informática para uso em produtos de segurança. Em sua forma original, ela já não é mais segura; no entanto, em uma forma modificada ela ainda é útil. O algoritmo, parametrizado por uma chave de 56 bits, tem 19 estágios distintos. O primeiro deles é uma transposição independente da chave no texto simples de 64 bits. O último estágio é exatamente o inverso dessa transposição. O penúltimo estágio troca os 32 bits mais à esquerda pelos 32 bits mais à direita. Os 16 estados restantes são funcionalmente idênticos, mas são parametrizados por diferentes funções da chave. O algoritmo foi projetado para permitir que a decodificação fosse feita com a mesma chave da codificação, uma propriedade necessária em qualquer algoritmo de chave simétrica. As etapas são simplesmente executadas na ordem inversa.

2) *Data Encryption Standard Triplo* (*3 DES)* – trata-se de uma melhoramento da técnica anterior (DES). Sobre o *3 DES*, TANNENBAUM e WETHERALL (2003, p. 557) tecem as seguintes considerações:

> No início de 1979, a IBM percebeu que o tamanho da mensagem *DES* era muito pequeno e criou uma forma de aumentá-lo usando a criptografia tripla (Tuchman, 1979). O método escolhido, que desde então foi incorporado ao padrão internacional 8732. Nesse caso, são usados três estágios e duas chaves. No primeiro estágio, o texto simples é criptografado com K1 da maneira usual do DES. No segundo estágio, o DES é executado no modo de descriptografia, com o uso de K2 como chave. Por fim, outra criptografia é feita com K1.

3) *Advanced Encryption Standard (AES)* – ainda que com o avanço do *3 DES* em relação ao *DES*, o *NIST* (Órgão do Departamento de Comércio dos Estados Unidos) determinou que o governo necessitava de um novo padrão criptográfico para utilização não confidencial. Após um concurso envolvendo especialistas e

pesquisadores de todo o mundo, o padrão Rijndael venceu. Sobre esse padrão, TANNENBAUM e WETHERALL (2003, p. 558) comentam:

> O Rijndael admite tamanhos de chaves e tamanhos de blocos desde 128 bits até 256 bits em intervalos de 32 bits. O comprimento da chave e o do bloco pode ser escolhido independentemente. Porém, o AES especifica que o tamanho do bloco deve ser 128 bits e o comprimento da chave deve ser 128, 192 ou 256 bits. É pouco provável que alguém utilize chaves de 192 bits; assim, de fato, o AES tem duas variantes: um bloco de 128 bits com uma chave de 128 bits e um bloco de 128 bits com uma chave de 256 bits (...). Sob uma perspectiva matemática, o Rijndael se baseia na teoria de campo de Galois, o que proporciona ao algoritmo algumas propriedades de segurança demonstráveis. Porém, ele também pode ser visto como código em C, sem a necessidade de entrarmos nos detalhes matemáticos. Como o *DES*, o Rijndael utiliza substituição e permutações, e também emprega várias rodadas. O número de rodadas depende do tamanho da chave e do tamanho do bloco, sendo 10 para chaves de 128 bits com blocos de 128 bits, passando para 14 no caso da maior chave ou do maior bloco. No entanto, diferente do *DES*, todas as operações envolvem *bytes* inteiros, a fim de permitir implementações eficientes, tanto em *hardware* quanto em *software*.

Por outro lado, no que diz respeito às chaves do tipo assimétrica, duas serão citadas – *Rivest Shamir Adleman* e Algoritmo da Mochila:

1) *Rivest Shamir Adleman (RSA)* – sobre essa técnica, TANNENBAUM e WETHERALL (2003, p. 566) lecionam:

> Um método muito interessante foi descoberto por um grupo de pesquisadores do *MIT* (Rivest et al., 1978) e é conhecido pelas iniciais dos três estudiosos que o criaram (Rivest, Shamir, Adleman): *RSA*. Ele sobreviveu a todas as tentativas de rompimento por mais de um quarto de século e é considerado um algoritmo muito forte. Grande parte da segurança prática se baseia nele. Sua principal desvantagem é exigir chaves de pelo menos 1024 bits para manter um bom nível de segurança (em comparação com 128 bits para os algoritmos de chave simétrica), e isso o torna bastante lento. O método RSA se baseia em alguns princípios

> da teoria dos números. Para criptografar a mensagem P, calcule C = P e (mod n). Para descriptografar C, calcule P = C d (mod n). É possível provar que, para todo P na faixa especificada, as funções de criptografia e descriptografia são inversas entre si. Para realizar a criptografia, você precisa de e e n, enquanto para a descriptografia, são necessários d e n. Portanto, a chave pública consiste no par (e, n) e a chave privada consiste em (d, n).

2) Algoritmo da Mochila – a respeito dessa técnica de chave assimétrica, TANNENBAUM e WETHERALL (2003, p. 567) explanam:

> Apesar de ser amplamente utilizado, o *RSA* não é de forma alguma o único algoritmo de chave pública conhecido. O primeiro algoritmo de chave pública foi o algoritmo da mochila (Merkle e Hellman, 1978). A ideia aqui é que alguém possui um grande número de objetos, cada objeto com um peso diferente. O dono dos objetos codifica a mensagem selecionando secretamente um subconjunto dos objetos e colocando-os na mochila. O peso total dos objetos contidos na mochila torna-se público, e o mesmo acontece com a lista de todos os objetos possíveis. A lista de objetos contidos na mochila é mantida em segredo. Com outras restrições específicas, o problema de descobrir uma lista de objetos possíveis com o peso fornecido foi considerado computacionalmente inviável e formou a base do algoritmo de chave pública. O inventor do algoritmo, Ralph Merkle, estava bastante seguro de que esse algoritmo não poderia ser decifrado; portanto, ofereceu um prêmio de 100 dólares a quem conseguisse fazê-lo. Adi Shamir (o "S" do *RSA*) se prontificou a decifrar o algoritmo e ganhou o prêmio. Indignado, Merkle reforçou o algoritmo e ofereceu um prêmio de 1.000 dólares a quem pudesse decifrá-lo. Ronald Rivest (o "R" do *RSA*) também conseguiu decifrar o novo algoritmo e ganhou o prêmio. Merkle não ousou oferecer 10.000 dólares pela nova versão revisada; portanto, "A" (Leonard Adleman) não teve sorte. Apesar de ter sido refeito, o algoritmo da mochila não é mais considerado seguro e não é usado.

5. DISSERTAÇÃO SOBRE O TRIPLO *DES*

5.1. ESTRUTURAÇÃO, CONCEITOS E FUNDAMENTAÇÃO

O Triple *DES* se revelou como uma das alternativas em potencial para vulnerabilidade do *DES*. O mesmo afirma STALLINGS (2015, p. 137) a seguir:

> Dada a potencial vulnerabilidade do *DES* a um ataque de força bruta, tem havido um grande interesse na descoberta de uma alternativa. Uma técnica é projetar um algoritmo completamente novo, e o *AES* é o exemplo principal. Outra alternativa, que preservaria o investimento existente em *software* e equipamento, é usar encriptação múltipla com *DES* e múltiplas chaves.

No que diz respeito à encriptação múltipla do *DES,* sua forma mais elementar é o *Double DES*, que consiste em duas etapas de encriptação e duas chaves. Sobre essa técnica, STALLINGS (2015, p. 137) explica:

> Dado o texto claro *P* e duas chaves de encriptação K_1 *e* K_2, o texto cifrado C é gerado como: $C = E(K_2, E(K_1,P))$. A decriptação exige que as chaves sejam aplicadas em ordem reversa: $P = D(K_1, D(K_2, C))$. Para o *DES*, esse esquema aparentemente envolve um tamanho de chave de 56 x 2 = 112 *bits*, o que resultaria em um aumento incrível na força criptográfica.

Outra forma de encriptação múltipla é a *Triple DES* com duas chaves. Ela foi proposta por Tuchman e se fundamenta na sequência encriptar – decriptar – encriptar, de acordo com as seguintes equações:

$$C = E(K_1, D(K_2, E(K_1, P)))$$
$$P = D(K_1, E(K_2, D(K_1, C)))$$
$$C = E(K_1, D(K_1, E(K_1, P))) = E(K_1, P)$$
$$P = D(K_1, E(K_1, D(K_1, C))) = D(K_1, C)$$

Sobre o *Triple DES* com duas chaves, STALLINGS (2015, p. 139) aponta que ele se trata de uma alternativa popular ao DES e que sua utilização se dá nos padrões de gerenciamento de chave ANSI X9.17 e ISO 8732.

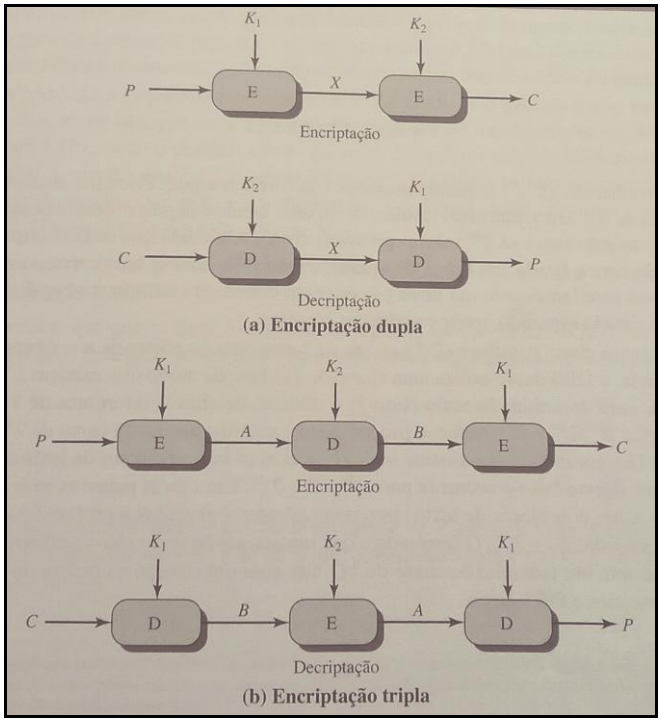
Uma terceira forma de encriptação múltipla é a *Triple DES* com três chaves e sobre esta STALLINGS (2015, p. 141) ensina:

> Qualquer um usando o 3 *DES* com duas chaves poderá sentir certa preocupação com os ataques criptoanalíticos. Assim, muitos pesquisadores agora consideram o 3 *DES* com três chaves a alternativa preferida (por exemplo, [KALI96a]). O *3 DES* com três chaves possui um tamanho de chave efetivo de 168 *bits* e é definido da seguinte maneira:
>
> $$C = E(K_3,D(K_2,E(K_1,P)))$$
>
> A compatibilidade com o *DES* é fornecida colocando-se $K_3 = K_2$ ou $K_1 = K_2$.

Ainda sobre o *3 DES*, CAVALCANTI, DESTRO explicam:

> O Triple DES utiliza três chaves para realizar o seu processamento de criptografia, a primeira chave criptografa a mensagem, a segunda chave decifra a mensagem criptografada pela primeira chave e a terceira chave torna a criptografar o resultado da operação anterior.

Imagem 03 – Encriptação Múltipla



Fonte: STALLINGS, William. *Criptografia e segurança de redes: princípios e práticas*,6.ed., p. 137

5.2. BENEFÍCIOS EM RELAÇÃO ÀS TÉCNICAS ANTERIORES

Os benefícios que o *3 DES* oferece é que nele se encontra uma fácil implementação em comparação com outros algoritmos: ele é baseado no *DES* e, mesmo apresentando a falha de poder ser quebrado pela força bruta, é mais rápido em comparação com métodos que utilizam chaves públicas. Acrescenta-se, ainda, que o *3 DES* possui uma segurança maior que o *DES*, o que o torna uma melhor opção em termos de privacidade de dados.

Ademais, o *3 DES* é muito útil para a criptografia de arquivos de dados pessoais e tem a vantagem de utilizar bem menos recursos do computador do que outras formas de criptografia. Esta técnica criptográfica pode também ser, relativamente, considerada em termos de segurança.

Por fim, em razão de gerar chaves diferentes, outro benefício é que ela pode ser compartilhada com pessoas distintas (no caso se uma das chaves que forem compartilhadas for descoberta, somente um usuário será atingido, preservando assim a integridade dos demais).

5.3. APLICAÇÕES QUE FAZEM/FIZERAM USO DA TÉCNICA

Sobre a utilização do *DES* e do *3 DES*, CAVALCANTI, DESTRO apontam:

> Um dos algoritmos de cifra simétrica moderno mais utilizado é o *Data Encryption Standard* (*DES*). Este algoritmo foi padronizado pelo *FIPS* (*Federal Information Processing Standard*) no ano de 1976 como padrão americano. O *DES* foi muito utilizado até meados dos anos 1990, quando a velocidade de processamento dos computadores aumentou tanto que a quebra da criptografia *DES* ficou mais fácil de ser realizada e, por falta de segurança, este padrão começou a ser substituído pelo padrão *Advanced Encryption Standard*. Porém o *DES* não foi totalmente abandonado e hoje é utilizado em sua forma conhecida como *Triple DES*, mais segura, para operações bancárias.

Cumpre ressaltar que, no passado, o 3 *DES* era implementado em *hardware* e tinha seu uso limitado à cifragem de dados não confidenciais, tendo sido bastante utilizado por órgãos governamentais de caráter não-militar e não-diplomático.

Exemplos de aplicações, fundamentadas na Internet, que utilizam o 3DES com três chaves são o PGP e o S/MIME.

5.4. DISCUSSÃO COMPARATIVA

A seguir, será realizada uma discussão comparativa entre a técnica criptográfica *Triplo DES* e as técnicas *DES, AES e Blowfish.*

O *Triplo DES* se diferencia do *DES* em termos de aperfeiçoamento nos níveis de segurança. Trata-se de igual algoritmo, contudo, no *Triplo DES,* a técnica criptográfica é aplicada triplamente, otimizando os níveis de segurança. O ponto negativo de ambas as técnicas é que possuem baixa performance no processamento, sendo, portanto bastante lentos, principalmente em softwares.

Corrobora com esse entendimento, a seguinte comparação entre *3 DES e DES* realizada por STALLINGS, BROWN (2014, p. 117-118):

> O *3 DES* tem dois atrativos (...) o primeiro é que, com o seu comprimento de chave de 168 bits, ele supera a vulnerabilidade do *DES* ao ataque de força bruta. O segundo é que o algoritmo de cifração subjacente ao *3DES* é o mesmo que no *DES*. Esse algoritmo foi submetido a mais escrutínio do que qualquer outro algoritmo de cifração por um período de tempo mais longo e nenhum ataque criptoanalítico efetivo baseado no algoritmo, a não ser o de força bruta, foi encontrado. Desse modo, há um alto nível de confiança de que o *3 DES* é muito resistente à criptoanálise. Se a segurança fosse a única consideração, o 3DES seria uma escolha apropriada para um algoritmo criptográfico padronizado ainda por muitas décadas.
>
> A principal desvantagem do *3 DES* é que o algoritmo é relativamente lento em *software.* O *DES* original foi projetado para a implementação em

> *hardware,* em meados da década de 1970, e não leva a um código em software eficiente. O *3 DES*, que requer três vezes mais cálculos, é correspondentemente mais lento. Uma desvantagem secundária é que ambos, o *DES* e o *3 DES*, usam um tamanho de bloco de 64 *bits*. Por questão tanto de eficiência quanto de segurança, um tamanho de bloco maior é desejável.

Já o *AES* foi criado pelo *Instituto Nacional de Norma e Tecnologia* (*NIST*) com o objetivo de substituir o *DES*. Trata-se de uma técnica criptográfica usada em muitos órgãos do governo norte-americano, por ser eficiente tanto em *hardware*, quanto em *software*. A desvantagem do AES é que, apesar da segurança que ele confere, é passível do ataque de força bruta, que consiste na técnica criptoanalítica de inúmeras tentativas de combinações até que o código seja aceito.

Sobre o *AES*, STALLINGS, BROWN (2014, p.118) complementam:

> Por causa dessas desvantagens, o *3 DES* não é um candidato razoável para utilização a longo prazo. Como substituto, o NIST publicou em 1997 uma chamada de propostas para um novo *Advanced Encryption Standard* (*AES*), que deveria ter um nível de segurança igual ou maior do que o do *3 DES* e eficiência significativamente melhor. Além desses requisitos gerais, o *NIST* especificou que o *AES* deveria ser uma cifra de bloco simétrica com comprimento de bloco de 128 *bits* e suporte para comprimento de chaves de 128, 192 e 256 *bits*. Os critérios de avaliação incluíam segurança, eficiência computacional, requisitos de memória, adequabilidade de *hardware* e de *software* e flexibilidade.

No tocante à técnica criptográfica *Blowfish*, ela foi fruto do trabalho de Bruce Schneier, presidente da *Counterpane Systems*, empresa especializada em criptografia e segurança. A *Blowfish* se caracteriza por ser a cifra de chave secreta que se vale de um número de *bits* variando entre 16 e 448 *bits*. Tal técnica é eficiente em determinadas plataformas de software e permite que dados sejam criptografados 16 vezes, tornando sua decifragem bastante difícil a um hacker.

Acrescenta-se ainda que, em conformidade com o artigo dos autores do *site* http://www.brighthub.com/computing/smb-security/articles/75099.aspx, é possível estabelecer o seguinte quadro comparativo entre as quatro técnicas de criptografia em análise:

Imagem 04 - Quadro comparativo entre *DES*, *3DES*, *AES*(256-bit) e *Blowfish*.

Algorítmo	Tamanho de dados	Tempo (segundos)	MB/Segundo Médio	Performance
DES	256 MB	10 - 11	22 - 23	Baixa
3DES	256 MB	12	12	Baixa
AES(256-bit)	256 MB	5	51,2	Média
Blowfish	256 MB	3,5 - 4	64	Alta

Fonte: http://www.brighthub.com/computing/smb-security/articles/75099.aspx

Depreende-se da interpretação desse quadro comparativo que:

a) a Blowfish possui o melhor desempenho;

b) a *AES* possui um desempenho satisfatório;

c) a *DES e a 3 DES* gastam o dobro do tempo da *AES*, apresentando baixas performances.

d) *Blowfish* e *AES* podem ser consideradas mais seguras que *DES* e *3 DES*.

5.5. VULNERABILIDADES E FALHAS

No que se refere à segurança do 3*DES*, RODRIGUES (2016, p. 114-116) aponta:

> Inicialmente, o *DES Triplo (3DES)* foi projetado para usar três chaves de criptografia, por isso o risco de ataque torna-se impraticável atualmente e até mesmo em um futuro distante. Porém, essa característica pode trazer a desvantagem de exigir um tamanho de chave 56 x 3 = 168 *bits,* o que pode ser muito pesado para a realização do processo(...) Atualmente, não existem ataques criptoanalíticos práticos contra o *3DES*, já que um ataque de força bruta no *DES* é da ordem de 2^{112}, aproximadamente (5×10^{33}). Desta forma, o *3 DES* aumenta a segurança triplicando o tempo de quebra de uma chave, por exemplo, se o atacante leva 24 horas para quebrar a chave usando o *DES duplo*, passará a gastar 72 horas para quebrar três

> chaves. Outro ponto forte é descobrir se realmente foi quebrada a chave de criptografia, pois para descobrir a primeira chave, o atacante terá que descobrir as outras duas chaves e, em seguida, verificar se ambas combinam com a primeira. Portanto, torna-se exaustiva a quebra completa da chave do *3 DES*.

Tal entendimento mostra que o nível de segurança ante ao ataque criptoanalítico é alto, uma vez que o *3 DES* apresenta uma tripla chave criptográfica.

Por outro lado, no que diz respeito ao processamento, RODRIGUES (2016, p. 117) assevera:

> Apesar de ser muito seguro, o *3 DES* é considerado lento, pois leva bastante tempo para criptografar e descriptografar (...). Outra desvantagem é o tamanho dos blocos, que são de 64*bits*, por motivos de eficiência e segurança, blocos maiores são mais desejados.

Assim, apesar do bom nível de segurança oferecido pelo *3 DES*, necessária se tornou a criação de criptografias de bloco com melhores níveis de processamento. Em razão disso, empresas e organizações optaram pelo desenvolvimento de algoritmos com alta velocidade com chaves bem maiores e modulares em relação às do *Triple DES*.

5.6. MELHORIAS PROPOSTAS OU IMPLEMENTADAS

Como explanado anteriormente, em razão do tamanho de chave ser triplo no *3 DES*, o processo de criptografia e descriptografia é muito lento – característica essa bastante indesejável em termos tecnológicos.

Algumas tentativas para sanar o problema foram propostas, como afirma RODRIGUES (2016, p. 115) a seguir:

> Como alternativa, Tuchman propôs um método de criptografia triplo, usando apenas duas chaves, conforme função especificada na fórmula abaixo:

$$C = E (K1,D (K2,E(K1, P)))$$

> Essa função deverá seguir uma determinada sequência: criptografar/ decriptografar/ criptografar (...). O *3 DES* com duas chaves é uma alternativa relativamente popular ao *DES* e tem sido adotado no uso dos padrões ISO 8732 e ANSX9.17.

No entanto, muitos pesquisadores se opõem ao 3 *DES* com duas chaves, como aponta RODRIGUES (2016, p. 116):

> Muitos pesquisadores acham que o 3DES com três chaves é a melhor alternativa, pois possui 168 *bits* de chave de criptografia. Pode ser representado da seguinte maneira:

$$C = E (K_1, D (K_2,E (K_1, P)))$$

Diante desse panorama, em que se constata, *no 3 DES*, o conflito existente entre as variáveis segurança e processamento, a necessidade de ambas conduziu ao desenvolvimento de técnicas criptográficas mais seguras e com melhor performance, como se obteve com o *AES – Advanced Encryption Standard*.

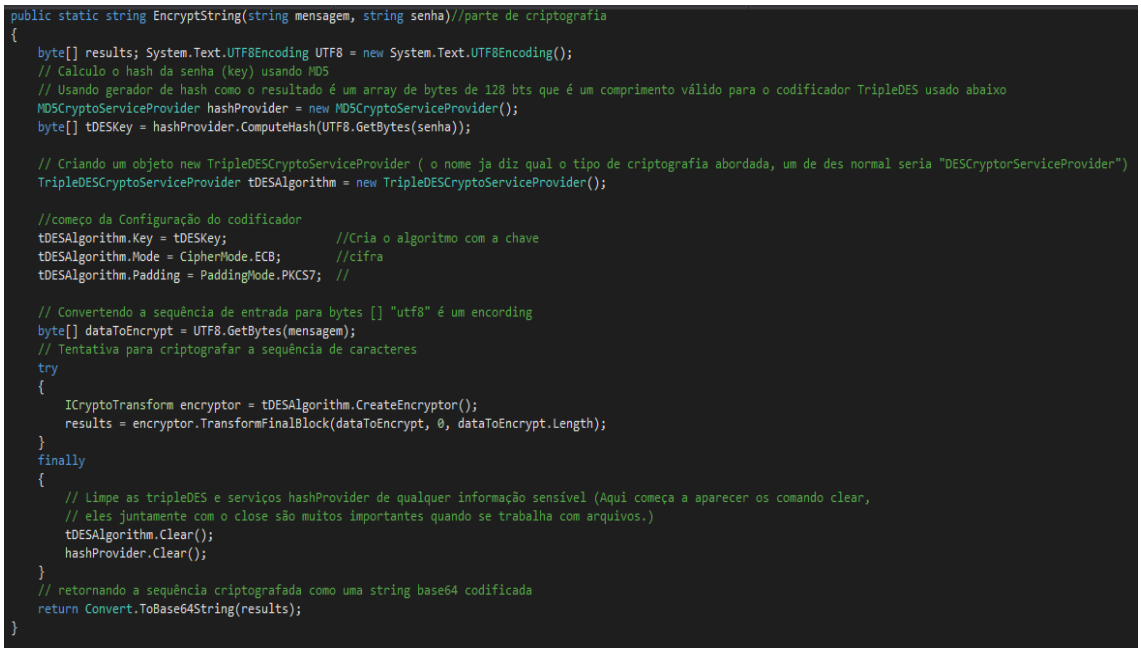
6. PROJETO – ESTRUTURA DO PROGRAMA

O *3 DES* utiliza o *modo de cifra* que se desenvolve por meio de *strings*. Estes são armazenados em um vetor específico, resultando, mediante a aplicação de uma chave, um bloco de saída. O bloco de saída é então codificado, utilizando uma chave de, aproximadamente, 64 *bits*. Esse processo gera um segundo bloco de saída, que passa por nova codificação, criando um terceiro bloco.

Criptografa-se, também, informações por blocos. Esse entendimento provém do algoritmo original *DES*. E, valendo-se desses conhecimentos prévios, as linhas de código do programa foram criadas com o uso de variáveis, vetores e comandos.

A ilustração abaixo mostra como foi desenvolvido a parte da criptografia:

Imagem 05 – Criptografia.



```
public static string EncryptString(string mensagem, string senha)//parte de criptografia
{
    byte[] results; System.Text.UTF8Encoding UTF8 = new System.Text.UTF8Encoding();
    // Calculo o hash da senha (key) usando MD5
    // Usando gerador de hash como o resultado é um array de bytes de 128 bts que é um comprimento válido para o codificador TripleDES usado abaixo
    MD5CryptoServiceProvider hashProvider = new MD5CryptoServiceProvider();
    byte[] tDESKey = hashProvider.ComputeHash(UTF8.GetBytes(senha));

    // Criando um objeto new TripleDESCryptoServiceProvider ( o nome ja diz qual o tipo de criptografia abordada, um de des normal seria "DESCryptorServiceProvider")
    TripleDESCryptoServiceProvider tDESAlgorithm = new TripleDESCryptoServiceProvider();

    //começo da Configuração do codificador
    tDESAlgorithm.Key = tDESKey;               //Cria o algoritmo com a chave
    tDESAlgorithm.Mode = CipherMode.ECB;       //cifra
    tDESAlgorithm.Padding = PaddingMode.PKCS7;  //

    // Convertendo a sequência de entrada para bytes [] "utf8" é um encording
    byte[] dataToEncrypt = UTF8.GetBytes(mensagem);
    // Tentativa para criptografar a sequência de caracteres
    try
    {
        ICryptoTransform encryptor = tDESAlgorithm.CreateEncryptor();
        results = encryptor.TransformFinalBlock(dataToEncrypt, 0, dataToEncrypt.Length);
    }
    finally
    {
        // Limpe as tripleDES e serviços hashProvider de qualquer informação sensível (Aqui começa a aparecer os comando clear,
        // eles juntamente com o close são muitos importantes quando se trabalha com arquivos.)
        tDESAlgorithm.Clear();
        hashProvider.Clear();
    }
    // retornando a sequência criptografada como uma string base64 codificada
    return Convert.ToBase64String(results);
}
```

Fonte: Própria, 2016.

Como já descrito a título de comentário no próprio programa, nesse momento, utilizou-se o cálculo do *hash* da senha (*key*) aplicando o uso do MD5, que é um esquema usado para a encriptação de dados: o MD5 transforma os dados em um código, servindo, portanto, para a conversão de uma senha em um determinado código criptografado, que será armazenado no banco de dados de um programa ou *site*.

No tocante à descriptografia, a ilustração a seguir a apresenta:

Imagem 06 – Descriptografia.

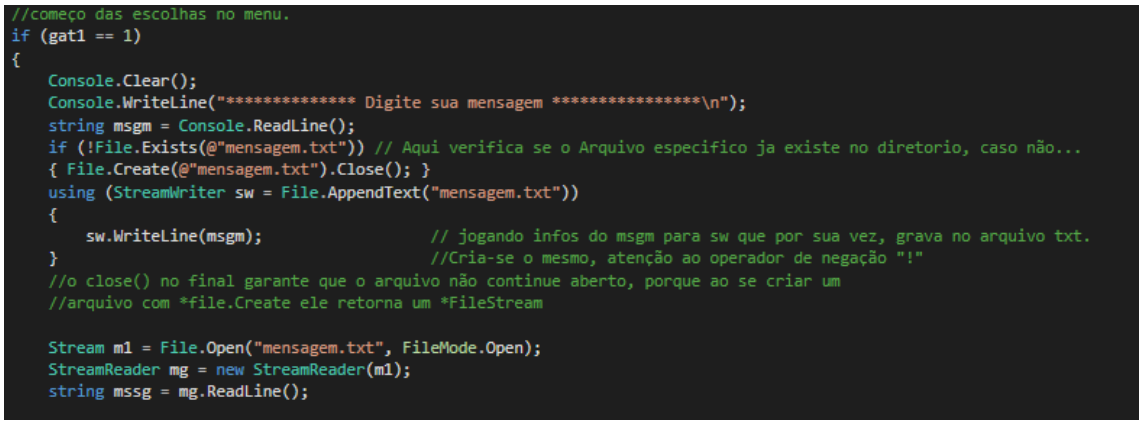
```
public static string DecryptString(string mensagem, string senha)//parte de descriptografia;
{
    byte[] results;
    System.Text.UTF8Encoding UTF8 = new System.Text.UTF8Encoding();

    // Calculamos o hash da senha usando MD5 (Mesmos processos feitos para a criptografia)
    MD5CryptoServiceProvider hashProvider = new MD5CryptoServiceProvider();
    byte[] tDESKey = hashProvider.ComputeHash(UTF8.GetBytes(senha));

    // Criando outro objeto new TripleDESCryptoServiceProvider
    TripleDESCryptoServiceProvider tDESAlgorithm = new TripleDESCryptoServiceProvider();

    // Mais uma vez configurando o codificador, atenção! Cuidado com os nomes
    tDESAlgorithm.Key = tDESKey;
    tDESAlgorithm.Mode = CipherMode.ECB;
    tDESAlgorithm.Padding = PaddingMode.PKCS7;
    // Convertendo a sequência de entrada para um byte []
    byte[] dataToDecrypt = Convert.FromBase64String(mensagem);
    // Tentativa para criptografar a sequência de caracteres
    try
    {   //aqui começa a diferenca entre os dois processos, no primeiro, era criado o TDESAlgorithm.CreateEncryptor
        // ja nesta parte fazemos o contrario, criamos o tDES...Decryptor;
        ICryptoTransform Decryptor = tDESAlgorithm.CreateDecryptor();
        //jogamos o resultado na variavel results, mas os dados ainda não estão utilizaveis.
        results = Decryptor.TransformFinalBlock(dataToDecrypt, 0, dataToDecrypt.Length);
    }
    finally
    {
        // Limpando novamente o tripleDES e serviços hashProvider de qualquer informação sensível
        tDESAlgorithm.Clear();
        hashProvider.Clear();
    }

    // agora encontramos a sequência criptografada como uma string base64 codificada, agora sim, os dados estão utilizaveis.
    return UTF8.GetString(results);
}
```

Fonte: Própria, 2016.

Nessa etapa, foram seguidos os mesmos procedimentos utilizados na criptografia, porém com a criação de um novo objeto para o trabalho em conjunto. Deu-se especial atenção à nomeação dos codificadores para evitar problemas futuros. Assim, a diferença entre os dois processos (o de criptografar com o de descriptografar) reside na aplicação de codificadores distintos: na parte da criptografia foi utilizado o *TDESAlgorithm.CreateEncryptor*; já na parte da descriptografia, o TDESAlgorithm.CreateDecryptor.

A seguir, as ilustrações mostram o funcionamento do *System.IO* e do *StreamWriter*:

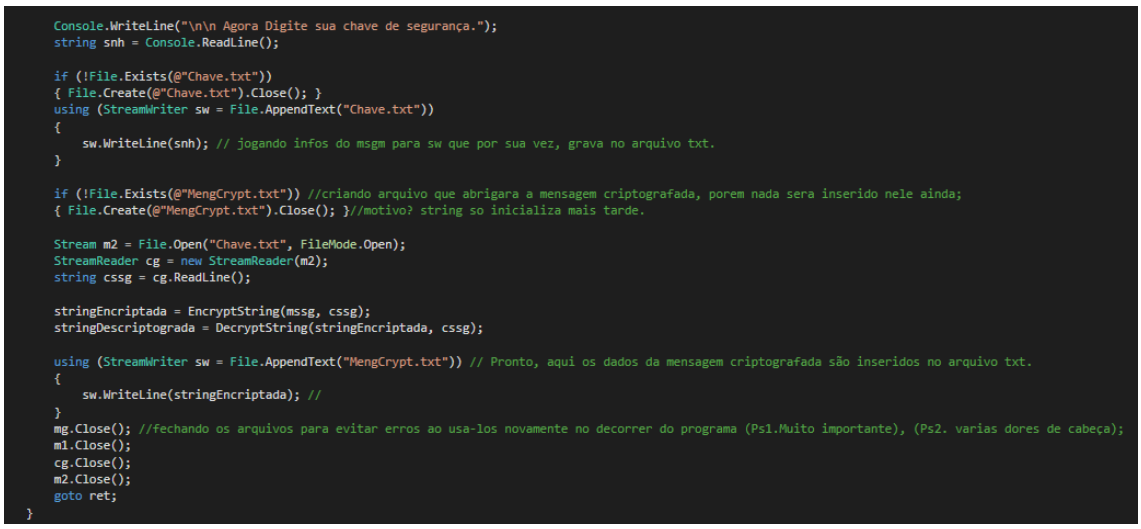
Imagem 07 – System.IO e StreamWriter (Criação da mensagem).



```
//começo das escolhas no menu.
if (gat1 == 1)
{
    Console.Clear();
    Console.WriteLine("************** Digite sua mensagem ***************\n");
    string msgm = Console.ReadLine();
    if (!File.Exists(@"mensagem.txt")) // Aqui verifica se o Arquivo especifico ja existe no diretorio, caso não...
    { File.Create(@"mensagem.txt").Close(); }
    using (StreamWriter sw = File.AppendText("mensagem.txt"))
    {
        sw.WriteLine(msgm);                 // jogando infos do msgm para sw que por sua vez, grava no arquivo txt.
    }                                       //Cria-se o mesmo, atenção ao operador de negação "!"
    //o close() no final garante que o arquivo não continue aberto, porque ao se criar um
    //arquivo com *file.Create ele retorna um *FileStream

    Stream m1 = File.Open("mensagem.txt", FileMode.Open);
    StreamReader mg = new StreamReader(m1);
    string mssg = mg.ReadLine();
```

Fonte: Própria, 2016.

Imagem 08 – System.IO e StreamWriter (Chave de segurança).



```
    Console.WriteLine("\n\n Agora Digite sua chave de segurança.");
    string snh = Console.ReadLine();

    if (!File.Exists(@"Chave.txt"))
    { File.Create(@"Chave.txt").Close(); }
    using (StreamWriter sw = File.AppendText("Chave.txt"))
    {
        sw.WriteLine(snh); // jogando infos do msgm para sw que por sua vez, grava no arquivo txt.
    }

    if (!File.Exists(@"MengCrypt.txt")) //criando arquivo que abrigara a mensagem criptografada, porem nada sera inserido nele ainda;
    { File.Create(@"MengCrypt.txt").Close(); }//motivo? string so inicializa mais tarde.

    Stream m2 = File.Open("Chave.txt", FileMode.Open);
    StreamReader cg = new StreamReader(m2);
    string cssg = cg.ReadLine();

    stringEncriptada = EncryptString(mssg, cssg);
    stringDescriptografada = DecryptString(stringEncriptada, cssg);

    using (StreamWriter sw = File.AppendText("MengCrypt.txt")) // Pronto, aqui os dados da mensagem criptografada são inseridos no arquivo txt.
    {
        sw.WriteLine(stringEncriptada); //
    }
    mg.Close(); //fechando os arquivos para evitar erros ao usa-los novamente no decorrer do programa (Ps1.Muito importante), (Ps2. varias dores de cabeça);
    m1.Close();
    cg.Close();
    m2.Close();
    goto ret;
}
```

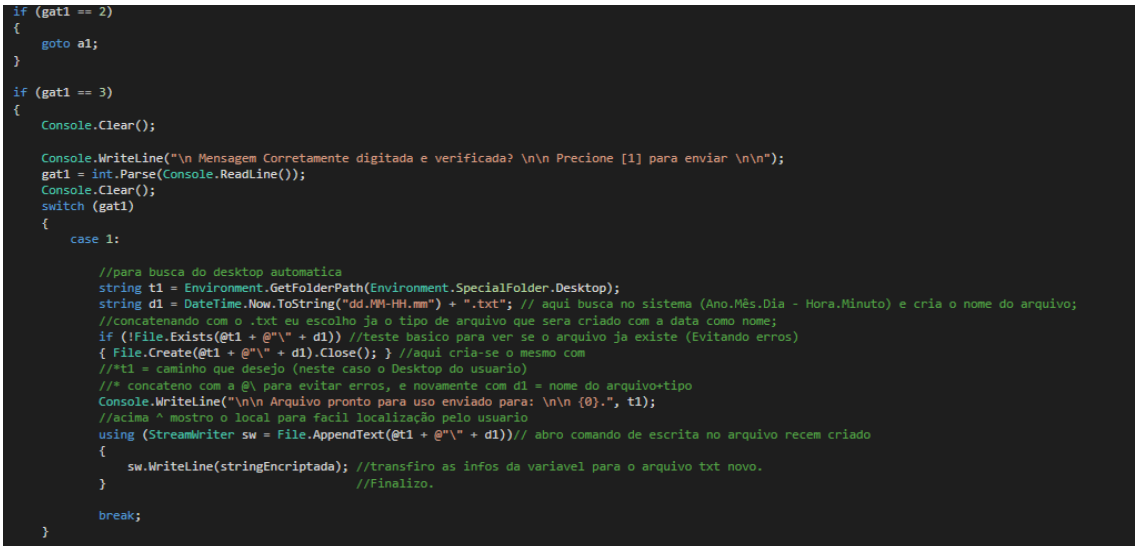
Fonte: Própria, 2016.

Com auxilio das funções do System.IO (*Input and Output*), o programa verifica se existem os arquivos necessários para pleno funcionamento. Caso eles ainda não existam, o programa os criará como 2 arquivos txt como pode ser observado acima. Importante frisar que o comando StreamWriter serve para a gravação de informações em arquivos, lembrando que as *strings* não podem ser simplesmente inseridas em arquivos distintos. Acrescenta-se, ainda, que a função StreamWriter se encarrega de dar continuidade ao procedimento: ele lê a *string* e escreve seu

conteúdo no arquivo destino (onde há o uso característico do comando). Em outras palavras, essa função trabalha diretamente no arquivo, inserindo o conteúdo que se encontra na variável e armazenando a mensagem que será criptografada.

Abaixo, a imagem mostra como foi desenvolvida a parte da busca automática:

Imagem 09 – Implementação do código e busca de arquivo.



```
if (gat1 == 2)
{
    goto a1;
}

if (gat1 == 3)
{
    Console.Clear();

    Console.WriteLine("\n Mensagem Corretamente digitada e verificada? \n\n Precione [1] para enviar \n\n");
    gat1 = int.Parse(Console.ReadLine());
    Console.Clear();
    switch (gat1)
    {
        case 1:

            //para busca do desktop automatica
            string t1 = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
            string d1 = DateTime.Now.ToString("dd.MM-HH.mm") + ".txt"; // aqui busca no sistema (Ano.Mês.Dia - Hora.Minuto) e cria o nome do arquivo;
            //concatenando com o .txt eu escolho ja o tipo de arquivo que sera criado com a data como nome;
            if (!File.Exists(@t1 + @"\" + d1)) //teste basico para ver se o arquivo ja existe (Evitando erros)
            { File.Create(@t1 + @"\" + d1).Close(); } //aqui cria-se o mesmo com
            //*t1 = caminho que desejo (neste caso o Desktop do usuario)
            //* concateno com a @\ para evitar erros, e novamente com d1 = nome do arquivo+tipo
            Console.WriteLine("\n\n Arquivo pronto para uso enviado para: \n\n {0}.", t1);
            //acima ^ mostro o local para facil localização pelo usuario
            using (StreamWriter sw = File.AppendText(@t1 + @"\" + d1))// abro comando de escrita no arquivo recem criado
            {
                sw.WriteLine(stringEncriptada); //transfiro as infos da variavel para o arquivo txt novo.
            }                                   //Finalizo.

            break;
    }
```

Fonte: Própria, 2016.

Nessa etapa, vislumbra-se a parte do código responsável por buscar e mostrar na tela todos os arquivos de texto que estão na pasta Padrão. Houve a utilização do comando *Environment* : primeiro esse comando diz que uma pasta será buscada; em seguida especifica em qual pasta se deseja obter o caminho em forma de dados *string* (no caso, foi a área de trabalho do usuário, onde as informações foram gravadas na variável "t2"). Após essa etapa, criou-se a *string* com o nome *arquivos* e houve a utilização do comando de busca em um diretório especifico. Acrescenta-se que, ao invés de se utilizar um diretório fixo, foram usadas as informações gravadas na variável t2 para se conseguir o endereço e para a posterior concatenação com o tipo de arquivo (“.txt”) que o usuário queira encontrar. Em seguida, implementou-se um laço para imprimir na tela todos os arquivos .txt que

forem encontrados na pasta especificada, oferecendo ao usuário uma lista completa para a sua escolha.

Em seguida, o usuário entra com o nome do arquivo desejado, que é armazenado na variável "z1". Logo após, inicia-se mais um variável com o nome de "mcpt2", que usará o comando de abertura e fará uma leitura do arquivo txt. No entanto, cumpre frisar que, ao invés de se usar um caminho específico, utilizam-se o nome do arquivo já armazenado em uma variável, o caminho armazenado em outra e sua extensão pré-definida por meio do método de concatenação. Assim sendo, necessários são o uso do caractere especial "@", que serve para evitar erros do compilador e a utilização de barras invertidas no código (/)). Após essa etapa, o arquivo finalmente é aberto, com o comando *streamwriter* e, com a ajuda da variável z1, todo conteúdo do arquivo (a mensagem criptografada) é transferido para a variável mcpt e depois convertido em *string*, ficando pronta para uso interno. Todavia, caso seja transferido para variável mcpt2, observar-se-á a mensagem sendo exibida na tela.

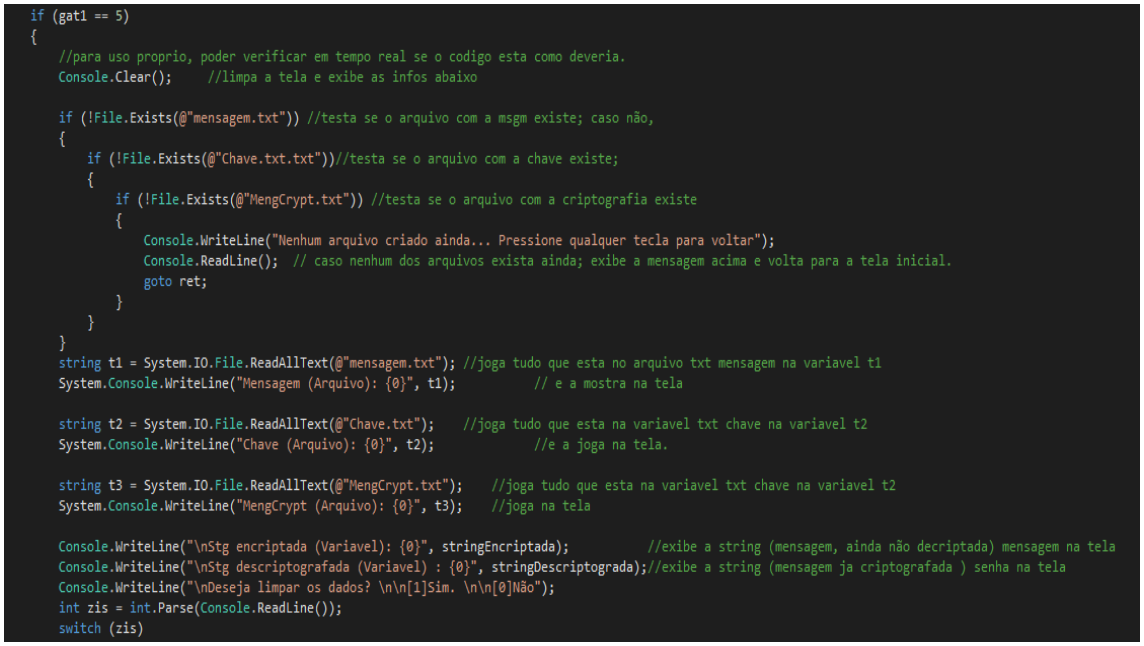
Imagem 10 – Continuação da implementação do código e busca de arquivo.

```
            Console.WriteLine("\n\n Agora digite a CHAVE de segurança para decriptar a msg.");
            string ssn = Console.ReadLine();


            stringDescriptografada = DecryptString(mcpt2, ssn);

            Console.WriteLine("\n\n Deseja descriptografar agora? \n[1] Sim. \n[0] Não");
            gat1 = int.Parse(Console.ReadLine());

            switch (gat1)
            {
                case 1:

                    Console.WriteLine("\n \n \n O conteudo da mensagem é:  {0}", stringDescriptografada);
                    Console.WriteLine("Pressione qualquer tecla para voltar.");
                    Console.ReadLine();
                    goto ret;

                case 0:
                    goto ret;
            }
            break;


        case 0:
            Console.Clear();
            goto ret3;
    }
}
```

Fonte: Própria, 2016.

Abaixo, a imagem apresenta uma medida de segurança:

Imagem 11 – Medida de segurança caso ocorra a tentativa de mudar uma mensagem não criada.



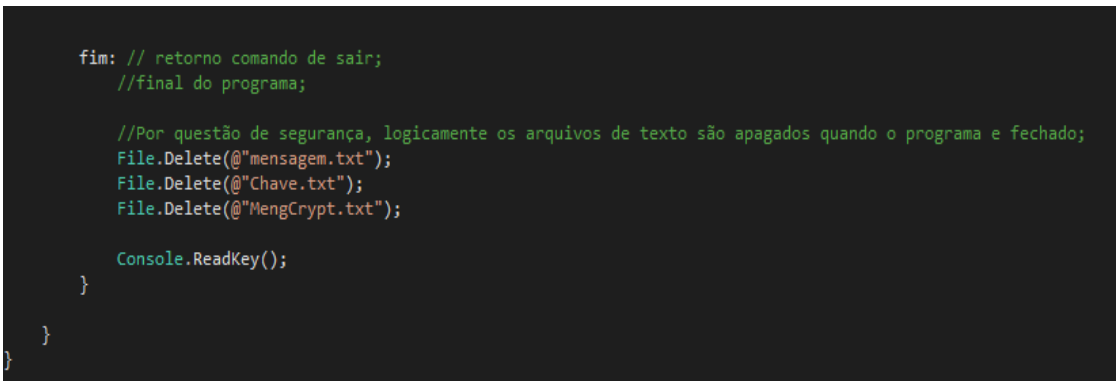
```
if (gat1 == 5)
{
    //para uso proprio, poder verificar em tempo real se o codigo esta como deveria.
    Console.Clear();     //limpa a tela e exibe as infos abaixo

    if (!File.Exists(@"mensagem.txt")) //testa se o arquivo com a msgm existe; caso não,
    {
        if (!File.Exists(@"Chave.txt.txt"))//testa se o arquivo com a chave existe;
        {
            if (!File.Exists(@"MengCrypt.txt")) //testa se o arquivo com a criptografia existe
            {
                Console.WriteLine("Nenhum arquivo criado ainda... Pressione qualquer tecla para voltar");
                Console.ReadLine();  // caso nenhum dos arquivos exista ainda; exibe a mensagem acima e volta para a tela inicial.
                goto ret;
            }
        }
    }
    string t1 = System.IO.File.ReadAllText(@"mensagem.txt"); //joga tudo que esta no arquivo txt mensagem na variavel t1
    System.Console.WriteLine("Mensagem (Arquivo): {0}", t1);          // e a mostra na tela

    string t2 = System.IO.File.ReadAllText(@"Chave.txt");    //joga tudo que esta na variavel txt chave na variavel t2
    System.Console.WriteLine("Chave (Arquivo): {0}", t2);             //e a joga na tela.

    string t3 = System.IO.File.ReadAllText(@"MengCrypt.txt");    //joga tudo que esta na variavel txt chave na variavel t2
    System.Console.WriteLine("MengCrypt (Arquivo): {0}", t3);    //joga na tela

    Console.WriteLine("\nStg encriptada (Variavel): {0}", stringEncriptada);          //exibe a string (mensagem, ainda não decriptada) mensagem na tela
    Console.WriteLine("\nStg descriptografada (Variavel) : {0}", stringDescriptografada);//exibe a string (mensagem ja criptografada ) senha na tela
    Console.WriteLine("\nDeseja limpar os dados? \n\n[1]Sim. \n\n[0]Não");
    int zis = int.Parse(Console.ReadLine());
    switch (zis)
```

Fonte: Própria, 2016.

A seguir, foi elaborado um teste simples com a negação (not) representada pelo sinal de exclamação (!). Esse teste consistiu em verificar a inexistência dos arquivos mensagem.txt, chave.txt e o mengcrypt. Caso não existam, aparecerá na tela a mensagem “Nenhum arquivo criado ainda".

Adiante, a imagem mostra o término do programa.

Imagem 12 – Término do programa e apagamento dos arquivos.



```
        fim: // retorno comando de sair;
            //final do programa;

            //Por questão de segurança, logicamente os arquivos de texto são apagados quando o programa e fechado;
            File.Delete(@"mensagem.txt");
            File.Delete(@"Chave.txt");
            File.Delete(@"MengCrypt.txt");

            Console.ReadKey();
        }

    }
}
```

Fonte: Própria, 2016.

Por fim, o programa automaticamente apaga todos os arquivos de texto criados na pasta raiz, mediante o comando "File.Delete" – aqui não foi utilizada nenhuma variável para direcionar o caminho, visto que os arquivos sem endereço específicos são criados na pasta *debug* do Visual Studio, o que torna desnecessário o seu diretório especifico (com exceção do uso do caractere “@” para se evitar erros).

7. RELATÓRIO COM AS LINHAS DE CÓDIGO DO PROGRAMA

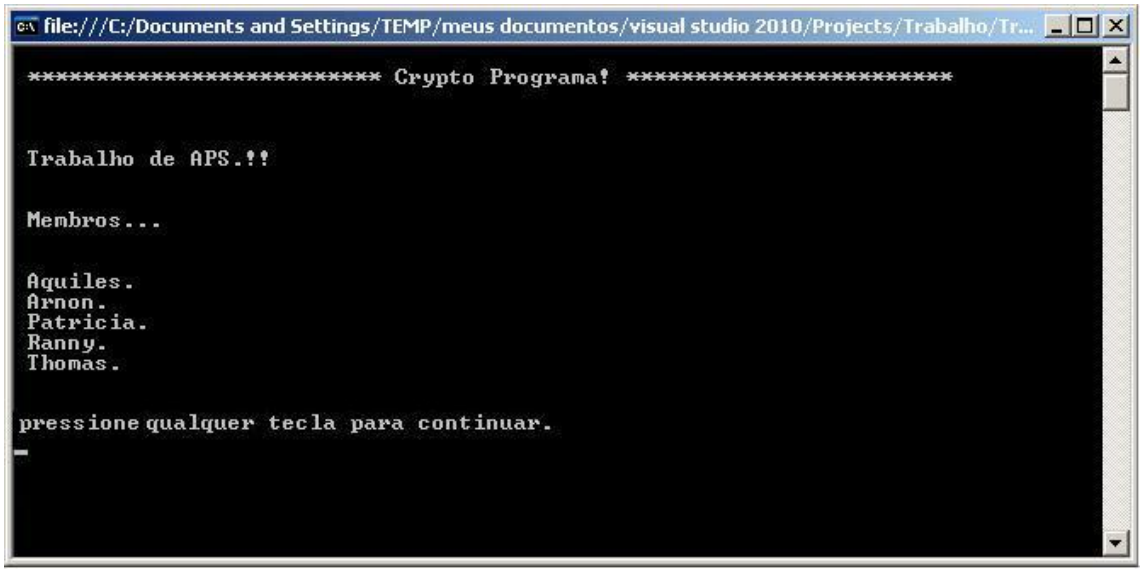
O programa foi desenvolvido em *Windows* 7 com o auxílio de um bloco de notas e executado na plataforma *Visual Studio* na linguagem c#. A criptografia utilizada foi do tipo simétrica (3 *DES*).

Buscou-se otimizar o programa com o uso do mínimo possível de recursos, criando códigos mais organizados e simples. Ao todo, foram escritas trezentos e cinquenta e nove linhas de código.

Home

A imagem abaixo mostra o inicio do programa com a listagem dos membros:

Imagem 13 – Início do programa e listagem dos membros.

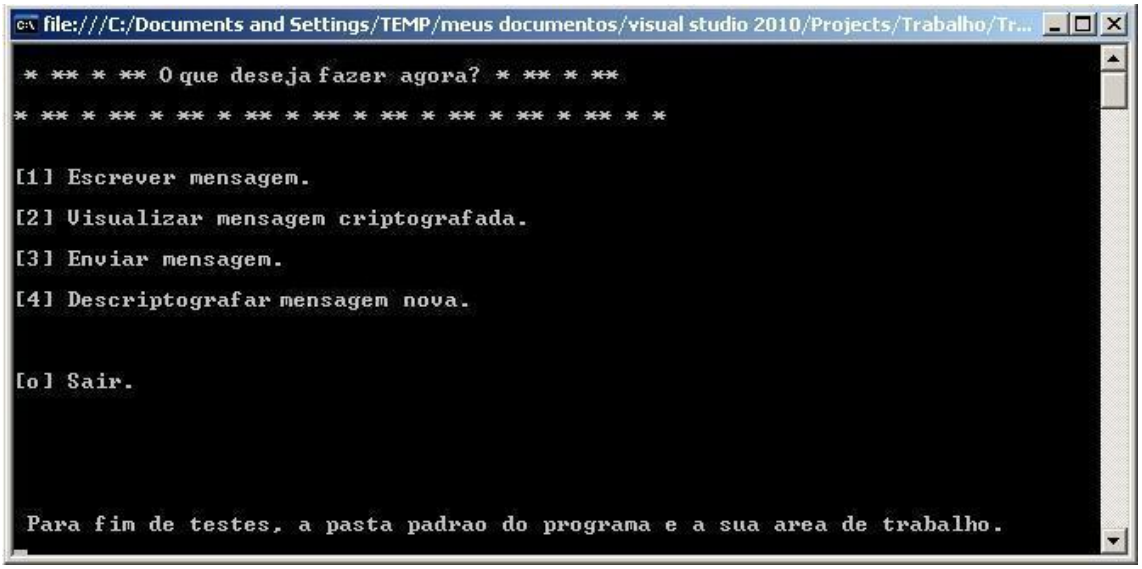


Fonte: Própria 2016.

Menu

A imagem a seguir mostra a função *menu*, que recebe a opção selecionada pelo usuário e retorna entrando na função escolhida:

Imagem 14 – Função *Menu*.



Fonte: Própria 2016.

As opções do *menu* são:

[1] Opção do usuário escrever um texto e criptografá-lo.

[2] Opção para o usuário verificar o texto escrito criptografado.

[3] Opção para envio de mensagem: retorna ao usuário seu texto criptografado em formato “arquivo de texto”.

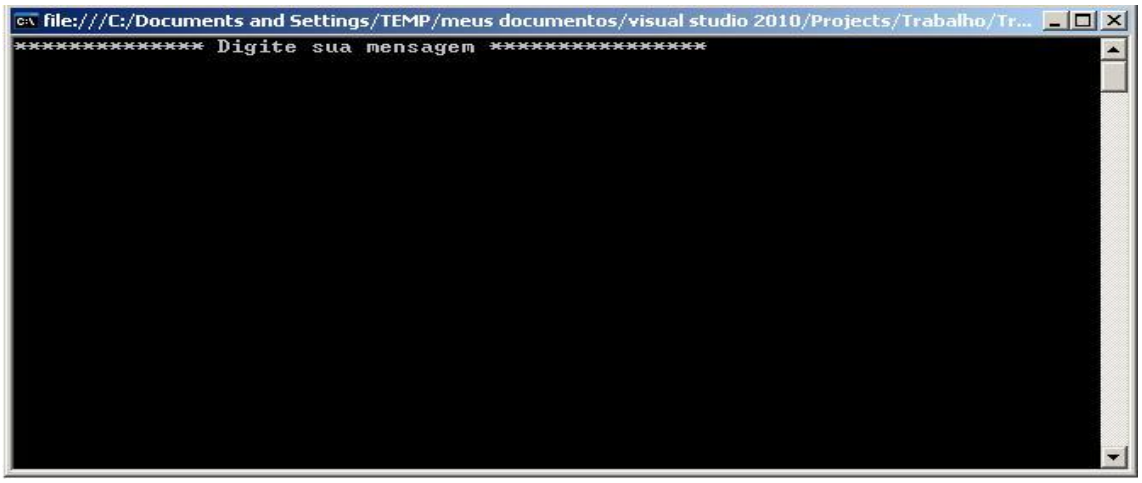
[4] O usuário pode escolher um arquivo de texto específico e decriptografá-lo, revelando assim sua mensagem original.

[0]. Finaliza a execução do programa.

Início da criptografia

Para iniciar a criptografia o usuário deverá escrever sua mensagem.

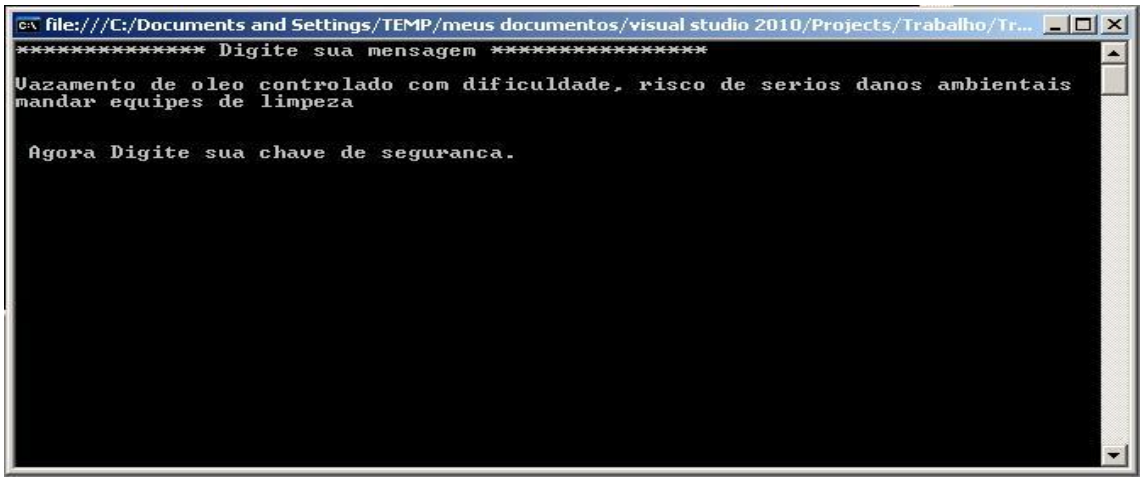
Imagem 15 – Início da criptografia.



Fonte: Própria 2016.

Abaixo, a imagem da continuação da criptografia:

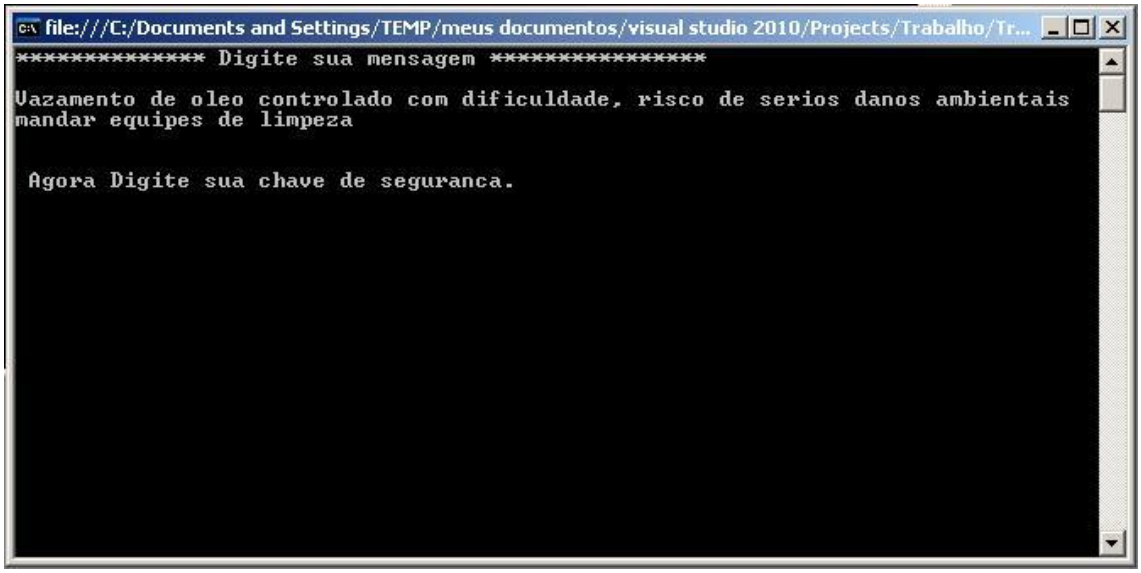
Imagem 16 – Continuação da criptografia.



Fonte: Própria 2016.

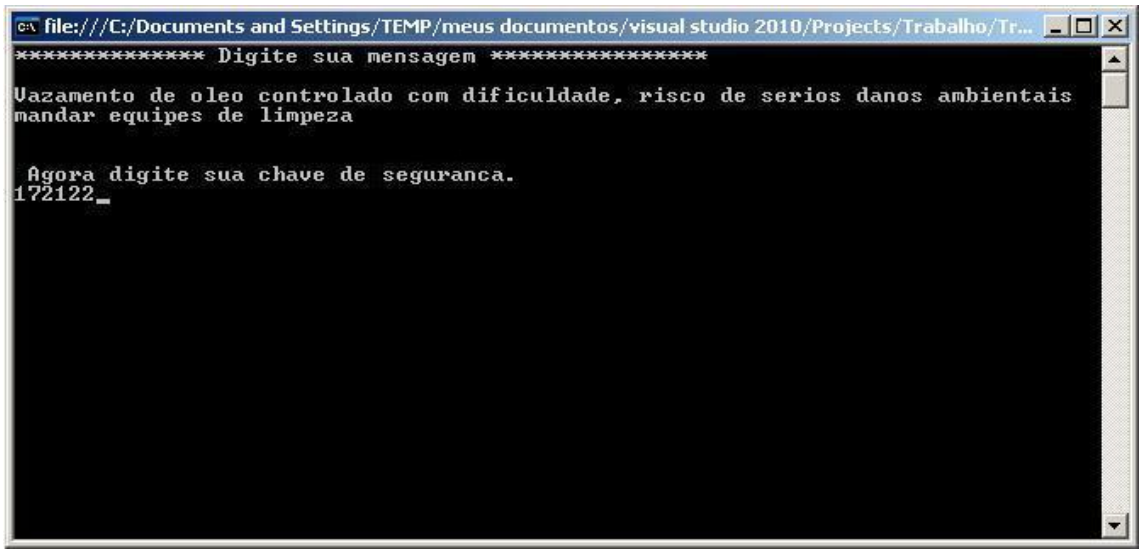
Após a mensagem ser digitada, o programa solicita uma chave para o usuário; esta chave pode ser composta por números ou algum outro texto. Isso pode ser vislumbrado nas seguintes imagens:

Imagem 17 – Solicitação de chave de segurança.



Fonte: Própria 2016.

Imagem 18 – Preenchimento da chave de segurança.



Fonte: Própria 2016.

Verificando a mensagem

Aqui, o usuário terá a opção de visualizar o seu texto criptografado para ter mais segurança antes do envio:

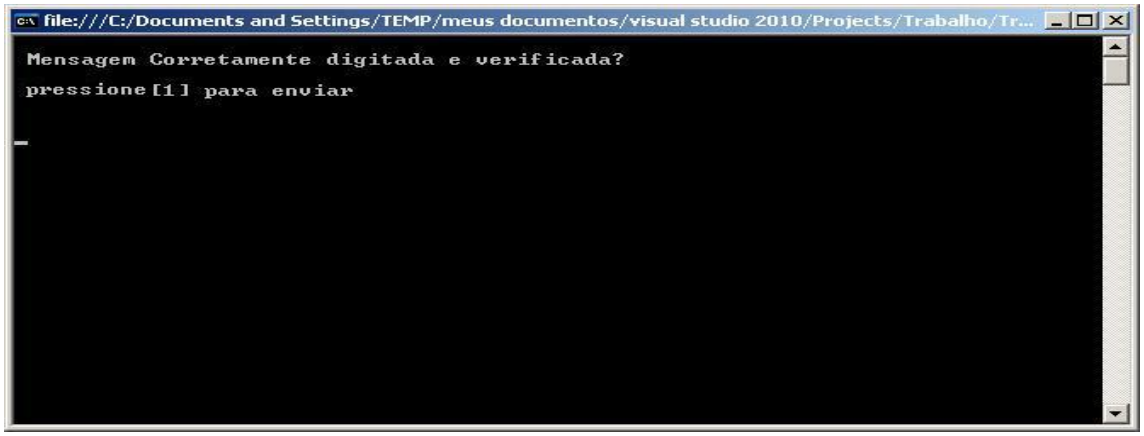
Imagem 19 – Primeira verificação de mensagem.

Fonte: Própria 2016.

Enviando a mensagem

As imagens abaixo mostram, passo a passo, como o programa tratará a mensagem do usuário. Primeiro, ele pede para o usuário verificar se a mensagem esta corretamente digitada e verificada.

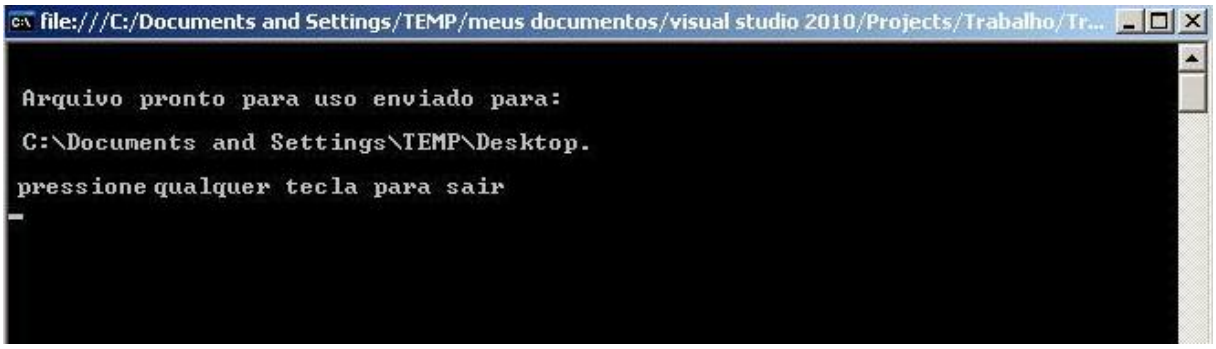
Imagem 20 – Segunda verificação de mensagem.



Fonte: Própria 2016.

Ao chegar nessa etapa, o programa já terá enviado a mensagem do usuário.

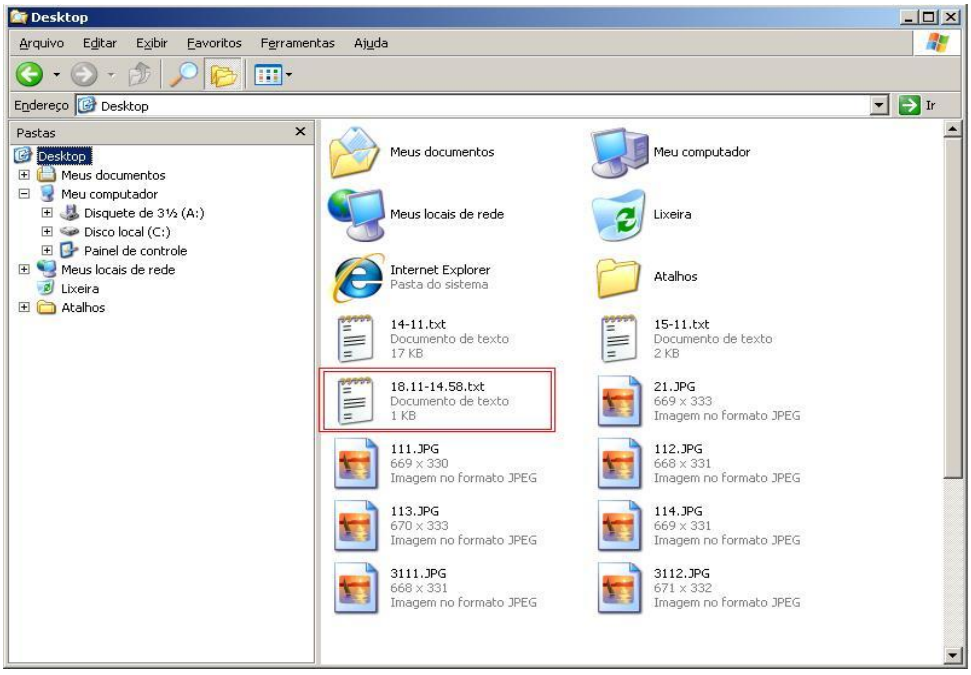
Imagem 21 – Envio de mensagem.



Fonte: Própria 2016.

Aqui, pode-se verificar como a criação do arquivo com a mensagem criptografada é realizada: o programa gera um novo arquivo para cada mensagem, usando como parâmetro de nomeação a data e a hora que constarem no computador do usuário.

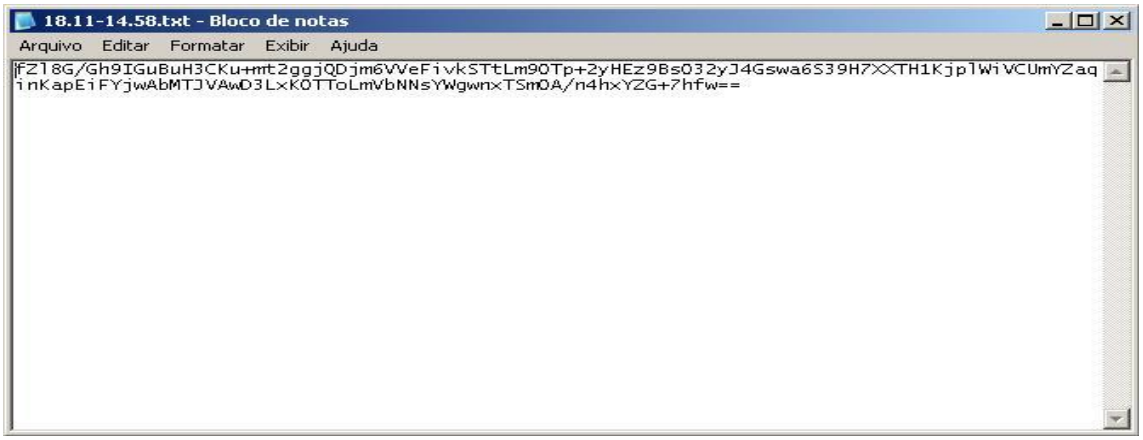
Imagem 22 – Criação de arquivo.



Fonte: Própria 2016.

Abaixo, observa-se a mensagem já criptografada e pronta para o envio no arquivo de texto.

Imagem 23 – Mensagem criptografada e pronta para envio.

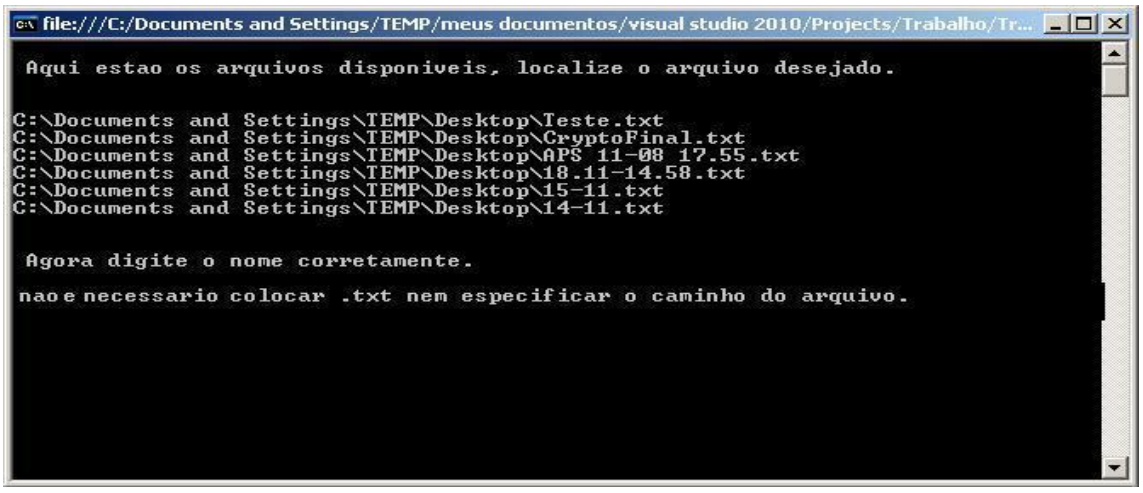


Fonte: Própria 2016.

Descriptografar mensagem

Essa parte do programa irá fazer uma varredura em uma pasta já pré-selecionada do computador do usuário, e exibirá todos os arquivos de texto que estiverem nela.

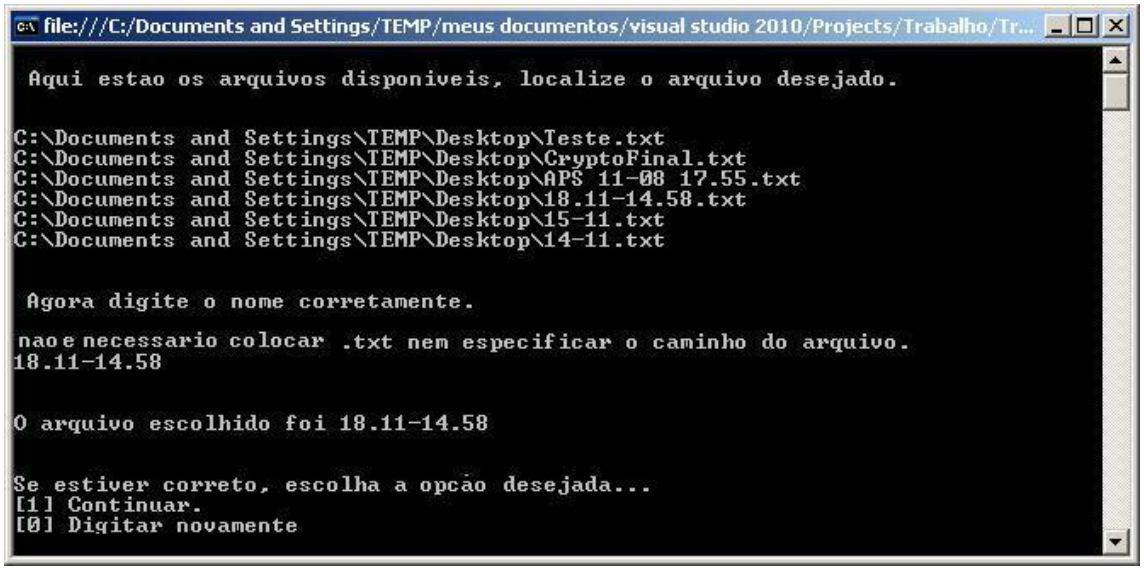
Imagem 24 – Exibição dos arquivos de texto.



Fonte: Própria 2016.

Logo após, o usuário deve localizar e digitar o nome da mensagem escolhida, como mostra a figura abaixo. O programa mostrará o nome escolhido e concederá uma chance do usuário conferir a digitação.

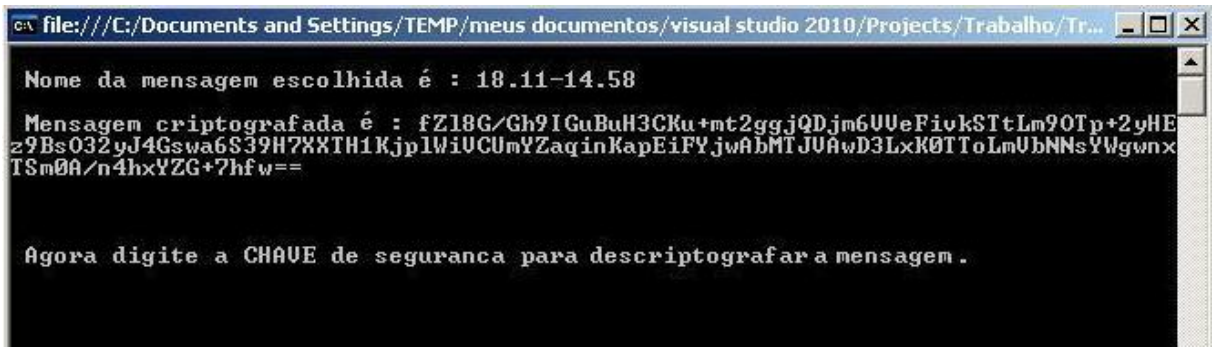
Imagem 25 – Digitar o nome da mensagem escolhida.



Fonte: Própria 2016.

Caso tudo esteja correto, o programa abrirá o arquivo de texto escolhido, exibindo seu conteúdo criptografado e pedirá a chave de segurança (que só o destinatário e o remetente devem conhecer).

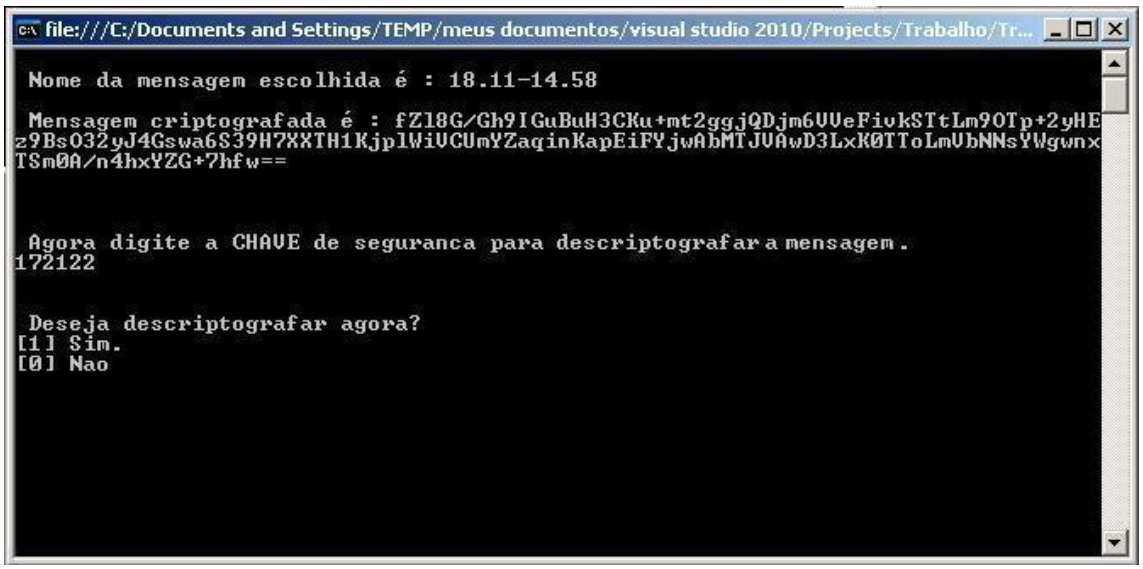
Imagem 26 – Digitar a chave de segurança.



Fonte: Própria 2016.

Em seguida, surgirá a pergunta “Deseja descriptografar agora?”:

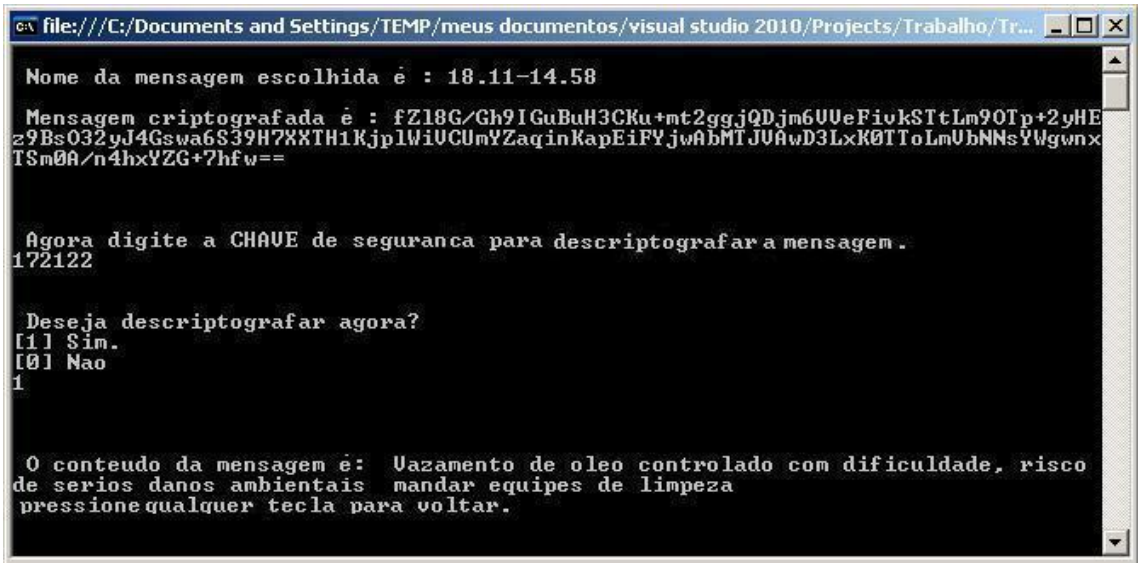
Imagem 27 – Mensagem de descriptografia.



Fonte: Própria 2016.

Se a chave estiver correta, o programa mostrará na tela o conteúdo original da mensagem.

Imagem 28 – Conteúdo original da mensagem.



Fonte: Própria 2016.

Finalizando a execução

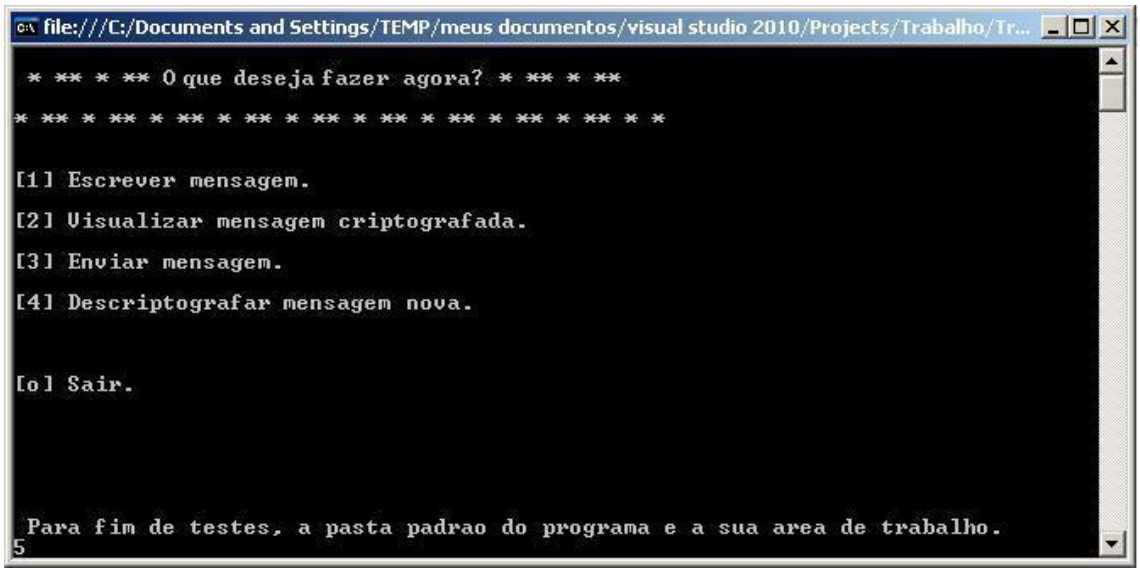
Para finalizar o programa, basta apenas escolher a opção zero “0” na tela inicial, o programa apagara automaticamente todo e qualquer dado que você tenha criado durante a sua utilização, exceto e claro, sua mensagem criptografada já pronta para o envio.

Extra

O programa possui mais uma função, que pode ser acessada na opção 5, para verificação do estado do código e do que algumas variáveis carregam, além de limpar todos os dados criados no computador.

Para tanto, no *menu* principal, digite 0 e pressione *enter*:

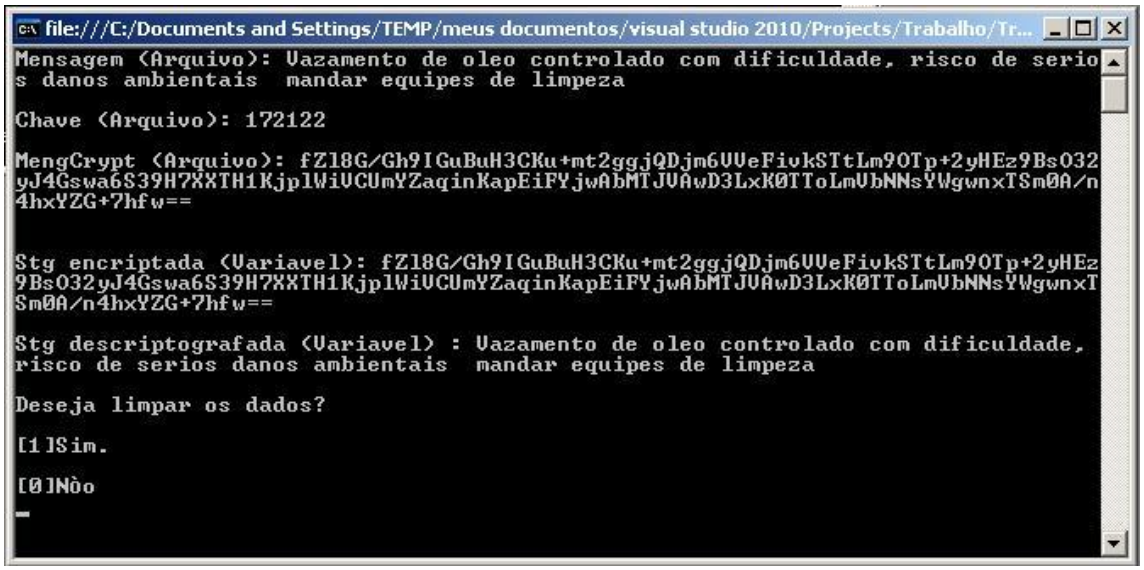
Imagem 29 – Opção 0 (Finalizando a execução).



Fonte: Própria 2016.

Na tela que será aberta, as informações serão divididas por linhas, como se observa na seguinte imagem:

Imagem 30 – Informações divididas por linhas.



Fonte: Própria 2016.

1. Mensagem <Arquivo)>: Refere-se a mensagem que está no arquivo de texto gerado.

2. Chave <Arquivo>: Também diz respeito ao arquivo de texto que armazena a chave de segurança.

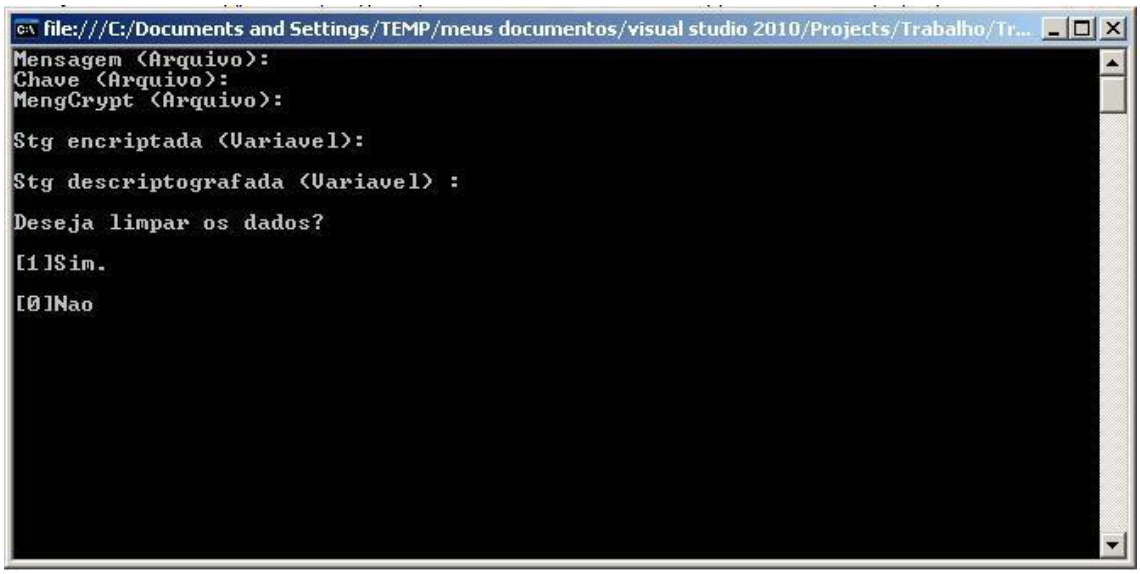
3. MengCrypt <Arquivo>: Este é o conteúdo do arquivo de texto que contém a mensagem criptografada: ele só é gerado quando o usuário escolhe enviar a mensagem.

4. Stg encriptada <Variavel>: Esta é a *string* que guarda a mensagem criptografada – ela é gerada antes do arquivo de texto do mesmo nome.

5. Stg descriptografada <Variavel> – Esta *string*, por sua vez, guarda o texto puro para este poder ser usado.

Para limpar todas as informações,e arquivos de textos criados, escolha, nesta janela, a opção “[1] sim”: aparecerá uma mensagem de confirmação e, logo após, voltará para a tela principal. Ao se entrar novamente, observar-se-á que os dados foram limpos.

Imagem 31 – Limpar dados.



Fonte: Própria 2016.

Este é o programa de criptografia, com suas funções e seu exemplo de funcionamento. A seguir, encontra-se disponibilizado o código completo do programa, com todas as suas 359 linhas e comentários específicos.

Linhas de código: Cripto_programa

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Security.Cryptography;
using System.Security;
using System.IO;

namespace APS_Trabalho
{
    class MainClass
    {
        public static string EncryptString(string mensagem, string senha)//parte de criptografia
        {

            byte[] results; System.Text.UTF8Encoding UTF8 = new System.Text.UTF8Encoding();
            // Calculo o hash da senha (key) usando MD5
            // Usando gerador de hash como o resultado é um array de bytes de 128 bts que é um
comprimento válido para o codificador TripleDES usado abaixo
            MD5CryptoServiceProvider hashProvider = new MD5CryptoServiceProvider();
            byte[] tDESKey = hashProvider.ComputeHash(UTF8.GetBytes(senha));

            // Criando um objeto new TripleDESCryptoServiceProvider ( o nome ja diz qual o tipo de
criptografia abordada, um de des normal seria "DESCryptorServiceProvider")
            TripleDESCryptoServiceProvider tDESAlgorithm = new TripleDESCryptoServiceProvider();

            //começo da Configuração do codificador
            tDESAlgorithm.Key = tDESKey;                //Cria o algoritmo com a chave
            tDESAlgorithm.Mode = CipherMode.ECB;        //cifra
```

```
            tDESAlgorithm.Padding = PaddingMode.PKCS7;  //

            // Convertendo a sequência de entrada para bytes [] "utf8" é um encording
            byte[] dataToEncrypt = UTF8.GetBytes(mensagem);
            // Tentativa para criptografar a sequência de caracteres
            try
            {
                ICryptoTransform encryptor = tDESAlgorithm.CreateEncryptor();
                results = encryptor.TransformFinalBlock(dataToEncrypt, 0, dataToEncrypt.Length);
            }
            finally
            {
                // Limpe as tripleDES e serviços hashProvider de qualquer informação sensível (Aqui
começa a aparecer os comando clear, eles juntamente com o close são muitos importantes quando se
trabalha com arquivos.)
                tDESAlgorithm.Clear();
                hashProvider.Clear();
            }
            // retornando a sequência criptografada como uma string base64 codificada
            return Convert.ToBase64String(results);
        }

        public static string DecryptString(string mensagem, string senha)//parte de descriptografia;
        {
            byte[] results;
            System.Text.UTF8Encoding UTF8 = new System.Text.UTF8Encoding();

            // Calculamos o hash da senha usando MD5 (Mesmos processos feitos para a criptografia)

            MD5CryptoServiceProvider hashProvider = new MD5CryptoServiceProvider();
            byte[] tDESKey = hashProvider.ComputeHash(UTF8.GetBytes(senha));

            // Criando outro objeto new TripleDESCryptoServiceProvider
            TripleDESCryptoServiceProvider tDESAlgorithm = new TripleDESCryptoServiceProvider();

            // Mais uma vez configurando o codificador, atenção! Cuidado com os nomes
            tDESAlgorithm.Key = tDESKey;
            tDESAlgorithm.Mode = CipherMode.ECB;
            tDESAlgorithm.Padding = PaddingMode.PKCS7;
            // Convertendo a sequência de entrada para um byte []
            byte[] dataToDecrypt = Convert.FromBase64String(mensagem);
            // Tentativa para criptografar a sequência de caracteres
            try
            {   //aqui começa a diferenca entre os dois processos, no primeiro, era criado o
TDESAlgorithm.CreateEncryptor
                // ja nesta parte fazemos o contrario, criamos o tDES...Decryptor;
                ICryptoTransform Decryptor = tDESAlgorithm.CreateDecryptor();
                //jogamos o resultado na variavel results, mas os dados ainda não estão utilizaveis.
                results = Decryptor.TransformFinalBlock(dataToDecrypt, 0, dataToDecrypt.Length);
            }
            finally
            {
                // Limpando novamente o tripleDES e serviços hashProvider de qualquer informação
sensível
                tDESAlgorithm.Clear();
                hashProvider.Clear();
            }

            // agora encontramos a sequência criptografada como uma string base64 codificada, agora
sim, os dados estão utilizaveis.
            return UTF8.GetString(results);
        }

        public static void Main(string[] args)
        {
            int gat1; // gatilho de comando para transitar pelo codigo;

            Console.WriteLine("\n ************************ Cripto Programa!
**********************\n"); // apresentação;
            Console.WriteLine("\n\n Trabalho de APS.!! ");
```

```
            Console.WriteLine("\n\n Membros... \n\n\n Aquiles. \n Arnon. \n Patricia. \n Ranny. \n
Thomas. ");
            Console.WriteLine("\n\nPressione qualquer tecla para continuar.");
            Console.ReadLine();

            /*
            Para localização de nomes;!
            Mensagem >> msg >> mssg
            Senha >> Senha/Key >> cssg
            Stg encriptada >> stringEncriptada
            Stg descriptografada >> stringDescriptograda
            */

            string mensagem = " ";
            string senha = " ";

            string stringEncriptada = EncryptString(mensagem, senha);
            string stringDescriptograda = DecryptString(stringEncriptada, senha);

                 //menu principal
        ret: // retorno para o menu principal, muito usado;
            Console.Clear();
            Console.WriteLine("\n * ** * ** O que deseja fazer agora? * ** * ** ");
            Console.WriteLine("\n* ** * ** * ** * ** * ** * ** * ** * ** * ** * *");
            Console.WriteLine("\n\n[1] Escrever mensagem. \n\n[2] Visualizar mensagem criptografada.
\n\n[3] Enviar mensagem. \n\n[4] Descriptografar mensagem nova. \n\n \n\n[o] Sair.");
            Console.WriteLine("\n\n\n\n\n\n Para fim de testes, a pasta padrao do programa e a sua area
de trabalho.");
            gat1 = int.Parse(Console.ReadLine());
            //começo das escolhas no menu.
            if (gat1 == 1)
            {
                Console.Clear();
                Console.WriteLine("************** Digite sua mensagem ***************\n");
                string msgm = Console.ReadLine();
                if (!File.Exists(@"mensagem.txt")) // Aqui verifica se o Arquivo especifico ja existe
no diretorio, caso não...
                { File.Create(@"mensagem.txt").Close(); }
                using (StreamWriter sw = File.AppendText("mensagem.txt"))
                {
                    sw.WriteLine(msgm);                 // jogando infos do msgm para sw que por sua
vez, grava no arquivo txt.
                }                                       //Cria-se o mesmo, atenção ao operador de
negação "!"
                //o close() no final garante que o arquivo não continue aberto, porque ao se criar um
                //arquivo com *file.Create ele retorna um *FileStream

                Stream m1 = File.Open("mensagem.txt", FileMode.Open);
                StreamReader mg = new StreamReader(m1);
                string mssg = mg.ReadLine();

                Console.WriteLine("\n\n Agora Digite sua chave de segurança.");
                string snh = Console.ReadLine();

                if (!File.Exists(@"Chave.txt"))
                { File.Create(@"Chave.txt").Close(); }
                using (StreamWriter sw = File.AppendText("Chave.txt"))
                {
                    sw.WriteLine(snh); // jogando infos do msgm para sw que por sua vez, grava no
arquivo txt.
                }

                if (!File.Exists(@"MengCrypt.txt")) //criando arquivo que abrigara a mensagem
criptografada, porem nada sera inserido nele ainda;
                { File.Create(@"MengCrypt.txt").Close(); }//motivo? string so inicializa mais tarde.

                Stream m2 = File.Open("Chave.txt", FileMode.Open);
                StreamReader cg = new StreamReader(m2);
                string cssg = cg.ReadLine();

                stringEncriptada = EncryptString(mssg, cssg);
```

```
                stringDescriptografada = DecryptString(stringEncriptada, cssg);

                using (StreamWriter sw = File.AppendText("MengCrypt.txt")) // Pronto, aqui os dados da
mensagem criptografada são inseridos no arquivo txt.
                {
                    sw.WriteLine(stringEncriptada); //
                }
                mg.Close(); //fechando os arquivos para evitar erros ao usa-los novamente no decorrer
do programa (Ps1.Muito importante), (Ps2. varias dores de cabeça);
                m1.Close();
                cg.Close();
                m2.Close();
                goto ret;
            }
            if (gat1 == 2)
            {
                goto a1;
            }

            if (gat1 == 3) // -> ideia! criar arquivo com data e hora, copiar o conteudo de
stringEncryptada para ele, ao comando, envia-la para a area de trabalho e apagar qualquer vestigio dela
no diretorio raiz
            {
                Console.Clear();

                Console.WriteLine("\n Mensagem Corretamente digitada e verificada? \n\n Precione [1]
para enviar \n\n");
                gat1 = int.Parse(Console.ReadLine());
                Console.Clear();
                switch (gat1)
                {
                    case 1:

                        //para busca do desktop automatica
                        string t1 = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
                        string d1 = DateTime.Now.ToString("dd.MM-HH.mm") + ".txt"; // aqui busca no
sistema (Ano.Mês.Dia - Hora.Minuto) e cria o nome do arquivo;
                        //concatenando com o .txt eu escolho ja o tipo de arquivo que sera criado com a
data como nome;
                        if (!File.Exists(@t1 + @"\" + d1)) //teste basico para ver se o arquivo ja
existe (Evitando erros)
                        { File.Create(@t1 + @"\" + d1).Close(); } //aqui cria-se o mesmo com
                        //*t1 = caminho que desejo (neste caso o Desktop do usuario)
                        //* concateno com a @\ para evitar erros, e novamente com d1 = nome do
arquivo+tipo
                        Console.WriteLine("\n\n Arquivo pronto para uso enviado para: \n\n {0}.", t1);
                        //acima ^ mostro o local para facil localização pelo usuario
                        using (StreamWriter sw = File.AppendText(@t1 + @"\" + d1))// abro comando de
escrita no arquivo recem criado
                        {
                            sw.WriteLine(stringEncriptada); //transfiro as infos da variavel para o
arquivo txt novo.
                        }                                   //Finalizo.

                        break;
                }

                Console.WriteLine("\n Pressione qualquer tecla para sair");
                Console.ReadLine();
                goto ret; // volta para o começo.
            }
            if (gat1 == 4) // descriptografar arquivo!
            {
            ret3:
                Console.Clear();
                Console.WriteLine("\n Aqui estão os arquivos disponiveis, localize o arquivo
desejado.\n\n");
                string t2 = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
                string[] arquivos = Directory.GetFiles(@t2, "*.txt", SearchOption.TopDirectoryOnly);
                for (int i = arquivos.Length - 1; i >= 0; --i)
                    // Caminho do arquivo .txt encontrado
```

```
                    Console.WriteLine(arquivos[i]); // exibindo na tela todos os arquivos txt
encontrados
                Console.WriteLine(" \n\n Agora digite o nome corretamente. \n");
                Console.WriteLine("(Não necessario colocar .txt nem especificar o caminho do
arquivo.)");
                string z1 = Console.ReadLine();
                Console.WriteLine("\n\nO arquivo escolhido foi {0}", z1);
                Console.WriteLine("\n\nSe estiver correto, escolha a opção desejada... \n[1] Continuar.
\n[0] Digitar novamente");
                gat1 = int.Parse(Console.ReadLine());
                switch (gat1)
                {
                    case 1:
                        Console.Clear();

                        Console.WriteLine("\n Nome da mensagem escolhida é : {0}", z1);
                        string mcpt2 = System.IO.File.ReadAllText(@t2 + @"\" + z1 + ".txt");

                        using (StreamWriter mcpt = File.AppendText(z1)) // obtendo o texto do arquivo
escolhido
                        {
                            mcpt.WriteLine(mcpt2); // transferindo o texto para uma variavel, para
poder ser usado;
                        }

                        Console.WriteLine("\n Mensagem criptografada é : {0}", mcpt2);


                        //*****************

                        Console.WriteLine("\n\n Agora digite a CHAVE de segurança para decriptar a
msg.");
                        string ssn = Console.ReadLine();


                        stringDescriptografada = DecryptString(mcpt2, ssn);

                        Console.WriteLine("\n\n Deseja descriptografar agora? \n[1] Sim. \n[0] Não");
                        gat1 = int.Parse(Console.ReadLine());

                        switch (gat1)
                        {
                            case 1:

                                Console.WriteLine("\n \n \n O conteudo da mensagem é:  {0}",
stringDescriptografada);
                                Console.WriteLine("Pressione qualquer tecla para voltar.");
                                Console.ReadLine();
                                goto ret;

                            case 0:
                                goto ret;
                        }
                        break;


                    case 0:
                        Console.Clear();
                        goto ret3;
                }
            }
            if (gat1 == 5)
            {
                //para uso proprio, poder verificar em tempo real se o codigo esta como deveria.
                Console.Clear();     //limpa a tela e exibe as infos abaixo

                if (!File.Exists(@"mensagem.txt")) //testa se o arquivo com a msgm existe; caso não,
                {
                    if (!File.Exists(@"Chave.txt.txt"))//testa se o arquivo com a chave existe;
                    {
```

```
                        if (!File.Exists(@"MengCrypt.txt")) //testa se o arquivo com a criptografia
existe
                        {
                            Console.WriteLine("Nenhum arquivo criado ainda... Pressione qualquer tecla
para voltar");
                            Console.ReadLine();  // caso nenhum dos arquivos exista ainda; exibe a
mensagem acima e volta para a tela inicial.
                            goto ret;
                        }
                    }
                }
                string t1 = System.IO.File.ReadAllText(@"mensagem.txt"); //joga tudo que esta no
arquivo txt mensagem na variavel t1
                System.Console.WriteLine("Mensagem (Arquivo): {0}", t1);           // e a mostra na
tela

                string t2 = System.IO.File.ReadAllText(@"Chave.txt");    //joga tudo que esta na
variavel txt chave na variavel t2
                System.Console.WriteLine("Chave (Arquivo): {0}", t2);              //e a joga na tela.

                string t3 = System.IO.File.ReadAllText(@"MengCrypt.txt");    //joga tudo que esta na
variavel txt chave na variavel t2
                System.Console.WriteLine("MengCrypt (Arquivo): {0}", t3);    //joga na tela

                Console.WriteLine("\nStg encriptada (Variavel): {0}", stringEncriptada);
//exibe a string (mensagem, ainda não decriptada) mensagem na tela
                Console.WriteLine("\nStg descriptografada (Variavel) : {0}",
stringDescriptograda);//exibe a string (mensagem ja criptografada ) senha na tela
                Console.WriteLine("\nDeseja limpar os dados? \n\n[1]Sim. \n\n[0]Não");
                int zis = int.Parse(Console.ReadLine());
                switch (zis)
                {
                    case 1:

                        File.Delete(@"mensagem.txt"); //deleta o arquivo txt da mensagem
                        File.Delete(@"Chave.txt");    //deleta o arquivo txt da chave
                        File.Delete(@"MengCrypt.txt");//deleta o arquivo txt da mensagem criptografada
                        stringEncriptada = " ";       //zera a string que armazena a mensagem
                        stringDescriptograda = " ";   //zera a string que armazena a mensagem
criptografada
                        //parte da rotina para testes, limpar o arquivo para recomeçar, sem precisar
fechar o programa;
                        File.Create(@"mensagem.txt").Close(); //cria novamente o txt da mensagem
                        File.Create(@"Chave.txt").Close();    //cria novamente o txt da chave
                        File.Create(@"MengCrypt.txt").Close();//cria novamente o txt para armazenar o
texto criptografado
                        Console.WriteLine("Limpo...");     // avisa que o processo foi finalizado
                        Console.ReadLine();                //espera alguma ação o usuario, para voltar
ao menu inicial
                        goto ret;                          //voltando...

                    case 0:

                        Console.WriteLine("\n\nPressione qualquer tecla para voltar ao menu
principal");
                        Console.ReadLine();//modo de voltar rapido para o menu principal.
                        goto ret;

                }
            }
            if (gat1 == 0)
            {
                goto fim; //pula ja para o encerramento do programa
            }
            else
            {
                Console.WriteLine("Opção não existe"); // tentando evitar break;
                Console.ReadLine();
                goto ret;
            }
```

```
        a1:
            Console.Clear();

            Console.Clear();
            Console.WriteLine("\n********************Criptografado********************\n");
            //string Crypt = System.IO.File.ReadAllText(@"Encrypted.txt");
            Console.WriteLine("\n{0}", stringEncriptada);
            Console.WriteLine("\nEste e seu texto criptografado");
            Console.WriteLine("\n\n\n\n\n\n Pressione qualquer tecla para voltar. \n\n");
            Console.ReadLine();
            goto ret; // volta para o menu principal


        fim: // retorno comando de sair;
            //final do programa;

            //Por questão de segurança, logicamente os arquivos de texto são apagados quando o programa
e fechado;
            File.Delete(@"mensagem.txt");
            File.Delete(@"Chave.txt");
            File.Delete(@"MengCrypt.txt");

            Console.ReadKey();
        }

    }
}
```

8. CONCLUSÃO

Os autores do presente trabalho compreendem que cumpriram com os objetivos da proposta de Atividade Prática Supervisionada (APS) do segundo semestre de 2016 do curso de Ciências da Computação da Universidade Paulista (UNIP).

Esse projeto se encerra com o entendimento de que ele só se tornou possível graças à atenção às exigências contidas nele – exigências essas sintetizadas no seguinte desafio: o de aliar vasto e especializado referencial teórico sobre criptografia com a aplicação da técnica criptográfica *3 DES* em uma simulação de um problema de viés socioambiental.

9. REFERÊNCIAS BIBLIOGRÁFICAS

O'CONNOR, Joseph. **Manual de Programação Neurolinguística.** 11.ed. Rio de Janeiro: Qualitymark Editora, 2015. p. 28 – 29.

UNIP. **Manual de APS.** 2016. p. 2.

TENENBAUM, A.S.; WETHERALL, D. **Rede de Computadores.** 4.ed. 15. reimpressão. São Paulo: Pearson Education, 2003. p. 24– 775.

STALLINGS, William. **Criptografia e Segurança de Redes.** 6.ed. São Paulo: Pearson Education do Brasil, 2015.p. 23-27; p. 136 – 150.

STALLINGS, William; BROWN, Laurie. **Segurança de Computadores.** 2. ed. São Paulo: Campus, 2014. p.117 – 118.

RODRIGUES, Rodrigo Vollo Antonio. **Uma análise acerca dos protocolos WEP e WPA utilizados como mecanismos de segurança nas redes wireless.** 1.ed. Franca, SP: Ed. do Autor, 2016. p. 114 – 117.

CAVALCANTI, Rafael; DE CARVALHO, Ricardo. ***3 DES* – Algoritmo de Decifração e Cifração em *Hardware*.** Disponível no seguinte sítio na *Web* (*website*): <http://fei.edu.br/70anos/simposio/trabalhos/El%C3%A9trica/3DES%20-%20Algoritmo%20de%20Decifra%C3%A7%C3%A3o%20e%20Cifra%C3%A7%C3%A3o%20em%20Hardware.pdf>. Acesso em: 13 set. 2016.

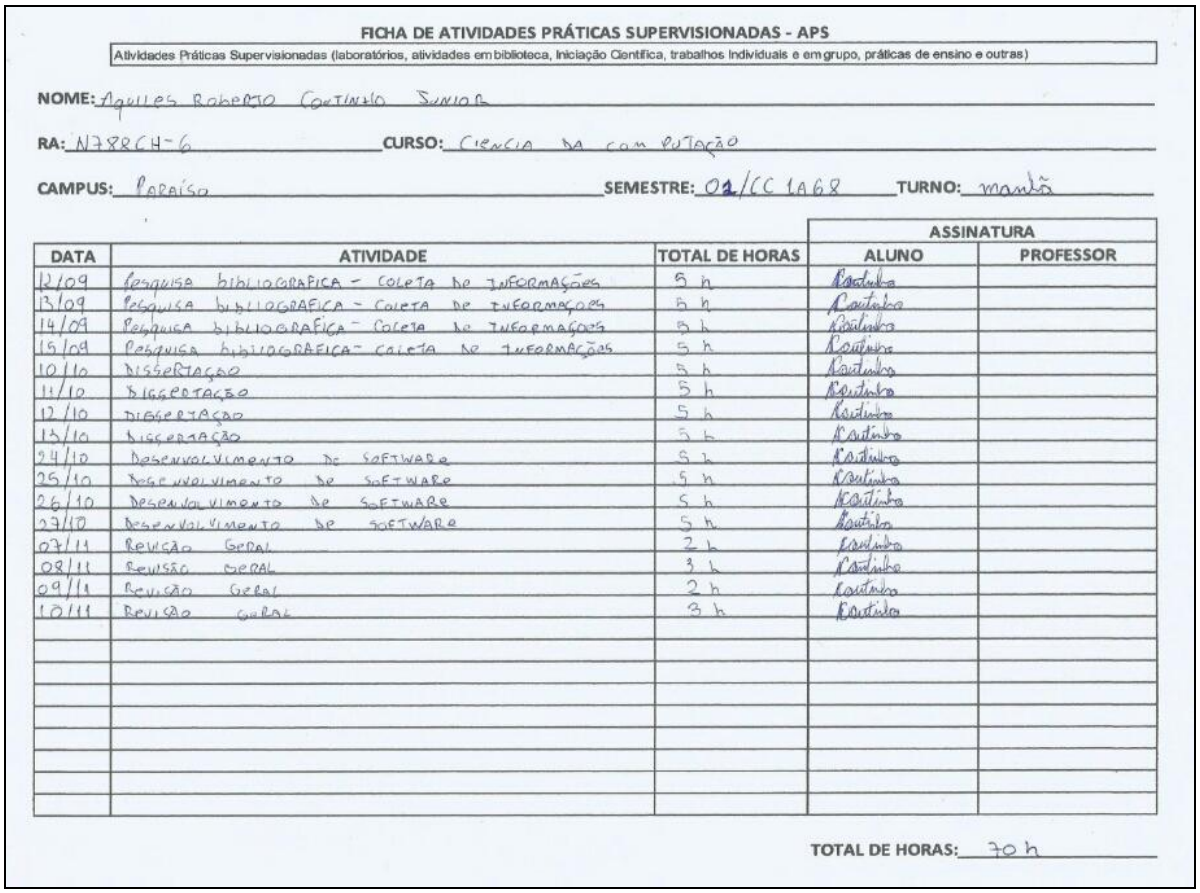
SOBRAL, Fábio. **Certificação Digital – Como garantir a segurança na transmissão ou no armazenamento da informação.** Disponível no *website*: <http:// http://biblioo.info/certificacao-digital/>. Acesso em: 14 set. 2016.

JOHN DOE, Precious. **Data Encryption Algorithm (DEA) Comparison.** Disponível em: < http://www.brighthub.com/computing/smb-security/articles/75099.aspx>. Acesso em: 14 set. 2016.

BRAGA, Alexandre; DAHAB, Ricardo. **Introdução à Criptografia para Programadores: evitando maus usos da criptografia em sistemas de *software*.** Disponível em: < http://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=ceseg:2015-sbseg-mc1.pdf> Acesso em: 15 set. 2016.

10. FICHAS DE ATIVIDADES SUPERVISIONADAS

Imagem 32 – Aquiles Roberto Coutinho Junior.



FICHA DE ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

Atividades Práticas Supervisionadas (laboratórios, atividades em biblioteca, Iniciação Científica, trabalhos Individuais e em grupo, práticas de ensino e outras)

NOME: Aquiles Roberto Coutinho Junior

RA: N78RCH-6 CURSO: Ciencia da computação

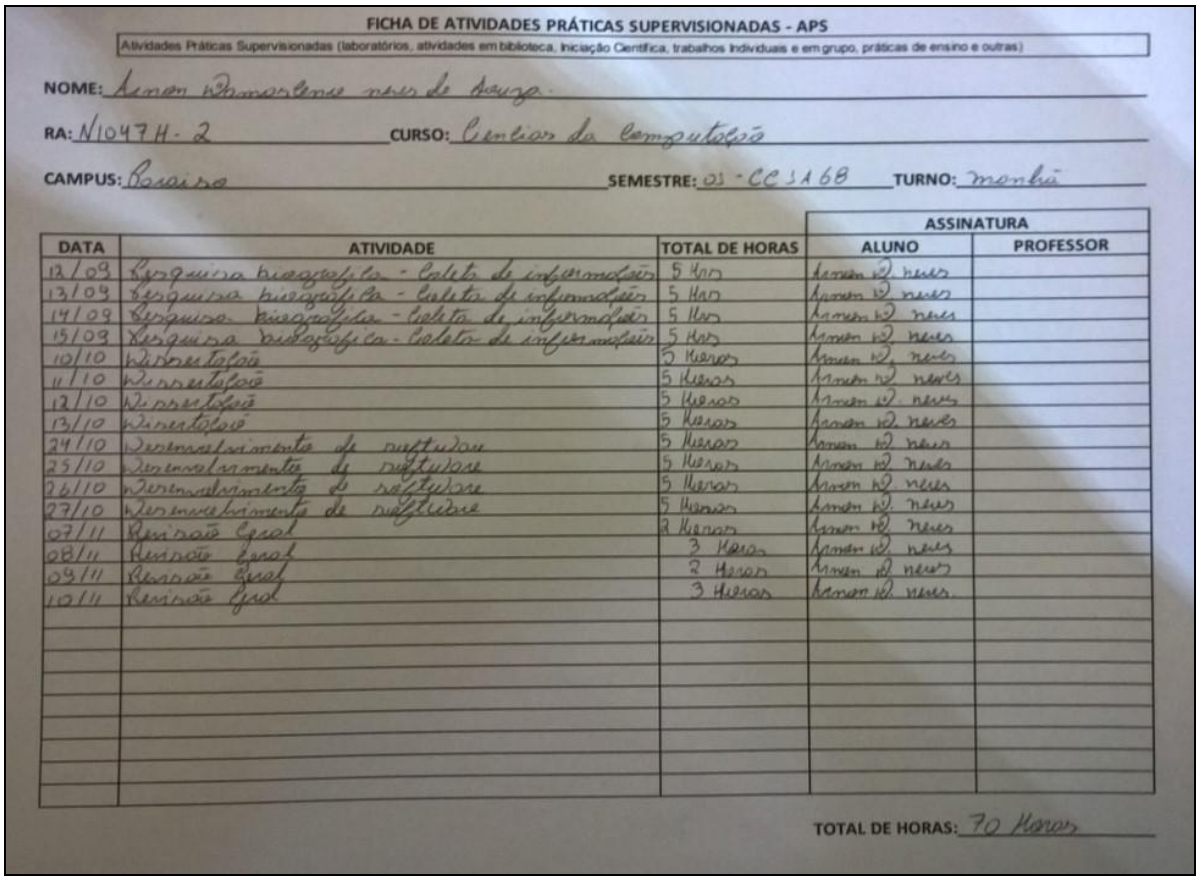
CAMPUS: Paraíso SEMESTRE: 02/CC 1A68 TURNO: manhã

DATA	ATIVIDADE	TOTAL DE HORAS	ASSINATURA	
			ALUNO	PROFESSOR
12/09	Pesquisa bibliografica - coleta de informações	5 h	Coutinho	
13/09	Pesquisa bibliografica - coleta de informações	5 h	Coutinho	
14/09	Pesquisa bibliografica - coleta de informações	5 h	Coutinho	
19/09	Pesquisa bibliografica - coleta de informações	5 h	Coutinho	
10/10	Dissertação	5 h	Coutinho	
11/10	Dissertação	5 h	Coutinho	
12/10	Dissertação	5 h	Coutinho	
13/10	Dissertação	5 h	Coutinho	
24/10	Desenvolvimento de software	5 h	Coutinho	
25/10	Desenvolvimento de software	5 h	Coutinho	
26/10	Desenvolvimento de software	5 h	Coutinho	
27/10	Desenvolvimento de software	5 h	Coutinho	
07/11	Revisão Geral	2 h	Coutinho	
08/11	Revisão Geral	3 h	Coutinho	
09/11	Revisão Geral	2 h	Coutinho	
10/11	Revisão Geral	3 h	Coutinho	

TOTAL DE HORAS: 70 h

Fonte: Própria, 2016.

Imagem 33 – Arnon Damasceno Neves de Souza.



FICHA DE ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

Atividades Práticas Supervisionadas (laboratórios, atividades em biblioteca, iniciação Científica, trabalhos individuais e em grupo, práticas de ensino e outras)

NOME: Arnon Damasceno neves de Souza.

RA: N1047H-2 CURSO: Ciências da Computação

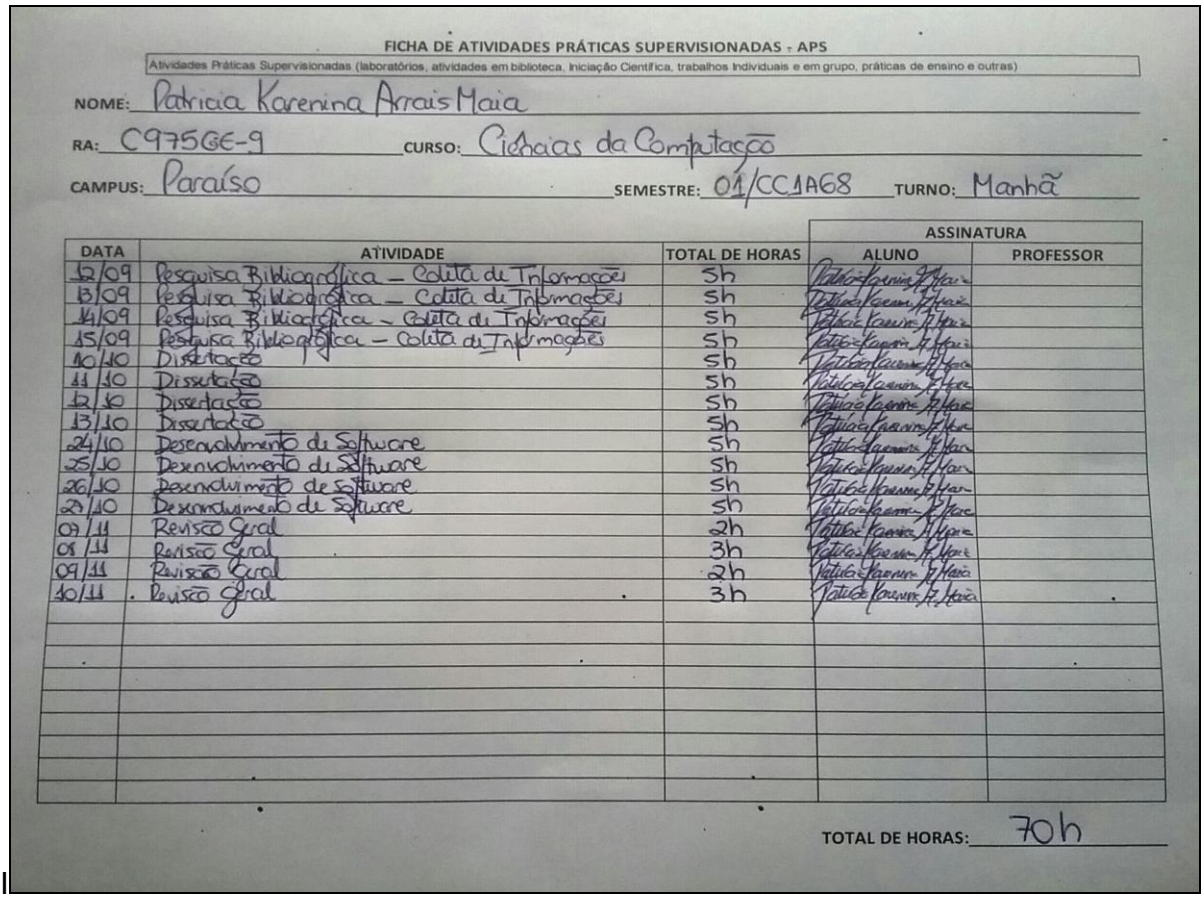
CAMPUS: Barreiro SEMESTRE: 01 - CC1A68 TURNO: manhã

DATA	ATIVIDADE	TOTAL DE HORAS	ASSINATURA ALUNO	ASSINATURA PROFESSOR
12/09	Pesquisa bibliográfica - Coleta de informações	5 Horas	Arnon D. neves	
13/09	Pesquisa bibliográfica - Coleta de informações	5 Horas	Arnon D. neves	
14/09	Pesquisa bibliográfica - Coleta de informações	5 Horas	Arnon D. neves	
15/09	Pesquisa bibliográfica - Coleta de informações	5 Horas	Arnon D. neves	
10/10	Dissertação	5 Horas	Arnon D. neves	
11/10	Dissertação	5 Horas	Arnon D. neves	
12/10	Dissertação	5 Horas	Arnon D. neves	
13/10	Dissertação	5 Horas	Arnon D. neves	
24/10	Desenvolvimento de software	5 Horas	Arnon D. neves	
25/10	Desenvolvimento de software	5 Horas	Arnon D. neves	
26/10	Desenvolvimento de software	5 Horas	Arnon D. neves	
27/10	Desenvolvimento de software	5 Horas	Arnon D. neves	
07/11	Revisão Geral	3 Horas	Arnon D. neves	
08/11	Revisão Geral	3 Horas	Arnon D. neves	
09/11	Revisão Geral	2 Horas	Arnon D. neves	
10/11	Revisão Geral	3 Horas	Arnon D. neves	

TOTAL DE HORAS: 70 Horas

Fonte: Própria, 2016.

Imagem 34 – Patricia Karenina Arrais Maia.



FICHA DE ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

Atividades Práticas Supervisionadas (laboratórios, atividades em biblioteca, Iniciação Científica, trabalhos Individuais e em grupo, práticas de ensino e outras)

NOME: Patricia Karenina Arrais Maia

RA: C975GE-9 CURSO: Ciências da Computação

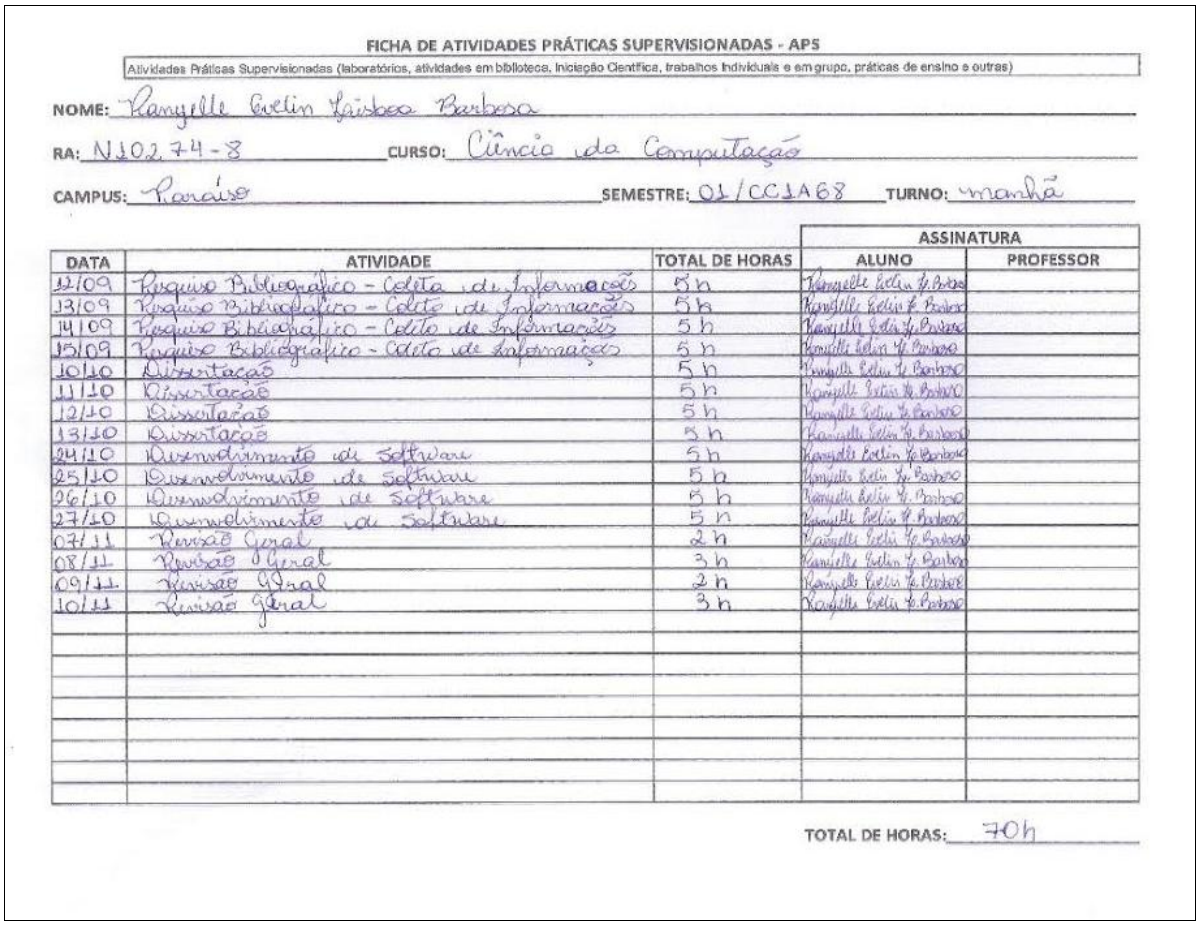
CAMPUS: Paraíso SEMESTRE: 01/CC1A68 TURNO: Manhã

DATA	ATIVIDADE	TOTAL DE HORAS	ASSINATURA	
			ALUNO	PROFESSOR
12/09	Pesquisa Bibliográfica – Coleta de Informações	5h		
13/09	Pesquisa Bibliográfica – Coleta de Informações	5h		
14/09	Pesquisa Bibliográfica – Coleta de Informações	5h		
15/09	Pesquisa Bibliográfica – Coleta de Informações	5h		
10/10	Dissertação	5h		
11/10	Dissertação	5h		
12/10	Dissertação	5h		
13/10	Dissertação	5h		
24/10	Desenvolvimento de Software	5h		
25/10	Desenvolvimento de Software	5h		
26/10	Desenvolvimento de Software	5h		
27/10	Desenvolvimento de Software	5h		
07/11	Revisão Geral	2h		
08/11	Revisão Geral	3h		
09/11	Revisão Geral	2h		
10/11	Revisão Geral	3h		

TOTAL DE HORAS: 70h

Fonte: Própria, 2016.

Imagem 35 – Ranyelle Evelin Lisboa Barbosa.



FICHA DE ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

Atividades Práticas Supervisionadas (laboratórios, atividades em biblioteca, Iniciação Científica, trabalhos individuais e em grupo, práticas de ensino e outras)

NOME: Ranyelle Evelin Lisboa Barbosa

RA: N102.74-8 CURSO: Ciência da Computação

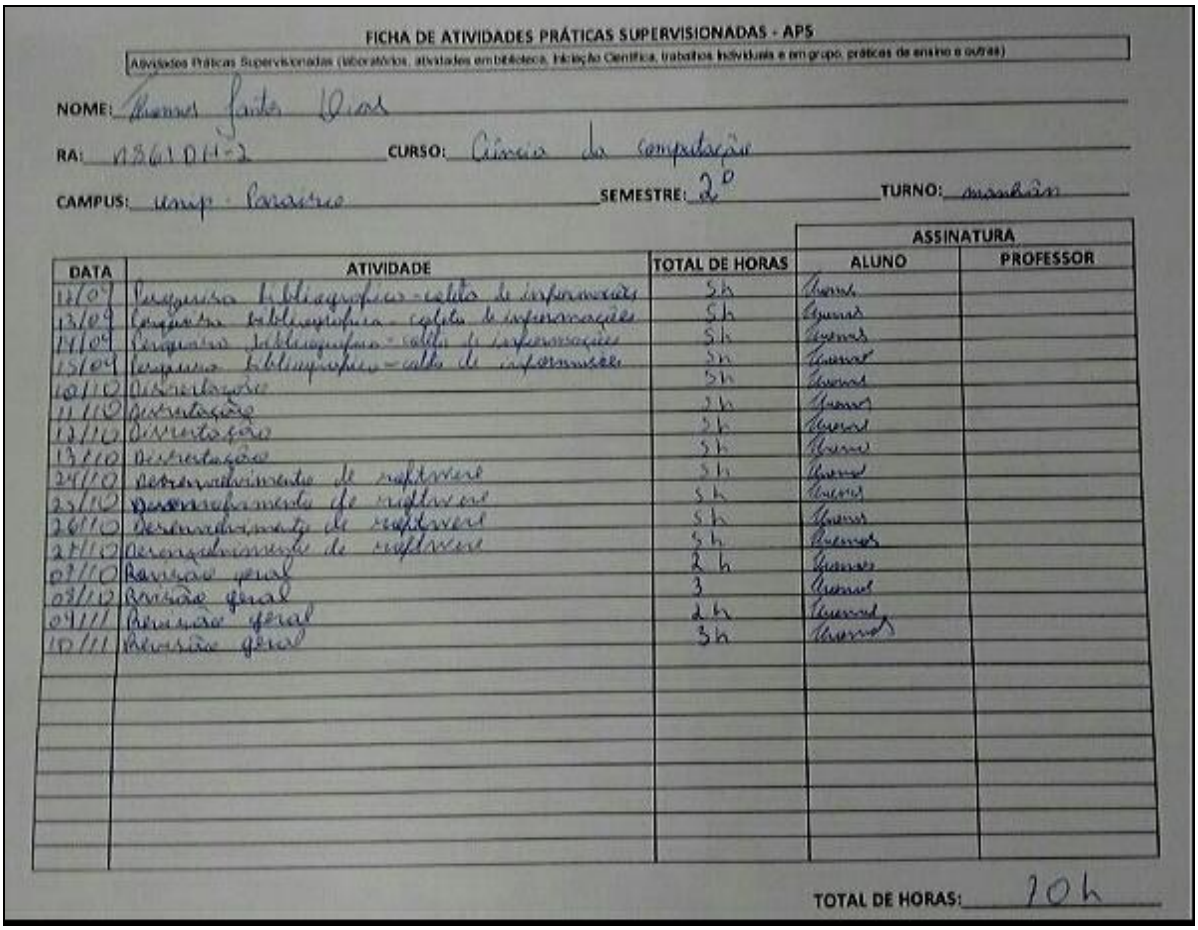
CAMPUS: Paraíso SEMESTRE: 01/CC1A68 TURNO: manhã

DATA	ATIVIDADE	TOTAL DE HORAS	ASSINATURA	
			ALUNO	PROFESSOR
12/09	Pesquisa Bibliográfico - Coleta de Informações	5h	Ranyelle Evelin L. Barbosa	
13/09	Pesquisa Bibliográfico - Coleta de Informações	5h	Ranyelle Evelin L. Barbosa	
14/09	Pesquisa Bibliográfico - Coleta de Informações	5h	Ranyelle Evelin L. Barbosa	
15/09	Pesquisa Bibliográfico - Coleta de Informações	5h	Ranyelle Evelin L. Barbosa	
10/10	Dissertação	5h	Ranyelle Evelin L. Barbosa	
11/10	Dissertação	5h	Ranyelle Evelin L. Barbosa	
12/10	Dissertação	5h	Ranyelle Evelin L. Barbosa	
13/10	Dissertação	5h	Ranyelle Evelin L. Barbosa	
24/10	Desenvolvimento de Software	5h	Ranyelle Evelin L. Barbosa	
25/10	Desenvolvimento de Software	5h	Ranyelle Evelin L. Barbosa	
26/10	Desenvolvimento de Software	5h	Ranyelle Evelin L. Barbosa	
27/10	Desenvolvimento de Software	5h	Ranyelle Evelin L. Barbosa	
07/11	Revisão Geral	2h	Ranyelle Evelin L. Barbosa	
08/11	Revisão Geral	3h	Ranyelle Evelin L. Barbosa	
09/11	Revisão Geral	2h	Ranyelle Evelin L. Barbosa	
10/11	Revisão Geral	3h	Ranyelle Evelin L. Barbosa	

TOTAL DE HORAS: 70h

Fonte: Própria, 2016.

Imagem 36 – Thomas Santos Dias.



FICHA DE ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

Atividades Práticas Supervisionadas (laboratórios, atividades em biblioteca, Iniciação Científica, trabalhos individuais e em grupo, práticas de ensino e outras)

NOME: Thomas Santos Dias

RA: N361DH-2 CURSO: Ciência da Computação

CAMPUS: Unip - Tatuapé SEMESTRE: 2º TURNO: manhã

DATA	ATIVIDADE	TOTAL DE HORAS	ASSINATURA ALUNO	ASSINATURA PROFESSOR
10/09	Pesquisa bibliográficas - coleta de informações	5h	Thomas	
13/09	Pesquisa bibliográficas - coleta de informações	5h	Thomas	
11/09	Pesquisa bibliográficas - coleta de informações	5h	Thomas	
15/09	Pesquisa bibliográficas - coleta de informações	5h	Thomas	
10/10	Diagramação	5h	Thomas	
11/10	Diagramação	5h	Thomas	
12/10	Diagramação	5h	Thomas	
13/10	Diagramação	5h	Thomas	
24/10	Desenvolvimento de software	5h	Thomas	
25/10	Desenvolvimento de software	5h	Thomas	
26/10	Desenvolvimento de software	5h	Thomas	
27/10	Desenvolvimento de software	5h	Thomas	
01/10	Revisão geral	2h	Thomas	
03/10	Revisão geral	3	Thomas	
09/11	Revisão geral	2h	Thomas	
10/11	Revisão geral	3h	Thomas	

TOTAL DE HORAS: 70h

Fonte: Própria, 2016.