

UNIVERSIDADE PAULISTA - UNIP
Instituto de Ciência e Tecnologia – ICET
Curso de Sistemas de Informação – Campus Tatuapé

ATIVIDADES PRÁTICAS SUPERVISIONADAS

Criptografia

Gabriel Costa Rossanesi	RA: N42905-4
Gabriel Y. C. Silqueira Perez	RA: F02DJB-0
Guylyan Russi Duarte	RA: D990JI-6
Richard Dos Santos Mendes Lins	RA: D89066-2
Renan Souza de Oliveira	RA: F055JJ-0

SÃO PAULO - SP
2019

UNIVERSIDADE PAULISTA – UNIP
Instituto de Ciência e Tecnologia – ICET
Curso de Sistemas de Informação – Campus Tatuapé

ATIVIDADES PRÁTICAS SUPERVISIONADAS

Criptografia

Pesquisa do Curso de Sistemas de Informação apresentado à Universidade Paulista, como requisito de avaliação na disciplina de Atividades Práticas Supervisionadas.

Profa. Orientadora: Ecila A. de Oliveira Migliori

Gabriel Costa Rossanesi	RA: N42905-4
Gabriel Y. C. Silqueira Perez	RA: F02DJB-0
Guylyan Russi Duarte	RA: D990JI-6
Richard Dos Santos Mendes Lins	RA: D89066-2
Renan Souza de Oliveira	RA: F055JJ-0

SÃO PAULO - SP
2019

LISTA DE FIGURAS

Figura 1 – Exemplo do conceito de criptografia. Fonte: Autoria própria ..	9
Figura 2 – Exemplo de criptografia simétrica. Fonte: Autoria própria.....	10
Figura 3 – Exemplo de criptografia assimétrica. Fonte: Autoria própria.	11
Figura 4 – Função gerador_primo(). Fonte: Autoria própria.....	17
Figura 5 – Função primo(). Fonte: Autoria própria.	18
Figura 6 – Função de Cadastro. Fonte Autoria própria.....	18
Figura 7 – Função totient(). Fonte: Autoria própria.	19
Figura 8- Função gerador_e(). Fonte: Autoria própria.....	19
Figura 9 – Função “calcular_chave_privada()”. Fonte: Autoria própria. .	19
Figura 10 – Função “criptografar()”. Fonte: Autoria própria.	20
Figura 11 – Usuário sendo cadastrado. Fonte: Autoria própria.	20
Figura 12 – Dados do usuário cadastrado. Fonte: Autoria própria.....	21
Figura 13 – Função Acessar() parte1. Fonte: Autoria própria.....	22
Figura 14 – Função Acessar() parte2. Fonte: Autoria própria.....	22
Figura 15 – Função Acessar() parte 3. Fonte: Autoria própria.....	23
Figura 16 – Função descifra(). Fonte: Autoria própria.	23
Figura 17 – Usuário realizando login. Fonte: Autoria própria.....	24
Figura 18 – Usuário realizando login para consultar. Fonte: Autoria própria.	24
Figura 19 – Sistema apresentando os acessos. Fonte: Autoria própria. .	25
Figura 20 – Usuário errando parte1. Fonte: Autoria própria.....	26
Figura 21 – Usuário errando parte2. Fonte: Autoria própria.....	26
Figura 22 – Função Main() parte1. Fonte: Autoria própria.	27
Figura 23 – Função Main() parte 2. Fonte: Autoria própria.	27
Figura 24 – Função Main() parte 3. Fonte: Autoria própria.	28
Figura 25 – Função Main() parte 4. Fonte: Autoria própria.	28
Figura 26 – Função Main() parte 5. Fonte: Autoria própria.	29
Figura 27 – Função Main() parte 6. Fonte: Autoria própria.	29
Figura 28 – Estrutura básica do sistema. Fonte: Autoria própria.....	32
Figura 29 – Como o sistema se comporta no cadastro. Fonte: Autoria própria.....	33
Figura 30 – Comportamento do sistema quando o usuário tenta acessar o navio. Fonte: Autoria própria.	34

Figura 31 – Comportamento do sistema quando o usuário tenta consultar os acessos. Fonte: Autoria própria. 34

Figura 32 – Comportamento do sistema quando o usuário deseja sair do sistema..... 35

LISTA DE ABREVIATURAS

AES – Advanced Encryption Standard

DES – Data Encryption Standard

MIT – Massachusetts Institute of Technology

NBS – National Bureau of Standards

NIST – National Institute of Standards in Technology

RSA – Rivest-Sharmir-Adleman

SUMÁRIO

1.	OBJETIVO	7
2.	INTRODUÇÃO.....	8
3.	CRIPTOGRAFIA – CONCEITOS GERAIS	9
3.1.	Criptografia simétrica	10
3.2.	Criptografia assimétrica.....	11
4.	TÉCNICAS CRIPTOGRÁFICAS MAIS UTILIZADAS E CONHECIDAS	12
4.1.	AES (Advanced Encryption Standard)	12
4.2.	RSA.....	13
4.3.	Blowfish.....	15
4.4.	DES.....	16
5.	DISSERTAÇÃO	17
5.1.	Estruturação, conceitos e fundamentação	29
5.2.	Benefícios em relação às técnicas anteriores	30
5.3.	Aplicações que fazem/fizeram uso da técnica	30
5.4.	Discussão comparativa entre esta técnica e outras conhecidas/utilizadas.....	31
5.5.	Vulnerabilidades e falhas.	31
5.6.	Melhorias propostas e/ou implementadas.....	32
6.	PROJETO (ESTRUTURA) DO PROGRAMA	32
7.	RELATÓRIO COM AS LINHAS DE CÓDIGO DO PROGRAMA	36
8.	REFERÊNCIAS BIBLIOGRÁFICAS	44
9.	FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS	46

1. OBJETIVO

Esta pesquisa 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 quilômetros 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. Após escolher o tipo de criptografia que melhor se adéqua a esta problemática, ela será implantada utilizando a linguagem de programação Python.

2. INTRODUÇÃO

Durante as grandes guerras que ocorreram no século XX ficou provado que o serviço de inteligência foi fundamental para o rumo que as guerras tomaram, sendo assim, nenhum dos lados envolvidos queriam ceder qualquer tipo de informação ao outro, pois, sabiam a vantagem que aquilo representava na batalha, a partir dessa necessidade então surgiu a criptografia, cujo seu principal objetivo era codificar as informações, para caso a mesma caísse em mãos inimigas não pudesse ser facilmente utilizada como vantagem na guerra.

Com o passar dos anos e os avanços tecnológicos surgiu a necessidade de desenvolver metodologias de criptografia cada vez mais seguras, porque atualmente a informação tem mais valor do que nunca teve em sua história, e a protege-la se tornou uma prioridade para todos, desde o ambiente na iniciativa privada, até organizações estatais.

Sabendo disso, é possível chegar no objetivo deste trabalho, que é desenvolver um algoritmo em Python 3 capaz de criptografar e descriptografar qualquer informação, para assim realizar o controle de acesso a um navio que está sobre segurança da marinha do Brasil com lixo tóxico em seu interior.

Para desenvolver esse trabalho foi necessário realizar pesquisas em fontes formais de informação, obtendo como resultados os conceitos básicos de criptografia, o surgimento e funcionamento de suas técnicas mais utilizadas, servindo assim como base para o desenvolvimento do algoritmo que realiza do controle de acesso do navio.

Inicialmente é abordado sobre os conceitos básicos de criptografia, no capítulo seguinte é tratado sobre as principais técnicas de criptografia mais famosas e mais utilizadas ao redor do mundo, após explicar sobre criptografia começa a dissertação, onde é abordado sobre a criptografia escolhida para o algoritmo, além de explicar os principais pontos do seu desenvolvimento. Depois é apresentado a estrutura do algoritmo, ou seja, como o sistema se comporta com a interação do usuário, e como funciona sua sequência de processos, e por fim é apresentado o código fonte do algoritmo.

3. CRIPTOGRAFIA – CONCEITOS GERAIS

Segundo Fiarresga (2010) a criptografia é uma técnica derivada da esteganografia, que tinha como objetivo ocultar a mensagem do inimigo, entretanto, diferentemente da esteganografia, a criptografia não esconde a existência da mensagem propriamente dita, mas sim o seu conteúdo, o tornando incompreensível usando uma determinada cifra, que faz com que só quem tenha conhecimento sobre a mesma, possa decifrar a comunicação.

Também sobre criptografia, a Cert (2017) em sua cartilha de segurança para a internet, a define como uma das mais famosas ferramentas de segurança para se defender dos potenciais riscos que o acesso à rede pode trazer. Isso é feito transformando mensagens em cifras, também conhecidas como códigos.

Já para Fiorim (2015) especialista de segurança da Trend Micro Brasil, em matéria escrita ao portal CanalTech, afirma que a criptografia é um conjunto de operações matemáticas que cifram dados de um usuário, para que só receptor possa ler. A criptografia é fundamental para aumentar a segurança de uma mensagem, pois, a mesma tem a capacidade de embaralhar o conteúdo, se tornando a maneira mais eficaz de esconder comunicações utilizando informações cifradas.

Exemplificando, o conceito de criptografia, na figura 1 têm-se um exemplo, neste caso cada letra do alfabeto equivale a um número, a letra “a = 1”, “b = 2”, “c = 3”, e assim consecutivamente até a letra z:

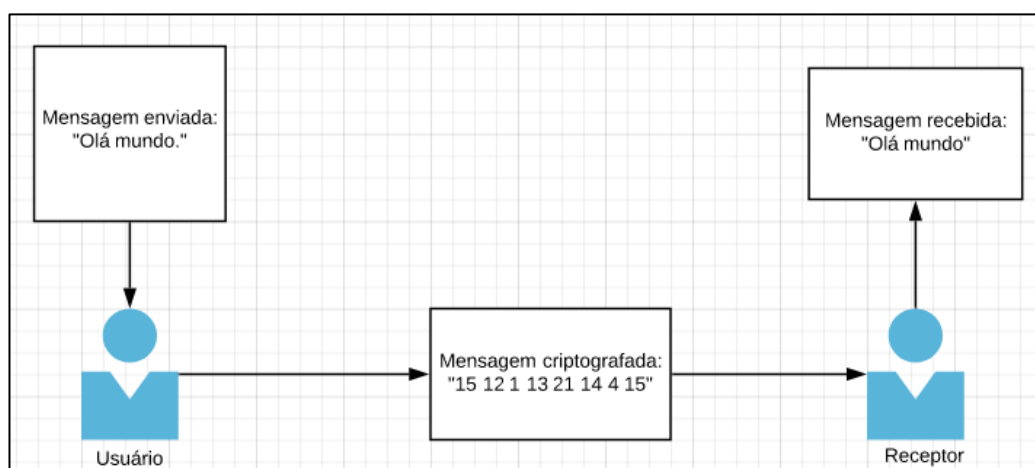


Figura 1 – Exemplo do conceito de criptografia. Fonte: Autoria própria

3.1. Criptografia simétrica

Segundo Fiarresga (2010) esse tipo de criptografia trabalha com uma única chave, tanto para criptografar, quanto para descriptografar. Isso faz com que o processo de descriptar uma comunicação, seja exatamente igual ao método que encriptou a mensagem.

Para Fiorim (2015) o tipo de criptografia simétrica tem a ver com a relação do remetente e do destinatário, onde existe uma única chave, a mesma que é usada para criptografar é usada para traduzir a mensagem.

Já para a Cert (2017) a criptografia de chave única, como é conhecida a simétrica, usa a mesma chave para criptografar quando para decodificar a mensagem. Quando esse processo é feito por apenas uma pessoa não é necessário compartilhar essa chave, mas quando essa operação é feita por pessoas diferentes é necessário que a chave seja combinada entre elas por um meio de comunicação seguro, o que mantém a confidencialidade da chave.

A figura 2 mostra como funciona a criptografia simétrica:

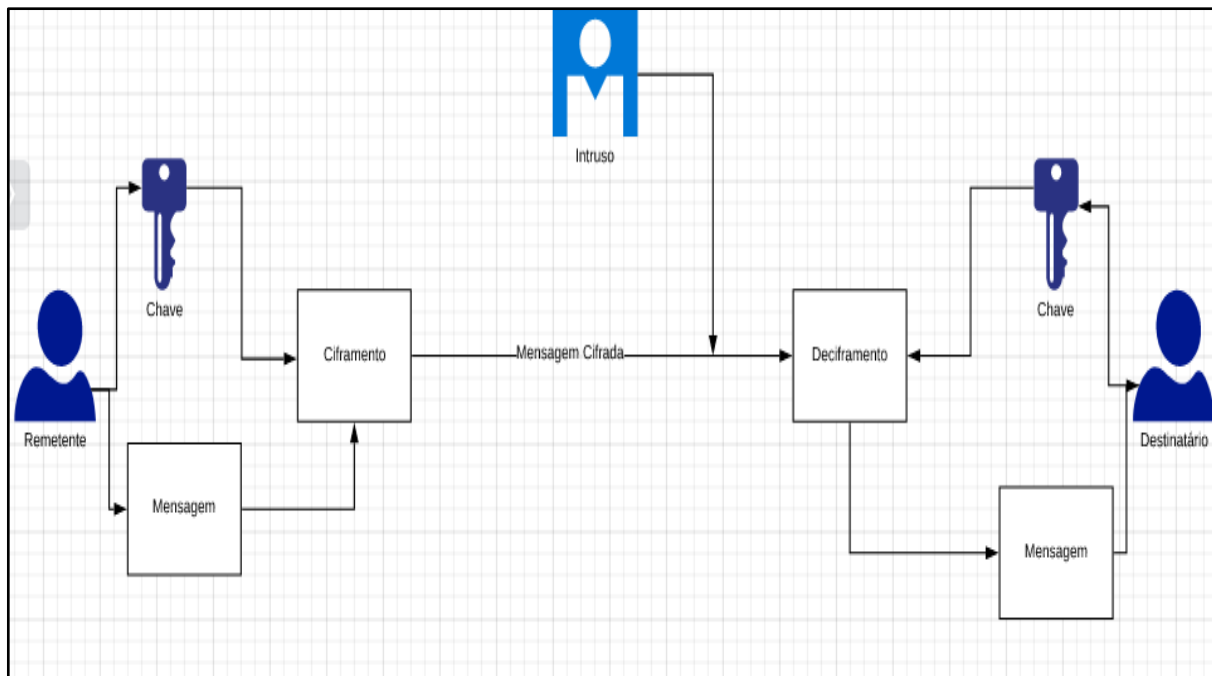


Figura 2 – Exemplo de criptografia simétrica. Fonte: Autoria própria.

3.2. Criptografia assimétrica

Segundo Fiorim (2015) a criptografia assimétrica é um tipo de codificação que utiliza uma dupla de chaves, denominadas de chave pública e privada, a primeira é usada para criptografar, já a segunda é necessária para a decodificação.

De acordo com a Cert (2017) a criptografia assimétrica chamada também de criptografia de chave pública, utiliza um par de chaves que são distintas. Quando uma mensagem é criptografada com uma das chaves, apenas a outra consegue decodificá-la.

Já para Fiarresga (2010) a criptografia assimétrica é uma espécie de criptografia que para ser manuseada necessita de duas chaves, uma pública e outra privada, onde uma tem a função de criptografar e a outra de decodificar a mensagem.

A figura 3 demonstra como funciona esse tipo de criptografia:

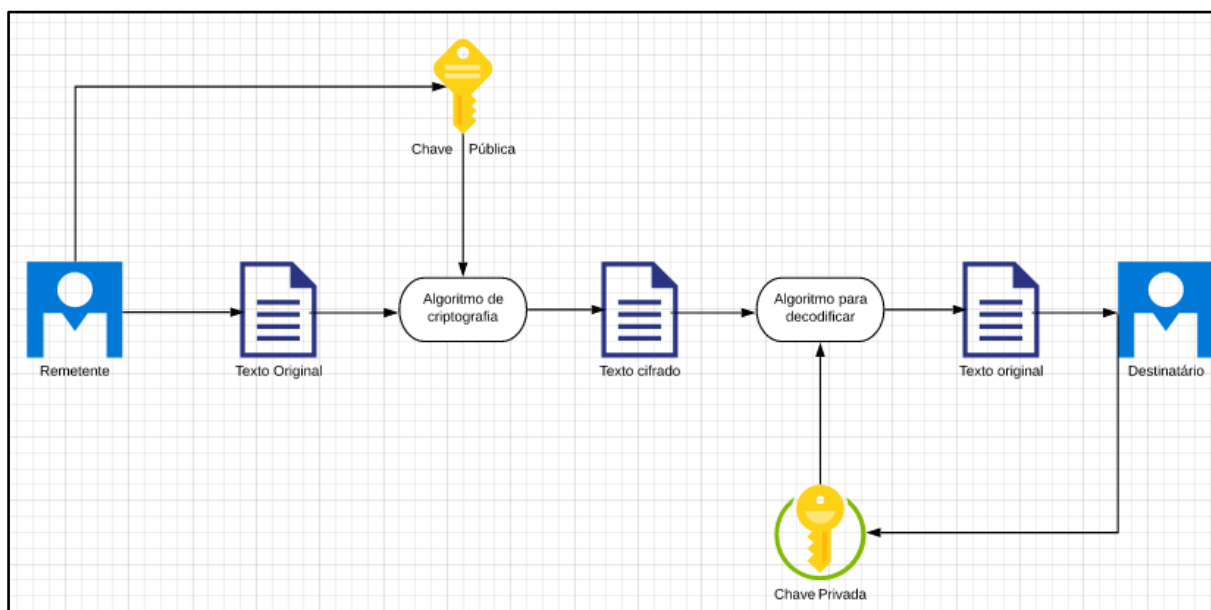


Figura 3 – Exemplo de criptografia assimétrica. Fonte: Autoria própria.

4. TÉCNICAS CRIPTOGRÁFICAS MAIS UTILIZADAS E CONHECIDAS

Nesse capítulo serão abordadas as técnicas de criptografia mais utilizadas e conhecidas.

4.1. AES (Advanced Encryption Standard)

Segundo o autor Campos (2008) o AES (*Advanced Encryption Standard*) surgiu quando o NIST (*National Institute of Standards in Technology*) abriu um concurso para criar um novo algoritmo padrão para a criptografia, com a exigência de que o ganhador não teria nenhum direito de propriedade intelectual sobre o algoritmo. Em 1998, o NIST escolheu 15 candidatos, entretanto muitos desses algoritmos tinham falhas e fraquezas e outros eram muito extensos e lentos, a lista então foi reduzida a 5 algoritmos em 1999, estes sendo MARS, RC6, Rijndael, Serpent e Twofish. No ano de 2000 a NIST escolheu o algoritmo de Rijndael como vencedor.

De acordo com Oliveira (2012) o AES contém sua amplitude de bloco fixo em 128 bits, e sua chave com três variáveis de tamanho, 128, 192 ou 256 bits. Uma das vantagens do AES, além de sua segurança é que ele consegue ser ágil tanto em software, quanto em hardware, sendo fácil de executar e exigindo pouca memória.

Ainda sobre o AES, Morale (2010) afirma que essa técnica funciona sobre um arranjo bidimensional de bytes com posições em 4x4, chamado de estado. No processo de criptografar o AES tem quatro etapas, uma de permutação e outras três de substituição:

1. SubBytes: Usa uma caixa-S para fazer a substituição linear byte a byte no bloco;
2. ShiftRows: É a etapa de permutação simples de modificação nas linhas do estado, cada fileira do estado é movida de um determinado número de posições;
3. MixColumns: É um procedimento nas colunas do estado. Fazendo um misto dos dados com cada coluna da matriz de estados;
4. AddRoundKey: Nessa operação os bytes do estado são combinados com a subchave própria do turno (RoundKey), onde cada subchave se deriva da chave principal usufruindo do algoritmo de agendamento de chaves.

4.2. RSA

Segundo Oliveira (2012) o RSA é um tipo de criptografia que usa um algoritmo assimétrico, o seu nome deriva dos seus criadores: Ron Rivest, Adi Shamir e Len Adleman, que o criaram em 1977 no MIT (*Massachusetts Institute of Technology*). Na atualidade é o algoritmo de chave pública mais utilizado, sendo um dos mais poderosos do momento. O RSA usa números primos, sua consistência se baseia na facilidade de multiplicar dois números primos para se ter um terceiro número, porém, se torna muito difícil de recuperar os dois números primos usados, a partir do terceiro gerado.

De acordo com Fiarresga (2010) se inicia com a criação de uma chave pública e uma privada, através das seguintes etapas:

1. É necessário escolher dois números primos, que serão denominados de p e q ;
2. Após isso é preciso calcular $n = pq$;
3. Agora deve-se calcular $\varphi(n) = (p - 1)(q - 1)$;
4. Feito isso é necessário escolher um número inteiro b , onde, $1 < b < \varphi(n)$ e m.d.c. $(b, \varphi(n)) = 1$;
5. Após isso é preciso selecionar um número inteiro a , onde $1 < a < \varphi(n)$ e $a \times b = 1 \pmod{\varphi(n)}$.

Para entender melhor como funciona esse tipo de algoritmo, abaixo será apresentado um exemplo, um indivíduo X cria uma chave pública e outra privada:

1. Geralmente por motivos de segurança, os números primos escolhidos são grandes, mas para esse exemplo serão usados números baixos com a finalidade de facilitar o entendimento; $p = 13$ e $q = 17$;
2. E $n = 221$;
3. Agora $\varphi(221) = 192$;
4. O número b escolhido foi 5; $1 < b < \varphi(221)$;
5. O número a escolhido foi 77; $5 \times 77 = 1 \pmod{192}$.

Nesse exemplo apresentado a chave pública é $(221, 5)$ e a chave privada é $(221, 77)$, sendo assim qualquer indivíduo Y pode cifrar e enviar uma mensagem para

o indivíduo X utilizando a chave pública, mas somente com a chave privada o indivíduo X pode decifrar a mensagem.

4.3. Blowfish

Segundo Silva (2018) o Blowfish foi inventado por Bruce Schneier, esse tipo de criptografia é baseada em chave de bloco. Também como o DES o seu algoritmo faz 16 iterações de uma mesma sequência de codificação sobre blocos de dados de 64 bits, funcionando ao mesmo tempo, sobre a metade esquerda e direita, respectivamente conhecidas como L e R, onde cada uma contém 32 bits. Entretanto, diferentemente do DES, o Blowfish no processo de codificação utiliza todos os 64 bits de um bloco de dados, para cada etapa da codificação.

Ainda sobre o Blowfish, Bugatti (2005) afirma que esse algoritmo é composto por duas fases, a primeira é conhecida como expansão de chave e a segunda fase é nomeada de cifragem de dados. A etapa de expansão de chave transforma uma chave de no máximo 448 bits em um grupo de subchaves, totalizando assim 4168 bytes.

Já a cifragem de dados acontece por meio de uma rede de Feistel com 16 iterações. Nessa fase é utilizado um método de cifragem fraco, usando várias iterações, de um jeito que acaba tornando o processo complexo. As iterações baseiam-se em uma permutação dependente apenas da chave e de uma troca dependente da chave e dos dados envolvidos. Todos os procedimentos feitos são XORs (exclusivos) e somas sobre palavras de 32 bits, onde as funções usadas nas iterações, XORs e somas, são funções eficientes e simples quando são efetuadas pelos microprocessadores. Além das funções simples como o XOR, ADD, e Deslocamentos, no Blowfish, também são usadas estruturas criptográficas nomeadas de “S-boxes” e “Rede Feistel”.

O autor Bugatti (2005) ainda diz que a “S-box” é basicamente uma caixa de substituição onde é colocado um texto inicial, sobre esse são realizadas trocas com o objetivo de embaralhar, aumentando assim a dificuldade de sua interpretação.

Segundo Silva (2019) a rede de Feistel é um bloco separado em duas partes, esquerda e direita, e somente uma delas é utilizada a cada “round”, já no “round” seguinte os lados são trocados se tornando assim opostos.

4.4.DES

Segundo Hinz (2000) após a segunda grande guerra, a denominada criptografia moderna ter avançado consideravelmente, até os anos 70 não existia um padrão para que equipamentos diferentes pudessem compartilhar entre si dados codificados. O surgimento de um padrão faria com que a proteção de dados se tornasse mais confiável e barata. Com esse objetivo, o NITS que naquela época era nomeado de NBS (*National Bureau of Standards*), em 1973, fez uma convocação a comunidade para apresentar a proposta de um algoritmo padrão. Mesmo com o esforço da comunidade, nenhum algoritmo chegava próximo das especificações do NIST, o algoritmo deveria:

1. Ter um nível alto de segurança;
2. Ter uma documentação de fácil compreensão;
3. A segurança do algoritmo deveria estar presente na chave, e não no sigilo do mesmo;
4. Ter disponibilidade a todos os usuários;
5. Ter a flexibilidade de se adequar a diversas aplicações;
6. Ser viável economicamente em dispositivos eletrônicos;
7. Ter eficiência;
8. Estar disponível para validação;
9. Ter a capacidade de ser exportável.

Sendo assim, no ano seguinte foi realizada uma nova convocação, onde um algoritmo desenvolvido pela IBM foi escolhido.

De acordo com o autor Câmara (2009) o DES (*Data Encryption Standard*) se tornou o algoritmo de chave simétrica mais difundido pelo mundo, e mesmo permitindo algo em torno de setenta e dois quadrilhões de combinações, o seu tamanho de chave que é de 56 bits é considerado pequeno, a prova disso é que em 1997 ele foi burlado em um desafio feito na internet. Esse foi um dos principais motivos que fez o NIST buscar um novo algoritmo, o AES apresentado no capítulo [4.1](#).

5. DISSERTAÇÃO

Nesse capítulo será abordado diretamente sobre a técnica criptográfica escolhida.

A técnica de criptografia utilizada no algoritmo de controle de acesso ao navio, foi a técnica RSA (Rivest-Shamir-Adleman), esse tipo de codificação usa o conceito de criptografia assimétrica, ou seja, uma chave pública é usada para criptografar, e somente uma chave privada pode realizar a decodificação como já mencionado no capítulo [4.2](#). O algoritmo de controle de acesso do navio utiliza seu sistema de criação de chaves, isso porque utilizando números primos grandes o trabalho do invasor se torna complicado, mesmo que o invasor consiga um método de chegar até os números primos que geraram a chave, ele será descoberto, pois, utilizando números primos grandes o processo de quebra da criptografia se torna lento, dando a possibilidade do mesmo ser descoberto, isso faz com que o invasor não tenha tempo útil de invadir o sistema.

O algoritmo foi desenvolvido em Python 3, e para realizar o processo de criptografia e decodificação, primeiro foi necessário gerar dois números primos usando a função “gerador_primo()”, essa função primeiro gera um número de forma randômica, no algoritmo o número gerado é entre 100 e 200, após isso esse número é passado como parâmetro a função “primo()” que verifica se o número gerado é primo ou não, as funções estão representadas nas figuras 4 e 5:

```
def gerador_primo():  
    # gera um numero primo aleatorio  
    while True:  
        x=random.randrange(100,200) # definindo o range de numeros  
        if(primo(x)==True):  
            return x
```

Figura 4 – Função gerador_primo(). Fonte: Autoria própria.

```
def primo(n):
    # verifica se o numero gerado é primo
    if (n <= 1):
        return False
    if (n <= 3):
        return True

    if (n%2 == 0 or n%3 == 0):
        return False

    i = 5
    while(i * i <= n):
        if (n%i == 0 or n%(i+2) == 0):
            return False
        i+=6
    return True
```

Figura 5 – Função primo(). Fonte: Autoria própria.

A função “gerador_primo()” é chamada duas vezes dentro da função “Cadastro()” representada na figura 6, que é onde a criptografia acontece. Primeiro é gerado um primo aleatório para “p” e outro para “q”, após isso é calculado o valor de “n”, multiplicando “p” e “q”:

```
def Cadastro(patente, login, senha):
    #Função responsável pelo cadastro de usuários
    p = gerador_primo() # gera um primo aleatório para P
    q = gerador_primo() # gera um primo aleatório para Q
    n = p * q # calculando n
    y = totient(p) # calculando totiente de P
    x = totient(q) # calculando totiente de Q
    totient_de_N = x * y # calculando totiente de N
    e = gerador_E(totient_de_N) # gerando o e E
    chave_publica = [n, e]
    print('Sua chave pública é {} e {}'.format(chave_publica[0], chave_publica[1]), end=" ")
    d = calcular_chave_privada(totient_de_N, e)
    print("e sua chave privada é {}".format(d))
    print("Guarde essas chaves, pois serão necessárias para acessar o navio.")
    print("Seu uso é pessoal, não é permitido seu compartilhamento.")
    senha_crip = criptografar(senha, e, n)
    senha_crip = str(senha_crip)
    gravando = open("cadastro.txt", 'a')
    gravando.write(patente + "|" + login + "|" + senha_crip + "|" + str(len(senha)) + "\n")
    gravando.close()
    status = True
    print("Cadastro realizado com sucesso!!!")
    return status
```

Figura 6 – Função de Cadastro. Fonte Autoria própria.

Depois de calcular “n”, é calculado o totiente de “p” e “q” utilizando a função “totient()” representada na figura 7:

```
def totient(numero):  
    # Calcula o totiente do numero primo  
    if(primo(numero)):  
        return numero-1  
    else:  
        return False
```

Figura 7 – Função totient(). Fonte: Autoria própria.

Após isso é calculado o totiente de “n”, multiplicando o totiente de “p” e “q”, o totiente de “n” é passado como parâmetro para a função “gerador_e()”, que gera um valor “e”, representada na figura 8:

```
def gerador_E(num):  
    #Gera um numero E aleatorio, obedecendo as condições  
    def mdc(n1,n2):  
        rest = 1  
        while(n2 != 0):  
            rest = n1%n2  
            n1 = n2  
            n2 = rest  
        return n1  
  
    while True:  
        e = random.randrange(2,num)  
        if(mdc(num,e) == 1):  
            return e
```

Figura 8- Função gerador_e(). Fonte: Autoria própria.

O valor de “n” e “e” são a chave pública do usuário, agora é preciso calcular a chave privada desse usuário, para isso é necessário o totiente de “n” e o valor “e” gerado, que são usados como parâmetros para a função “calcular_chave_privada()” representada na figura 9:

```
def calcular_chave_privada(toti, e):  
    #Função responsável por calcular a chave privada  
    d = 0  
    while(mod(d * e, toti)!=1):  
        d += 1  
    return d
```

Figura 9 – Função “calcular_chave_privada()”. Fonte: Autoria própria.

Agora o algoritmo já pode criptografar a senha, são passados para a função “criptografar()” representada na figura 10, a senha, o valor de “n” e o valor de “e” que são a chave pública do usuário, criptografando cada caractere da senha:

```
def criptografar(senha,e,n):  
    #Função responsável por descriptografar  
    tam = len(senha)  
    i = 0  
    lista = []  
    while(i < tam):  
        letra = senha[i]  
        k = ord(letra)  
        k = k**e  
        d = mod(k,n)  
        lista.append(d)  
        i += 1  
    return lista
```

Figura 10 – Função “criptografar()”. Fonte: Autoria própria.

A função “criptografar()” retorna uma lista com todos os caracteres criptografados, devido a isso é necessário converter essa lista em string, para poder a gravar no arquivo “cadastro.txt” juntamente com o login, a patente, e o tamanho da senha, a figura 11 mostra o sistema cadastrando um usuário:

```
MARINHA DO BRASIL  
Marinha do Brasil, protegendo nossas riquezas, cuidando da nossa gente.  
-----  
CONTROLE DE ACESSO.  
-----  
Escolha o número correspondente ao que você deseja realizar no sistema:  
1 - Entrar no navio;  
2 - Cadastrar;  
3 - Consultar acessos;  
4 - Sair.  
-----  
Opção:2  
-----  
CADASTRO  
-----  
digite sua patente:almirante  
Digite seu login:usuario123  
Digite sua senha:1234  
Sua chave pública é 85 e 51, e sua chave privada é 59.  
Guarde essas chaves, pois serão necessárias para acessar o navio.  
Seu uso é pessoal, não é permitido seu compartilhamento.  
Cadastro realizado com sucesso!!!  
-----
```

Figura 11 – Usuário sendo cadastrado. Fonte: Autoria própria.

Quando o usuário é cadastrado é retornado a ele sua chave pública e privada, pois, ele precisara de ambas para acessar o navio, ou consultar os acessos. A figura 12 mostra os dados do usuário cadastrado no arquivo “cadastro.txt”:

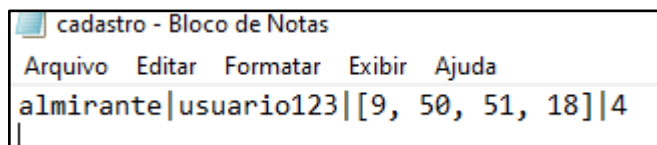


Figura 12 – Dados do usuário cadastrado. Fonte: Autoria própria.

Agora o usuário já pode acessar o navio, para isso ele precisará fornecer o primeiro valor do seu par de chave pública, que é o valor “n”, e também a sua chave privada, além da sua patente, login e senha. Esses dados fornecidos são usados como parâmetros para a função “Acessar()”, que primeiro pega todos os dados armazenados no arquivo “cadastro.txt” e os separa em listas, é criada uma lista somente com as patentes cadastradas, outra com os logins, outra com as senhas, e outra com o tamanho das senhas. Após a separação ter ocorrido, primeiro é verificado se a patente e o login daquele usuário está cadastrado, depois disso começa o processo de descriptografia da senha.

Para isso é usada a função “descifra()” que recebe como parâmetro cada caractere da senha criptografada, além do primeiro valor da chave pública (valor “n”) e a sua chave privada. Depois de cada caractere das senhas cadastradas serem descriptografadas, apenas uma das senhas vai estar correta, pois, cada usuário tem o seu par de chaves, o que interfere no processo de descriptografar.

Agora é preciso verificar se a senha informada pelo usuário é igual a alguma das senhas descriptografadas, para isso primeiro é necessário quebrar a senha informada pelo usuário em uma lista, e depois criar uma nova lista, que vai receber apenas os caracteres que estão presentes na lista que tem a senha informada e também estão presentes na lista que tem as senhas descriptografadas. Depois que esse processo é feito a senha criptografada é recriada e comparada a senha informada pelo usuário, se a patente, o login, e a senha estiverem corretos então o acesso é liberado, e a patente, além da data e hora, são gravados no arquivo “acessos.txt”, que armazena todos os usuários que entraram no navio, dessa forma é possível ter um controle, pois, se alguém fizer algo de ilegal dentro do navio é possível analisar quando isso aconteceu e quem são os principais suspeitos do ocorrido.

As figuras 13, 14, 15, e 16 mostram a função “Acessar()” e a função “descifrar()”:

```
def Acessar(pk1, p, l, s, d):
    #Função responsável por realizar o acesso no navio
    lista_total = [] # Recebe todos os dados cadastrados
    lista_total2 = [] # é usada para separar em login
    lista_login = [] # Lista contém somente os logins
    lista_patente = [] # lista contém somente as patentes
    lista_senha = [] # lista contém apenas as senhas
    lista_tamanho = []
    senha_int = [] #senhas convertidas em inteiros
    log = False #Sinaliza se o login do usuário bate
    pat = False #Sinaliza se a patente do usuário bate
    sen = False #Sinaliza se a patente do usuário bate
    senha_certa = [] #recebe a senha do usuário quebrando ela em letras
    senhafinal = "" # senha descriptografada
    status = True # se o acesso foi cadastrado com sucesso ou não
    arquivo = open("cadastro.txt", 'r')
    dados = arquivo.readlines()
    arquivo.close()

    for linha in dados:
        lista_total.append(linha.replace("\n", "")) # remove o '\n' da lista
    for dado in lista_total:
        lista_total2 = dado.split("|") #o separador é usado para dividir os dados dentro do arquivo patente|login|senha|len(senha)
        lista_patente.append(lista_total2[0]) # recebe somente as patentes cadastradas
        lista_login.append(lista_total2[1]) # recebe apenas os logins cadastrados
        lista_senha.append(lista_total2[2]) # recebe as senha criptografadas
        lista_tamanho.append(lista_total2[3]) # recebe o tamanho da senha
```

Figura 13 – Função Acessar() parte1. Fonte: Autoria própria.

```
    for patente in lista_patente:
        #verifica se a patente informada está cadastrada
        if p == patente:
            pat = True
    for login in lista_login:
        #verifica se o login informado está cadastrado
        if l == login:
            log = True
    for senha in lista_senha:
        # remove os separadores das senhas e descriptografa elas
        senha = senha.replace("[", "")
        senha = senha.replace("]", "")
        senha = senha.split(",")
        for num in senha:
            senha_int.append(int(num))
        senha_dec = descifra(senha_int, pk1, d)
    tamanho = len(s)
    for i in range(0, tamanho):
        senha_certa.append(s[i])

    lista_mega = [x for x in senha_dec if x in senha_certa] #atribui a lista mega somente as letras descriptografadas q
    cont = 0 #indice da string senha
    for letra in lista_mega:
        # recria a senha descriptografada
        if letra == s[cont]:
            senhafinal += letra
            cont += 1
```

Figura 14 – Função Acessar() parte2. Fonte: Autoria própria.

```

tamanho_senha = 0
for i in lista_tamanho:
    tamanho_senha = int(i)
    if senhafinal == s and tamanho_senha == len(s):
        # se a senha recriada for igual a senha informada atribui 'True' ao status da senha
        sen = True

if pat == True and log == True and sen == True:
    # se todos os dados do usuário forem válidos libera o acesso e grava no arquivo acessos
    print("Acesso liberado!!!")
    print("Bem-vindo {}".format(p))
    dh = datetime.now()
    dht = dh.strftime('%d/%m/%y %H:%M')
    acessos = open("acessos.txt", 'a')
    acessos.write(p + " - " + dht + "\n")
    acessos.close()
    time.sleep(5)
    exit()
else:
    print("Acesso negado!!!")
return sen

```

Figura 15 – Função Acessar() parte 3. Fonte: Autoria própria.

```

def descifra(cifra, n, d):
    #Função responsável por descriptografar
    lista = []
    i = 0
    tamanho = len(cifra)
    # texto=cifra ^ d mod n
    while i < tamanho:
        result = cifra[i]**d
        texto = mod(result,n)
        letra = chr(texto)
        lista.append(letra)
        i += 1
    return lista

```

Figura 16 – Função descifra(). Fonte: Autoria própria.

Quando o usuário quer consultar os acessos ao navio, o sistema informa que é necessário realizar login, após isso a função “Acessar()” é chamada e o mesmo processo descrito anteriormente acontece, entretanto não é registrado um acesso ao navio, apenas os acessos são listados, apresentando a patente do usuário que acessou o navio, juntamente com a data e a hora.

A figura 17 apresenta um usuário realizando login no sistema:

```
-----
MARINHA DO BRASIL
Marinha do Brasil, protegendo nossas riquezas, cuidando da nossa gente.
-----
CONTROLE DE ACESSO.
-----
Escolha o número correspondente ao que você deseja realizar no sistema:
1 - Entrar no navio;
2 - Cadastrar;
3 - Consultar acessos;
4 - Sair.
-----
Opção:1
-----
ACESSO
-----
Digite o primeiro valor do seu par de chaves públicas:85
agora digite sua patente:almirante
Digite seu login:usuario123
Digite sua senha:1234
Digite sua chave privada:99
Acesso liberado!!!
Bem-vindo almirante
```

Figura 17 – Usuário realizando login. Fonte: Autoria própria.

Já as figuras 18, e 19 mostram o comportamento do sistema quando o usuário deseja consultar os acessos:

```
-----
MARINHA DO BRASIL
Marinha do Brasil, protegendo nossas riquezas, cuidando da nossa gente.
-----
CONTROLE DE ACESSO.
-----
Escolha o número correspondente ao que você deseja realizar no sistema:
1 - Entrar no navio;
2 - Cadastrar;
3 - Consultar acessos;
4 - Sair.
-----
Opção:3
-----
CONSULTA
-----
Primeiro é necessário realizar login.
Digite o primeiro valor do seu par de chaves públicas:85
agora digite sua patente:almirante
Digite seu login:usuario123
Digite sua senha:1234
Digite sua chave privada:99
Acesso liberado!!!
Bem-vindo almirante
```

Figura 18 – Usuário realizando login para consultar. Fonte: Autoria própria.


```
-----
                                ACESSOS
-----
almirante - 05/11/19 16:34

almirante - 05/11/19 16:35

almirante - 06/11/19 11:38

almirante - 06/11/19 11:39

almirante - 06/11/19 11:40

almirante - 06/11/19 11:49

almirante - 06/11/19 11:51
-----
```

Figura 19 – Sistema apresentando os acessos. Fonte: Autoria própria.

E por fim a última função do sistema, a função “main()” que é a função principal. É nela que é desenhado o menu, além de realizar o controle sobre qual o momento correto de chamar as outras funções apresentadas acima, essa é a função que tem a principal interação com o usuário. Nela também são realizadas algumas validações, por exemplo, evita que o usuário digite letras em campos onde se deve digitar números, além de limitar a quantidade de erros. Se o usuário errar mais de três vezes na escolha da funcionalidade, no cadastro, no acesso ao navio, ou no login para a consulta dos acessos, à função “main()” encerra o algoritmo por questões de segurança, isso evita com que um possível invasor não tenha tantas chances assim para testar diferentes combinações para invadir o sistema. Outra validação que a função “main()” faz é no momento em que o usuário deseja se cadastrar, a função verifica se a patente informada pelo mesmo é válida ou não, no sistema apenas membros do quadro de oficiais superior ou acima podem se cadastrar no sistema, sendo assim as patentes válidas são: Capitão de Corveta, Capitão de Fragata, Capitão de Mar e Guerra, Contra-Almirante, Vice-Almirante, Almirante de Esquadra, e Almirante.

As figuras 20, e 21 demonstram o que acontece no sistema quando o usuário erra mais de três vezes, na figura o exemplo apresentado é no momento de acesso

ao navio, entretanto, como dito acima também funciona para o cadastro, escolha de funcionalidade, e consulta de acessos ao navio:

```
-----
MARINHA DO BRASIL
Marinha do Brasil, protegendo nossas riquezas, cuidando da nossa gente.
-----
CONTROLE DE ACESSO.
-----
Escolha o número correspondente ao que você deseja realizar no sistema:
    1 - Entrar no navio;
    2 - Cadastrar;
    3 - Consultar acessos;
    4 - Sair.
-----
Opção:1
-----
ACESSO
-----
Digite o primeiro valor do seu par de chaves públicas:36
agora digite sua patente:almirante
Digite seu login:usuario
Digite sua senha:123
Digite sua chave privada:12
Acesso negado!!!
Você só tem mais 2 tentativas de tentar acessar o navio.
-----
```

Figura 20 – Usuário errando parte1. Fonte: Autoria própria.

```
-----
ACESSO
-----
Digite o primeiro valor do seu par de chaves públicas:11
agora digite sua patente:usuario
Digite seu login:123
Digite sua senha:11
Digite sua chave privada:12
Acesso negado!!!
Você só tem mais 1 tentativas de tentar acessar o navio.
-----
ACESSO
-----
Digite o primeiro valor do seu par de chaves públicas:96
agora digite sua patente:almirante
Digite seu login:usuario
Digite sua senha:123
Digite sua chave privada:14
Acesso negado!!!
Você tentou acessar o navio mais de 3 vezes.
Por motivos de segurança o sistema irá encerrar.
-----
```

Figura 21 – Usuário errando parte2. Fonte: Autoria própria.

E por fim as figuras 22, 23, 24, 25, 26 e 27 abaixo apresentam a função “main()”:

```
def main():
    #Desenhando o menu
    dec = "-" * 80
    print(dec)
    print("MARINHA DO BRASIL".center(80))
    print("Marinha do Brasil, protegendo nossas riquezas, cuidando da nossa gente.".center(80))
    print(dec)
    print("CONTROLE DE ACESSO.".center(80))
    print(dec)
    escolha = 0
    cont = 0

    while escolha != 4:
        #Se a escolha do usuário for diferente de 4 que é a opção para sair o while roda
        print("Escolha o número correspondente ao que você deseja realizar no sistema:".center(80))
        print("1 - Entrar no navio:".center(80))
        print("2 - Cadastrar:".center(74))
        print("3 - Consultar acessos:".center(82))
        print("4 - Sair:".center(70))
        print(dec)
        #Tratamento de erro caso o usuário digitar uma letra
        try:
            escolha = int(input("Opção:"))
        except ValueError:
            print("Opção inválida, digite apenas números.")

        #A lista abaixo contém as patentes aptas a se cadastrar no sistema
        patentes_validas = ["capitãodecorveta", "capitãodefragata", "capitãodemareguerra", "contra-almirante", "vice-almirante", "almirante"]
```

Figura 22 – Função Main() parte1. Fonte: Autoria própria.

```
#Determinando as funções para cada funcionalidade
if escolha == 1:
    cont = 0
    v_acesso = False
    while cont < 3:
        print(dec)
        print("ACESSO".center(80))
        print(dec)
        try:
            pk1 = int(input("Digite o primeiro valor do seu par de chaves públicas:"))
            p = input("Agora digite sua patente:".lower())
            l = input("Digite seu login:")
            s = input("Digite sua senha:")
            d = int(input("Digite sua chave privada:"))
            if len(str(pk1)) > 5:
                pk1 = 1
                d = 1
                print("Valor de chave pública inválido.")
                print("Digite apenas o primeiro valor do seu par de chave pública.")
                print("Por exemplo se seu par de chave é 12345 e 678910, digite apenas 12345.")
            if len(str(d)) > 5:
                d = 1
                pk1 = 1
            v_acesso = Acessar(pk1, p, l, s, d, escolha)
        except ValueError:
            print("Nesse campo são permitidos apenas números.")
```

Figura 23 – Função Main() parte 2. Fonte: Autoria própria.

```

if v_acesso == False:
    #limita o usuário a tentar login somente três vezes
    cont += 1
    if cont == 3:
        print("Você tentou acessar o navio mais de 3 vezes.")
        print("Por motivos de segurança o sistema irá encerrar.")
        time.sleep(5)
        exit()
    print("Você só tem mais {} tentativas de tentar acessar o navio.".format(3-cont))

```

Figura 24 – Função Main() parte 3. Fonte: Autoria própria.

```

elif escolha == 2:
    print(dec)
    print("CADASTRO".center(80))
    print(dec)
    #Cadastro de usuário
    cont_tent = 0
    cadastro = False
    while cont_tent < 3:
        # Limita o usuário a apenas 3 tentativas
        cad_pat = input("Digite sua patente:".lower())
        cad_pat = cad_pat.replace(" ", "")
        # verifica se a patente inserida é válida
        for patente in patentes_validas:
            if cad_pat == patente:
                cadastro = True
        if cadastro != True:
            #Se a patente não for apta
            cont_tent += 1
            tent = 3 - cont_tent
            print("Sua patente não é apta a se cadastrar.")
            if cont_tent == 3:
                #Quando o usuário tenta cadastrar mais de três vezes
                print("O algoritmo está encerrando por motivos de segurança.")
                time.sleep(5)
                exit()
            print("Você ainda tem {} tentativas de cadastro.".format(tent))
    if cadastro == True:

```

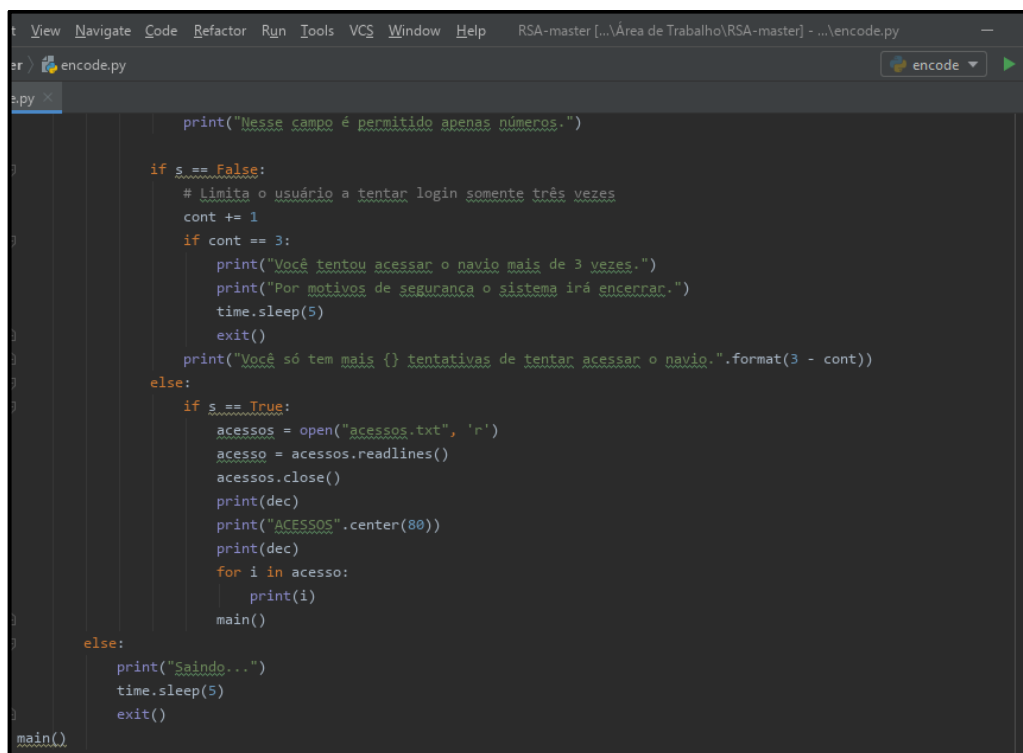
Figura 25 – Função Main() parte 4. Fonte: Autoria própria.

```

elif escolha == 3:
    print(dec)
    print("CONSULTA".center(80))
    print(dec)
    print("Primeiro é necessário realizar login.")
    s = False
    while cont < 3:
        try:
            pk1 = int(input("Digite o primeiro valor do seu par de chaves públicas:"))
            p = input("Agora digite sua patente:".lower())
            l = input("Digite seu login:")
            s = input("Digite sua senha:")
            dd = int(input("Digite sua chave privada:"))
            if len(str(pk1)) > 5:
                pk1 = 1
                dd = 1
                print("Valor de chave pública inválido.")
                print("Digite apenas o primeiro valor do seu par de chave pública.")
                print("Por exemplo se seu par de chave é 12345 e 678910, digite apenas 12345.")
            if len(str(dd)) > 5:
                pk1 = 1
                dd = 1
            s = Acessar(pk1, p, l, s, dd, escolha)
        except ValueError:
            print("Nesse campo é permitido apenas números.")

```

Figura 26 – Função Main() parte 5. Fonte: Autoria própria.



```

print("Nesse campo é permitido apenas números.")

if s == False:
    # Limita o usuário a tentar login somente três vezes
    cont += 1
    if cont == 3:
        print("Você tentou acessar o navio mais de 3 vezes.")
        print("Por motivos de segurança o sistema irá encerrar.")
        time.sleep(5)
        exit()
    print("Você só tem mais {} tentativas de tentar acessar o navio.".format(3 - cont))
else:
    if s == True:
        acessos = open("acessos.txt", 'r')
        acesso = acessos.readlines()
        acessos.close()
        print(dec)
        print("ACESSOS".center(80))
        print(dec)
        for i in acesso:
            print(i)
        main()
    else:
        print("Saindo...")
        time.sleep(5)
        exit()

main()

```

Figura 27 – Função Main() parte 6. Fonte: Autoria própria.

5.1. Estruturação, conceitos e fundamentação

Segundo Oliveira (2012) um dos pontos principais do RSA é a facilidade de multiplicar dois números primos para gerar um terceiro número, entretanto quando o

processo é inverso a dificuldade para chegar até os dois números primos que geraram esse terceiro é altíssima. Esse processo é conhecido como fatoração, por exemplo, para chegar até o número 3.337 são necessários os fatores primos 47 e 71.

O autor ainda afirma que a segurança desse tipo de criptografia está em escolher números primos grandes para a multiplicação que irá gerar a chave pública, e a partir da mesma fatorar um grande número para gerar a chave privada. Se o número o número for bem escolhido e tiver um tamanho considerável, o invasor não irá conseguir descobrir as chaves em tempo útil, por esse motivo esse foi a técnica escolhida para o algoritmo como já mencionado no capítulo 5. Um exemplo disso ocorreu em 1999, quando uma chave RSA de 512 bits foi decifrada pelo Instituto Nacional de Pesquisa da Holanda, com ajuda de cientista de outros países, esse processo durou em torno de 7 meses, além de utilizar de mais de 300 estações de trabalho para o deciframento.

Para gerar as chaves RSA são necessários cálculos, que estão detalhados no capítulo [4.2](#) conforme o autor Fiarresga (2010).

5.2. Benefícios em relação às técnicas anteriores

Os benefícios da criptografia RSA, além, claro de sua facilidade em multiplicar números primos para gerar um terceiro como já mencionado antes são os seguintes:

- Tempo de quebra: Segundo Ladeira e Raugust (2017) para gerar as chaves no algoritmo RSA costuma levar tempo polinomial, no entanto, para quebrar o algoritmo o tempo hábil para isso é exponencial, o que sustenta a alta segurança da técnica;
- Assinatura Digital: Segundo Barbosa et al. (2003) o RSA é o principal algoritmo com suporte para a implementação de assinaturas digitais e troca de chaves, entre algoritmos mais populares;
- Não é necessário que o indivíduo que irá criptografar a mensagem tenha acesso a chave que decodifica, como em algoritmos de chave simétrica (AES, e Blowfish).

5.3. Aplicações que fazem/fizeram uso da técnica

Segundo Petean et al. (2017) o RSA é amplamente utilizado na internet, como em e-mails, e compras online. Entretanto, é na área de assinaturas digitais que essa

técnica se destaca nas aplicações, pois, quando uma mensagem é assinada de forma digital o receptor tem a certeza que a mensagem que o mesmo recebeu, ou o arquivo é verídico, e que foi o remetente esperado que há enviou, como o RSA foi uma das primeiras técnicas a dar suporte a este tipo de tecnologia, ele é o principal algoritmo utilizado para este fim.

5.4. Discussão comparativa entre esta técnica e outras conhecidas/utilizadas

Todos os tipos de criptografia têm seus pontos fortes e pontos fracos, escolher qual é o “melhor” depende de sua necessidade. Dos algoritmos apresentados até aqui (AES, RSA, Blowfish, DES) se analisar o quesito segurança, o RSA é um dos mais seguros, entretanto toda essa segurança tem um custo: processamento. Quanto maior a segurança da chave, mais processamento o RSA consome, então se a sua prioridade é a velocidade, ou então você não tiver infraestrutura suficiente para rodar o RSA, o Blowfish ou o AES podem ser boas soluções, o único que não é recomendado entre os apresentados é o DES, pois, mesmo tendo sido por um determinado tempo um dos mais difundidos do mundo, o mesmo se tornou obsoleto, podendo ser facilmente quebrado.

O AES é uma boa alternativa ao RSA, pois, segundo Mathias (2005) é bastante usado por conta da sua fácil e rápida implementação, simplicidade do projeto, e a facilidade de possíveis modificações, pecando apenas em sua função inversa.

5.5. Vulnerabilidades e falhas.

Algumas das vulnerabilidades, falhas, e desvantagens do RSA segundo Barbosa et al. (2003) são:

- Ataques de força bruta: Esse tipo de ataque consiste em tentar todas as combinações possíveis de chaves, embora isso demande muito tempo, e possivelmente o invasor possa ser detectado;
- Ataques matemáticos: Existem algumas formas matemáticas de atacar o RSA, uma delas é encontrar uma técnica que calcula a enésima raiz mod n , pois dessa forma $C = M^e \text{ mod } n$, a enésima (e th) raiz de $c \text{ mod } n$ é a mensagem M , mas ainda não existe nenhum método capaz de realizar este cálculo;
- Expoente d pequeno para decriptografia: As vezes são utilizados expoentes d um pouco menores para ter ganhos de performance no processo de

decriptografar, entretanto apesar do ganho de performance a segurança do algoritmo diminui consideravelmente;

- Expoente e pequeno para criptografia: Assim como no exemplo acima, as vezes o expoente e usado na criptografia é pequeno para obter o ganho de performance, entretanto como já dito no tópico acima, a segurança do algoritmo é reduzida consideravelmente.

5.6.Melhorias propostas e/ou implementadas.

Após ser abordado as características do RSA, apresentando suas vantagens, e suas desvantagens, fica claro que uma possível melhoria no sistema de criptografia seria manter o alto nível de segurança, sem perder tanta performance. A performance do RSA pode ser o seu grande calcanhar de Aquiles, pois, existem outros sistemas de criptografia que apresentam um nível bom de segurança, com uma performance melhor do que o RSA.

6. PROJETO (ESTRUTURA) DO PROGRAMA

Nesse capítulo será abordado sobre a estrutura do sistema, e como o mesmo se comporta dependendo das ações do usuário.

A estrutura básica do sistema é a seguinte: Entrar no navio, cadastrar, consultar e sair. Essas funcionalidades são as que o usuário encontra assim que entra no sistema, como é visível na figura 28:

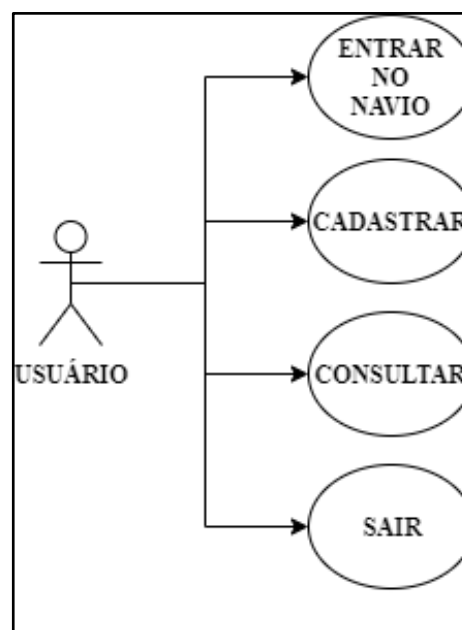


Figura 28 – Estrutura básica do sistema. Fonte: Autoria própria.

A figura 29 mostra o processo para o usuário se cadastrar no sistema, ele fornece sua patente, login, e senha e se a patente for válida o sistema gera as chaves do usuário, criptografa a senha e grava no arquivo “cadastro.txt” e retorna ao usuário suas chaves, caso a patente não seja válida um contador com as tentativas do usuário incrementado e depois é feita uma verificação se é a terceira tentativa do usuário ou não, se for o sistema irá informar ao usuário que ele errou três vezes e irá encerrar:

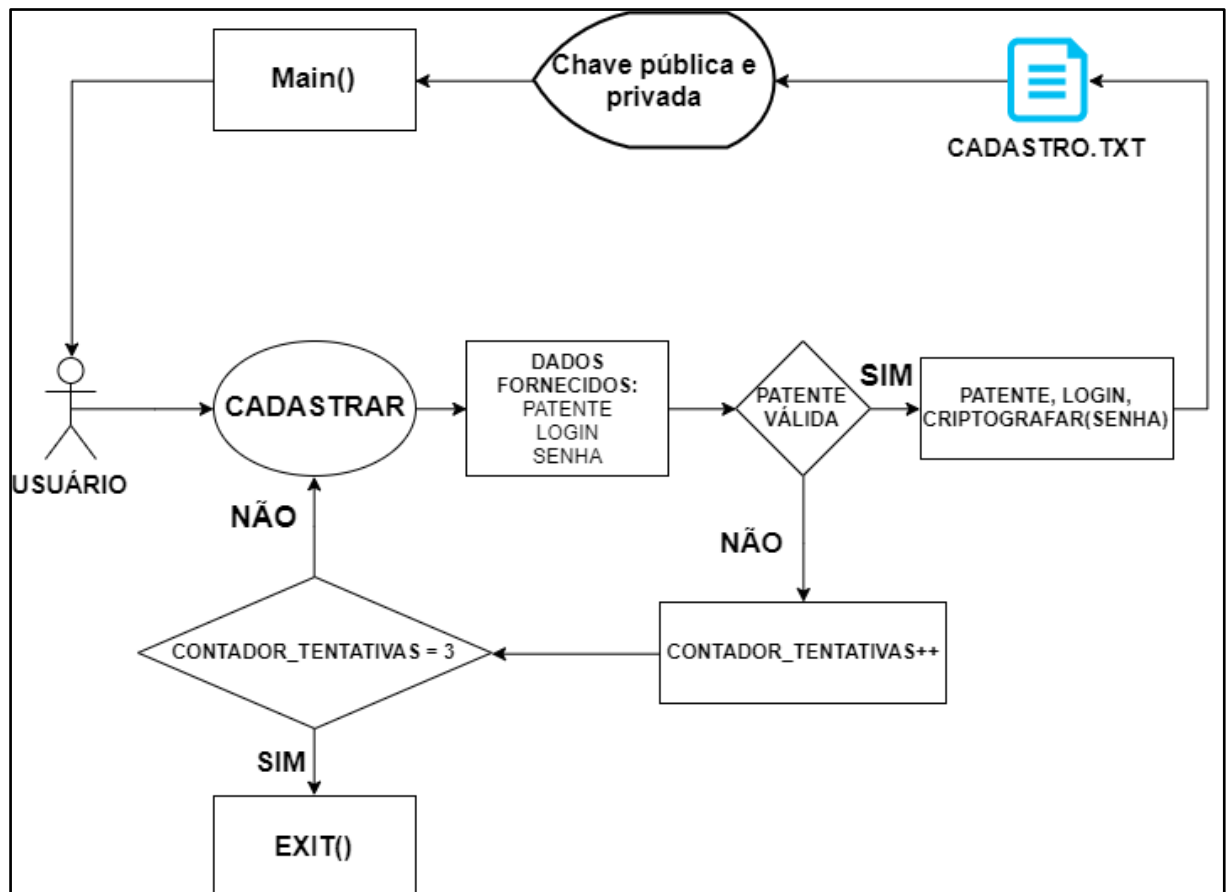


Figura 29 – Como o sistema se comporta no cadastro. Fonte: Autoria própria.

A figura 30 apresenta como o sistema se comporta quando o usuário deseja entrar no navio. O usuário deve fornecer o primeiro valor da sua chave pública, sua patente, seu login, sua senha, e sua chave privada, após isso o sistema verifica nos dados cadastrados no “cadastro.txt” se são válidos ou não, para isso é necessário descriptografar as senhas cadastradas e comparar todos os dados. Se os dados forem validos, ele informa ao usuário que o acesso foi liberado e logo após isso pega a patente do usuário, a data e a hora em que o acesso ocorreu e grava esses dados no arquivo “acessos.txt” que será usado para consultar os acessos.

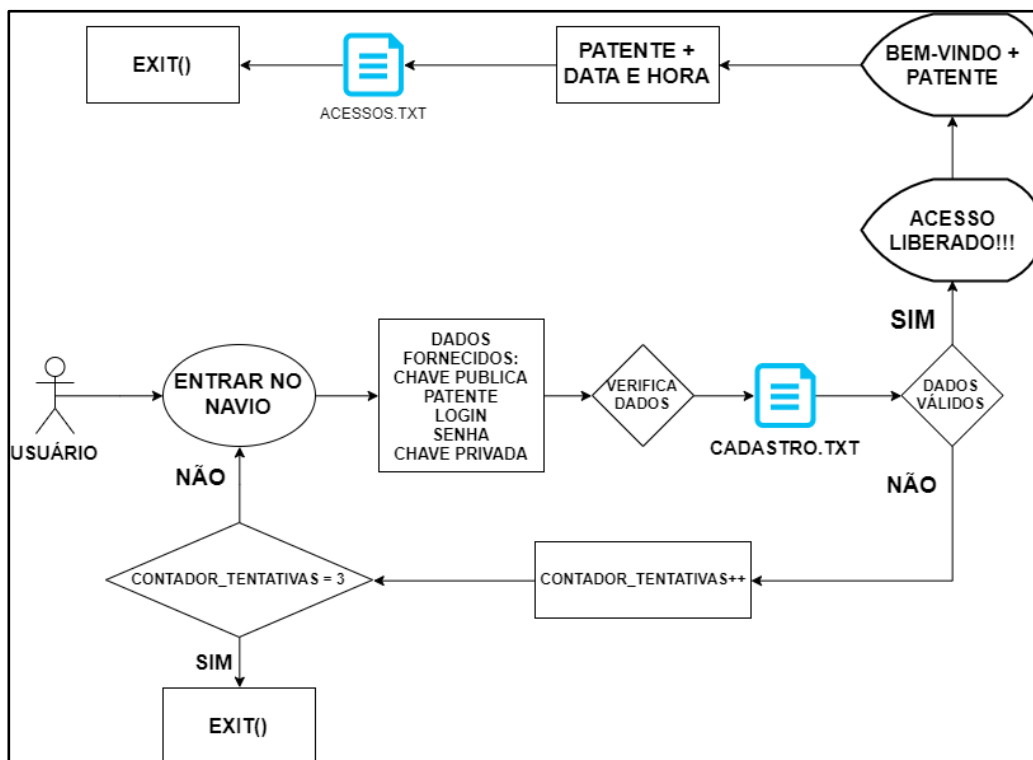


Figura 30 – Comportamento do sistema quando o usuário tenta acessar o navio. Fonte: Autoria própria.

A figura 31 apresenta o comportamento do sistema quando o usuário deseja consultar os acessos, o processo é semelhante ao de acesso do navio, entretanto não é registrado um acesso do usuário, pois, ele não entrou no navio, e são apresentados os dados cadastrados no arquivo “acessos.txt”:

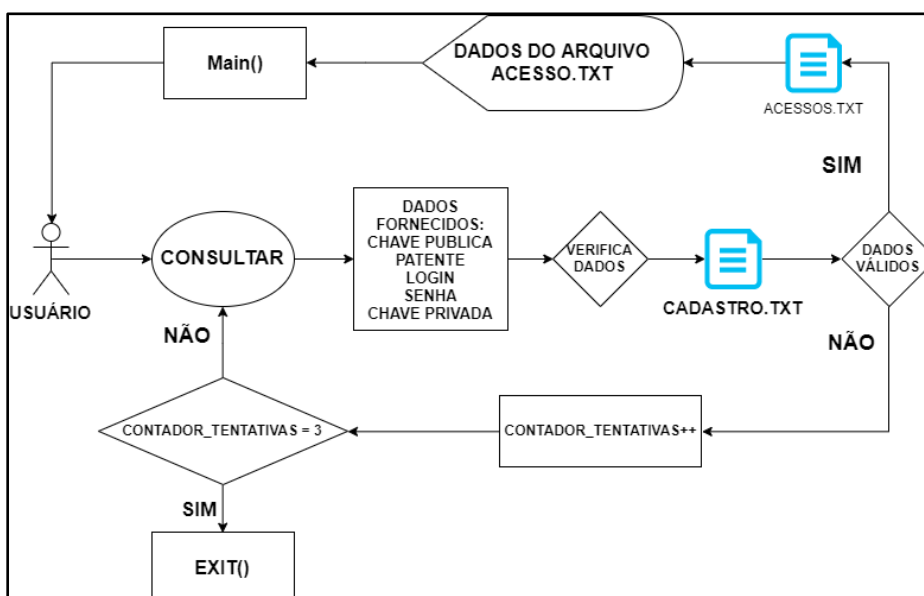


Figura 31 – Comportamento do sistema quando o usuário tenta consultar os acessos. Fonte: Autoria própria.

A figura 32 apresenta o comportamento do sistema quando o usuário deseja sair:

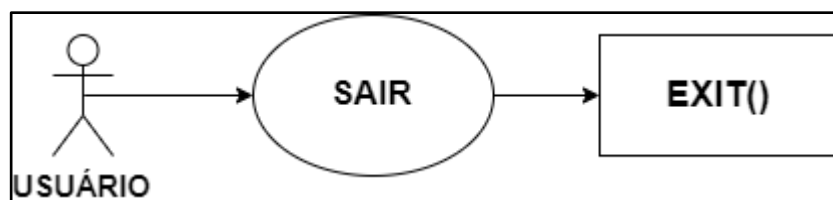


Figura 32 – Comportamento do sistema quando o usuário deseja sair do sistema.

7. RELATÓRIO COM AS LINHAS DE CÓDIGO DO PROGRAMA

Esse capítulo contém o código fonte do sistema:

```
import random
import time
from datetime import datetime
def totient(numero):
    # Calcula o totiente do numero primo
    if(primo(numero)):
        return numero-1
    else:
        return False

def primo(n):
    # verifica se o numero gerado é primo
    if (n <= 1):
        return False
    if (n <= 3):
        return True

    if (n%2 == 0 or n%3 == 0):
        return False

    i = 5
    while(i * i <= n):
        if (n%i == 0 or n%(i+2) == 0):
            return False
        i+=6
    return True

def gerador_E(num):
    #Gera um numero E aleatorio, obedecendo as condições
    def mdc(n1,n2):
        rest = 1
        while(n2 != 0):
            rest = n1%n2
            n1 = n2
            n2 = rest
        return n1

    while True:
        e = random.randrange(2,num)
        if(mdc(num,e) == 1):
            return e
```

```
def gerador_primo():
    # gera um numero primo aleatorio
    while True:
        x=random.randrange(100,256) # definindo o range de numeros
        if(primo(x)==True):
            return x
```

```
def mod(a,b):
    # função modular entre dois números
    if(a<b):
        return a
    else:
        c = a % b
        return c
```

```
def criptografar(senha,e,n):
    #Função responsável por descriptografar
    tam = len(senha)
    i = 0
    lista = []
    while(i < tam):
        letra = senha[i]
        k = ord(letra)
        k = k**e
        d = mod(k,n)
        lista.append(d)
        i += 1
    return lista
```

```
def descifra(cifra, n, d):
    #Função responsável por descriptografar
    lista = []
    i = 0
    tamanho = len(cifra)
    # texto=cifra ^ d mod n
    while i < tamanho:
        result = cifra[i]**d
        texto = mod(result,n)
        letra = chr(texto)
```

```

        lista.append(letra)
        i += 1
    return lista

```

```

def calcular_chave_privada(toti, e):
    #Função responsável por calcular a chave privada
    d = 0
    while(mod(d * e, toti)!=1):
        d += 1
    return d

```

```

def Cadastro(patente, login, senha):
    #Função responsável pelo cadastro de usuários
    p = gerador_primo() # gera um primo aleatório para P
    q = gerador_primo() # gera um primo aleatório para Q
    n = p * q # calculando n
    y = totient(p) # calculando totiente de P
    x = totient(q) # calculando totiente de Q
    totient_de_N = x * y # calculando totiente de N
    e = gerador_E(totient_de_N) # gerando o e E
    chave_publica = [n, e]
    print('Sua chave pública é {} e {}'.format(chave_publica[0],
    chave_publica[1]), end=" ")
    d = calcular_chave_privada(totient_de_N, e)
    print("e sua chave privada é {}".format(d))
    print("Guarde essas chaves, pois serão necessárias para acessar o navio.")
    print("Seu uso é pessoal, não é permitido seu compartilhamento.")
    senha_crip = criptografar(senha, e, n)
    senha_crip = str(senha_crip)
    gravando = open("cadastro.txt", 'a')
    gravando.write(patente + "|" + login + "|" + senha_crip + "|" + str(len(senha))
    + "\n")
    gravando.close()
    status = True
    print("Cadastro realizado com sucesso!!!")
    return status

```

```

def Acessar(pk1, p, l, s, d, escolha):
    pa = p
    p = p.replace(" ", "")
    #Função responsável por realizar o acesso no navio
    lista_total = [] # Recebe todos os dados cadastrados
    lista_total2 = [] # é usada para separar em login

```

```

lista_login = [] # Lista contém somente os logins
lista_patente = [] #Lista contém somente as patentes
lista_senha = [] # lista contém apenas as senhas
lista_tamanho = []
senha_int = [] #senhas convertidas em inteiros
log = False #Sinaliza se o login do usuário bate
pat = False #Sinaliza se a patente do usuário bate
sen = False #Sinaliza se a patente do usuário bate
senha_certa = [] #recebe a senha do usuário quebrando ela em letras
senhafinal = "" # senha descryptografada
status = True # se o acesso foi cadastrado com sucesso ou não
arquivo = open("cadastro.txt", 'r')
dados = arquivo.readlines()
arquivo.close()

for linha in dados:
    lista_total.append(linha.replace("\n", "")) # remove o '\n' da lista
for dado in lista_total:
    lista_total2 = dado.split("|") #o separador é usado para dividir os dados
    dentro do arquivo patente|login|senha|len(senha)
    lista_patente.append(lista_total2[0]) # recebe somente as patentes
    cadastradas
    lista_login.append(lista_total2[1]) # recebe apenas os logins cadastrados
    lista_senha.append(lista_total2[2]) # recebe as senha criptografadas
    lista_tamanho.append(lista_total2[3]) # recebe o tamanho da senha

for patente in lista_patente:
    #verifica se a patente informada está cadastrada
    if p == patente:
        pat = True
for login in lista_login:
    #verifica se o login informado está cadastrado
    if l == login:
        log = True
for senha in lista_senha:
    # remove os separadores das senhas e descryptografa elas
    senha = senha.replace("[", "")
    senha = senha.replace("]", "")
    senha = senha.split(",")
    for num in senha:
        senha_int.append(int(num))
    senha_dec = descifra(senha_int, pk1, d)
tamanho = len(s)
for i in range(0, tamanho):
    senha_certa.append(s[i])

```

```

    lista_mega = [x for x in senha_dec if x in senha_certa] #atribui a lista_mega
    somente as letras descritografadas que estão na lista senha_certa
    cont = 0 #indice da string senha
    for letra in lista_mega:
        # recria a senha descritografada
        if letra == s[cont]:
            senhafinal += letra
            cont +=1
    tamanho_senha = 0
    for i in lista_tamanho:
        tamanho_senha = int(i)
        if senhafinal == s and tamanho_senha == len(s):
            # se a senha recriada for igual a senha informada atribui 'True' ao
            status da senha
            sen = True

    if pat == True and log == True and sen == True:
        # se todos os dados do usuário forem válidos libera o acesso e grava no
        arquivo acessos
        print("Acesso liberado!!!")
        print("Bem-vindo {}".format(pa))
        if escolha == 1:
            dh = datetime.now()
            dht = dh.strftime('%d/%m/%y %H:%M')
            acessos = open("acessos.txt", 'a')
            acessos.write(pa + " - " + dht + "\n")
            acessos.close()
            time.sleep(5)
            exit()
        else:
            print("Acesso negado!!!")
            return sen

def main():
    #Desenhando o menu
    dec = "-" * 80
    print(dec)
    print("MARINHA DO BRASIL".center(80))
    print("Marinha do Brasil, protegendo nossas riquezas, cuidando da nossa
    gente.".center(80))
    print(dec)
    print("CONTROLE DE ACESSO.".center(80))
    print(dec)
    escolha = 0
    cont = 0

```



```

while escolha != 4:
    #Se a escolha do usuário for diferente de 4 que é a opção para sair o
while roda
    print("Escolha o número correspondente ao que você deseja realizar no
sistema:".center(80))
    print("1 - Entrar no navio;".center(80))
    print("2 - Cadastrar;".center(74))
    print("3 - Consultar acessos;".center(82))
    print("4 - Sair;".center(70))
    print(dec)
    #Tratamento de erro caso o usuário digitar uma letra
    try:
        escolha = int(input("Opção:"))
    except ValueError:
        print("Opção inválida, digite apenas números.")

    #A lista abaixo contém as patentes aptas a se cadastrar no sistema
    patentes_validas = ["capitãodecorveta", "capitãodefragata",
"capitãodemareguerra", "contra-almirante", "vice-almirante",
"almirantedeesquadra", "almirante"]

    #Determinando as funções para cada funcionalidade
    if escolha == 1:
        cont = 0
        v_acesso = False
        while cont < 3:
            print(dec)
            print("ACESSO".center(80))
            print(dec)
            try:
                pk1 = int(input("Digite o primeiro valor do seu par de chaves
públicas:"))
                p = input("Agora digite sua patente:".lower())
                l = input("Digite seu login:")
                s = input("Digite sua senha:")
                d = int(input("Digite sua chave privada:"))
                v_acesso = Acessar(pk1, p, l, s, d, escolha)
            except ValueError:
                print("Nesse campo são permitidos apenas números.")
            if v_acesso == False:
                #Limita o usuário a tentar login somente três vezes
                cont += 1
                if cont == 3:
                    print("Você tentou acessar o navio mais de 3 vezes.")
                    print("Por motivos de segurança o sistema irá encerrar.")

```

```

        time.sleep(5)
        exit()
        print("Você só tem mais {} tentativas de tentar acessar o
navio.".format(3-cont))

elif escolha == 2:
    print(dec)
    print("CADASTRO".center(80))
    print(dec)
    #Cadastro de usuário
    cont_tent = 0
    cadastro = False
    while cont_tent < 3:
        # Limita o usuário a apenas 3 tentativas
        cad_pat = input("Digite sua patente:".lower())
        cad_pat = cad_pat.replace(" ", "")
        # verifica se a patente inserida é válida
        for patente in patentes_validas:
            if cad_pat == patente:
                cadastro = True
        if cadastro != True:
            #Se a patente não for apta
            cont_tent += 1
            tent = 3 - cont_tent
            print("Sua patente não é apta a se cadastrar.")
            if cont_tent == 3:
                #Quando o usuário tenta cadastrar mais de três vezes
                print("O algoritmo está encerrando por motivos de segurança.")
                time.sleep(5)
                exit()
            print("Você ainda tem {} tentativas de cadastro.".format(tent))
        if cadastro == True:
            #Quando a patente do usuário é válida
            login = input("Digite seu login:")
            senha = input("Digite sua senha:")
            s = Cadastro(cad_pat, login, senha)
            if s == True:
                main()

elif escolha == 3:
    print(dec)
    print("CONSULTA".center(80))
    print(dec)
    print("Primeiro é necessário realizar login.")
    s = False
    while cont < 3:

```

```

try:
    pk1 = int(input("Digite o primeiro valor do seu par de chaves
públicas:"))
    p = input("Agora digite sua patente:".lower())
    p = p.replace(" ", "")
    l = input("Digite seu login:")
    s = input("Digite sua senha:")
    dd = int(input("Digite sua chave privada:"))
    s = Acessar(pk1, p, l, s, dd, escolha)
except ValueError:
    print("Nesse campo é permitido apenas números.")

if s == False:
    # Limita o usuário a tentar login somente três vezes
    cont += 1
    if cont == 3:
        print("Você tentou acessar o navio mais de 3 vezes.")
        print("Por motivos de segurança o sistema irá encerrar.")
        time.sleep(5)
        exit()
    print("Você só tem mais {} tentativas de tentar acessar o
navio.".format(3 - cont))
    else:
        if s == True:
            acessos = open("acessos.txt", 'r')
            acesso = acessos.readlines()
            acessos.close()
            print(dec)
            print("ACESSOS".center(80))
            print(dec)
            for i in acesso:
                print(i)
            main()
        else:
            print("Saindo...")
            time.sleep(5)
            exit()
main()

```

8. REFERÊNCIAS BIBLIOGRÁFICAS

BARBOSA, Luis Alberto de Moraes et al. **RSA: Criptografia Assimétrica e Assinatura Digital**. 2003. Disponível em: <<http://www.braghetto.eti.br/files/Trabalho%20Oficial%20Final%20RSA.pdf>>. Acesso em: 24 out. 2019.

BUGATTI, Pedro Henrique. **Implementação e Avaliação do Algoritmo Blowfish em C, Java e Microcontroladores PIC**. 2005. 145 f. TCC (Graduação) - Curso de Ciência da Computação, Centro Universitário "Eurípides de Marília" - Univem, Marília, 2005. Disponível em: <<https://aberto.univem.edu.br/bitstream/handle/11077/447/Implementa%c3%a7%c3%a3o%20e%20Avalia%c3%a7%c3%a3o%20do%20Algoritmo%20Blowfish%20em%20C%2c%20Java%20e%20Microcontroladores%20PIC.pdf?sequence=1&isAllowed=y>>. Acesso em: 1 out. 2019.

CÂMARA, Marco. **Criptografia E Compressão de Dados**. 2009. Disponível em: <<http://www.logicengenharia.com.br/mcamara/ALUNOS/Cripto&Compr.pdf>>. Acesso em: 1 out. 2019.

CAMPOS, Alessandro Augusto Nunes. **Algoritmo de Criptografia AES em Hardware, Utilizando Dispositivo de Lógica Programável (FPGA) e Linguagem de Descrição de Hardware (VHDL)**. 2008. 94 f. TCC (Pós-Graduação) - Curso de Pós-graduação em Engenharia Elétrica, Universidade Federal de Itajubá, Itajubá, 2008. Disponível em: <https://repositorio.unifei.edu.br/xmlui/bitstream/handle/123456789/1693/dissertacao_0032910.pdf?sequence=1&isAllowed=y>. Acesso em: 1 out. 2019.

CERT. **Cartilha de Segurança para Internet**. 2017. Disponível em: <<https://cartilha.cert.br/criptografia/>>. Acesso em: 27 set. 2019.

FIARRESGA, Victor Manuel Calhabrês. **Criptografia e Matemática**. 2010. 144 f. TCC (Mestrado) - Curso de Mestrado em Matemática Para Professores, Universidade de Lisboa Faculdade de Ciências, Lisboa, 2010. Disponível em: <https://repositorio.ul.pt/bitstream/10451/3647/1/ulfc055857_tm_Victor_Fiarresga.pdf>. Acesso em: 27 set. 2019.

FIORIM, Franzvitor. **Criptografia para iniciantes: o que é, como funciona e por que precisamos dela?** 2015. Disponível em: <<https://canaltech.com.br/seguranca/criptografia-para-iniciantes-o-que-e-como-funciona-e-por-que-precisamos-dela-46753/>>. Acesso em: 27 set. 2019.

HINZ, Marco Antônio Mielke. **Um estudo descritivo de novos algoritmos de criptografia**. 2000. 58 f. TCC (Graduação) - Curso de Informática - Ênfase em Sistemas de Computação, Universidade Federal de Pelotas - Instituto de Física e Matemática, Pelotas, 2000. Disponível em: <<https://archive.alvb.in/bsc/material-disciplinas/mtm5512/Trabalho%20Final/Mono-MarcoAntonio.pdf>>. Acesso em: 1 out. 2019. Acesso em: 1 out. 2019.

LADEIRA, Ricardo de La Rocha; RAUGUST, Anderson Schwede. **Uma análise da complexidade do algoritmo RSA implementado com o teste probabilístico de Miller-Rabin**. 2017. Disponível em: <<https://seer.imed.edu.br/index.php/revistas/article/view/1639/1296>>. Acesso em: 24 out. 2019.

MATHIAS, Leopoldo A. P.. **Vantagens e Desvantagens do uso do AES**. 2005. Disponível em: <https://www.gta.ufrj.br/grad/05_2/aes/vandesv.htm>. Acesso em: 28 out. 2019.

MORALE, Leonardo Araújo de Oliveira. **ANÁLISE DE ALTERNATIVAS PARA TRANSMISSÃO DE DADOS DE DISPOSITIVO MÓVEL PARA SERVIDOR REMOTO UTILIZANDO CRIPTOGRAFIA SIMÉTRICA**. 2010. 90 f. TCC (Graduação) - Curso de Engenharia da Computação, Fatecs – Faculdade de Tecnologia e Ciências Sociais Aplicada, Brasília, 2010. Disponível em: <<https://repositorio.uniceub.br/jspui/bitstream/123456789/3217/2/20016011.pdf>>. Acesso em: 1 out. 2019.

OLIVEIRA, Ronielton Rezende. **Criptografia simétrica e assimétrica: os principais algoritmos de cifragem**. 2012. Disponível em: <<http://www.ronielton.eti.br/publicacoes/artigorevistasegurancadigital2012.pdf>>. Acesso em: 1 out. 2019.

PETTEAM, Fernanda Bia et al. **Criptografia: O Método RSA**. 2017. Disponível em: <https://www.ime.unicamp.br/~ftorres/ENSINO/MONOGRAFIAS/Fernando_TN17M2.pdf>. Acesso em: 25 out. 2019.

SILVA, Cristovam Lage da. **DESENVOLVIMENTO DE UM MÓDULO DE CRIPTOGRAFIA LEVE PARA FPGA UTILIZANDO O ALGORITMO SIMON AND SPECK**. 2019. 36 f. TCC (Graduação) - Curso de Engenharia da Computação, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2019. Disponível em: <http://www.inf.pucrs.br/~moraes/docs/tcc/tcc_cristovam.pdf>.

SILVA, Guilherme Gomes Felix da. **Formalização de Algoritmos de Criptografia em um Assistente de Provas Interativo**. 2018. 70 f. Dissertação (Mestrado) - Curso de Mestre em Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2018. Disponível em: <<https://www.maxwell.vrac.puc-rio.br/35851/35851.PDF>>. Acesso em: 1 out. 2019.

9. FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS

[illegible]

FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Gabriel Vinícius Mendes Selquira Perez TURMA: ST2P33 RA: F0688B-0
CURSO: Sistema de Informação CAMPUS: Toledo SEMESTRE: 2º TURNO: Noturno
CÓDIGO DA ATIVIDADE: 7881 SEMESTRE: 2º Semestre ANO GRADE: 2019/2

[illegible]

[1] Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS:		75h	

AVALIAÇÃO: _____

NOTA:

DATA: / /

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO

FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Guyllan Rocio Duarte TURMA: SIQP33 RA: D990JI-6
 CURSO: Sistema de Informaçõ CAMPUS: Petropolis SEMESTRE: 2º TURNO: noite
 CÓDIGO DA ATIVIDADE: 78B1 SEMESTRE: 2º ANO GRADE: 2019/2

[illegible]

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUIDAS: 75h

AVALIAÇÃO: _____

NOTA: _____

DATA: ____/____/____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO



NOME: Richard Dos Santos Mendes Lima TURMA: SIQ P33 RA: D89066-2
CURSO: Sistemas de Informação CAMPUS: Totopó SEMESTRE: 2º TURNO: noite
CÓDIGO DA ATIVIDADE: 78B1 SEMESTRE: 2º ANO GRADE: 2019/2

[illegible]

TOTAL DE HORAS ATRIBUÍDAS:

75h

NOTA:

DATA: / /

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO

FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Renan Augusto de Oliveira TURMA: SIQP33 RA: F055JJ-0
CURSO: Sistemas de Informação CAMPUS: Todolândia SEMESTRE: 2º TURNO: noite
CÓDIGO DA ATIVIDADE: 72B1 SEMESTRE: 2º Semestre ANO GRADE: 2019/2

[illegible]

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS:		75h
----------------------------	--	-----

AVALIAÇÃO: _____

NOTA:

DATA: ____/____/____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO