**Universidade Paulista – UNIP**

**Sistemas de Informação**

**Tema:**

As técnicas criptográficas, conceitos, usos e aplicações.

Integrantes:

Gustavo Pereira do Amor Divino / RA: N508612

Ian Cozachevici de Jesus / RA: N503530

Jennifer Satirio Lima Santos / RA: N516089

**Paraiso**

**Novembro 2019**

Índice

[1. Objetivo do trabalho 3](#_Toc25424931)

[2. Introdução 4](#_Toc25424932)

[3. Criptografia (conceitos gerais) 7](#_Toc25424933)

[4. Técnicas criptográficas 9](#_Toc25424934)

[5. Estruturação, conceitos e fundamentação 15](#_Toc25424935)

[6. Benefícios em relação às técnicas anteriores 15](#_Toc25424936)

[7. Discussão comparativa entre esta técnica e outras conhecidas/utilizadas 15](#_Toc25424937)

[8. Vulnerabilidades e falhas 15](#_Toc25424938)

[9. Estudo de caso 16](#_Toc25424939)

[10. Interdisciplinaridade 16](#_Toc25424940)

[11. Briefing do programa 17](#_Toc25424941)

[12. Relatório com as linhas de código do programa 19](#_Toc25424942)

[13. Conclusão 64](#_Toc25424943)

[14. Bibliografia 65](#_Toc25424944)

[15. Fichas de horas APS 66](#_Toc25424945)

# Objetivo do trabalho

Demonstrar a importância e a razão de utilizar um meio de segurança excelente para mensagens ou diálogos entre indivíduos que já havia sendo utilizada anteriormente, com intuito de transmitir segurança às informações, sempre preservando a autenticidade e o sigilo da mesma.

# Introdução

Mesmo a muito tempo atrás, a sociedade humana, transmite mensagens por meio de diálogos, desenhos e outras formas de comunicação, permitindo assim a propagação de dados e informações, que poderiam ser originados de elementos simples ou até sobre situações críticas. Tais situações ocasionalmente recebem uma importância de que nem todo mundo poderia ter conhecimento de seu significado. Por meio desta forma de repassar este conteúdo a frente, mantendo sua confidencialidade e integridade, várias técnicas foram criadas para poder dar cabo nisso. E a partir dessa necessidade que diversas sociedades criaram o que chamamos atualmente de criptografia.

A humanidade desde 4500 A.C, no Egito antigo, e afins mensuraram maneiras tangíveis de transmitir informações entre si, sem que o conteúdo, ou seja, o significado de suas palavras, símbolos ou desenhos, os conhecidos hieróglifos, fossem decifrados por qualquer indivíduo. Por meio disto, recordações, tábuas de pedras oriundas daquela época que contêm os hieróglifos e símbolos dos mais diversos formatos conservados até hoje, sendo que diversas destas mensagens contém suas definições e propósitos desconhecidos até os tempos modernos.

Posteriormente outras civilizações como os gregos, romanos, hebreus e etc, empregaram esta ideia de criptografia para poder propagar suas informações de cunha importância para outros indivíduos mantendo assim sua segurança, ou seja, que a mensagem seja confidencial (poucas pessoas ou só os destinatários poderiam saber o significado da mensagem) e tenha integridade (de que a mensagem seja só reconhecida pelo destinatário e se a mesma sofreu alguma alteração).

Estas sociedades antigas marcaram o início desta maneira de assegurar essa transmissão de informação, sendo caracterizadas de criptografia clássica, marcando assim a época em que elas vivenciavam. Porém com o decorrer do tempo, as pessoas viriam a descobrir inéditas maneiras de criar, e inovar sua forma de pensar diante a suas necessidades e deveres, como dizia o filósofo Platão (428 a.C.347 a.C.) “A necessidade é a mãe da inovação”. Posteriormente, a humanidade vinha passando por evoluções não só no ramo econômico, social e político mas sim como um todo, ou seja, cada ramificação que provinha de uma sociedade vinha passando por uma reformulação, e assim a tecnologia era criada e se desenvolvia mais e mais sendo para a realização de atividades diárias, como também para desempenhar atividades que antes, ao olhos das humanos, era impossível, se tornando assim essencial dali pra frente, Levitt Theodore como afirmava em seu livro *Ideas are Useless Unless Used* (1981) “A criatividade é pensar coisas novas. A inovação é fazer coisas novas”.

A tecnologia trouxe consigo não só maneiras de realizar e acelerar atividades de forma exponencial, melhorando assim sua produção e seu cotidiano. Como também, Maneiras de propagar informações, se tornaram não só de uma forma mais simples (com os estabelecimentos de jornais, artigos e principalmente da televisão) como do mesmo modo evoluíram em transmitir de um âmbito imenso.

Sendo assim, informação se converteu em algo demasiadamente valiosa, como citava o escritor americano John Naisbitt “A nova fonte de poder não é o dinheiro nas mãos de poucos, mas informação nas mãos de muitos”, pois a mesma poderia ser usada não unicamente para a transmissão de artigos e diálogos de valor “simples”, poderiam também carregar um peso de cunha importância para a própria como um todo, pois uma informação de alto calão nas mãos de terceiros poderia significar perdas econômicas, até a derrota do estado como um todo ( vindo até de referências bélicas).

A partir do século XX, esse tesouro que passou a ser a informação, começava a ganhar importância, pois aquele século vinha a ser marcado por várias disputas bélicas, envolvendo grandes potências mundiais, e que este tal tesouro vinha a ser disputado por seus envolvidos, porém mais especificamente na segunda guerra mundial (1939-1945), a criptografia se tornou um dos protagonistas, pelas mãos do exército alemão ( possivelmente o principal causador da próprio conflito bélico) pela invenção da máquina enigma, uma maneira feita pela própria Alemanha para que suas comunicações se tornassem “incomunicáveis” para seus inimigos, se tornou um dos maiores clímax da segunda guerra, se tornando assim umas das maiores armas do exército alemão, como também um de seu maiores algozes.

Com o decorrer dos anos, após algumas demonstrações de como a criptografia poderia ser considerada como o “baú a sete chaves” deste tesouro que é a informação, multinacionais e empresas utilizam este método de assegurar a transmissão de mensagens entre colaboradores e decisões de cunha importância para ela, para poder evoluir e se desenvolver, como cita o executivo Alfredo Martini Júnior “Informações que não circulam não são informações, são apenas recordações. São como fotos antigas que guardamos em nossas gavetas!.... Uma boa comunicação empresarial exige a constante circulação das informações entre os colaboradores.”.

E para poder manter esse carrossel girando, a criptografia se converteu em algo essencial atualmente refutando e alterado segurança de informações, várias maneiras foram sendo criadas, desenvolvidas para atender essa demanda. Na pesquisa a seguir veremos como se originou a criptografia e a sua importância desde os tempos antigo até os modernos para as mais diversas ocasiões e propósitos e alguns exemplos e como desempenham no âmbito internacional atualmente.

# Criptografia (conceitos gerais)

A utilidade da comunicação com segurança entre duas fontes, não envolvendo terceiros não autorizados tenham acesso ao conteúdo da mensagem é um problema antigo da Humanidade e como resposta a isso surgiu a criptografia. O termo *criptografia* origina-se do grego, *kryptós*, “escondido”, e *gráphein*, “escrita”. Define-se a criptografia como sendo: o conjunto de princípios e técnicas de comunicação sigilosa e segura entre indivíduos. A técnica consiste em transformar um texto legível em um texto ilegível de modo que apenas indivíduos autorizados na comunicação possam decifrar o conteúdo.

Entre 600 e 500 a.c., os hebreus utilizavam cifras monoalfabéticas de substituição simples, substituindo cada letra do texto legível por outra letra no texto cifrado. A Cifra de César,sendo uma das técnicas mais clássicas da criptografia**,** também utiliza um método de substituição, porém substituindo uma letra por outra situada três posições à frente no alfabeto. A decifração é o processo contrário, ou seja, troca a letra cifrada por outra anterior à três posições no alfabeto.

Entre as cifras de substituição, destaca-se ainda a cifra poli alfabética de Vigenère - erroneamente atribuída a Blaise de Viginère, sendo, na verdade, de autoria do italiano Giovan Batista Belaso. Nesta cifra, denominada “grade de Vigenère”, é escrito o alfabeto 26 vezes em linhas e colunas diferentes formando 26 possíveis cifras de César. Na cifragem ocorre a escolha de uma palavra-chave e cada letra desta palavra corresponde a uma linha na grade e cada letra do texto legível representa uma coluna.

Os gregos antigos inventaram a cifra de transposição**,** que consiste em reordenar as letras da mensagem de forma que o receptor saiba a lógica usada na construção para poder reconstrui-la. Os espartanos usavam uma cifra de transposição que basicamente enrolavam uma tira de couro em torno de uma cítala e escreviam as cifras sobre ela. Bastava uma cítala de mesma espessura que quando aplicada a tira de couro com a mensagem conseguia-se decifrar a mensagem.



(Fonte: http://cryptograma.blogspot.com/2015/06/historia-da-criptografia-cifras-e.html)

Para além do Ocidente, a civilização árabe-islâmica também contribuiu significativamente para o avanço da criptografia, sobretudo, na área da criptoanálise. Por volta de 800 d.C., o matemático árabe, Alcindi, inventou a técnica de análise de frequência que fora responsável por quebrar muitas cifras de substituição monoalfabéticas até o desenvolvimento das cifras poli alfabéticas, além de outras contribuições importantes para a criptoanálise.

Na Primeira Guerra Mundial, a criptografia teve um papel importante e terminou alterando os rumos do conflito. O Telegrama Zimmermann, carta criptografada do ministro do exterior alemão, Arthur Zimmermann, para o embaixador alemão no México orientando-o a propor uma aliança com o país para atacar os Estados Unidos foi interceptado e decodificado pelos britânicos e culminou na entrada dos EUA na guerra.

No entanto, foi na Segunda Guerra Mundial que a criptografia sofreu uma revolução com a introdução de máquinas (eletro) mecânicas e do uso da matemática por parte da inteligência militar dos governos envolvidos na guerra. O caso mais emblemático foi o da “indecifrável” máquina Enigma, da Alemanha Nazista, semelhante à uma máquina de escrever, operava com rotores à direita e à esquerda do teclado e quando era pressionada, o rotor à direita avançava ocasionando a rotação dos rotores à esquerda. O grupo liderado por Alan Turing - considerado o pai da Computação - conseguiram decifrar os códigos da máquina nazista, e encurtando assim a guerra.



(Fonte: <http://cryptograma.blogspot.com/2015/06/historia-da-criptografia-cifras-e.html>)

Claude Shannon, após a guerra desenvolveu pesquisas que estabeleceram a base teórica para a criptografia e criptoanálise moderna. Seus trabalhos, especialmente o livro *The Mathematical Theory Of Communication,* foram tão relevantes que a criptografia deixou de ser restrita a governos e organizações e começou a ser difundida comercialmente.

Nos anos 70 surgiu a cifra DES, empreendida pela IBM a fim de desenvolver instalações seguras de comunicações eletrônicas para as empresas, como bancos e centros financeiros. Foi oficialmente substituída em 2001 pela cifra AES pois tinha se tornado ultrapassada já que utilizava uma chave de 56 bits facilmente quebrável por ataque de força bruta.

Em 1976, W. Diffie e M. Helmann publicaram um artigo introduzindo um revolucionário sistema de criptografia de chave pública que utiliza um par de chaves. O sistema foi aperfeiçoado por pesquisadores do MIT em 1978 e originou-se o algoritmo RSA.

# Técnicas criptográficas

**Chave simétrica**

Na criptografia simétrica é utilizado uma chave para criptografar como para de criptografar dados. Os algoritmos que são usados para a criptografia simétrica são mais simples do que os algoritmos usados na criptografia assimétrica.

Em função desses algoritmos mais simples, e porque a mesma chave é usada tanto para criptografar como para de criptografar dados, a criptografia simétrica é muito mais rápida que a criptografia assimétrica. Portanto, a criptografia simétrica é adequada para a criptografia e a de criptografia de uma grande quantidade de dados.

Os algoritmos de chave simétrica podem ser divididos em cifras de fluxos (ou continuas) e em cifras por blocos. As cifras de fluxo cifram os bits da mensagem um a um, enquanto que as cifras por bloco pegam um número de bits e cifram como uma única.

Existem vários algoritmos que usam chaves simétricas, como o **DES**, o **IDEA**, **RC** e o **AES**.

**DES** (Data Encryption Standard): criado no ano de 1977 pela IBM, fazendo o uso de chaves de 56 bits, correspondendo a 72 quadrilhões de combinações. No mesmo ano foi promovido um desafia na internet fazendo com que esse algoritmo fosse quebrado por técnicas de “força bruta” (tentativa e erro).

**IDEA** (International Data Encryption Algorithm): criado por James Massey e Xuejia Lai em 1991, o IDEA é um algoritmo que utiliza chaves de 128 bits e possui uma estrutura bem semelhante ao DESporem sendo mais fácil a sua implementação em software.

**RC** (Ron’s Code ou Rivest Cipher): criado em 1994 por Ron Rivest na empresa RSA Data Security, este algoritmo é bastante utilizado em e-mails utilizando chaves que vão de 8 a 1024 bits. Cada versão se diferencia uma da outra por utilizarem chaves maiores, possuindo as versões: RC2, RC4, RC% e RC6.

**AES** (Advanced Encryption Standard): utilizado pelo governo dos Estados Unidos e hoje sendo utilizada no mundo todo. Possui 4 etapas: **AddRoundKey**, **ShiftRows**, **SubBytes** e **MixColumns**.

**AddRoundKey**: Cada sub-chave deriva da chave principal utilizando o algoritmo de escalonamento de chaves do Rijndael. Cada byte é combinado com a sub-chave própria do turno (RoundKey).

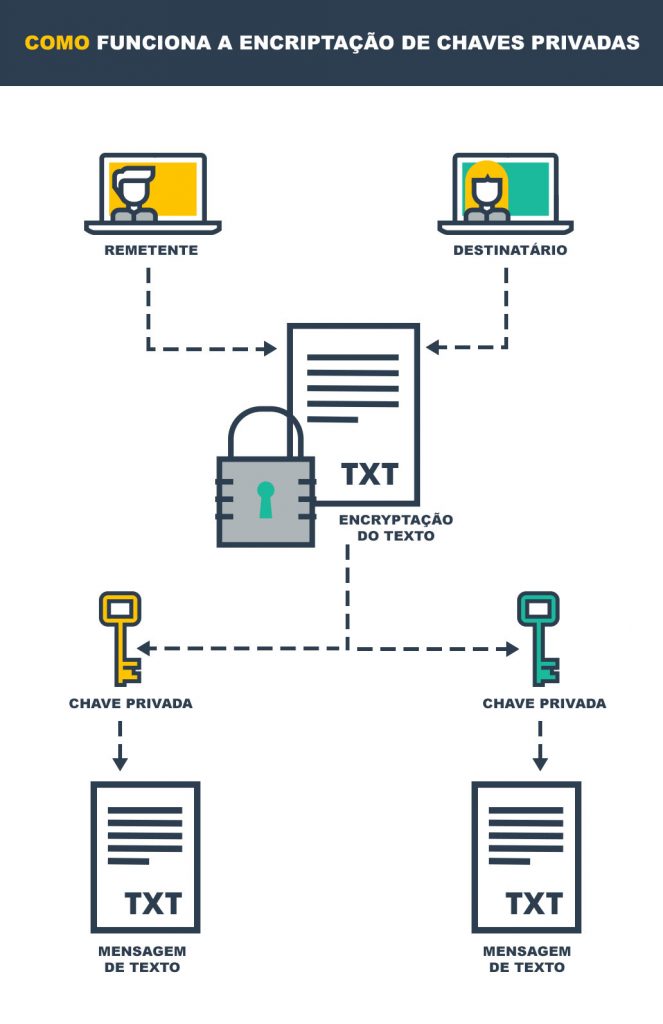
**ShiftRows**: Etapa em que cada fileira do estado é deslocada em um determinado número de posições.

**SubBytes**: Substituição não linear onde é substituído cada byte por outro de acordo com uma tabela de referência. No arranjo, cada byte é atualizado utilizando uma S-box de 8 bits.

**MixColumns**: Etapa de mescla, onde opera nas colunas do estado e as combina os quatro bytes de cada coluna usando uma transformação linear. Junto ao ShiftRows, o MixColumns fornece difusão a cifra.

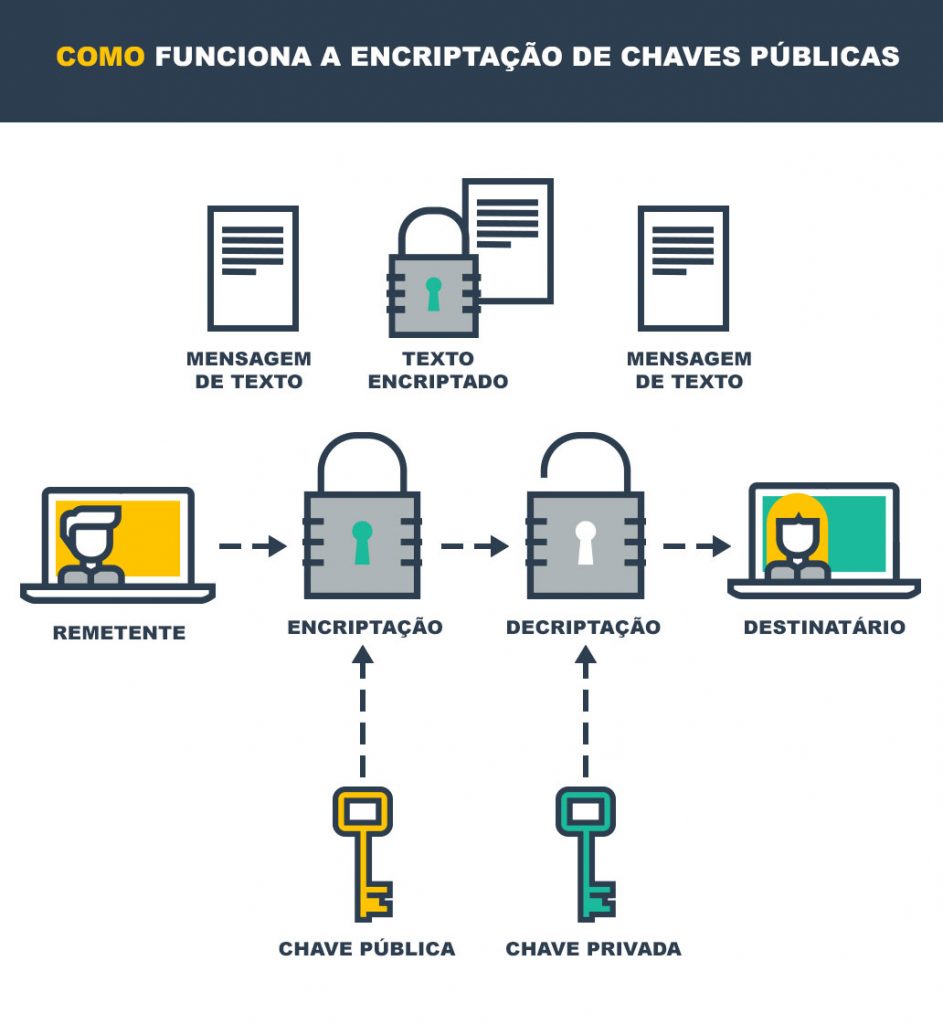
**Chave assimétrica**

Na criptografia de chave assimétrica todas as partes envolvidas na comunicação utilizam duas chaves distintas (assimétricas) e complementares, sendo uma privada e uma pública.



(Fonte:<https://www.brasilwebhost.com.br/blog/se-proteja-usando-os-aplicativos-de-mensagens/>)

A chave pública qualquer um pode ter acesso para se comunicar de forma segura, porem a chave privada somente os titulares podem ter acesso, sendo com a chave primada que será onde o destinatário poderá realizar a decodificação de uma mensagem criptografada diretamente pela sua chave pública, como por exemplo uma senha de um cartão.



(Fonte:<https://www.brasilwebhost.com.br/blog/se-proteja-usando-os-aplicativos-de-mensagens/>)

Esse tipo de criptografia possui muita segurança, pois como mostrado na imagem não é necessária compartilhar a chave privada para conseguir visualizar a mensagem, assim permitindo que qualquer uma possa enviar qualquer mensagem secreta, apenas tento a chave pública.

Assim não sendo necessário realizar o envio da chave, sendo uma desvantagem para a chame simétrica por ter que ser realizado esse envio de chave. Porém o processo que ocorre na criptografia assimétrica acaba sendo mais lento se comparado com a criptografia simétrica, sendo necessário realizar uma análise para verificar o mais viável para cada situação.

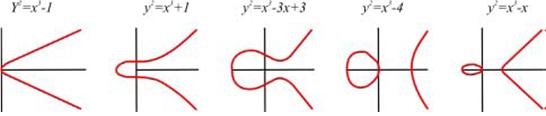
Esse procedimento mais lento se deve por conta da complexidade do desenvolvimento dos algoritmos da chave assimétrica, já que eles precisam ser capazes de reconhecer as chaves existentes e relaciona-las no momento oportuno.

No momento em que a chave privada estiver protegida, a mensagem que estiver criptografada possui extrema confidencialidade, caso a chave privada não esteja segura, qualquer um que possuir acesso a esta chave privada poderá ter o acesso a mensagem.

Existem alguns algoritmos que utilizam da criptografia assimétrica ou chave pública, como o **ElGamal, Curvas Elípticas**, o **RSA** e o **Diffie-Hellman**.

**ElGamal**: Algoritmo que utiliza a chave pública para realizar o gerenciamento de chaves. Este algoritmo manipula a matemática com enormes quantidades numéricas, a segurança vem de algo chamado de problema do logaritmo discreto. Portanto, o ElGamal possui sua segurança na dificuldade de realizar os cálculos de logaritmos discretos de um corpo finito.

**Curvas Elípticas**: Neal koblitz e V.S. Miller, em 1985, sugeriram de forma independente faze a utilização de curvas elípticas em sistemas criptografados de chave pública. Eles realizaram a implementação de Curvas Elípticas em algoritmos de chave pública já existentes. Portanto o sistema criptográfico de curvas elípticas se consiste em realizar modificações em outros sistemas, como o ElGamal por exemplo, que passa a trabalhar para o domínio das CurvasElípticas**,** em vez de trabalhar para o domínio dos corpos finitos.



(Fonte:https://sites.google.com/site/lucienefo/disciplinas/ce-281/criptografia-comcurvas-elipticas)

**RSA**:(Rivest-Shamir-Adleman). Este algoritmo foi criado no MIT em 1977 e possui este nome por conta de seus inventores: Ron Rivest, Adi Shamir e Len Adleman. Este é o algoritmo mais amplamente utilizado de chave pública, além de suas formas de criptografia serem umas das mais poderosas de chave pública conhecidas até hoje, fazendo o uso de diversos números primos.

**Diffie-Hellman**: Criado por Whitfield Diffie e Martin Hellman em 1976, baseado no problema do logaritmo discreto, também sendo o criptosistema mais antigo em uso de chave pública. Este sistema foi projetado com a intenção de fazer com que dois indivíduos entrem em um acordo quando compartilharem um segredo como uma chave, porem apenas trocando mensagens em público.

# Estruturação, conceitos e fundamentação

A criptografia é o estudo de uma comunicação segura, a qual é formada por algumas técnicas que consistem em manter em sigilo expressões, textos, frases, recados, mensagens, e, as mesmas só podem ser entendidas por quem tem o conhecimento do segredo da decifragem da técnica utilizada.

Na década de 70 dois cientistas da computação que trabalhavam no Massachussets Institute or Technology (M.I.T.), Ronald Rivest e Adi Shamir, criaram um método de criptografia de chave assimétrica eficiente, recebendo auxilio do matemático Leonard Adleman, que concordava com a ideia pelo ponto de vista matemático. Em 1977 os três registaram a patente do método RSA.

O método de criptografia RSA é o método de criptografia mais conhecido e mais usados atualmente.

# **Benefícios em relação às técnicas anteriores**

Uma das principais vantagens deste tipo de criptografia é a segurança, pois não é preciso compartilhar a chave privada.

Qualquer pessoa que tenha em mãos uma chave pública, podem enviar uma mensagem secreta.

# Discussão comparativa entre esta técnica e outras conhecidas/utilizadas

A principal vantagem de se utilizar a técnica RSA seria a segurança por utilizar a fatoração de números extensos e por ter duas chaves que fazem funções distintas, porém, para construir as duas chaves, para fazer todos os calculas, exige tempo e isso é um fator que as outras técnicas de criptografia se sobressaem, fazendo com que a escolha do método de criptografia a ser utilizado deve ser estudado para ficar de acordo com o objetivo do trabalho.

# Vulnerabilidades e falhas

Pelo simples fato do sistema de criptografia RSA ser muito utilizado, dá pra se imaginar que ele vem tendo suas vulnerabilidades estudadas e analisadas.

Um exemplo de ataque de força bruta ao método RSA, uma abordagem com o objetivo a chave pública por meio da fatoração do modulo n, ou seja, dada uma fatoração de n, pode-se chegar ao expoente da de criptografia.

Apesar da melhora constante dos algoritmos de fatoração de números inteiros, ainda é uma ameaça distante da realidade, caso o sistema RSA seja corretamente implementado, devido à dificuldade para a fatoração de números inteiros com a tecnologia atual.

# Estudo de caso

Foi proposta uma situação problema sobre um caso especifico que envolve restrição de acesso a uma área contaminada ambientalmente que contenha riscos a saúde pública, sendo o problema, um navio apreendido pela guarda costeira brasileira por transportar lixo toxico 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.

De acordo com o que o grupo apurou das horas estudadas sobre os tipos de criptografia chegou-se a uma conclusão de que uma possível solução seria uma troca de mensagens entre o navio e quem está em terra seria utilizando o método RSA.

# Interdisciplinaridade

Nosso projeto foi construído na linguagem de programação Python e outras matérias do nosso curso de Sistemas de Informação nos auxiliaram em alguns pontos do trabalho.

Portanto, a interdisciplinaridade de um projeto acadêmico é fundamental para o aprendizado e principalmente para o conhecimento teórico e técnico do sistema envolvido.

# Briefing do programa

O programa é feito na linguagem Python, e para rodar é necessário importar uma dependência Numpy para conseguirmos executar o programa. O programa necessita de usuários cadastrados na base de dados para conseguir enviar mensagens ou ler mensagens. Se o usuário não existir no banco, há uma opção de cadastro.

A base de dados é um JSON que constitui: um nome / login, uma senha para acessar o programa, um número primo sortido A, um outro número primo sortido B e os outros campos do banco são as chaves formadas por cálculos com os números A e B. Os números sortidos foram pegos de uma lista de números primos entre 10 a 300.

Na criação de um novo usuário, é aplicado a validação para ver se o nome ou login já existe no banco, se existir, um laço de repetição é instruído até o usuário digitar um login válido, se não existir, prosseguimos com o cadastro do usuário no banco. Neste momento, é feito também os cálculos RSA para gerar as chaves públicas e privadas.

Após o login ou o cadastro do novo usuário no sistema, temos o menu principal composto por: Ler mensagem, Escrever Mensagem, Créditos de quem construiu o programa, Logoff e qualquer tecla para encerrar o programa.

A opção de ler mensagem, é feita a busca na pasta de mensagens para verificar se existe uma outra pasta com o nome do usuário. Se não existir, criamos uma pasta e retornamos os arquivos contido na pasta, ou seja 0. Se existir uma pasta, retornamos os arquivos contido na pasta. Se não existir arquivos voltamos para o menu principal, se existir arquivos, listamos os arquivos e podemos escolher. Quando abrimos um arquivo, é feita a leitura do arquivo e utilizamos as chaves privadas do usuário logado para decriptar a mensagem. Quando os cálculos são feitos, retornamos a mensagem original.

A opção de escrever mensagem, listamos os usuários cadastrados no banco. Também é possível escrever uma mensagem apenas para o usuário logado, ou seja, uma mensagem de rascunho encriptada. Após selecionar a opção, é possível escrever a mensagem e para finalizar, na próxima linha, é necessário digitar “out”. Após a finalização da mensagem, é feito dois arquivos iguais se houver um destinatário, se não houver, é feito apenas um único arquivo. O primeiro arquivo é destinado ao próprio usuário, e os cálculos do RSA é aplicado utilizando as chaves públicas do usuário. Se houver um destinatário, é aplicado os cálculos RSA utilizando as chaves públicas do destinatário. É possível retornar para o menu principal ou escrever novamente a mensagem.

# Relatório com as linhas de código do programa

from random import choice, randint, sample, randrange

from math import pi, sqrt as r2

from datetime import datetime

from sys import exit

import os

import shutil

from getpass import getuser, getpass

from platform import system

import json

import numpy as np

# import jsonpickle as Njson

ascii\_table = {

"32":" ",

"33":"!",

"34":'"',

"35":"#",

"36":"$",

"37":"%",

"38":"&",

"39":"'",

"40":"(",

"41":")",

"42":"\*",

"43":"+",

"44":",",

"45":"-",

"46":".",

"47":"/",

"48":"0",

"49":"1",

"50":"2",

"51":"3",

"52":"4",

"53":"5",

"54":"6",

"55":"7",

"56":"8",

"57":"9",

"58":":",

"59":";",

"60":"<",

"61":"=",

"62":">",

"63":"?",

"64":"@",

"65":"A",

"66":"B",

"67":"C",

"68":"D",

"69":"E",

"70":"F",

"71":"G",

"72":"H",

"73":"I",

"74":"J",

"75":"K",

"76":"L",

"77":"M",

"78":"N",

"79":"O",

"80":"P",

"81":"Q",

"82":"R",

"83":"S",

"84":"T",

"85":"U",

"86":"V",

"87":"W",

"88":"X",

"89":"Y",

"90":"Z",

"91":"[",

"92":"\\",

"93":"]",

"94":"^",

"95":"\_",

"96":"`",

"97":"a",

"98":"b",

"99":"c",

"100":"d",

"101":"e",

"102":"f",

"103":"g",

"104":"h",

"105":"i",

"106":"j",

"107":"k",

"108":"l",

"109":"m",

"110":"n",

"111":"o",

"112":"p",

"113":"q",

"114":"r",

"115":"s",

"116":"t",

"117":"u",

"118":"v",

"119":"w",

"120":"x",

"121":"y",

"122":"z",

"123":"{",

"124":"|",

"125":"}",

"126":"~",

# "127":"00",

"128":"Ç",

"129":"ü",

"130":"é",

"131":"â",

"132":"ä",

"133":"à",

"134":"å",

"135":"ç",

"136":"ê",

"137":"ë",

"138":"è",

"139":"ï",

"140":"î",

"141":"ì",

"142":"Ä",

"143":"Å",

"144":"É",

"145":"æ",

"146":"Æ",

"147":"ô",

"148":"ö",

"149":"ò",

"150":"û",

"151":"ù",

"152":"ÿ",

"153":"Ö",

"154":"Ü",

"155":"ø",

"156":"£",

"157":"Ø",

"158":"×",

"159":"ƒ",

"160":"á",

"161":"í",

"162":"ó",

"163":"ú",

"164":"ñ",

"165":"Ñ",

"166":"ª",

"167":"º",

"168":"¿",

"169":"®",

"170":"¬",

"171":"½",

"172":"¼",

"173":"¡",

"174":"«",

"175":"»",

"176":"░",

"177":"▒",

"178":"▓",

"179":"│",

"180":"┤",

"181":"Á",

"182":"Â",

"183":"À",

"184":"©",

"185":"╣",

"186":"║",

"187":"╗",

"188":"╝",

"189":"¢",

"190":"¥",

"191":"┐",

"192":"└",

"193":"┴",

"194":"┬",

"195":"├",

"196":"─",

"197":"┼",

"198":"ã",

"199":"Ã",

"200":"╚",

"201":"╔",

"202":"╩",

"203":"╦",

"204":"╠",

"205":"═",

"206":"╬",

"207":"¤",

"208":"ð",

"209":"Ð",

"210":"Ê",

"211":"Ë",

"212":"È",

"213":"ı",

"214":"Í",

"215":"Î",

"216":"Ï",

"217":"┘",

"218":"┌",

"219":"█",

"220":"▄",

"221":"¦",

"222":"Ì",

"223":"Ó",

"224":"ß",

"225":"Ô",

"226":"Ò",

"227":"õ",

"228":"Õ",

"229":"µ",

"230":"þ",

"231":"Þ",

"232":"Ú",

"233":"Û",

"234":"Ù",

"235":"Ù",

"236":"ý",

"237":"Ý",

"238":"¯",

"239":"´",

# "240":" ",

"241":"±",

"242":"‗",

"243":"¾",

"244":"¶",

"245":"§",

"246":"÷",

"247":"¸",

"248":"°",

"249":"¨",

"250":"·",

"251":"¹",

"252":"³",

"253":"²",

"254":"■"

# "255":" "

}

# Achar os números primos:

def primos():

primos = []

for x in range(10, 300):

cont = 0

for y in range(1, x+1):

if x % y == 0:

cont+=1

if cont <= 2:

primos.append(x)

return primos

# Ler o banco de dados:

def checkDB():

# Nome do banco:

db\_File = "users.txt"

# Pega o path atual:

origin\_Path = os.getcwd()

# Verifica se existe um diretório; Se não é criado:

if ( not os.path.isdir('banco') ):

os.mkdir('banco')

# Entra no diretório do banco:

os.chdir('banco')

# Verifica se existe o arquivo do banco; Se não é criado:

if ( not os.path.isfile(db\_File) ):

f = open(db\_File, mode='w', encoding="utf8")

f.close()

# Lemos o arquivo do banco:

db = open(db\_File, mode='r', encoding="utf8")

data = db.read()

db.close()

# Voltamos para o path original:

os.chdir(origin\_Path)

# Retornamos os dados:

return data

# Achar um usuário do banco:

def listDB(field):

data\_JSON = checkDB()

data\_DICT = json.loads(data\_JSON)

r = []

for index, fields in data\_DICT.items():

for attr, value in fields.items():

if ( attr == field ):

r.append([field, value])

return r

# Listar os membros para mandar mensagem:

def listMembers(\_user):

# Pegamos os dados do arquivo do banco:

data\_JSON = checkDB()

# Transformamos o JSON do banco em DICT:

data\_DICT = json.loads(data\_JSON)

# Criação de variáveis:

# list\_members -> DICT que contém: { 'ID' : 'NOME' };

# members -> DICT que contém todas as chaves;

list\_members = {}

members = {}

# Pegamos o ID

id\_user = int

for index, fields in \_user.items():

id\_user = index

# Printamos a lista de usuários cadastrados no banco, removend o própio usuário logado:

cont = 1

turn = False

for index, fields in data\_DICT.items():

for attr, value in fields.items():

if ( attr == "name" ):

if ( index != id\_user ):

print(f"-----> {cont} : {value}")

list\_members[str(cont)] = value

members[str(index)] = fields

else :

if ( id\_user == "1" ):

cont = 0

else:

turn = True

if ( turn == True ):

turn = False

else:

cont += 1

print(f"-----> {len(data\_DICT)} : Apenas para mim")

# Retorna as variáveis:

return list\_members, members

# Verifica se é membro:

def isMember():

print('Bem vindo ao programa\n')

print('Já é membro ?\n--> 1 : Sim\n--> 2 : Não\n--> 3 : Sair\n')

member = input('--> ')

# Menu:

looping = 0

while ( looping == 0 ):

if ( member == '1'):

looping = 1

res = login()

break

elif ( member == '2' ):

looping = 1

res = register()

break

elif ( member == '3' ):

clear()

print('\nGoodbye \*-\*')

exit()

clear()

print('Bem vindo ao programa\n')

print('Já é membro ?\n--> 1 : Sim\n--> 2 : Não\n--> 3 : Sair\n')

member = input('--> ')

# Retorna a resposta da opção escolhida:

# Se for um DICT ou um INT;

return res

# Menu interno do programa:

def opening(user):

clear()

# Printa o nome do usuário:

for index\_user, attr\_user in user.items():

print(f"Bem Vindo: {attr\_user['name']}")

print('\*\*\* O que deseja fazer? \*\*\*\n')

print('-----> 1 - Ler Mensagem:')

print('-----> 2 - Escrever Mensagem:')

print('-----> 3 - Créditos:')

print('-----> 4 - Logoff:')

print('-----> Qualquer tecla para sair...\n')

response = input("--------> ")

# Escolha do usuário, passando como parâmetro os dados de quem fez login:

if ( response == '1' ):

return read(user)

elif ( response == '2' ):

return write(user)

elif ( response == '3' ):

return credit(user)

elif ( response == '4' ):

clear()

return main()

else:

clear()

print('\nGoodbye \*-\*')

exit()

# Tela de login:

def login():

clear()

print('\*\*\* LOGIN \*\*\*\n')

print("Digite seu login:")

name = input("--------> ")

print("Digite sua senha:")

passw = input("--------> ")

# Variáveis que utilizaremos neste escopo:

# guest -> DICT com os dados de quem fará o login;

# defined -> Bool para verificação dos dados em relação ao banco;

guest = {}

defined = False

# Acessamos o banco:

data\_JSON = checkDB()

# Se não existir dados no banco, retornamos para o menu principal;

if ( len(data\_JSON) == 0 ):

return 0

# Senão, convertemos o dado do banco que está em JSON, para DICT:

data\_DICT = json.loads(data\_JSON)

# Check dos dados inserios com o banco de dados:

for i, val in data\_DICT.items():

# Se o login e senha forem correspondente aos dados do banco:

# defined = True

if ( val["name"] == name and val["password"] == passw ):

guest[i] = val

defined = True

# De acordo com o Bool criado:

# Se os dados inseridos não estiverem no banco corretamente:

# É retornado um INT;

if ( defined == False ):

return 0

# Se os dados inseridos forem compatíveis com o banco:

# É retornado um DICT;

else:

return guest

# Tela de registro:

def register():

clear()

# Pegamos o Path atual:

origin\_Path = os.getcwd()

print('\*\*\* REGISTER \*\*\*\n')

print("Você quer fazer um registro?\n\n--> 1 : Sim\n--> 2 : Voltar ao Menu\n--> Qualquer Tecla para sair\n")

q = input("--------> ")

# Menu de Registro:

# Se o input for igual a '1'

if ( q == "1" ):

clear()

# Um banco de números primos, gerado através da função:

privateKeys = np.array([11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293])

print('\*\*\* REGISTER \*\*\*\n')

print("Digite seu login: ")

name = input("--------> ")

# Importamos o banco de dados:

data\_JSON = checkDB()

# Se contém algum dado do banco:

if ( len(data\_JSON) > 0 ):

# Transforma o JSON em DICT:

data\_DICT = json.loads(data\_JSON)

# Verificação se existe algum nome igual ao do banco:

checkName = False

while ( checkName == False ):

checkName = True

# Laço FOR para acessar o ATTR do DICT chamado 'name':

# Se o nome digitado pelo usuário for igual ao do banco:

# Validação feita se o nome for igual independente do tipo de letra (UPPER ou LOWER);

# Continuamos no laço WHILE;

for i, field in data\_DICT.items():

for j, val in field.items():

\_thisName = name

\_nameUpper = \_thisName.upper()

\_nameLower = \_thisName.lower()

\_thisNameDB = val

\_nameDBUpper = \_thisNameDB.upper()

\_nameDBLower = \_thisNameDB.lower()

if ( name == val or

\_nameUpper == \_nameDBUpper or

\_nameLower == \_nameDBLower ):

checkName = False

if ( checkName == True ):

break

clear()

print('\*\*\* REGISTER \*\*\*\n')

print("Digite um login válido.")

name = input("Name --> ")

# Se não existir um nome igual ao do banco, passamos para as próximas etapas:

print("\nDigite uma senha: ")

passw = input("--------> ")

# Se existe algum dado guardado no banco:

# Se existir, pegamos o tamanho dos dados e somamos + 1, para gerar o ID automático;

if ( len(data\_JSON) > 0 ):

user\_Id = len(data\_DICT) + 1

# Se não existir, criamos um DICT e definimos o primeiro ID que será inserido do banco como 1;

else :

data\_DICT = {}

user\_Id = 1

# Pegamos os números primos aleatóriamente:

# Primeira chave privada:

privateKeyA = choice(privateKeys)

# Segunda chave privada:

privateKeyB = choice(privateKeys)

# Enquanto o KeyB for igual ao KeyA, pegamos aleatóriamente outro número para KeyB

while ( privateKeyA == privateKeyB ):

privateKeyB = choice(privateKeys)

# Geramos a primeira chave pública:

publicKeyAB = privateKeyA \* privateKeyB

# Geramos o número Totiente / Phi de Euler de publicKeyAB:

fiAB = (privateKeyA - 1) \* (privateKeyB - 1)

# Chave pública E:

# Regra para achar o número: 1 > E < fiAB;

# O número E tem que ser Coprimo de fiAB, ou seja, o MDC de X e fiAB precisa ser APENAS 1;

# Variável que começará a achar o número E:

increm = 2

# Pegamos o resto do MDC entre 2 e fiAB:

rMDC = searchNumE(increm, fiAB)

# Enquanto o resto do MDC for maior que 1 ( Utilizaremos esse WHILE para achar um número COPRIMO de fiAB ):

while ( rMDC > 1 ):

# Incremento: Antes era 2. Se entramos nesse WHILE é pq é maior que 1; Então incrementamos:

increm += 1

rMDC = searchNumE(increm, fiAB)

# Quando o resto do MDC de increm e fiAB for igual 1, achamos o nossa outra chave pública E com increm:

publicKeyE = increm

# Geramos a chave inversa:

for x in range(fiAB):

if ( (x\*publicKeyE)%fiAB == 1 ):

privateInverse = x

# Criamos um DICT de todos os dados gerados:

data\_DICT[str(user\_Id)] = {"name":name, "password":passw, "privateKeyA":str(privateKeyA), "privateKeyB":str(privateKeyB), "publicKeyAB":str(publicKeyAB), "fiAB":str(fiAB), "publicKeyE":str(publicKeyE), "privateInverse":str(privateInverse)}

# Entramos na pasta do banco e adicionamos os dados:

os.chdir('banco')

db = open('users.txt', 'w')

data\_JSON = json.dumps(data\_DICT, ensure\_ascii=False, indent=4, separators=(',', ': '))

db.write(data\_JSON)

db.close()

# Voltamos para a pasta do executável:

os.chdir(origin\_Path)

# Retornamos o DICT que contém o ID e todos os valores:

for \_i\_db, \_attr\_db in data\_DICT.items():

if ( \_i\_db == str(user\_Id) ):

return {\_i\_db : \_attr\_db }

# Se o input for igual a '2'

elif ( q == "2" ):

clear()

main()

# Encerramos o programa:

else:

print('Programa Encerrado!')

exit()

# Função MDC (Máximo Divisor Comum):

def searchNumE(increment, fiAB):

r = increment % fiAB

if ( r==0 ):

return r

else:

while ( r != 0 ):

increment = fiAB

fiAB = r

r = increment % fiAB

return fiAB

# Função para encripitar:

def encripty(decimal, publicKeyE, publicKeyAB):

decimal = int(decimal)

publicKeyE = int(publicKeyE)

publicKeyAB = int(publicKeyAB)

result = ((decimal\*\*publicKeyE) % publicKeyAB)

return str(result)

# Função para decripitar:

def decripty(encripty, privateInverse, publicKeyAB):

encripty = int(encripty)

privateInverse = int(privateInverse)

publicKeyAB = int(publicKeyAB)

result = ((encripty\*\*privateInverse) % publicKeyAB)

return str(result)

# Tela de Créditos:

def credit(user):

clear()

# Printa o nome do usuário:

for index\_user, attr\_user in user.items():

print(f"Bem Vindo: {attr\_user['name']}")

print('\*\*\* CRÉDITOS \*\*\*\n')

grupo = [

["Gustavo Pereira do Amor Divino", "N508612"],

["Ian Cozachevici de Jesus", "N503530"],

["Jennifer Satirio Lima Santos", "N51608-9"]

]

for alunos in grupo:

for order, aluno in enumerate(alunos):

if ( order == 0 ):

print(f"==> Nome: {aluno}")

elif ( order == 1 ):

print(f"==> R.A: {aluno}")

print("<---------------------->")

input('-----> Digite algo para continuar... ')

clear()

# Printa o nome do usuário:

for index\_user, attr\_user in user.items():

print(f"Bem Vindo: {attr\_user['name']}")

print('\*\*\* O que deseja fazer? \*\*\*\n')

print('-----> 1 - Voltar ao Menu de Opções:')

print('-----> 2 - Logoff:')

print('-----> Qualquer tecla para sair...\n')

response = input("--------> ")

# Menu:

if ( response == '1' ):

return opening(user)

elif ( response == '2' ):

clear()

return main()

else:

clear()

print('\nGoodbye \*-\*')

exit()

# Ler mensagens:

def read(\_user):

# Guardo os dados do usuário logado no dict:

this\_user = {}

for id\_user, fields in \_user.items():

this\_user = fields

# Path do executável:

origin\_Path = os.getcwd()

# Check da pasta mensagens;

# Se não existir, criamos.

if ( not os.path.isdir('msgs') ):

os.mkdir('msgs')

# Entrando na pasta mensagens:

os.chdir('msgs')

# Path da pasta mensagens:

msg\_Path = os.getcwd()

# Se não existir a pasta do usuários de mensagens:

if ( not os.path.isdir(this\_user['name']) ):

# Menu:

clear()

print('\*\*\* READ \*\*\*\n')

print('Não há mensagens para você...\n')

print('\*\*\* O que deseja fazer? \*\*\*\n')

input('Digite algo para voltar ao menu de opções: ')

# Voltamos para o path do executável:

os.chdir(origin\_Path)

opening(\_user)

# Entrando na pasta mensagens do usuário:

os.chdir(this\_user['name'])

# Path da pasta do Usuário de mensagens:

user\_path = os.getcwd()

# Check da mensagem selecionada:

iteration = False

while ( iteration == False ):

clear()

print('\*\*\* READ \*\*\*\n')

print("--------> Lista de arquivos:\n")

# Variável que iremos adicionar o nome dos arquivos:

files\_dict = {}

# Listamos os arquivos e adicionamos no DICT:

for index, files in enumerate(os.listdir('.')):

files\_dict[index+1] = files

print(f"--> {index+1} : {files}")

print("\nDigite o arquivo desejado: ")

# Variável do arquivo selecionado;

file = input("--------> ")

# Pequena validação dos espaços em branco:

if ( file == '' or file == ' '):

file = 0

file = int(file)

# Verificação do número digitado:

for i in range(1, len(os.listdir('.'))+1):

if ( file == i ):

iteration = True

break

# Verificamos no DICT criado dos arquivos:

# Se o número digitado pra ler for igual ao DICT criado dos arquivos:

# Pegamos o nome completo do arquivo e guardamos na variável f\_name;

for index, files in files\_dict.items():

if ( index == file ) :

f\_name = files

# Variáveis para ler a

encript = []

decript = []

return\_ascii = []

# Lendo o arquivo com o nome desejado:

f\_read = open(f\_name, mode="r", encoding="utf8")

for x in f\_read.readlines():

# Retirando as quebras de linhas:

r = x.rstrip('\n')

# Retirando os espaços de #

r = x.split('#')

# Adicionando no array:

encript.append(r)

f\_read.close()

# Decripita a Mensagem:

for frase in encript:

frase\_decript\_To\_Me = []

# Chamada da função, passando como parâmetro ( NºdaLetra, ChaveInversa, ChavePublicaAB ):

for letter in frase:

to\_me = decripty(letter, this\_user['privateInverse'], this\_user['publicKeyAB'])

# Passo o resultado para a lista criada:

frase\_decript\_To\_Me.append(to\_me)

# Passo o vetor do valor para outra lista:

decript.append(frase\_decript\_To\_Me)

# Converte os valores ASCII para Caracteres:

for value in decript:

ascii\_by\_letter\_To\_Me = []

for uni\_value in value:

for decimal, char in ascii\_table.items():

# Se o valor decripitado for igual ao decimal do banco ASCII, adicionamos em uma lista

if ( uni\_value == decimal ):

ascii\_by\_letter\_To\_Me.append(char)

# Quando a frase estiver toda convertida, tratamos o vetor para adicionar um única frase em um índice da lista

return\_ascii.append(''.join(ascii\_by\_letter\_To\_Me))

# Resultado da mensagem:

retorno = False

while ( retorno == False ):

clear()

# Print da mensagem decripitada:

print('\n-------> MESSAGE: <-------')

for response in return\_ascii:

print(response)

print('\n-------> END: MESSAGE <-------')

# Menu:

print('\n\*\*\* O que deseja fazer? \*\*\*\n')

print('-----> 1 - Ler mensagens:')

print('-----> 2 - Voltar a menu de opções:')

choice = input("--> ")

if ( choice == '1' ):

retorno = True

os.chdir(origin\_Path)

read(\_user)

break

elif ( choice == '2' ):

retorno = True

os.chdir(origin\_Path)

opening(\_user)

break

# Escrever mensagens:

def write(\_user):

# Só podemos escrever algum arquivos se tivemos PELO MENOS UM dado no banco:

# Variáveis para pegar a data e horário antes de escrever a mensagem:

now = datetime.now()

year = now.strftime('%Y')

month = now.strftime('%B')

day = now.strftime('%d')

hour = now.strftime('%H')

minute = now.strftime('%M')

second = now.strftime('%S')

dateToday = '-'.join([hour, minute, second, year, month, day])

# Conexão com o banco:

data\_JSON = checkDB()

# Convertendo o JSON em DICT:

data\_DICT = json.loads(data\_JSON)

# Path do executável:

origin\_Path = os.getcwd()

# Se não existir o diretórito msgs, criamos:

if ( not os.path.isdir('msgs') ):

os.mkdir('msgs')

# Pegando os campos do usuário logado e criação da variável Sketch (rascunho):

sketch = False

this\_user = {}

for id\_user, fields in \_user.items():

this\_user = fields

# Validação do número digitado:

iteration = False

while ( iteration == False ):

clear()

print('\*\*\* WRITE \*\*\*\n')

print("--------> Lista de usuários:\n")

# Listando os usuários disponíveis no banco para mandar a mensagem, passando como parâmetro o usuário logado:

result\_members = listMembers(\_user)

# Guardando a lista de usuários:

list\_members = result\_members[0]

# Guardando as chaves dos respectivos usuários:

members = result\_members[1]

print("\nDigite o usuário: ")

who = input("--------> ")

# Pequena validação dos espaços em branco:

if ( who == '' or who == ' '):

who = 0

who = int(who)

# Verificando se o número digitado consta na lista de membros, tirando o número zero e adicionando o próprio usuários -> len(list\_members)+2

for i in range(1, len(list\_members)+2):

if ( who == i ):

iteration = True

break

# Validando se o usuário escolheu ele mesmo:

if ( len(list\_members)+1 == who ):

for id\_user, fields in \_user.items():

sketch = True

who = fields

# Validando se foi outra pessoa:

else :

# Verificação do nome em relação ao numero da lista:

for choose, fields in list\_members.items():

if ( who == int(choose) ):

who = fields

# Verificação do campo das chaves em relação ao nome:

for user\_id, fields in members.items():

if ( fields["name"] == who ):

who = fields

# Entrando na pasta mensagens:

os.chdir('msgs')

# Path das Mensagens:

path\_msgs = os.getcwd()

# Criação da pasta do usuário, se ainda não estiver criado:

if ( not os.path.isdir(f"{this\_user['name']}") ):

os.mkdir(f"{this\_user['name']}")

# Criação da pasta do destinatário e validando se foi apenas pra ele mesmo:

# Se ele escolheu ele mesmo, a pasta não é criada. Se for diferente dele, criaremos a pasta desta pessoa, se ainda não estiver criada;

if ( not os.path.isdir(f"{who['name']}") and sketch == False ):

os.mkdir(f"{who['name']}")

# Entrando na pasta do usuário:

os.chdir(f"{this\_user['name']}")

# Path da pasta do usuário:

path\_user = os.getcwd()

# Retornando ao path das Mensagens:

os.chdir(path\_msgs)

# Entrando e pegando o path da pasta do destinátario, lembrando que só fará isto se for para uma pessoa do banco, diferente dele mesmo:

if ( sketch == False ):

os.chdir(f"{who['name']}")

path\_who = os.getcwd()

# Retornando ao path das Mensagens:

os.chdir(path\_msgs)

# Preparação para escrever a mensagem:

clear()

# Variável de saída do texto:

out = "out"

# Variável do texto:

txt = []

# Laço WHILE para escrever a mensagem:

print('\*\*\* Escreva sua mensagem \*\*\*\n')

print("-----> Para encerrar sua mensagem, na próxima linha digite: 'out' \n")

loop = 0

while ( loop == 0):

msg = input('--> ')

if (out in msg):

loop = 1

break

txt.append(msg)

# Variáveis para encripitar:

encript\_To\_Who = []

encript\_To\_Me = []

# Converte o texto digitado em ASCII e Encripita:

for frase in txt[:]:

frase\_encripty\_To\_Who = []

frase\_encripty\_To\_Me = []

for letter in frase:

for decimal, char in ascii\_table.items():

if ( letter == char ):

# Função encripty;

to\_who = encripty(decimal, who['publicKeyE'], who['publicKeyAB'])

to\_me = encripty(decimal, this\_user['publicKeyE'], this\_user['publicKeyAB'])

frase\_encripty\_To\_Who.append(to\_who)

frase\_encripty\_To\_Me.append(to\_me)

encript\_To\_Who.append(frase\_encripty\_To\_Who)

encript\_To\_Me.append(frase\_encripty\_To\_Me)

# Ajustando para gravar a mensagem:

# Aqui criamos as novas variáveis, antes de gravar no arquivo;

# Cada letra será separado por '#'

# Cada frase, está em sua respectiva ordem:

encript\_To\_Who\_New = []

encript\_To\_Me\_New = []

for x in encript\_To\_Who:

r = "#".join(x)

encript\_To\_Who\_New.append(r)

for y in encript\_To\_Me:

r = "#".join(y)

encript\_To\_Me\_New.append(r)

# Nome do arquivo será a data e hora do momento:

name\_file\_user = dateToday+".txt"

# Criando o arquivos:

f\_user = open(name\_file\_user, mode="w", encoding="utf8")

for frase in encript\_To\_Me\_New:

f\_user.write(frase)

f\_user.write('\n')

f\_user.close()

# Movemos o arquivo para a pasta do usuário:

shutil.copy2(name\_file\_user, path\_user)

# Removendo o arquivo no diretório msgs:

os.remove(name\_file\_user)

# Se não for rascunho, fazemos o mesmo procedimento:

if ( sketch == False ):

# Criando o arquivos:

f\_user = open(name\_file\_user, mode="w", encoding="utf8")

for frase in encript\_To\_Who\_New:

f\_user.write(frase)

f\_user.write('\n')

f\_user.close()

# Movemos o arquivo para a pasta do usuário:

shutil.copy2(name\_file\_user, path\_who)

# Removendo o arquivo no diretório msgs:

os.remove(name\_file\_user)

# Menu

# Verificando o que devemos fazer após a operação:

retorno = False

while ( retorno == False ):

clear()

print('\*\*\* O que deseja fazer? \*\*\*\n')

print('-----> 1 - Enviar novamente uma mensagem:')

print('-----> 2 - Voltar a menu de opções:')

choice = input("--> ")

if ( choice == '1' ):

retorno = True

os.chdir(origin\_Path)

write(\_user)

break

elif ( choice == '2' ):

retorno = True

os.chdir(origin\_Path)

opening(\_user)

break

# Função para limpar a tela:

def clear():

return os.system('cls')

# Início do programa:

def main():

# Verifica se é membro:

res = isMember()

# Verifica se é um DICT:

# Se for, chama a função do menu interno, com os dados de quem fez login:

if ( isinstance(res, dict) ):

opening(res)

# Se for um INT:

# Retorna pra função principal, main();

elif ( isinstance(res, int) ):

clear()

main()

return 0

exit()

# Chamada do programa:

main()

# Conclusão

Diante do tema abordado e discutido no trabalho identificou-se que o armazenamento de dados por meios eletrônicos vem aumentando cada vez mais com o passar dos anos, as quais necessitam de proteção contra o acesso de terceiros não autorizados. A criptografia é uma das ferramentas mais utilizadas para se comunicar secretamente, estabelecendo assim uma mensagem na qual só o remetente e destinatário possa decifrar a mensagem.

Tem como objetivo codificar os dados a serem transmitidos de forma que um texto legível se transforme em um texto ilegível. Um dos instrumentos necessários para o uso de criptografia é o uso de algoritmos criptográficos.

A criptografia é importante para estabelecer uma comunicação de forma segura, historicamente utilizadas desde os gregos, criptografia RSA foi criada no MIT em 1977 é o algoritmo mais amplamente utilizado de chave pública, além de suas formas de criptografia serem uma das mais poderosas de chave pública conhecidas até hoje, utilizando de diversos números primos.

A realização deste trabalho mostrou-se muito interessante, pois conclui-se que a criptografia é totalmente necessária e importante para os meios de comunicação e informação que utilizamos hoje, já que podemos constatar um crescente aumento no uso de da internet com uma frequente troca de dados entre servidores e usuários que transmitem dados pessoais que necessitam de sigilo e segurança.

# Bibliografia

ASSOCIAÇÃO DA JUVENTUDE ISRAELITA HEHAVER (Portugal). Mezuzah. 2017. Disponível em: <http://rivkah.com.br/tradicoes/mezuzah.htm>.

BENATO, Gabriel (Ed.). MENSAGENS CRIPTOGRAFADAS, O QUE É, E COMO FUNCIONAM. 2015. Disponível em: <https://www.brasilwebhost.com.br/blog/se-proteja-usando-os-aplicativos-de-mensagens/#>.

MYERS, Lysa (Ed.). Tudo sobre criptografia: o que é e quando devemos usar? 2017. Disponível em: <https://www.welivesecurity.com/br/2017/08/31/tudo-sobre-criptografia-quando-usar/>.

NETWORKING (Ed.). **Criptografia simétrica e assimétrica. Sabe a diferença?**2010. Disponível em: <https://pplware.sapo.pt/tutoriais/networking/criptografia-simetrica-e-assimetrica-sabe-a-diferenca/>.

PINTO, Pedro (Ed.). Criptografia simétrica e assimétrica. Sabe a diferença? Sabe a diferença? 2010. Disponível em: <https://pplware.sapo.pt/tutoriais/networking/criptografia-simetrica-e-assimetrica-sabe-a-diferenca/>.

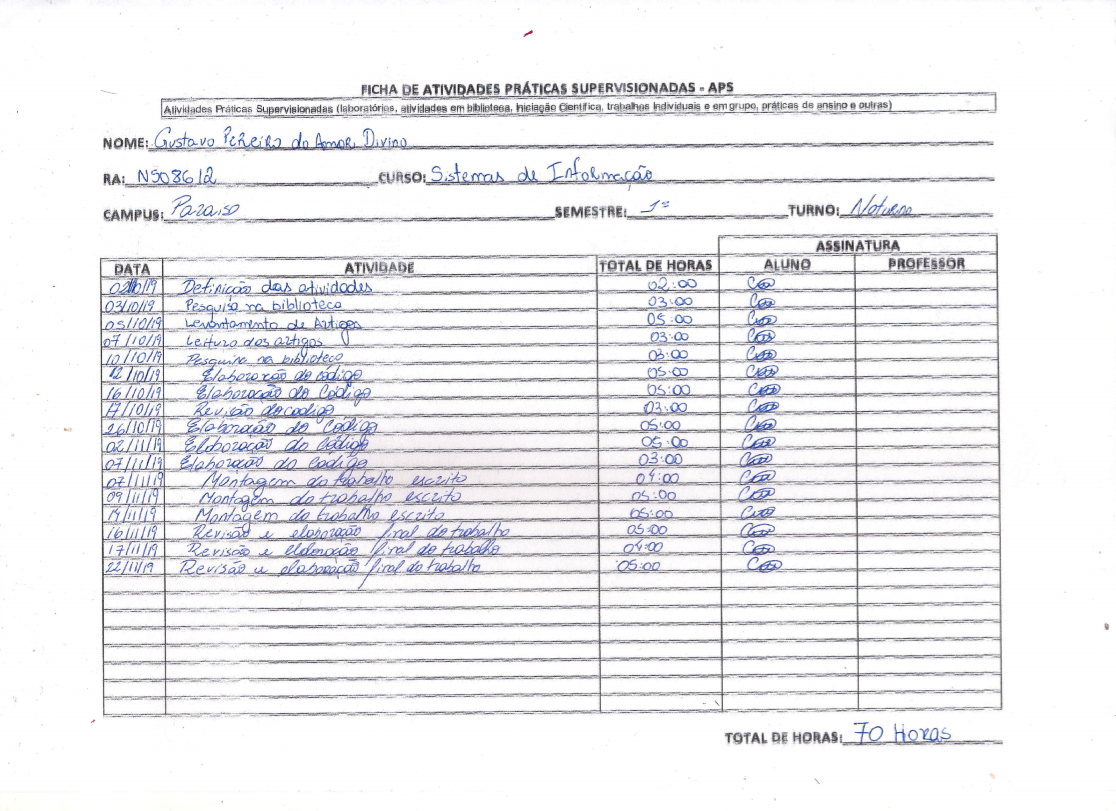
ROMAGNOLO, Cesar Augusto (Ed.). O que é Criptografia. 2017. Disponível em: <https://www.oficinadanet.com.br/artigo/443/o\_que\_e\_criptografia>.

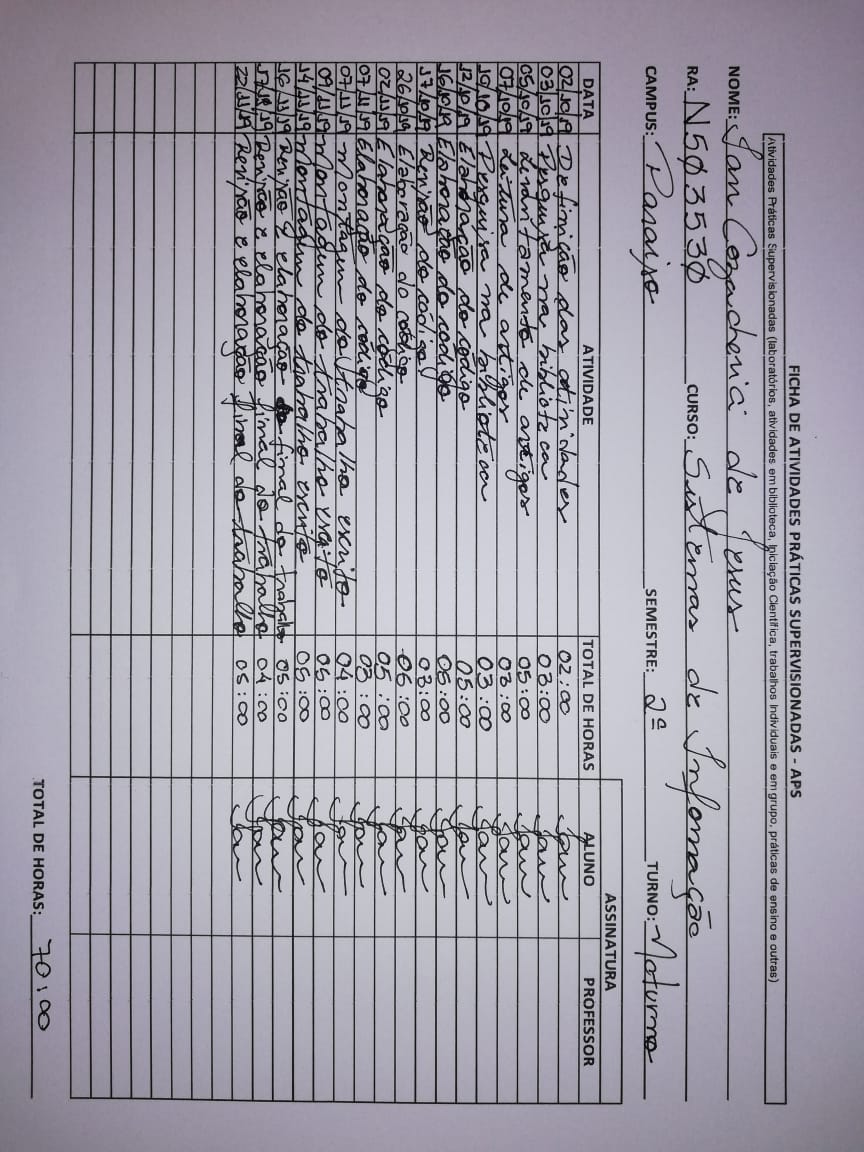
VAZ, Marcela. **O que é SSL?**2013. Disponível em: <https://www.techtudo.com.br/artigos/noticia/2012/01/o-que-e-ssl.html>.

SILVA, Maria Piedade Teodoro da. A criptografia na Segunda guerra mundial. 2015. Disponível em: <https://pt.slideshare.net/Reberthsantos/reberth>.

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO (Rio de Janeiro). Universidade Federal do Rio de Janeiro (Ed.). Criptografia Quântica. 2007. Disponível em: <https://www.gta.ufrj.br/grad/07\_1/quantica/Substituio.html>.

# Fichas de horas APS

****

****