

UNIVERSIDADE FEDERAL FLUMINENSE
INSTITUTO DE COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Raphael Bernardino Ferreira Lima
Igor Gonçalves Agarra Galvão

FERRAMENTA PARA AUXÍLIO DE
APRENDIZADO EM CRIPTOGRAFIA

Niteroi-RJ

2016

RAPHAEL BERNARDINO FERREIRA LIMA
IGOR GONÇALVES AGARRA GALVÃO

FERRAMENTA PARA AUXILIO DE APRENDIZADO EM CRIPTOGRAFIA

Trabalho submetido ao Curso de
Bacharelado em Ciência da Computação
da Universidade Federal Fluminense como
requisito parcial para a obtenção do título
de Bacharel em Ciência da Computação.

Orientador: Prof. Célio Vinicius Neves de Albuquerque

Co-orientador: Prof. Luis Antonio Brasil Kowada

Niteroi-RJ

2016

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

L732 Lima, Raphael Bernardino Ferreira

Ferramenta para auxílio de aprendizado em criptografia / Raphael Bernardino Ferreira Lima, Igor Gonçalves Agarra Galvão. – Niterói, RJ : [s.n.], 2016.
63 f.

Trabalho (Conclusão de Curso) – Departamento de Computação, Universidade Federal Fluminense, 2016.

Orientadores: Célio Vinicius Neves de Albuquerque, Luis Antonio Brasil Kowada.

1. Ferramenta computacional. 2. Criptografia. 3. Algoritmo. I. Título.


CDD 004

RAPHAEL BERNARDINO FERREIRA LIMA
IGOR GONÇALVES AGARRA GALVÃO

FERRAMENTA PARA AUXILIO DE APRENDIZADO EM CRIPTOGRAFIA

Trabalho submetido ao Curso de
Bacharelado em Ciência da Computação
da Universidade Federal Fluminense como
requisito parcial para a obtenção do título
de Bacharel em Ciência da Computação.

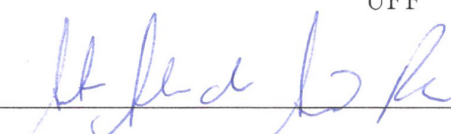
Aprovado por:



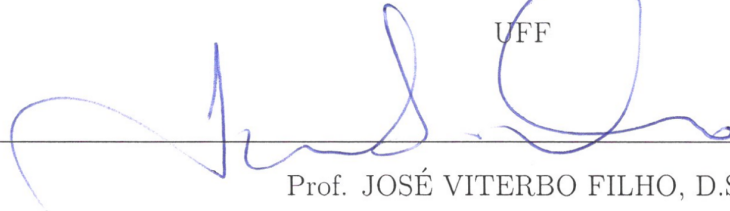
Prof. CÉLIO VINICIUS NEVES DE ALBUQUERQUE, Ph.D. - Orientador
UFF



Prof. LUIS ANTONIO BRASIL KOWADA, D.Sc. - Co-orientador
UFF



Prof. ANTONIO-AUGUSTO DE ARAGÃO ROCHA, D.Sc.
UFF



Prof. JOSÉ VITERBO FILHO, D.Sc.
UFF

Niterói-RJ

2016

Dedicamos este trabalho a todas aquelas pessoas que nos ajudaram a concluí-lo.

Agradecimentos

Gostaríamos de agradecer primeiramente a Deus.

Em seguida, aos nossos pais por tudo o que fizeram por nós.

Também queremos agradecer aos nossos orientadores Célio Vinicius de Albuquerque e Luis Antonio Kowada, por permitir a realização deste trabalho.

Agradecemos a Antonio Augusto Rocha e José Viterbo Filho por terem aceitado participar da banca deste trabalho.

E por fim, queremos deixar registrado nosso agradecimento a todos os que participaram dos testes desta ferramenta.

Resumo

Este trabalho tem como propósito desenvolver uma ferramenta computacional que ajude o processo de ensino-aprendizagem, através de visualização, dos fundamentos da Criptografia e de alguns dos principais algoritmos usados para Criptografia Simétrica, Chave Pública, e Funções de Dispersão. Os algoritmos escolhidos foram AES para chave simétrica, RSA para chave assimétrica e MD5 para função de dispersão. A ferramenta desenvolvida é inspirada em algumas ferramentas como SHAVisual, RSAVisual e DESVisual. Para mensurar a eficácia da ferramenta, aplicamos testes práticos com grupos de pessoas, antes e depois do uso da ferramenta.

Palavras-chave: *Criptografia, Algoritmos criptográficos, Funções hash, Ferramenta de visualização, AES, RSA, MD5*

Abstract

This work has the intention to develop a computational tool that helps teaching-learning process, using the visualization, the fundamentals of cryptography and some of the main algorithms used for symmetric encryption, public key, and hash functions. The chosen algorithms were AES for symmetric key, RSA for asymmetric key and MD5 for hash function. The developed tool is inspired by a few tools as SHAVisual, RSAVisual and DESVisual. In order to measure the effectiveness of the tool, we apply practical tests with groups of people, before and after the use of the tool.

Keywords: *Cryptography, Cryptographic algorithms, Hash functions, Visualization tool.*

Sumário

Resumo	vi
Abstract	vii
Lista de Figuras	xi
Lista de Tabelas	xii
1 Introdução	1
1.1 Algoritmos e funções criptográficas	2
1.1.1 Funções Hash	2
1.1.2 Criptografia de chave simétrica	3
1.1.3 Criptografia de chave assimétrica ou de Chave Pública	3
1.2 Objetivos	5
1.3 Metodologia	5
2 Algoritmos criptográficos e Funções Hash	7
2.1 MD5	7
2.1.1 Passo 1	8
2.1.2 Passo 2	8
2.1.3 Passo 3	8
2.1.4 Passo 4	8
2.2 AES	10
2.2.1 SBOX	11
2.2.2 RCON	11
2.2.3 Expansão de chaves	11
2.2.4 AddRoundKey	14

2.2.5	SubBytes	14
2.2.6	ShiftRows	15
2.2.7	MixColumns	15
2.2.8	Encriptação	15
2.2.9	Decriptação	16
2.3	RSA	16
2.3.1	Geração das chaves	17
2.3.2	Encriptação	17
2.3.3	Decriptação	18
3	Ferramenta	19
3.1	Implementação	20
3.2	Imagens	20
4	Testes	28
4.1	Questionário Pré Utilização	28
4.2	Questionário Pós Utilização	32
4.3	Análise dos resultados	35
5	Conclusão	36
	Apêndices	40
	Apêndice A Instalação da Ferramenta	41
A.1	Instalação	41
A.1.1	Instalando Python	41
A.1.2	Instalando Flask	41
A.1.3	Instalando a Ferramenta	41
	Apêndice B Perguntas Testes	43
B.1	Questionário Pós Utilização	43
B.1.1	Perguntas referentes ao MD5	43
B.1.2	Perguntas referentes a chave simétrica e assimétrica	46
B.1.3	Perguntas referentes ao AES	48
B.1.4	Perguntas referentes ao RSA	49

Lista de Figuras

2.1	Ilustração do passo 4 do MD5	10
3.1	Página inicial da ferramenta.	21
3.2	Detalhes teóricos sobre o algoritmo MD5.	21
3.3	Formulário responsável pelas entradas do algoritmo MD5.	22
3.4	Demonstração do funcionamento no passo 4.	23
3.5	Detalhes das funções utilizadas em cada rodada.	23
3.6	Explicação teórica do funcionamento do RSA.	24
3.7	Formulário de entrada para os parâmetros do RSA.	24
3.8	Etapa inicial de converter o texto de entrada em caracteres ASCII.	25
3.9	Exposição das chaves pública e privada geradas.	25
3.10	Detalhes teóricos do AES.	26
3.11	Formulário que captura as entradas para o algoritmo AES.	26
3.12	Utilização da tabela RCON na etapa de expansão de chaves.	27
3.13	Demonstração do procedimento <i>SubBytes</i> nas rodadas intermediárias.	27
4.1	Pergunta pré utilização sobre idade.	29
4.2	Pergunta pré utilização sobre escolaridade.	29
4.3	Pergunta pré utilização sobre chave assimétrica.	30
4.4	Pergunta pré utilização sobre chave simétrica.	30
4.5	Pergunta pré utilização sobre funções de dispersão.	30
4.6	Pergunta pré utilização sobre interesse na ferramenta.	31
4.7	Pergunta pré utilização sobre nível de conhecimento do usuários.	31
4.8	Pergunta pré utilização sobre utilização de outras ferramentas.	31
4.9	Distribuição de pontos questionário de chave simétrica.	33
4.10	Distribuição de pontos questionário de chave assimétrica.	33

4.11 Distribuição de pontos questionário de função hash.	34
4.12 Opinião sobre a ferramenta no questionário de chave simétrica.	34
4.13 Opinião sobre a ferramenta no questionário de chave assimétrica.	34
4.14 Opinião sobre a ferramenta no questionário de funções hash.	35

Lista de Tabelas

2.1	Tabela SBOX	12
2.2	Tabela SBOX-Inv	13
2.3	Tabela RCON. Em negrito as posições que são, de fato, utilizadas.	13
2.4	Tabela de corpo finito de Galois	15
2.5	Tabela Inversa de corpo finito de Galois	15

Capítulo 1

Introdução

Proteger informações sempre foi muito importante em nossa sociedade, com a chegada da tecnologia computacional, algoritmos criptográficos modernos foram desenvolvidos e otimizados para que essa segurança fosse muito mais forte. Utilizamos desde uma simples Cifra de César, homenagem a Júlio César, que usava substituição de letras para troca de mensagens entre seus generais em torno do ano 44 a.C. E o que mais surpreende é que tal algoritmo foi utilizado até 1915, onde as grandes guerras aceleraram os estudos para o desenvolvimento de novas técnicas, criando novos e mais robustos algoritmos, tendo como seu principal destaque a máquina Enigma [6].

Com a evolução dos computadores foi necessário aprimorar ainda mais a criptografia, criando algoritmos mais seguros, tais como o RSA (Rivest-Shamir-Adleman) [9] que foi desenvolvido em 1977 por empresa de mesmo nome, MD5 (*Message Digest 5*) [7] desenvolvido em 1991 para suceder o MD4 (*Message Digest 4*) e o AES (*Advanced Encryption Algorithm*) [2] desenvolvido em 2001 em um concurso para escolher o padrão de criptografia do governo dos Estados Unidos da América.

Tais algoritmos são amplamente utilizados por empresas, na comunicação entre sistemas computacionais, preservação de dados sensíveis, integridade, autenticidade e não-repúdio de informações, visando proteger seu conteúdo e de seus clientes. E por isso esse tema é presente no ensino de segurança da informação, tendo como prioridade mostrar a qualidade do sigilo da mensagem, em detrimento da descrição do mecanismo de funcionamento.

A utilização de uma ferramenta que mostra o passo-a-passo desses algoritmos criptográficos auxiliaria docentes e alunos no entendimento, facilitando o aprendizado sobre

como tais algoritmos funcionam, tanto para suas vantagens quanto para as suas falhas.

Abordaremos três principais algoritmos, sendo cada um o mais popular e utilizados dentre as categorias escolhidas: MD5 (Funções *Hash*), AES (Algoritmo de Chave Simétrica) e RSA (Algoritmo de Chave Assimétrica). Mostrando cada um através de animações feitas utilizando slides. Tendo como inspiração ferramentas já desenvolvidas com propósitos parecidos como o SHAvistual [4], DESvistual [12], RSAvistual [11]. Tais ferramentas não entram em detalhes em relação aos passos dos algoritmos, o que nos auxiliou a desenvolver uma ferramenta mais didática e explicativa.

1.1 Algoritmos e funções criptográficas

1.1.1 Funções Hash

Uma função *hash*, ou de dispersão, é uma função definida por um algoritmo que aplicado a uma entrada de tamanho arbitrário retorna uma sequência de tamanho fixo. Quando o tamanho da saída é menor do que o tamanho da entrada, a função não pode ser bijetora, e portanto não é possível recuperar a entrada a partir da saída. Neste caso, dizemos que a função é de via única (*one way*). Se o algoritmo associado à função *hash* for simples e rápido, podemos usá-lo para obter um resumo (*checksum*) do arquivo de forma fácil e prática.

Muitos consideram a função *hash* uma função criptográfica, visto que é utilizada comumente no contexto de segurança e transforma a informação original, tornando-a ilegível. Diferentemente dos algoritmos criptográficos que são de via dupla (reversíveis), ou seja, é possível recuperar a informação original a partir da saída do algoritmo ¹.

O valor retornado por uma função *hash* é chamado de *hash*, valor *hash*, código *hash* ou *checksum*. Devemos ressaltar também que por questões de codificação, geralmente, são representados em base hexadecimal.

É altamente recomendável utilizar funções *hash* em casos de armazenamento de senhas ou na disponibilização de arquivos na internet, com a finalidade de verificar a autenticidade e a integridade do arquivo baixado através de *checksum*. Neste último caso pode ocorrer alguma perda de pacote no meio e corromper o arquivo final, ou até mesmo

¹Apesar de alguns autores não concordarem com essa distinção entre função criptográfica e algoritmo criptográfico.

ocorrer alguma mudança do(s) dado(s) por um agente mal intencionado (intruso).

Considerando que várias entradas de uma função *hash* produzem a mesma saída, podemos medir a qualidade pela uniformidade do número de colisões para cada *hash* [10]. Uma função *hash* é dita perfeita se for injetora, ou seja, se não houver colisão. No contexto de Criptografia, as entradas de funções *Hash* possuem tamanho arbitrário, ou com tamanho muito maior do que o tamanho das saídas. Sendo assim, neste contexto não são usadas funções *hash* perfeitas.

1.1.2 Criptografia de chave simétrica

Uma criptografia de chave simétrica consiste em utilizar a mesma chave para encriptar e decriptar uma determinada mensagem. Mas note que no caso de comunicação entre dois indivíduos esse segredo compartilhado, a chave, é combinado anteriormente fazendo uso de sistemas criptográficos que podem utilizar funções *hash* e algoritmos de criptografia assimétrica, que veremos mais a frente. Esse problema é uma das maiores desvantagens da criptografia de chave simétrica. Em contra-partida, são muito mais rápidos e usam menos recursos computacionais para serem executados. Possuem a vantagem de possuir métodos de encriptação mais úteis, tais como, encriptar mensagens em blocos e em fluxo.

Como dito anteriormente, a chave precisa ser compartilhada, e dado que os computadores estão cada vez mais rápidos, essa chave precisa ser trocada em intervalos de tempo cada vez menores para evitar possíveis vazamentos de dados em uma comunicação, já que um atacante pode realizar um ataque de força-bruta e conseguir obter a chave. Mas já quando estamos tratando de arquivos locais (arquivos de backup, por exemplo) não precisamos trocar a chave constantemente, visto que essa chave secreta jamais deverá ser compartilhada entre os usuários do ambiente.

Ultimamente, alguns processadores vêm adicionando otimizações para o AES, que é o algoritmo simétrico mais utilizado hoje em dia dada a sua alta resistência aos ataques mais conhecidos.

1.1.3 Criptografia de chave assimétrica ou de Chave Pública

Comumente chamada de criptografia de chave pública, esse tipo de criptografia requer o uso de duas chaves distintas e matematicamente inversas. Uma sendo privada

e secreta, e outra sendo pública e que pode ser compartilhada com qualquer pessoa (até mesmo com um atacante). Em termos de uso, pode-se dizer que é a mais utilizada para assinar digitalmente documentos (juntamente com funções *hash*) e no procedimento citado anteriormente de compartilhar a chave inicial na criptografia simétrica.

No algoritmo de chave assimétrica, uma das chaves serve para encriptar e a outra para decriptar, qual será utilizada irá depender de qual objetivo deseja-se alcançar. Por exemplo, se utilizarmos a chave pública para encriptar um texto, podemos garantir que apenas o dono da chave privada poderá ler o texto, visto que para decriptar o texto será necessário da chave privada. Agora, se quisermos garantir que qualquer indivíduo possa verificar se um texto foi escrito por determinada pessoa, ou seja, garantir a autenticidade, iremos utilizar a chave privada para encriptar e a chave pública para decriptar. E já que apenas um indivíduo poderia ter escrito a mensagem, visto que apenas este possui a chave privada, conseguimos alcançar a propriedade de não-repúdio (ou incontestabilidade).

Esse tipo de criptografia não é difícil de ser quebrado se conhecemos uma trapdoor² ou uma parte da chave privada [1]. O principal desafio nos dias atuais é a quantidade de operações necessárias para resolver problemas como fatoração, encontrar o logaritmo discreto ou encontrar a ordem de um grupo de Curvas Elípticas. Para se ter uma ideia do tamanho dos valores envolvidos em tais problemas, por exemplo, uma chave de 1024 bits, que é usada pelo RSA, armazena valores até 10^{616} , algo absurdamente maior do que a quantidade de segundos passados desde a criação do universo 13,3 bilhões de anos atrás (aproximadamente de $4,2 \times 10^{17}$ segundos).

Diversos sistemas criptográficos fazem uso de chaves assimétricas, consequentemente, estratégias distintas de trocas de chaves (*key exchange*) foram desenvolvidas. Uma delas consiste em criptografar a chave escolhida por uma das partes (cliente), envolvida na troca da mensagem, utilizando a chave pública da outra parte (servidor), assim garantindo que o segredo seja mantido apenas entre as duas partes (cliente e servidor), mesmo que ocorra interceptação no momento da transmissão por parte de um atacante.

²determinada parte do algoritmo que pode ser explorada por uma função matematicamente inversa e que seja fácil de calcular

1.2 Objetivos

A ferramenta proposta neste trabalho tem como objetivo auxiliar no ensino dos algoritmos de criptografia mais avançados e mais utilizados, tais como, AES, MD5 e RSA. Iremos tentar detalhar na ferramenta o máximo possível dos passos executados, com a finalidade de tornar mais didático o processo de aprendizagem e fazer com que aluno e professor possam acompanhar no grão fino os detalhes dos algoritmos apresentados.

Queremos também que a ferramenta possa ser utilizada através de computadores e qualquer outro sistema computacional que possua acesso à internet, por isso, optamos por fazer uma versão para web. Outro objetivo da ferramenta é que o aluno possa aprender sozinho, se assim desejar.

Os objetivos secundários que buscamos atingir são fazer com que mais pessoas se interessem por criptografia e, através de uma linguagem mais simples e objetiva, facilitar a compreensão de grande parte das complicações matemática's.

1.3 Metodologia

A metodologia de avaliação utilizada foi a realização de testes, que foram feitos de forma online através da ferramenta Google Forms, antes e depois de utilizar a ferramenta proposta. Criamos um formulário de perguntas com o objetivo de avaliar o usuário de acordo com seu conhecimento antes e depois do uso. No primeiro caso, serão 5 perguntas de conhecimento geral sobre criptografia, e o segundo caso será dividido em duas partes, uma com 5 perguntas sobre a criptografia utilizada no algoritmo especificamente, e 5 perguntas sobre o algoritmo de criptografia ou função *hash*.

O objetivo do teste é ter um parâmetro de comparação para medir a diferença de conhecimento após o uso da ferramenta, assim podemos medir o nível de aprendizado adquirido pelo usuário, além de conseguir mensurar a qualidade da mesma.

O questionário preenchido anteriormente ao uso da ferramenta possui perguntas com propriedades mais gerais, fazendo com que o usuário classifique seu conhecimento através de 4 respostas: Nenhum conhecimento, conhecimento baixo, médio e alto, com relação a diversos algoritmos criptográficos, além de testar seu conhecimento para identificar algoritmos de chave simétrica, assimétrica e funções de dispersão.

O formulário de perguntas posterior ao uso do aplicativo possui um total de 5

questões específicas para o algoritmo apresentado e outras 5 questões mais gerais sobre o tipo de criptografia utilizado, avaliando assim o conhecimento adquirido na ferramenta. O objetivo das perguntas específicas é testar a parte animada da ferramenta, onde mostramos passo-a-passo do algoritmo, e as outras 5 perguntas mais gerais são referentes aos textos de funcionamento dos algoritmos a fim de testar o conhecimento sobre o tipo de criptografia utilizado.

Utilizamos dois tipos de usuários bases, com experiência e sem experiência, tendo como maior foco usuários sem nenhum conhecimento no assunto, pois foram esses os principais motivadores da criação da ferramenta. A utilização de usuários, com experiência, que possam questionar erros didáticos e conceituais é importante para correção dos mesmos, logo mesmo sendo minoria, possuem papel fundamental na ferramenta. Os usuários que nunca tiveram contato com o assunto, podem avaliar, principalmente, a parte didática, mostrando como melhorar a transmissão desse conhecimento.

Ao final, comparamos esses resultados para mensurar a qualidade do aplicativo, além de utilizar as opiniões e sugestões dos usuários para melhorar cada vez mais a ferramenta.

Capítulo 2

Algoritmos criptográficos e Funções Hash

Abaixo iremos detalhar o funcionamento dos algoritmos criptográficos AES (simétrico), RSA (assimétrico), além do MD5 (função *hash*). Iremos contar um pouco da história, como o algoritmo funciona, quais tabelas são necessárias para os cálculos e brevemente como realizar estas contas. Não iremos entrar no detalhe de como essas tabelas são geradas, nem como as funções utilizadas foram criadas. Iremos também supor que o leitor tenha conhecimento de álgebra booleana, ou seja, que saiba fazer operações lógicas de $XOR(\oplus)$, $OR(\vee)$, $AND(\wedge)$ e $NOT(\neg)$.

2.1 MD5

Falando um pouco da história do MD5, sucessor do MD4, temos que esse foi criado em 1991 por Ronald Rivest¹ na empresa RSA Data Security Inc. Para efeitos de curiosidade, em 2008 uma variação do MD5 foi proposta em um concurso, no qual seria escolhido o novo SHA-3 (*Secure Hash Algorithm 3*), porém o MD6 (*Message Digest 6*) [8] foi considerado muito lento para os computadores atuais. Não iremos nos aprofundar neste assunto pois foge do foco desta seção.

Voltando ao assunto principal desta seção, o funcionamento do MD5 pode ser descrito em quatro passos bem definidos que serão descritos detalhadamente a seguir.

¹Ronald Rivest: laureado com o Prêmio Turing de 2002, juntamente com Adi Shamir e Leonard Adleman, pelo algoritmo RSA (Rivest, Shamir e Adleman).

2.1.1 Passo 1

Primeiramente, o texto a ser usado é alterado para que tenha tamanho igual a 512 bits, utilizando dois procedimentos descritos neste passo e no passo 2. Essa mudança é necessária para realização do passo 4, que será visto futuramente.

Haverá uma verificação para que a entrada seja congruente a 448 módulo 512, caso a mesma seja maior do que o esperado, será dividida em blocos de 448 bits, nesta situação esse passo não será realizado. E ao fim do passo 2, cada bloco terá 512 bits.

Como primeiro procedimento adicionaremos o bit “1” no final da mensagem. Logo em seguida, como segundo procedimento, a mensagem terá a adição de múltiplos bits “0”, até que seu tamanho chegue a 448 bits, assim após esses dois procedimentos o resultado será um mensagem congruente a 448 módulo 512. Note que mesmo que se inicialmente a mensagem seja congruente à 448 módulo 512 o bit “1” será adicionado.

2.1.2 Passo 2

Como passo 2, o tamanho da entrada, em uma representação de 64 bits, é concatenado no final da mensagem produzida pelo passo anterior, assim completando o tamanho de 512 bits. A saída deste passo será usada no passo 4.

2.1.3 Passo 3

O terceiro passo é independente dos passos anteriores. Ele consiste em inicializar quatro variáveis de 32 bits cada A , B , C e D , que serão utilizadas no passo subsequente, formando um bloco de 128 bits. Os seguintes valores serão alocados, utilizando notação hexadecimal:

$$A = 0x67452301$$

$$B = 0xEFCDAB89$$

$$C = 0x98BADCFE$$

$$D = 0x10325476$$

2.1.4 Passo 4

Neste passo 4, como ilustrado na Figura 2.1, são usados 3 elementos: os valores A , B , C e D obtidos no passo 3, a saída do passo 2 e uma função K que é explicada adiante.

A partir dos valores A , B , C e D , obtemos as seguintes funções:

$$F_1 = (B \wedge C) \vee ((\neg B) \wedge D)$$

$$F_2 = (D \wedge B) \vee ((\neg D) \wedge C)$$

$$F_3 = B \oplus C \oplus D$$

$$F_4 = C \oplus (B \vee (\neg D))$$

A função K é gerada à partir da equação $K(i) = 2^{32} * |\text{sen}(i + 1)|$, com i variando de 0 a 63.

A mensagem resultante do passo 2 é dividida em 16 blocos de 32 bits, totalizando os seus 512 bits, e alocada na variável M . Cada rodada desse passo utiliza os 16 blocos dessa divisão, uma em cada operação, sendo 4 rodadas.

Cada rodada será controlada pela variável i , variando a mesma de 0 até 15 na primeira rodada, 16 a 31 na segunda, e assim sucessivamente, ou seja, será realizado a mesma sequência de ações, porém as funções utilizadas serão diferentes em cada rodada. Além de i também usaremos a variável g que utilizará os seguintes valores para cada rodada, respectivamente:

$$g = i$$

$$g = (5 * i + 1) \mod 16$$

$$g = (3 * i + 5) \mod 16$$

$$g = (7 * i) \mod 16$$

Na primeira rodada a função F_1 será executada em cima do valor inicial de A , gerando um A' que será somado a M_g e aplicado em K_i . Após feito isso será feita uma operação conhecida como *left-rotate*, rotacionando o resultado de s espaços à esquerda. Este resultado obtido será somado com B e atribuído ao próprio. Note que todos os passos descritos alteram apenas a variável A , mas pelas questões das trocas realizadas, todos as variáveis serão modificadas no final da rodada. Ao final desse passo, o valor resultante será um hash de 128 bits, que será a agregação dos valores finais de A , B , C e D , sendo cada um de 32 bits.

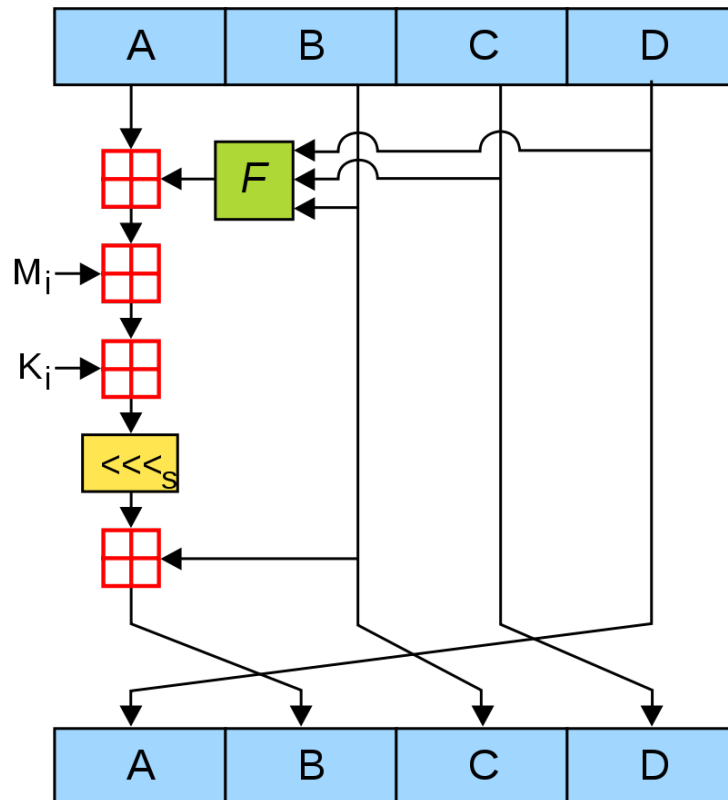


Figura 2.1: Ilustração do passo 4 do MD5

2.2 AES

O algoritmo AES é um caso particular do algoritmo Rijndael, criado por dois criptologistas búlgaros de nomes Joan Daemen e Vincent Rijmen[2] e anunciado pelo NIST em 26 de novembro de 2001 como novo padrão de criptografia simétrica. A sua particularidade em relação ao Rijndael é que o tamanho de blocos é fixo com tamanho igual a 128 bits, enquanto que no algoritmo original possui tamanho variável. Sendo o sucessor do DES (*Data Encryption Standard*), como padrão de fato e de direito, vem cada vez mais aumentando sua popularidade no contexto de criptografia simétrica. Além disso, diversas otimizações a nível de *hardware* são feitas para que o AES fique ainda mais rápido. Podemos citar, por exemplo, o AES-NI (*AES New Instructions*) [3] que é um novo conjunto de instruções que reduzem o tempo de execução de 3 a 10 vezes em comparação a uma implementação de *software*. E mesmo desconsiderando essas otimizações, o algoritmo ainda permanece eficiente, pois as instruções utilizadas são permutação e substituição de bytes em cada bloco. Estas operações são facilmente implementadas nos diversos sistemas computacionais, tanto em *hardware* quanto em *software*. Essas permutações e

substituições são feitas através de matrizes denominadas SBOX (ou *Substitution Box*) e RCON (ou *Round Constants*), e que serão mostradas brevemente abaixo. Não iremos nos aprofundar em como essas matrizes são geradas pois para isso iríamos precisar nos aprofundar em um Corpo Finito de Galois.

O algoritmo utiliza funções como o *AddRoundKey*, *SubBytes*, *ShiftRows* e *MixColumns* que são comentadas mais a frente neste documento. Para decifração também são usadas as funções *ShiftRowsInv* e *MixColumnsInv*, que são as respectivas inversas de *ShiftRows* e *MixColumns*.

2.2.1 SBOX

A SBOX do AES realiza duas operações consecutivas nos dados: 1) substitui o valor pelo seu inverso multiplicativo no Corpo Finito de Galois $GF(2^8)$ e 2) aplica uma transformação Afim no resultado [2]. Isto é equivalente a substituir cada byte pelo resultado da Tabela 2.1, onde o dígito hexadecimal mais significativo indica a linha e o menos significativo indica a coluna.

Abaixo podemos ver os dois tipos de tabela SBOX, a SBOX “direta” que pode ser visualizada na Tabela 2.1 e que é utilizada na encriptação, e a SBOX inversa (Tabela 2.2) utilizada na decifração. Note que ambas são consideradas *lookup table*.

2.2.2 RCON

A tabela RCON é utilizada na etapa de expansão de chaves e assim como a SBOX ela também pode ser calculada usando o Corpo Finito de Galois $GF(2^8)$. Abaixo podemos observar o resultado das operações em hexadecimal, formando assim a tabela RCON (Tabela 2.3). A RCON guarda todas as constantes de rodada, ou seja, nem todas as posições são utilizadas, pois depende da quantidade de rodadas que são executados.

2.2.3 Expansão de chaves

Na fase de expansão de chaves, utilizamos a chave da entrada que possui 128, 192 ou 256 bits e expandimos para um conjunto de chaves da rodada (ou chave expandida) que contém, respectivamente, para 176, 208 ou 240 bits. A *expanded key* pode ser vista

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
10	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
20	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
30	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
40	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
50	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
60	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
70	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
80	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
90	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A0	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B0	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C0	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D0	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E0	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F0	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Tabela 2.1: Tabela SBOX

com um array de *word*². Para calcular o valor da *expanded key* fazemos os seguintes procedimentos:

Passo 1. As 4 primeiras colunas da *expanded key* serão compostas pela própria *cipher key*.

Passo 2. Enquanto a quantidade de bytes da *expanded key* for menor do que a quantidade necessária de $(rodadas + 1) \times 16$, iremos executar um total de 11 ciclos para expandir a chave. Onde a quantidade de rodadas necessária para uma chave de 128, 192 e 256 são, respectivamente, 10, 12 e 14 (ou simplesmente $rodada = key_size_bits/32 + 6$).

Passo 2.1. No primeiro procedimento do ciclo iremos aplicar um *left-rotate* de 1 byte na *word*.

Passo 2.2. Então iremos aplicar o procedimento de *SubBytes* para cada byte da *word* gerada no passo 2.2.

Passo 2.3. Daí faremos um XOR do byte mais a esquerda (o primeiro byte da

²uma parte da *expanded key* com tamanho de 4 bytes

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
10	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
20	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
30	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
40	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
50	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
60	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
70	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
80	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
90	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A0	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B0	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C0	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D0	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E0	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F0	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Tabela 2.2: Tabela SBOX-Inv

0x8d	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80	0x1b	0x36	0x6c	0xd8	0xab	0x4d	0x9a
0x2f	0x5e	0xbc	0x63	0xc6	0x97	0x35	0x6a	0xd4	0xb3	0x7d	0xfa	0xef	0xc5	0x91	0x39
0x72	0xe4	0xd3	0xbd	0x61	0xc2	0x9f	0x25	0x4a	0x94	0x33	0x66	0xcc	0x83	0x1d	0x3a
0x74	0xe8	0xcb	0x8d	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80	0x1b	0x36	0x6c	0xd8
0xab	0x4d	0x9a	0x2f	0x5e	0xbc	0x63	0xc6	0x97	0x35	0x6a	0xd4	0xb3	0x7d	0xfa	0xef
0xc5	0x91	0x39	0x72	0xe4	0xd3	0xbd	0x61	0xc2	0x9f	0x25	0x4a	0x94	0x33	0x66	0xcc
0x83	0x1d	0x3a	0x74	0xe8	0xcb	0x8d	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80	0x1b
0x36	0x6c	0xd8	0xab	0x4d	0x9a	0x2f	0x5e	0xbc	0x63	0xc6	0x97	0x35	0x6a	0xd4	0xb3
0x7d	0xfa	0xef	0xc5	0x91	0x39	0x72	0xe4	0xd3	0xbd	0x61	0xc2	0x9f	0x25	0x4a	0x94
0x33	0x66	0xcc	0x83	0x1d	0x3a	0x74	0xe8	0xcb	0x8d	0x01	0x02	0x04	0x08	0x10	0x20
0x40	0x80	0x1b	0x36	0x6c	0xd8	0xab	0x4d	0x9a	0x2f	0x5e	0xbc	0x63	0xc6	0x97	0x35
0x6a	0xd4	0xb3	0x7d	0xfa	0xef	0xc5	0x91	0x39	0x72	0xe4	0xd3	0xbd	0x61	0xc2	0x9f
0x25	0x4a	0x94	0x33	0x66	0xcc	0x83	0x1d	0x3a	0x74	0xe8	0xcb	0x8d	0x01	0x02	0x04
0x08	0x10	0x20	0x40	0x80	0x1b	0x36	0x6c	0xd8	0xab	0x4d	0x9a	0x2f	0x5e	0xbc	0x63
0xc6	0x97	0x35	0x6a	0xd4	0xb3	0x7d	0xfa	0xef	0xc5	0x91	0x39	0x72	0xe4	0xd3	0xbd
0x61	0xc2	0x9f	0x25	0x4a	0x94	0x33	0x66	0xcc	0x83	0x1d	0x3a	0x74	0xe8	0xcb	0x8d

Tabela 2.3: Tabela RCON. Em negrito as posições que são, de fato, utilizadas.

word) com a posição i (identificador do ciclo) da tabela RCON.

Passo 2.4. Iremos repetir este passo 4 vezes a fim de misturar os 4 bytes da *word* fazendo um XOR com os n últimos bytes da *expanded key*, ou de forma mais clara, com os bytes da última *subkey* gerada. Iremos concatenar o resultado deste passo à *expanded*

key.

Passo 2.5 Neste ponto iremos checar se a chave expandida possui 176, 208 ou 240 bits, dependendo da quantidade de bits da chave (128, 192 ou 256 bits, respectivamente). Se a quantidade já foi atingida, iremos para o passo 3. Senão, iremos para o passo 2.6.

Passo 2.6 Este passo é especial e apenas precisa ser executado caso a chave possua 256 bits. Iremos aplicar a S-Box na *word* e depois aplicar um XOR do resultado com os bytes da última *subkey* gerada. Iremos concatenar o resultado deste passo à *expanded key*.

Passo 2.7 Caso a chave possua 192 ou 256 bits, este passo também deverá ser executado. Iremos realizar um procedimento parecido com o passo 2.4, porém aqui iremos repetir o procedimento 2 vezes no caso da chave possuir 192 bits e 3 vezes no caso da chave possuir 256 bits. Iremos concatenar o resultado deste passo à *expanded key*.

Passo 3. No último ciclo iremos expandir sem utilizar os casos especiais 2.6 e 2.7 (no caso das chaves de 192 e 256 bits).

Passo 4. Ao final deste processo iremos ter, finalmente, a *expanded key* com 176, 208 ou 240 bits.

2.2.4 AddRoundKey

Realiza um XOR entre cada coluna da *key* com cada coluna da *expanded key* de mesma posição, gerando assim, uma nova *key* ao final do processo. Por essa função não possuir inversa, no processo de encriptação a primeira rodada a ser utilizado é o de número 0, já no processo de decríptação as rodadas serão passadas na ordem inversa (na ordem decrescente).

2.2.5 SubBytes

Neste passo, iremos realizar a substituição dos bytes através da tabela S-Box (Tabela 2.1) na encriptação, ou a S-BoxInv (Tabela 2.2), no caso da decríptação. Esse procedimento tem como objetivo mitigar ataques algébricos simples.

2.2.6 ShiftRows

Aplicaremos um deslocamento de n bytes, sendo n igual ao número da linha menos um, assim a primeira linha se manterá igual, rotacionamos (deslocamento circular) à esquerda a segunda linha de um byte, a terceira de dois bytes e a quarta linha de três bytes. Na decifração rotacionamos a mesma quantidade de bytes, porém à direita.

2.2.7 MixColumns

Haverá uma multiplicação entre cada coluna da *key* recebida pela Tabela de Corpo Finito de Galois (Tabela 2.4), cada coluna é processada separadamente, gerando ao final uma nova *key*.

0x02	0x03	0x01	0x01
0x01	0x02	0x03	0x01
0x01	0x01	0x02	0x03
0x03	0x01	0x01	0x02

Tabela 2.4: Tabela de corpo finito de Galois

0x02	0x03	0x01	0x01
0x01	0x02	0x03	0x01
0x01	0x01	0x02	0x03
0x03	0x01	0x01	0x02

Tabela 2.5: Tabela Inversa de corpo finito de Galois

2.2.8 Encriptação

A encriptação do algoritmo do AES começa com a expansão de chaves citada na Seção 2.2.3 e logo após os dados são divididos em blocos que possuem 128 bits cada, onde cada um passará pelos seguintes passos:

Passo 1. Na rodada inicial iremos aplicar a etapa *AddRoundKey* com a rodada valendo zero.

Passo 2. Nas rodadas intermediárias iremos aplicar, na seguinte ordem, os passos *SubBytes* utilizando a *S-Box*, *ShiftRows*, *MixColumns* e executar o *AddRoundKey*

utilizando o valor da rodada.

Passo 3. Na rodada final iremos utilizar a mesma ordem de etapas citados no passo 2, mas não iremos executar o passo de *MixColumns*.

2.2.9 Decriptação

Na decriptação também iremos dividir os dados de entrada em blocos de 128 bits e realizar a expansão de chaves. A diferença é que utilizaremos as funções inversas do *ShiftRows* e *MixColumns*, e a função de *SubBytes* utilizará a *S-BoxInv* assim como a etapa de *AddRoundKey* começará na última posição.

Passo 1. Iniciaremos a rodada inicial com o *AddRoundKey* e passando como entrada o valor do última rodada.

Passo 2. Nas rodadas intermediárias faremos a seguinte sequência de etapas: *ShiftRowsInv*, *SubBytes* usando a *S-BoxInv*, *AddRoundKey* e *MixColumnsInv*.

Passo 3. Na rodada final, assim como na encriptação, também não iremos usar o *MixColumns*. Iremos apenas executar o *ShiftRowsInv*, *SubBytes* com a *S-BoxInv* e *AddRoundKey* com o valor da rodada igual a zero.

2.3 RSA

O nome RSA deriva dos nomes de seus criadores, Ronald Rivest, Adi Shamir e Leonard Adleman [9], sendo que Rivest e Shamir criavam os algoritmos criptográficos e Adleman os testava. Para efeito de curiosidade, esses testes duraram cerca de um ano até que em Abril de 1977 Rivest conseguiu criar a função de mão-única (para a qual não se conhece modo eficiente de invertê-la) tão procurada por eles.

O RSA, como a grande maioria dos algoritmos criptográficos, se divide em 3 etapas básicas: geração das chaves, encriptação e decriptação.

Essas etapas podem ser entendidas da seguinte forma: Na fase de geração das chaves, iremos gerar as chaves públicas e privadas, e apenas ressaltando que a última nunca deve ser compartilhada mas a primeira, por ser pública, pode ser compartilhada livremente. Na etapa de encriptação iremos criptografar uma mensagem qualquer e na etapa de decriptação iremos descriptografar a mensagem criptografada voltando ao texto original.

2.3.1 Geração das chaves

A geração das chaves pode ser descrita da seguinte forma:

1. Escolha dois números primos P e Q muito grandes (na ordem de 2^{1024} ou maior).
2. Calcule $N = P \times Q$ e guarde esse valor, pois este será utilizado na chave pública e na chave privada.
3. Calcule ϕ , a função totiente de N , $\phi(N) = (P - 1) \times (Q - 1)$
4. Escolha um número E tal que $1 < E < \phi$, e E e ϕ sejam co-primos ³.
5. Calcule D que será a inversa de E usando o algoritmo de Euclides Estendido [5].

A saída esperada do algoritmo será a chave pública formada pela chave de encriptação E e o valor de N , obtido através de P e Q , e a chave privada formada pela chave de deciptação D (a inversa de E) e N .

2.3.2 Encriptação

Para encriptar uma mensagem M , teremos que realizar o seguinte cálculo:

$$M_{ENC} = M^E \mod N$$

Note que realizar esse cálculo diretamente é bastante complexo devido a limitação de memória e quantidade de bits utilizados para representação dos números. E por isso, devemos usar a exponenciação modular. O algoritmo de maneira simplificada, e não eficiente, pode ser entendido da seguinte forma:

1. Criamos uma variável *result* e atribuímos a ela o valor de M .
2. Enquanto o expoente E for maior do que 1 repita os passos seguintes:
3. Multiplicamos *result* com M , calculamos o resto da divisão deste com N e atualizamos o valor da variável *result* com este resultado.
4. Decrementamos o expoente E em 1 unidade.

Abaixo um pseudocódigo exemplificando o algoritmo acima.

³Ou seja, que E e ϕ sejam primos entre si. Podemos facilmente encontrar esse valor se o máximo divisor comum entre E e ϕ for igual a 1.

```

1:  $result \leftarrow M$ 
2: while  $E > 1$  do
3:    $result = (result * M) \bmod N$ 
4:    $E = E - 1$ 
5: end while

```

2.3.3 Decifração

A decifração acontece de maneira análoga à encriptação, porém aqui iremos usar o expoente D e M será a mensagem criptografada no passo anterior (M_{ENC}).

$$M_{DEC} = M^D \bmod N$$

Note que esse cálculo também é complexo, por isso devemos utilizar novamente a exponenciação modular descrita acima.

Capítulo 3

Ferramenta

A ferramenta consiste em uma página web (Figura 3.1) para que o acesso seja facilitado, além de simplificar a divulgação. Por este motivo, a ferramenta não precisa ser instalada na máquina do usuário e pode ser utilizada em diversas plataformas computacionais. O único fator limitante é possuir um browser baseado no Chromium (Google Chrome da Google ou o Chromium de fato) e acesso à internet.

Cada algoritmo da ferramenta possui duas seções: animação de execução e funcionamento, de acordo com a Figura 3.2, Figura 3.6 e Figura 3.10. Na primeira seção, temos um conjunto de slides que explica o passo-a-passo detalhadamente e visa representar a maior quantidade de informações sobre o algoritmo. Na parte do funcionamento fazemos um detalhamento teórico do algoritmo, explicando a fundo alguns termos envolvidos na execução do mesmo. Algumas partes do algoritmo não são explicadas pois fogem do escopo de ensino e entram no escopo da lógica booleana, por exemplo. Logo, detalhes como a tabela verdade de AND, OR e XOR, funcionamento do algoritmo de Euclides Estendido e o corpo Finito de Galois não são explicados na ferramenta.

A seção animação de execução tem o papel mais importante, pois ela ajudaria os professores a criarem slides para suas aulas, assim facilitando o ensino do algoritmo. Será necessário que o usuário escolha uma mensagem de entrada como na Figura 3.3, Figura 3.7 e Figura 3.11 para que o todos os passos e slides da criptografia sejam gerados. Utilizando o estilo carrossel, aonde cada slide pode ser passado ou retrocedido quantas vezes forem necessárias, além de conter ao lado do carrossel um descrição do algoritmo em português, e ao clicar em alguma linha do texto será focado o slide referente aquele procedimento, como mostrado na Figura 3.4, Figura 3.8, Figura 3.12 e Figura 3.13, algumas informações serão

mostradas ao passar o mouse em cima da imagem representada Figura 3.5. Cada repetição (*loop*) do algoritmo será representado pelo movimento vertical do carrossel, tendo todos os passos e operações sendo mostradas. Além de conter uma figura de uma interrogação (Figura 3.9), contendo a descrição explicativa daquele slide.

3.1 Implementação

A linguagem escolhida foi Python devido a sua grande facilidade de uso e desempenho em servidores web. A ferramenta utiliza o framework Flask que provê um conjunto de facilidades adicionais tais como gerenciamento de rota e tráfego. Utilizamos para as apresentações o Reveal.Js que solucionou o problema de um algoritmo ter vários slides e o usuário precisar passar individualmente cada slide, agora é possível avançar uma etapa completa sem precisar ver todos os passos da mesma.

A instalação da ferramenta, do Python e do Flask pode ser visualizada no Apêndice A.

3.2 Imagens

Ao entrar no site, como ilustrado na Figura 3.1, o usuário se depara com um menu lateral, usado para acessar cada algoritmo, e um questionário usado para a fase de testes da ferramenta, ao clicar em um algoritmo será dada a opção da visualização da animação de execução ou do funcionamento. Ao clicar em animação de execução uma nova tela será apresentada (Figura 3.3, Figura 3.11 e Figura 3.7) para escolha da mensagem a ser criptografada, assim gerando uma apresentação de slides referente exclusivamente aquela mensagem escolhida (Figura 3.4, Figura 3.8, Figura 3.12 e Figura 3.13).

Ao passar o mouse na imagem de um ponto de interrogação (Figura 3.9) ao lado de cada slide dará mais informações sobre aquele passo ao usuário, além de ser possível ter informações do valor de determinada variáveis (Figura 3.5) com o mesmo procedimento.

A outra opção mostrada no menu será a tela de funcionamento (Figura 3.2, Figura 3.6 e Figura 3.10), onde o usuário poderá ler, de forma resumida, a descrição do algoritmo.

Por favor, responda esse questionário apenas 1 vez antes de utilizar o menu ao lado.

ALL
MD5
AES
RSA

Identificação

Pedimos que as informações abaixo sejam preenchidas para que possamos analisar melhor o desempenho obtido.

***Obrigatório**

Nome ou apelido *

Sua resposta

Idade

Escolher

Figura 3.1: Página inicial da ferramenta.

Na Figura 3.2 temos a descrição teórica do algoritmo MD5, que detalha os passos vistos na Seção 2.1 e auxilia o usuário na compreensão do algoritmo com informações além do que as que são exibidas nos slides.

Voltando ao assunto principal desta seção, o funcionamento do MD5 pode ser descrito em quatro passos bem definidos que serão descritos detalhadamente a seguir.

Passo 1:

Primeiramente, o texto a ser usado é alterado para que tenha tamanho igual a 512 bits, utilizando dois procedimentos descritos neste passo e no passo 2. Essa mudança é necessária para realização do passo 4, que será visto futuramente.

Haverá uma verificação para que a entrada seja congruente a 448 módulo 512, caso a mesma seja maior do que o esperado, será dividida em blocos de 448 bits, nesta situação esse passo não será realizado. E ao fim do passo 2, cada bloco terá 512 bits.

Como primeiro procedimento adicionaremos o bit "1" no final da mensagem. Logo em seguida, como segundo procedimento, a mensagem terá a adição de múltiplos bits "0", até que seu tamanho chegue a 448 bits, assim após esses dois procedimentos o resultado será um mensagem congruente a 448 módulo 512. Note que mesmo que se inicialmente a mensagem seja congruente à 448 módulo 512 o bit "1" será adicionado.

Passo 2:

Ao fim do primeiro passo, o tamanho original da entrada (em uma representação de 64 bits) será acrescentado à mensagem produzida pelo passo anterior, assim completando o tamanho de 512 bits, logo, ao final haverá uma saída utilizável no passo 4.

Passo 3:

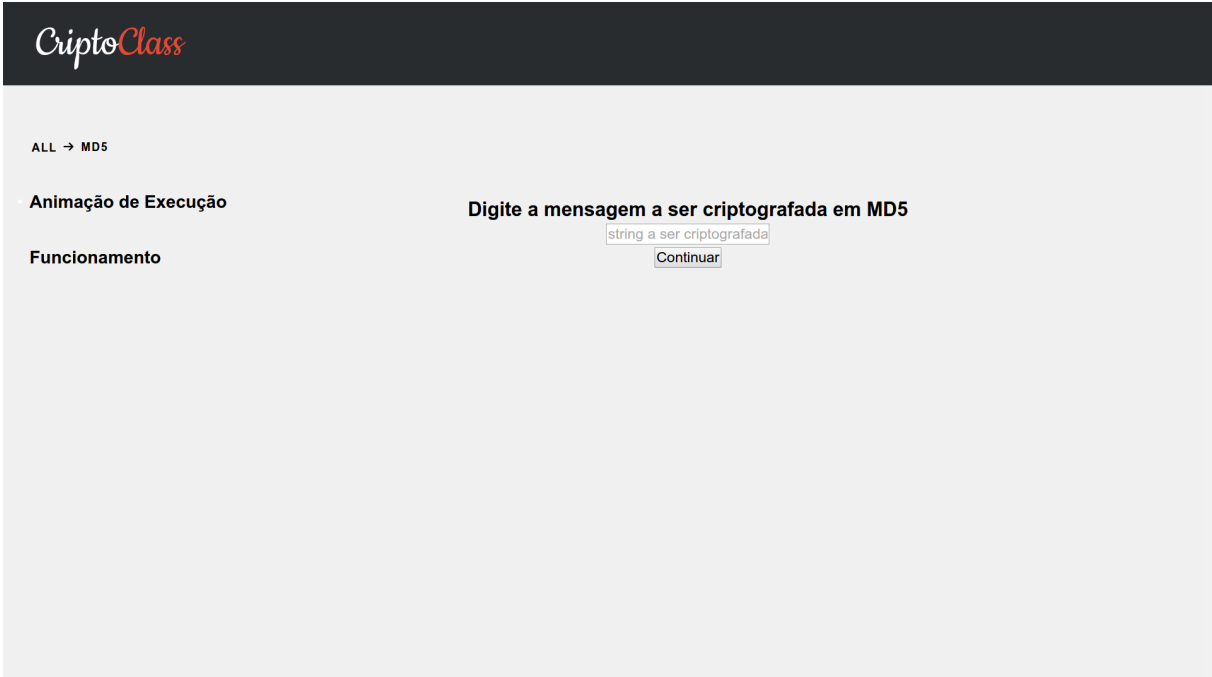
O quarto passo consiste em inicializar as variáveis A, B, C e D, utilizadas no passo subsequente, variáveis com 32 bits cada, formando um bloco de 128 bits. Os seguintes valores serão alocados, utilizando notação hexadecimal:

A = 0x67452301
B = 0x52795860

Figura 3.2: Detalhes teóricos sobre o algoritmo MD5.

Na Figura 3.3 temos o campo para entrada a mensagem a ser criptografada em

MD5, procedimento necessário para que seja gerado os slides de apresentação.



CriptoClass

ALL → MD5

Animação de Execução

Funcionamento

Digite a mensagem a ser criptografada em MD5

string a ser criptografada

Continuar

Figura 3.3: Formulário responsavel pelas entradas do algoritmo MD5.

Na Figura 3.4 e Figura 3.5 vemos o slide referente ao passo 4. Esse passo consiste de uma sequência de imagens verticais, onde cada imagem representa uma pequena etapa do passo. Podemos visualizar na Figura 3.5 que ao colocarmos o mouse sobre as variáveis podemos obter mais informações, no caso, a função que está sendo utilizada.

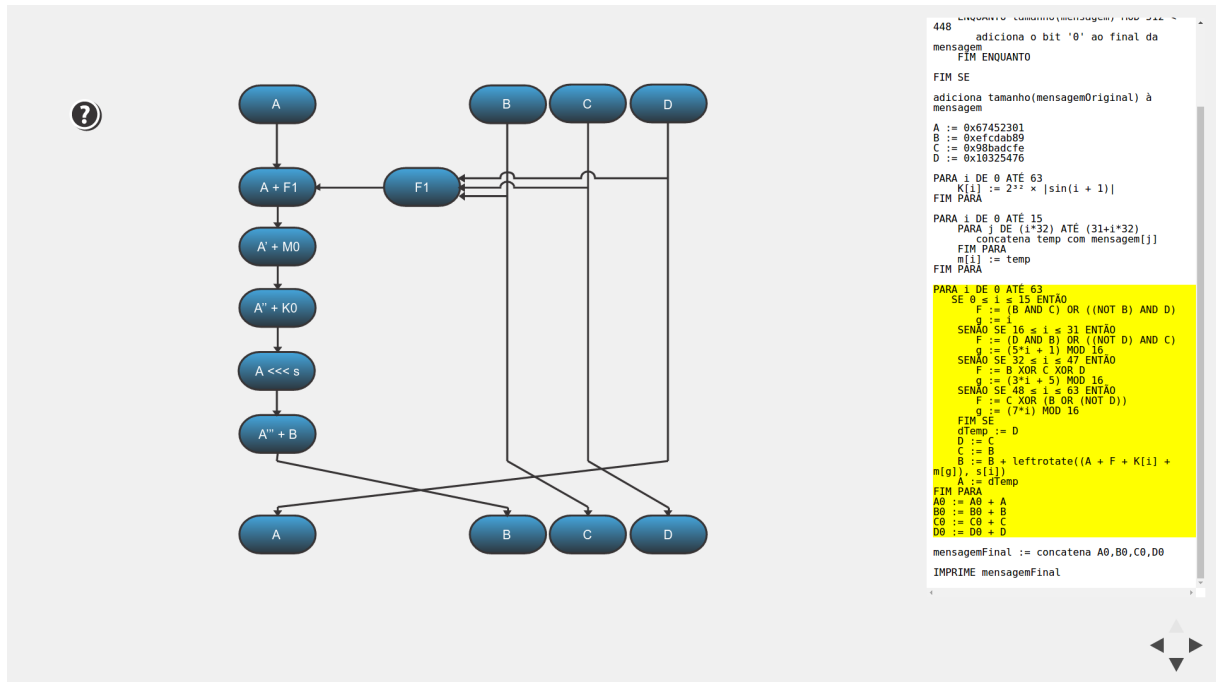


Figura 3.4: Demonstração do funcionamento no passo 4.

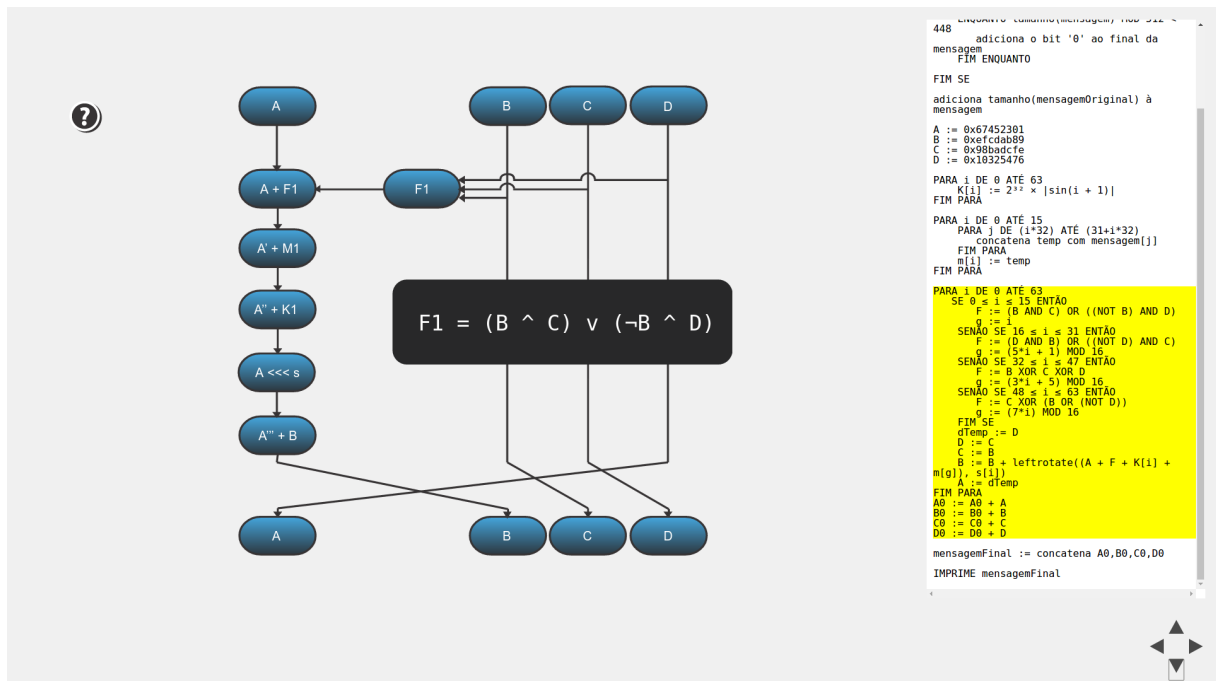


Figura 3.5: Detalhes das funções utilizadas em cada rodada.

Na Figura 3.6 temos a descrição teórica do algoritmo RSA, que detalha os passos vistos na Seção 2.3 e auxilia o usuário na compreensão do algoritmo com informações além das exibidas nos slides.



Figura 3.6: Explicação teórica do funcionamento do RSA.

Na Figura 3.7 temos os campos da mensagem de entrada e quantidade de bits que serão utilizados (e que varia de 16 a 30) no algoritmo RSA, procedimento necessário para que seja gerado os slides de apresentação.

Figura 3.7: Formulário de entrada para os parâmetros do RSA.

Na Figura 3.8 temos um slide referente ao RSA, estando o mouse fora do símbolo

de interrogação na esquerda, ao ser colocado em cima será exibido informações sobre o slide. Como representado na Figura 3.9.



Figura 3.8: Etapa inicial de converter o texto de entrada em caracteres ASCII.

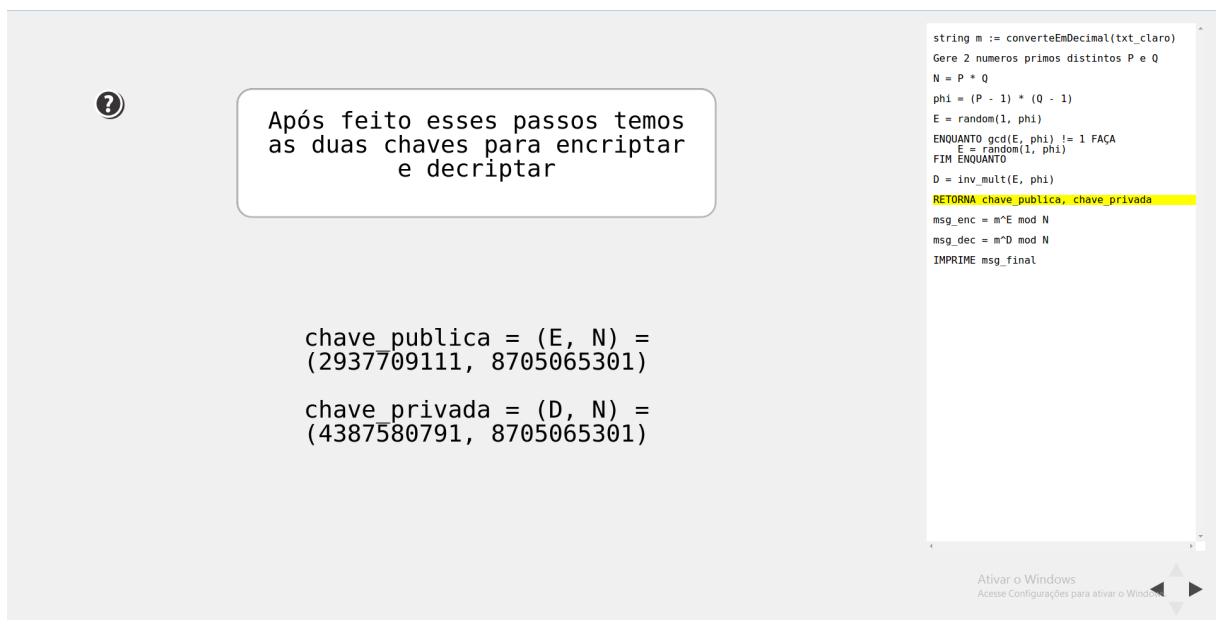


Figura 3.9: Exposição das chaves pública e privada geradas.

Na Figura 3.10 temos a descrição teórica do algoritmo AES, que detalha os passos vistos na Seção 2.2 e auxilia o usuário na compreensão do algoritmo com informações além do que as que são exibidas nos slides.

slides.

74 65 0B 0B
65 0B 0B 0B
73 0B 0B 0B
74 0B 0B 0B

↓

74 2B 01
65 2B 00
73 2B 00
74 2B 00

RCON

5E
4E
58
5F

```

converteMensagemChaveParaHexadecimal(Mensagem)
PARA i = 1 ATE 11 FAÇA:
  PARA j = 0 ATE 4 FAÇA:
    word[j] = SBOX[word[j]] ^ exp_key
  FIM PARA
  word[0] = word[0] ^ RCON[i]
  exp_key.extend(word)
  PARA j = 0 ATE 4 FAÇA:
    word = word ^ exp_key
  FIM PARA
  exp_key.extend(word)
FIM PARA

expanded_key
initialRound(bloco, 0)
PARA i = 1 ATE 9 FAÇA:
  SubBytes(bloco)
  ShiftRow(bloco)
  MixColumn(bloco)
  AddRoundKey(bloco, expanded_key)
finalRound(bloco, 10)
Imprime a mensagem criptografada
expanded_key
initialRound(bloco, 0)
PARA i = 1 ATE 9 FAÇA:
  InvShiftRow(bloco)
  InvSubBytes(bloco)
  AddRoundKey(bloco, expanded_key)
  InvMixColumn(bloco)
finalRound(bloco, 10)
Imprime a mensagem descriptografada
  
```

Figura 3.12: Utilização da tabela RCON na etapa de expansão de chaves.

3D 58 53 58
2D 26 2D 26
3B 30 3B 30
3C 37 3C 37

→

27 6A ED 6A
D8 F7 D8 F7
E2 04 E2 04
EB 9A EB 9A

SubBytes

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	63	7C	77	7B	F2	6B	6F	C5	30	01	07	2B	FE	D7	AB	76
10	CA	82	C9	7D	FA	59	47	F0	AD	D4	A1	AF	9C	A4	72	C0
20	87	FD	93	26	38	FF	F7	CC	34	A3	83	F1	71	D8	31	15
30	94	C7	23	C3	18	96	05	8A	07	12	90	52	8B	09	02	75
40	00	61	0C	1A	1B	0E	5A	A0	52	8B	D6	9B	29	E3	2F	A1
50	53	D1	00	ED	20	FC	B1	58	6A	CB	BE	39	4A	AC	56	C9
60	D0	EF	AA	F8	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
70	11	A3	89	0F	92	08	36	F3	BC	86	D4	21	19	FF	F8	D2
80	C0	AC	13	E2	0F	07	41	17	C1	A7	78	3D	46	40	10	7E
90	60	81	0F	DC	22	2A	90	88	4E	8B	1A	1E	5E	0B	D9	DD
A0	E0	82	1A	0A	49	06	24	3C	C2	D3	AC	62	91	95	E4	79
B0	87	C8	37	40	4D	05	4E	A9	6C	56	F4	EA	65	7A	AE	08
C0	9A	79	25	2E	AC	A6	D8	5B	E0	71	1F	4B	BD	43	6A	54
D0	70	3E	D5	68	48	03	F9	0E	61	35	57	B9	6C	C1	DD	0E
E0	E1	F8	98	11	69	D9	8E	94	9B	1E	87	CE	55	28	DF	6D
F0	8C	A1	89	0D	DF	E3	42	68	41	99	2D	0F	BD	54	BB	16

```

converteMensagemChaveParaHexadecimal(Mensagem)
PARA i = 1 ATE 11 FAÇA:
  PARA j = 0 ATE 4 FAÇA:
    word[j] = SBOX[word[j]] ^ exp_key
  FIM PARA
  word[0] = word[0] ^ RCON[i]
  exp_key.extend(word)
  PARA j = 0 ATE 4 FAÇA:
    word = word ^ exp_key
  FIM PARA
  exp_key.extend(word)
FIM PARA

expanded_key
initialRound(bloco, 0)
PARA i = 1 ATE 9 FAÇA:
  SubBytes(bloco)
  ShiftRow(bloco)
  MixColumn(bloco)
  AddRoundKey(bloco, expanded_key)
finalRound(bloco, 10)
Imprime a mensagem criptografada
expanded_key
initialRound(bloco, 0)
PARA i = 1 ATE 9 FAÇA:
  InvShiftRow(bloco)
  InvSubBytes(bloco)
  AddRoundKey(bloco, expanded_key)
  InvMixColumn(bloco)
finalRound(bloco, 10)
Imprime a mensagem descriptografada
  
```

Figura 3.13: Demonstração do procedimento *SubBytes* nas rodadas intermediárias.

Capítulo 4

Testes

Os testes foram realizados através do Google Forms, ferramenta do Google que provê um formulário e algumas análises básicas sobre as respostas obtidas. Neste capítulo iremos detalhar o resultado dos testes obtidos, mesmo que breves. Os testes foram realizados totalizando 6 respostas para os algoritmos de chave assimétrica, 8 para os algoritmos de chave simétrica, 11 para a função de dispersão e usando 13 usuários.

4.1 Questionário Pré Utilização

O questionário pré-utilização foi feito para termos uma ideia geral de como era o perfil dos usuários da ferramenta. Não tínhamos o intuito de medir o nível de cada pessoa específica, e sim de um grupo de pessoas de maneira uniforme. Para isso, utilizamos apenas os dados gerais do antes e após.

Pudemos perceber que grande parte veio em busca de estudo pessoal e não tinha utilizado nenhuma ferramenta anteriormente. O nível de escolaridade era em mais da metade do ensino médio (Figura 4.1), algo que nos surpreendeu, visto que conseguimos incentivar pessoas mais novas, e que estarão entrando em breve no ensino superior, a aprender criptografia.

Nessas perguntas testando primeiramente o conhecimento do usuário em reconhecer algoritmos de chaves assimétricas (Figura 4.3), chaves simétricas (Figura 4.4) e de dispersão (Figura 4.5). Após definir a idade (Figura 4.1) e escolaridade (Figura 4.2) de nossos usuários, além do interesse (Figura 4.6) no uso, assim definindo um perfil para os mesmos.

Em seguida, precisávamos definir o nível de conhecimento (Figura 4.7) dos usuários sobre os algoritmos, antes de utilizar a ferramenta e se o mesmo já tinha utilizado outra ferramenta (Figura 4.8) de mesmo propósito. O que demonstrou que poucas pessoas utilizaram outras ferramentas parecidas, em nossos testes, apenas um usuário teve acesso à aplicativos semelhantes.

Idade (13 respostas)

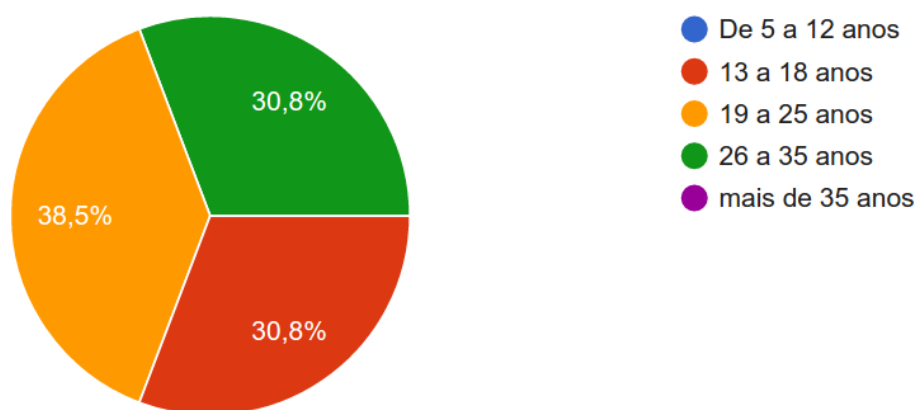


Figura 4.1: Pergunta pré utilização sobre idade.

Escolaridade (13 respostas)

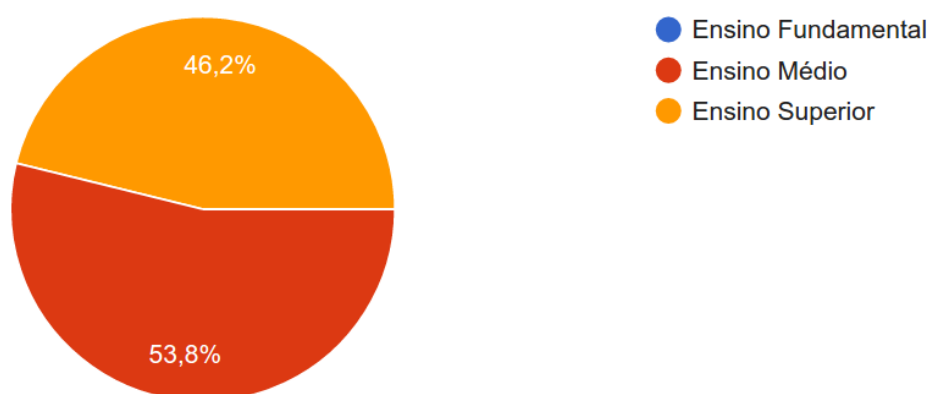


Figura 4.2: Pergunta pré utilização sobre escolaridade.

Marque os algoritmos de chave assimétrica (ou chave pública):

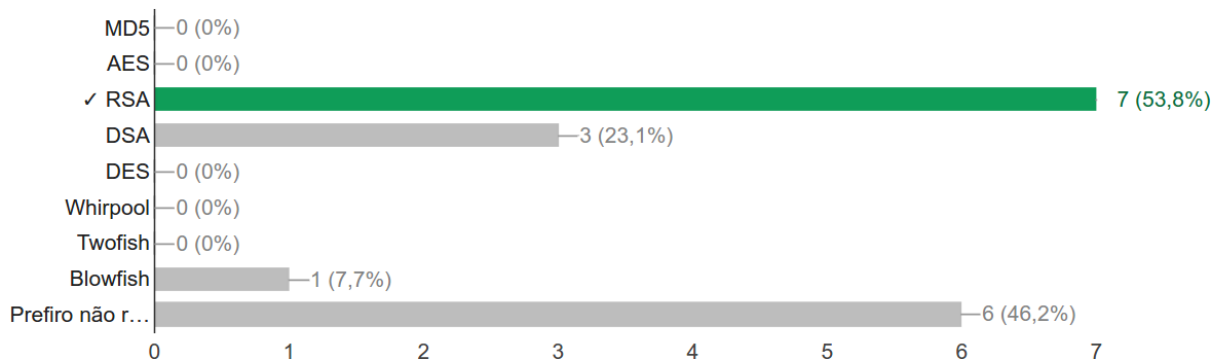


Figura 4.3: Pergunta pré utilização sobre chave assimétrica.

Marque os algoritmos de chave simétrica:

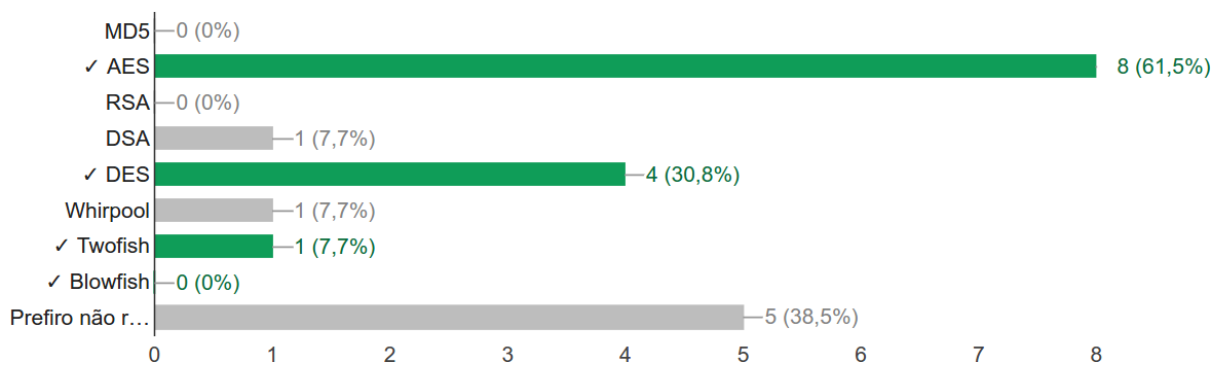


Figura 4.4: Pergunta pré utilização sobre chave simétrica.

Marque as funções hash ou de dispersão

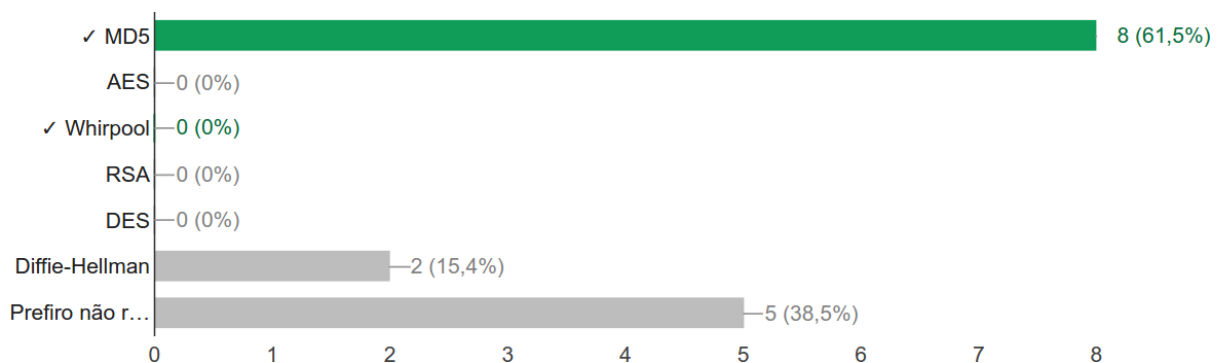


Figura 4.5: Pergunta pré utilização sobre funções de dispersão.

Qual é seu interesse na utilização na ferramenta?

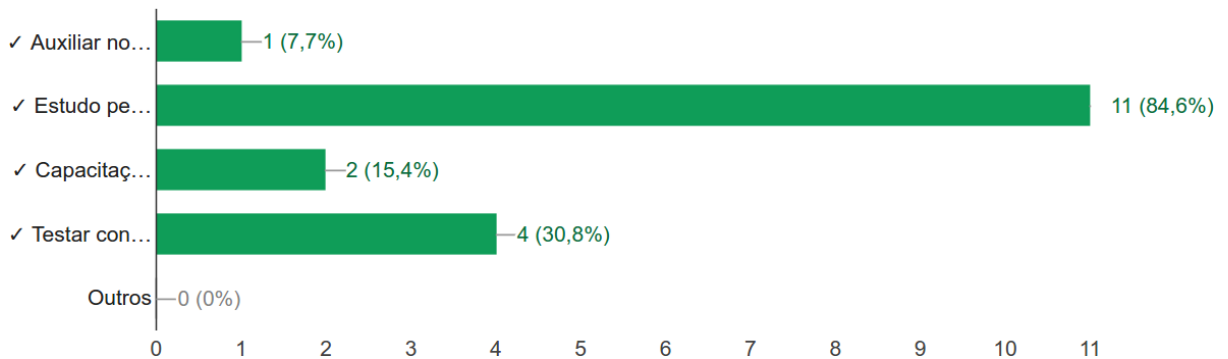


Figura 4.6: Pergunta pré utilização sobre interesse na ferramenta.

Qual seu nível de conhecimento nos seguintes algoritmos/funções criptográficas:

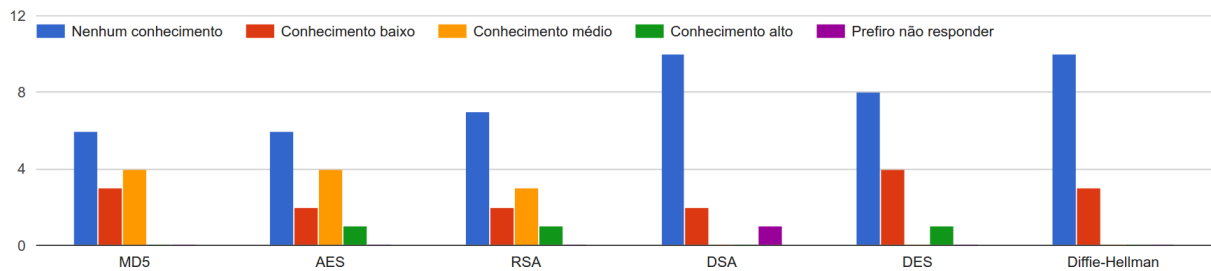


Figura 4.7: Pergunta pré utilização sobre nível de conhecimento do usuários.

Já utilizou alguma ferramenta de mesma finalidade anteriormente? Se sim, qual?

(12 respostas)

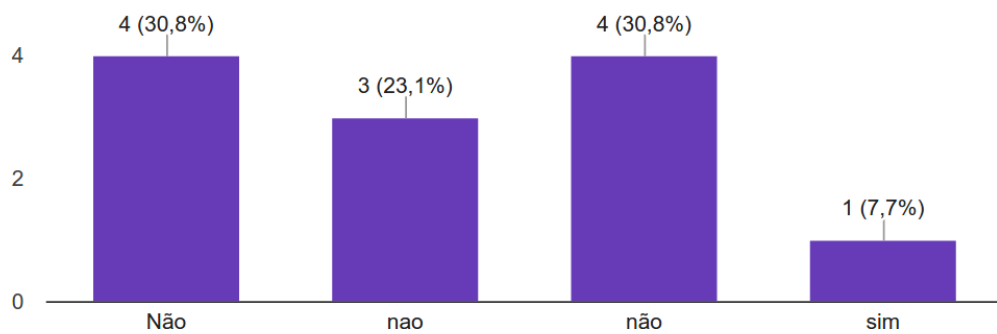


Figura 4.8: Pergunta pré utilização sobre utilização de outras ferramentas.

4.2 Questionário Pós Utilização

A partir da utilização da ferramenta, outras perguntas eram feitas, referente ao que lhes era apresentado. No total eram 10 perguntas, cada uma valendo 1 ponto, e obtivemos os resultados abaixo para o nível de conhecimento adquirido. Essas perguntas podem ser visualizadas no Apêndice B.

O MD5 demonstrou ter maior nível de explicação, pois tivemos maior número de acertos nas respostas (Figura 4.11), tendo maior índice de pontos entre 7 e 8 e média de 6 pontos. Mostrando assim que os usuário conseguiram reter um bom nível de conhecimento da ferramenta. Resultado comum por ser o algoritmo considerado de mais fácil entendimento dos apresentados na ferramenta, utilizando em seus passos operações matemáticas triviais.

O RSA teve um desempenho abaixo da média (Figura 4.10), tendo como maior índice os valores 2 e 3 e média de 3.5, mostrando que o nível de explicação não condiz com as perguntas realizadas no teste. Vale ressaltar que o uso de problemas matemáticos complexos dificultam o entendimento do algoritmo, mesmo sendo uma média baixa já era esperado um resultado abaixo dos outros algoritmos.

O AES teve desempenho abaixo da média (Figura 4.9), como o RSA, tendo como maior índice o valor 4 e média igual ao do RSA, mostrando que precisamos melhorar e facilitar o entendimento do mesmo. Resultado não esperado, por ser o algoritmo mais popular e conhecido entre os apresentados, mas também explicado por usar uma base matemática complexa para gerações das tabelas. O valor de maior repetição sendo maior que o RSA mostra que por ter operações mais simples torna-se de mais fácil entendimento, mas por conter um nível de dificuldade similar, iguala a mesma média.

Tivemos dois *feedbacks* negativos no RSA (Figura 4.13) e no MD5 (Figura 4.14) a respeito da interface, porém não justificados, o que dificulta em determinar o porquê da ferramenta não ter agradado. Alguns não concordaram com a metodologia de ensino e qualidade, mas acreditamos que seja porque os usuários não estejam acostumados ou familiarizados com este tipo de ensino.

Os outros *feedbacks* que a ferramenta recebeu tiveram um nível de aceitação de razoável para bom (Figura 4.12), tendo alguns desvios em interface e qualidade.

Informações

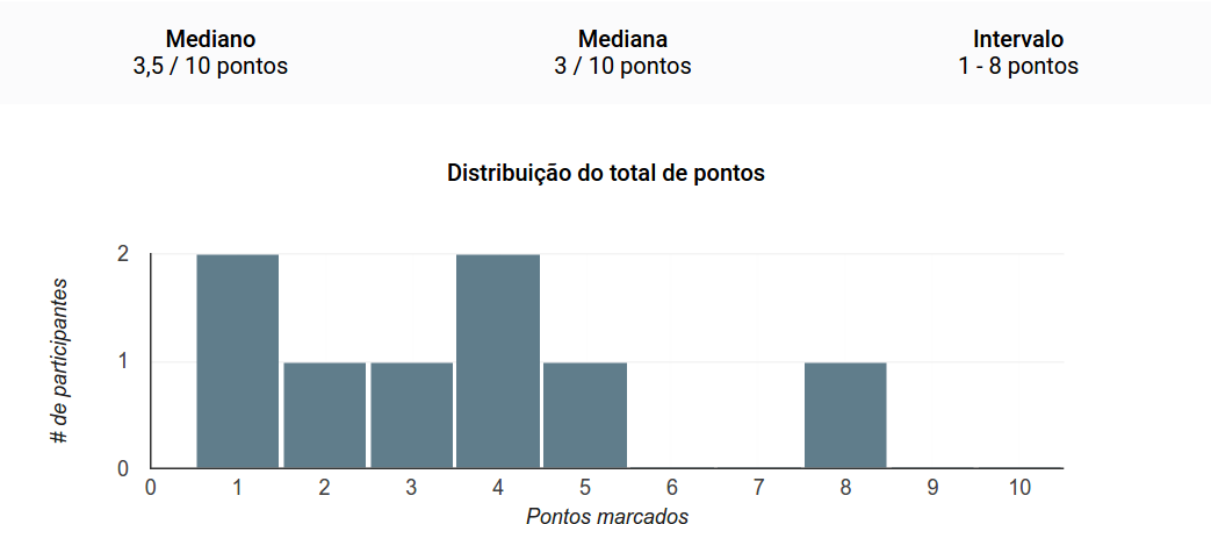


Figura 4.9: Distribuição de pontos questionário de chave simétrica.

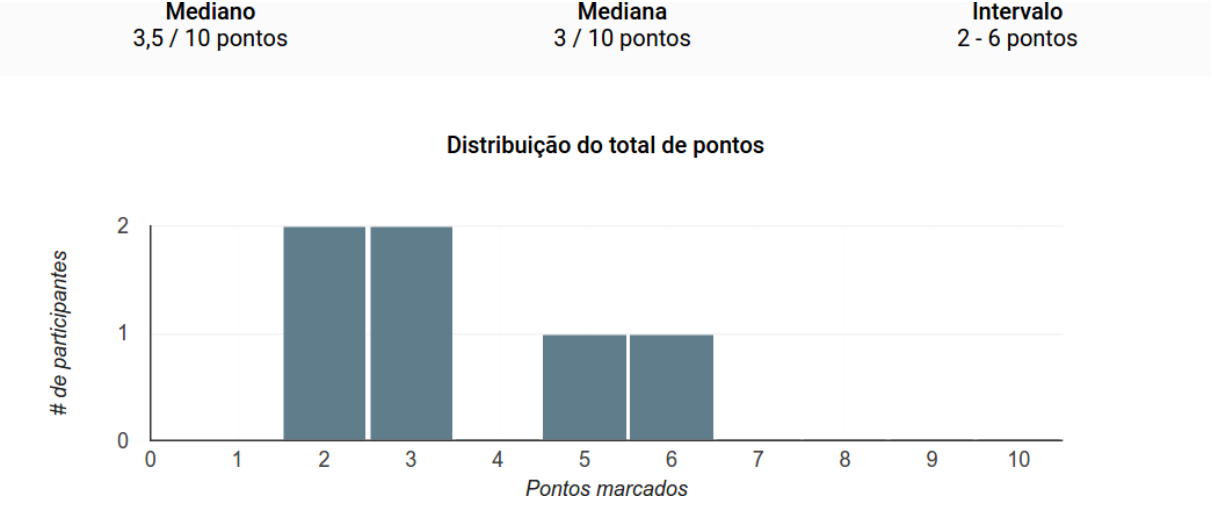


Figura 4.10: Distribuição de pontos questionário de chave assimétrica.

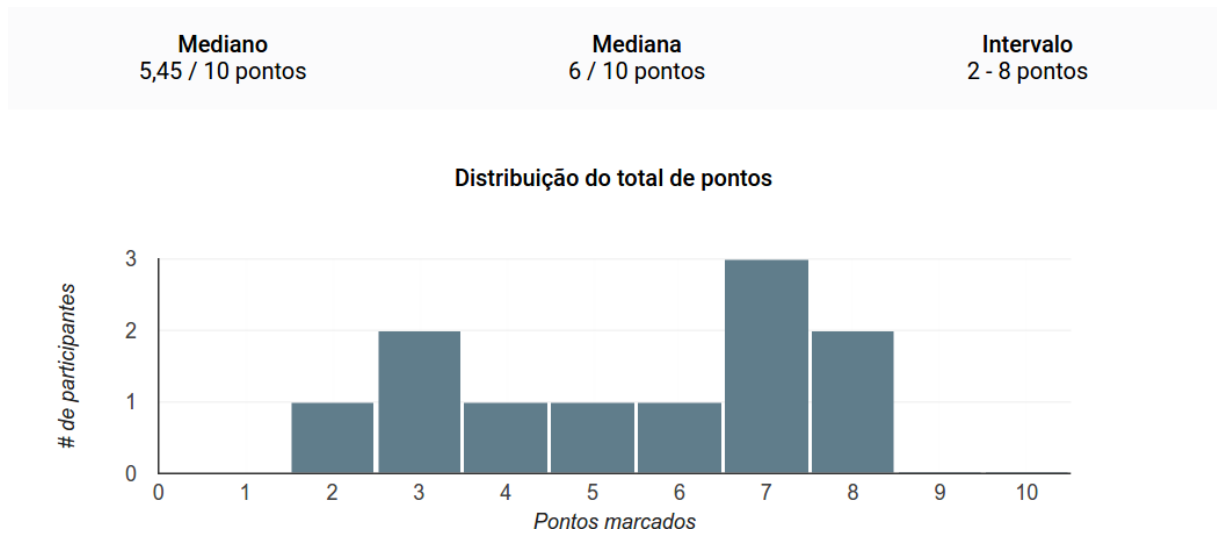


Figura 4.11: Distribuição de pontos questionário de função hash.

O que achou da ferramenta utilizada?

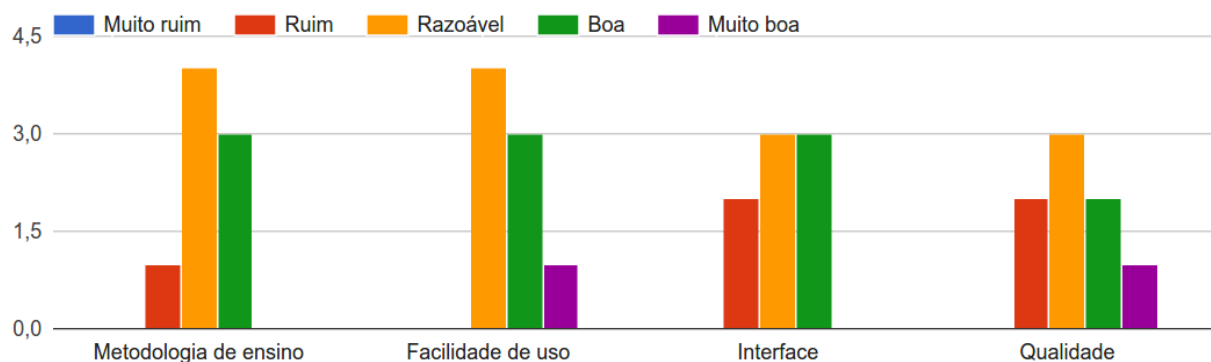


Figura 4.12: Opinião sobre a ferramenta no questionário de chave simétrica.

O que achou da ferramenta utilizada?

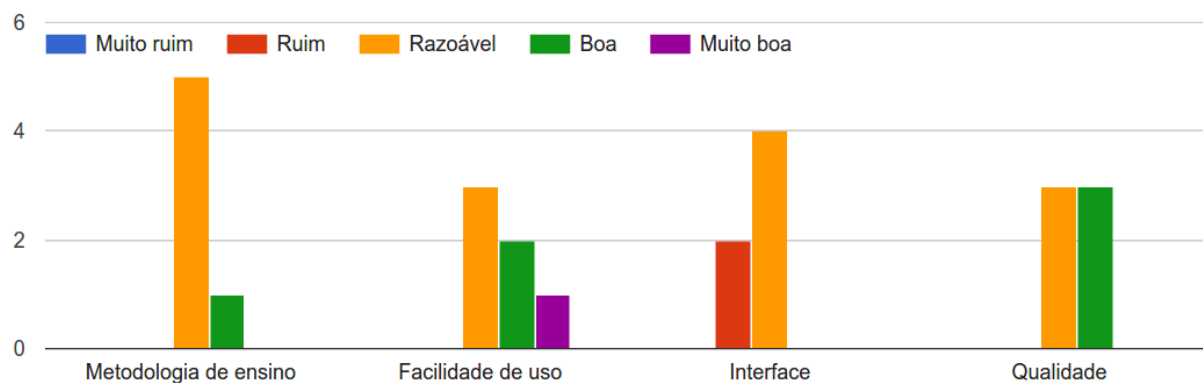


Figura 4.13: Opinião sobre a ferramenta no questionário de chave assimétrica.

O que achou da ferramenta utilizada?

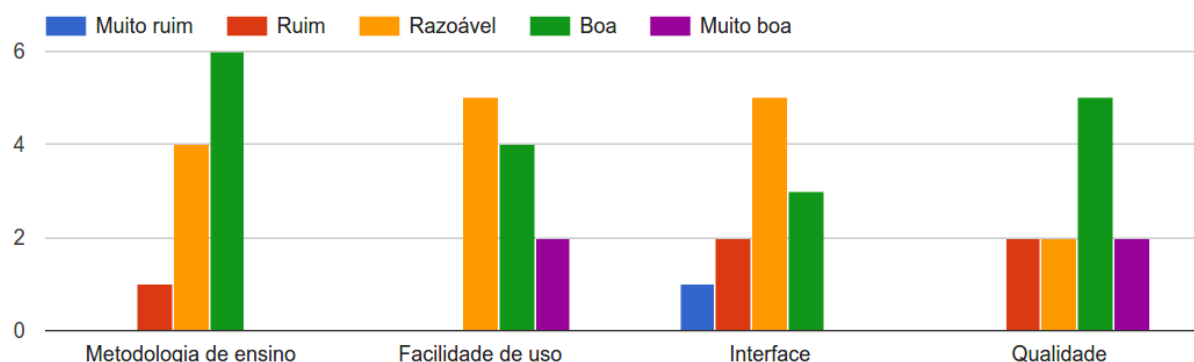


Figura 4.14: Opinião sobre a ferramenta no questionário de funções hash.

4.3 Análise dos resultados

Analisando os resultados podemos dizer que a ferramenta ainda está devendo na parte didática, médias baixas nas respostas das perguntas para os algoritmos AES e RSA, mostram que eles estão didaticamente inferior ao esperado, assim tendo que ser melhorados. Mas os resultados da metodologia de ensino, facilidade de uso, interface e qualidade da ferramenta, tiveram resultados acima do esperado, tendo como resposta mais comum o valor razoável-bom, mostrando um bom desempenho da ferramenta quanto ao seu projeto.

Outro ponto a se ressaltar é a discrepância do preenchimento do formulário pré e pós utilização, a utilização de testadores sem compromisso com a ferramenta acabou tornando os testes menos eficientes, pois os usuários não terminavam ou só utilizam alguns de nossos algoritmos. Tendo como escolha de uso os mais populares, o RSA sofreu com isso e assim pode ter tido uma análise ineficaz de resultado.

Além disso, percebemos ao longo dos testes, que o número e tamanho das perguntas também influenciavam na desistência dos testadores, mesmo que no começo do processo tentamos gerar testes mais sucintos e objetivos, não esperávamos tal número de desistências. Ao percebermos essas falhas, já era tarde para podermos refazer os testes com um número maior de usuários.

Capítulo 5

Conclusão

A ferramenta desenvolvida tem como principal objetivo auxiliar no ensino de criptografia, tendo foco principal em algoritmos mais populares. Por isso, a partir do conhecimento adquirido através do curso de Ciência da Computação, podemos entender e explicar de forma mais didática e resumida tais algoritmos, operações de difícil entendimento começaram a ser tornar mais triviais a partir do estudo dos mesmos.

Nossos objetivos propostos no início do projeto acabaram não sendo todos concretizados, pois alguns problemas foram encontrados no meio do caminho, como dificuldade na implementação e auxílio de um servidor para testes. Mas os objetivos principais foram realizados, os quais são a ferramenta desenvolvida com os algoritmos propostos, e a realização de testes para a definição da qualidade. E a partir desses testes percebemos que a ferramenta deveria ser melhorada quanto ao conteúdo referente a parte dos slides de explicação do algoritmo. Como a ferramenta tem um objetivo de auxiliar, mas não de substituir, pois sem um professor o aprendizado acaba sendo prejudicado, isso torna o problema menos importante, mas não descartável.

Acreditamos que o projeto ficou ideal para uma fase inicial, esperamos que a ferramenta seja aprimorada por outros estudantes e profissionais da área, visto que a mesma será *open-source* e estará disponível para que qualquer um possa alterar e utilizar, mas não vendê-la. A ferramenta necessita ser mais testada, ser mais completa em termos de algoritmos, e também explicar de forma mais detalhada alguns passos. Por exemplo, no RSA, nesta versão da ferramenta estamos considerando que o usuário possui conhecimento sobre o algoritmo de Euclides Estendido, que tem como o objetivo calcular o inverso modular. Talvez nas próximas versões essa explicação já possa vir detalhada na própria ferramenta,

para que o usuário não precise visitar links externos para aprender o algoritmo.

Todo esse conhecimento adquirido nesse período de estudo aumentou nosso interesse em criptografia, conhecer a complexidade para se desenvolver cada algoritmo nos fascinou a estudar cada vez mais sobre cada um deles, além de outros como MD6, SHA-1, SHA-2 e DES. Assim criando uma excelente experiência desenvolvendo esse projeto.

Em relação aos trabalhos futuros, iremos divulgar a ferramenta na Universidade Federal Fluminense (UFF) e pedir que nossos professores e colegas de curso a utilize. Gostaríamos também que profissionais da área possam contribuir com o conhecimento prático da criptografia no dia-a-dia, assim como, adicionar curiosidades ou conceitos que ultrapassam simplesmente o entendimento dos algoritmos.

Referências Bibliográficas

- [1] Dan Boneh e Glenn Durfee. Cryptanalysis of rsa with private key d less than $n^{0.292}$. *IEEE transactions on Information Theory*, 46(4):1339–1349, 2000.
- [2] Joan Daemen e Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.
- [3] Nadeem Firasta, Mark Buxton, Paula Jinbo, Kaveh Nasri e Shihjong Kuo. Intel avx: New frontiers in performance improvements and energy efficiency. *Intel white paper*, 2008.
- [4] Jun Ma, Jun Tao, Melissa Keranen, Jean Mayo, Ching-Kuang Shen e Chaoli Wang. Shavisual: a secure hash algorithm visualization tool. Em *Proceedings of the 2014 conference on Innovation & technology in computer science education*, pp. 338–338. ACM, 2014.
- [5] Evgeny Milanov. The RSA algorithm. *RSA Laboratories*, 2009.
- [6] A Ray Miller. The cryptographic mathematics of enigma. *Cryptologia*, 19(1):65–80, 1995.
- [7] Ronald Rivest. The md5 message-digest algorithm. 1992.
- [8] Ronald L Rivest, Benjamin Agre, Daniel V Bailey, Christopher Crutchfield, Yevgeniy Dodis, Kermin Elliott Fleming, Asif Khan, Jayant Krishnamurthy, Yuncheng Lin, Leo Reyzin et al. The md6 hash function—a proposal to nist for sha-3. *Submission to NIST*, 2:3, 2008.
- [9] Ronald L Rivest, Adi Shamir e Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

- [10] Phillip Rogaway e Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. Em *International Workshop on Fast Software Encryption*, pp. 371–388. Springer, 2004.
- [11] Jun Tao, Jun Ma, Melissa Keranen, Jean Mayo, Ching-Kuang Shene e Chaoli Wang. Rsavisual: a visualization tool for the rsa cipher. Em *Proceedings of the 45th ACM technical symposium on Computer science education*, pp. 635–640. ACM, 2014.
- [12] Jun Tao, Jun Ma, Jean Mayo, Ching-Kuang Shene e Melissa Keranen. Desvisual: a visualization tool for the des cipher. *Journal of Computing Sciences in Colleges*, 27(1):81–89, 2011.

Apêndices

Apêndice A

Instalação da Ferramenta

A.1 Instalação

A ferramenta está disponível para Linux nas distribuições baseadas em Debian e em outras que são semelhantes. Abaixo iremos descrever o passo-a-passo da instalação em uma distribuição Ubuntu. Ou seja, talvez em outras distribuições os comandos listados abaixo não funcionem.

A.1.1 Instalando Python

Por padrão, o Python já vem instalado em grande parte das distribuições Linux. Mas caso não esteja instalada, use o seguinte comando no terminal como root (ou administrador):

```
$ apt-get install python python3 -y
```

A.1.2 Instalando Flask

Para instalar o Flask, execute o seguinte comando no terminal como root (privilegios administrativos):

```
$ apt-get install python-flask -y
```

A.1.3 Instalando a Ferramenta

A ferramenta pode ser instalada usando o shell script “install.sh” ou executando os passos descritos abaixo, também como root:

```
$ apt-get install python-pip python-dev -y
$ pip install --upgrade pip
$ pip install -r requirements.txt
```

Caso queira hospedar em um servidor, poderá utilizar o Apache e configurá-lo seguindo as instruções descritas abaixo:

```
$ apt-get install apache2 libapache2-mod-wsgi -y
$ a2enmod wsgi
$ cat >/etc/apache2/sites-available/$appName.conf <<EOL
<VirtualHost *:${appPort}>
    ServerName 127.0.0.1
    ServerAdmin admin@mywebsite.com
    WSGIScriptAlias / /var/www/$appName/server.wsgi
    <Directory /var/www/$appName/>
        Order allow,deny
        Allow from all
    </Directory>
    Alias /static /var/www/$appName/static
    <Directory /var/www/$appName/static/>
        Order allow,deny
        Allow from all
    </Directory>
    ErrorLog ${APACHE_LOG_DIR}/error.log
    LogLevel warn
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
EOL
$ a2ensite $appName
$ service apache2 restart
```

Onde *\$appName* é o nome escolhido para a sua aplicação e *\$appPort* é a porta na qual estará rodando a sua aplicação.

Apêndice B

Perguntas Testes

B.1 Questionário Pós Utilização

B.1.1 Perguntas referentes ao MD5

- O que é uma função hash ou função de dispersão?
 - Uma função hash é um algoritmo de via única que mapeia dados com comprimento variável para dados de comprimento fixo, com isso, podemos "compactar" grandes volumes de dados de forma fácil e prática. (Resposta Correta)
 - Função hash é algoritmo criptográfico simétrico que utiliza uma chave como entrada e gera um hash como saída.
 - Função hash é algoritmo criptográfico assimétrico que utiliza uma chave como entrada e gera um hash como saída, que irá utilizar uma segunda chave para recuperar a mensagem original.
 - Uma função hash é um algoritmo que mapeia dados com comprimento variável para dados de comprimento fixo, com isso, podemos "compactar" grandes volumes de dados de forma fácil e prática, e ainda podemos recuperar a informação original descompactando o hash.
 - Prefiro não responder.
- Marque em quais situações devemos usar funções hash:
 - Garantir integridade da mensagem. (Resposta correta)

- Comunicação de mensagens sigilosas, como mensagens de email.
 - Ocultar dados sigilosos, como senhas. (Resposta correta)
 - Compactar arquivos e/ou informações grandes.
 - Prefiro não responder.
- Quais as características de boa função hash?
 - A taxa de colisão é muito alta, irreversível e fácil de calcular.
 - A taxa de colisão é muito alta, reversível e difícil de calcular.
 - A taxa de colisão é muito baixa, reversível e difícil de calcular.
 - A taxa de colisão é muito baixa, irreversível e fácil de calcular. (Resposta correta)
 - A taxa de colisão é muito baixa, irreversível e difícil de calcular.
 - Prefiro não responder.
- É possível recuperar a informação original com garantia depois de ter executado uma função hash?
 - Sim, com certeza podemos!
 - Não, definitivamente não! (Resposta correta)
 - Prefiro não responder.
- Quais as principais diferenças entre uma função hash e um algoritmo criptográfico?
 - Algoritmos criptográficos não precisam de uma chave de entrada e funções hash precisam.
 - Em uma podemos recuperar a informação com precisão, na outra não. (Resposta correta)
 - Uma função hash possui tamanho fixo, já algoritmos criptográficos possuem tamanho variável. (Resposta correta)
 - Prefiro não responder.
- Qual o tamanho, em bits, da mensagem produzida pela função MD5?

- 512 bits
 - 256 bits
 - 128 bits (Resposta correta)
 - 64 bits
 - Prefiro não responder.
- O vetor constante K, possui quantos elementos?
 - 16 elementos
 - 32 elementos
 - 64 elementos (Resposta correta)
 - 128 elementos
 - Prefiro não responder
- Ao início do quarto passo, a mensagem é dividida em 16 palavras de qual tamanho, em bits?
 - 16 bits
 - 32 bits (Resposta correta)
 - 64 bits
 - 128 bits
 - Prefiro não responder
- Em uma mensagem inicial de 128 bits, quantas vezes o procedimento leftrotate é executado?
 - 4 vezes
 - 16 vezes
 - 32 vezes
 - 64 vezes (Resposta correta)
 - Prefiro não responder
- Ao início do passo 3, qual será o tamanho da mensagem, em bits?

- 64 bits
- 128 bits
- 256 bits
- 512 bits (Resposta correta)
- Prefiro não responder

B.1.2 Perguntas referentes a chave simétrica e assimétrica

- Marque os algoritmos de chave simétrica:
 - MD5
 - AES (Resposta correta)
 - RSA
 - DSA
 - DES (Resposta correta)
 - Whirpool
 - Twofish (Resposta correta)
 - Blowfish (Resposta correta)
- Marque os algoritmos de chave assimétrica:
 - MD5
 - ElGamal (Resposta correta)
 - AES
 - RSA (Resposta correta)
 - DSA (Resposta correta)
 - DES
 - Twofish
 - Blowfish
 - Diffie-Hellman (Resposta correta)

- Qual a diferença entre criptografia por chave simétrica e chave assimétrica (ou chave pública)?
 - Na criptografia por chave simétrica as chaves escolhidas são opostas matematicamente. Já na criptografia de chave pública as chaves são distribuídas de maneira pública, ou seja, que qualquer um pode ter acesso a elas.
 - Na criptografia de chave pública utilizamos duas chaves, uma para encriptar e outra para decriptar. Enquanto na criptografia por chave simétrica, utilizamos apenas uma chave para encriptar e decriptar. (Resposta correta)
 - Nos dois casos são geradas duas chaves. A diferença é que na criptografia de chave pública, podemos compartilhar uma das chaves geradas e continuar seguros. Já na criptografia de chave simétrica, não podemos pois seria muito fácil de calcular a chave simétrica, ou oposta matematicamente.
 - Prefiro não responder.
- Por que a chave criptográfica dos algoritmos de chave assimétrica costumam ser maiores do que as chaves dos algoritmos de chave simétrica?
 - Porque o algoritmo de chave assimétrica usa duas chaves enquanto o algoritmo de chave simétrica utiliza apenas uma.
 - Porque os algoritmos de chave assimétrica são baseados em problemas matemáticos difíceis para serem resolvidos nos computadores clássicos. (Resposta correta)
 - Porque os algoritmos de chave assimétrica são mais rápidos que os de chave simétrica para o mesmo tamanho de chave.
 - Prefiro não responder.
- É correto afirmar que os algoritmos de chave simétrica são mais simples do que os algoritmos de chave assimétrica (ou chave pública)?
 - Sim, pois os algoritmos de chave simétrica utilizam apenas substituição e operações lógicas como XOR, AND e OR. Enquanto os algoritmos de chave assimétrica, utilizam algoritmos para calcular o inverso modular, operações com expoentes grandes e cálculo de números primos, algo que requer muito mais poder de processamento computacional. (Resposta correta)

- Não, pois os algoritmos de chave simétrica são muito mais seguros do que os de chave assimétrica.
- Não, os algoritmos de chave simétrica utilizam cálculos super complexos, enquanto que os algoritmos de chave assimétrica podem ser executados em até pequenos circuitos, placas e computadores, como Arduino, Raspberry PI e smartphones.
- Prefiro não responder.

B.1.3 Perguntas referentes ao AES

- Marque os nomes das tabelas utilizadas:
 - S-Box e S-BoxInv (Resposta correta)
 - SCON
 - RCON (Resposta correta)
 - R-Box
 - Tabelas do Corpo Finito de Galois (Resposta correta)
- Qual a sequência correta de etapas na fase de encriptação?
 - No passo inicial teremos apenas a etapa do AddRoundKey. Nos passos intermediários teremos as etapas de SubBytes, ShiftRows, MixColumns e AddRoundKey. E no passo final aplicaremos SubBytes, ShiftRows e MixColumns apenas. (Resposta correta)
 - No passo inicial teremos apenas a etapa de SubBytes. Nos passos intermediários teremos as etapas de AddRoundKey, ShiftRows, MixColumns e SubBytes. E no passo final aplicaremos SubBytes, ShiftRows e MixColumns apenas.
 - No passo inicial teremos a etapa de AddRoundKey, ShiftRows e MixColumns. Nos passos restantes teremos as etapas de AddRoundKey, ShiftRows, MixColumns e SubBytes.
 - Prefiro não responder.
- Qual o objetivo de utilizarmos a tabela uma tabela de substituição? (S-Box)

- Embaralhar os dados para dificultar um ataque linear.
 - Mitigar ataques algébricos simples. (Resposta correta)
 - Criar difusão na cifra através de funções afim e transformações do corpo finito de Galois.
 - Criar confusão utilizando uma sequência de trocas aleatórias.
 - Prefiro não responder.
- Quais os procedimentos que geram difusão?
 - AddRoundKey e SubBytes
 - SubBytes e MixColumns
 - MixColumns e ShiftRows (Resposta correta)
 - ShiftRows e SubBytes
 - SubBytes e AddRoundKey
 - Prefiro não responder.
- Quais são a quantidade de rounds e tamanho da chave expandida de uma chave com 128 bits (AES-128)?
 - 10 e 176. (Resposta correta)
 - 11 e 196.
 - 12 e 240.
 - 14 e 256.
 - Prefiro não responder.

B.1.4 Perguntas referentes ao RSA

- Como um usuário pode recuperar a informação original depois de um algoritmo assimétrico ter sido executado no servidor?
 - Podemos usar a chave pública do servidor para descriptografar uma mensagem criptografada com a chave pública do usuário.

- Podemos requisitar a chave privada do servidor para descriptografar uma mensagem criptografada com a chave pública do servidor.
 - Podemos utilizar a chave pública do servidor para descriptografar a mensagem previamente criptografada com a chave privada do servidor. (Resposta correta)
 - Prefiro não responder.
- Como podemos garantir o não-repúdio (ou incontestabilidade) de mensagens do usuário através de chaves assimétricas?
 - O usuário deverá criptografar a mensagem utilizando a sua chave pública e enviar para o destinatário desejado.
 - O usuário irá criptografar a mensagem usando a sua chave privada e enviar para o destinatário desejado. (Resposta correta)
 - A mensagem será criptografada utilizando a chave pública do destinatário e enviada para o mesmo.
 - Prefiro não responder.
- A segurança do RSA é baseada em:
 - Quantidade de bits utilizados (Resposta correta)
 - Problemas matemáticos complexos (Resposta correta)
 - Encadeamento de blocos
 - Operações XOR
 - Utilização de duas chaves distintas (pública e privada)
 - Prefiro não responder.
- Qual a quantidade de bits que o RSA necessita para ser considerado seguro?
 - A partir de 128
 - A partir de 256
 - A partir de 512
 - A partir de 1024 (Resposta correta)
 - Prefiro não responder.

- Qual a quantidade mínima de informações adicionais que precisamos para calcular a chave privada a partir do conhecimento da chave pública, considerando também o custo computacional?
 - P e Q .
 - N e K .
 - Φ (totiente). (Resposta correta)
 - P , Q e E .
 - Prefiro não responder.