Codificação de texto formatado em imagens para uso em aplicações web

Emanuel Cauê Nolasco Rodrigues
Mossoró, RN
Brasil
Emanuel.caue@gmail.com

Resumo *Esse trabalho apresenta quatro abordagens de codificar texto na linguagem de formatação HTML em imagens. As abordagens investigadas foram o mapeamento direto, a tabela diferente com lixo, a criptografia com chave aleatória a partir da cifra de Viginère e a criptografia baseada nos algoritmos AES e MD5. As abordagens foram comparadas e a segurança em cada abordagem é gradualmente atingida para a consecução do objetivo de guardar e proteger os dados de forma segura em uma imagem.*

Palavras-Chave: *Processamento digital de imagem, codificação de texto, esteganografia, criptografia, HTML, página Web.*

Abstract *This work presents four approaches to encoding text in the HTML formatting language in images. The approaches investigated were the direct mapping, the different table with garbage, the random key cryptography from the Viginère cipher and the encryption based on the AES and MD5 algorithms. Approaches have been compared and security in each approach is gradually achieved to achieve the goal of securely storing and protecting data in an image.*

Keywords: *Digital image processing, text coding, steganography, encryption, HTML, Web page*

1 Introdução

Guardar, enviar e receber informações sigilosas, é uma necessidade há séculos, com o crescimento da internet, esse é assunto cada vez mais presente. Neste trabalho, será abordado a aplicação de codificação de texto em forma de imagens, e como essa codificação pode ser segura e imprevisível. O objetivo é criar uma aplicação que gere imagens para qualquer pessoa guardar, de forma que ela não consiga ser lida, sem a ajuda da aplicação e consequentemente, não consiga entender o padrão por trás da imagem gerada para ler imagens de outras pessoas.

Utilizar este tipo de imagem codificada pode vir a ser útil, tanto para guardar dados pessoais, de forma a não poderem ser lidos, quanto para enviar dados sob quaisquer circunstâncias dentro de uma rede não protegida, podendo ser esses dados uma página inteira em HTML.

Uma aplicação pode usar esse tipo de codificação para enviar as imagens para serem interpretadas pelo cliente através do decodificador, e só então mostrar os dados, o que dispensaria um método de autenticação diretamente na aplicação, ao invés de somente enviar uma cópia do HTML guardado no servidor, como é feito normalmente em aplicações sem privacidade de conteúdo.

Antes de ser possível inserir um texto dentro de uma imagem, é necessário um método de mapear os valores do texto lá dentro, que possa ser lidos de volta à transformar a imagem num texto. Será abordado, então, o básico de como as imagens e texto são guardados nos computadores modernos, o básico sobre imagens digitais, e como unir os dois conceitos para gerar imagens com texto criptografado.

2 Abordagens para codificação de texto formatado em imagem

Nessa seção apresentam-se as 4 abordagens investigadas para a codificação de texto formatado em imagens coloridas: i) O mapeamento direto, ii) a tabela diferente com lixo, iii) a criptografia com chave aleatória a partir da cifra de Viginère e iv) a criptografia baseada nos algoritmos AES e MD5.

2.1 Abordagem 1: Mapeamento Direto

O mapeamento direto busca codificar textos em padrão ASCII, em imagens. Como cada caractere tem um valor de até 8 bits na tabela ASCII, cada pixel, pode conter uma tripla de caracteres. Sendo assim, essa estratégia pode ser usada para converter um texto em imagem diretamente, ou seja, cada valor corresponde diretamente à uma componente de um pixel. A seguinte expressão, ilustra o valor de cada pixel: I(i,j) = T(a, b, c), sendo ‘i’ e ‘j’, as coordenadas do pixel na imagem, e ‘a’,’b’ e ‘c’, os caracteres lidos no texto. Ao final da estrutura de pixels, um novo I(i,j) = T (1, 2, 3), é adicionado, para definir que esta, é uma imagem codificada usando o algoritmo de mapeamento direto. Os componentes vazios, após o fim do texto, serão preenchidos com o valor 0, até o final da imagem, já que esse valor não é mapeado para nenhum caractere útil na tabela ASCII, da seguinte forma: I(i,j) = T(0,0,0);

Essa abordagem tende a proteger os dados de qualquer um, desde que este não entenda como os dados são guardados nas imagens e textos. Mas, uma simples conversão de cor para caractere, pode transformar a imagem de volta. A imagem a seguir, ilustra um texto maior aleatório de 7 parágrafos, retirado do site

Lipsum.com

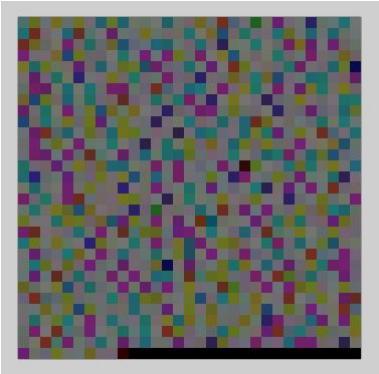

Figura 1: Texto aleatório mapeado diretamente para imagem

Essa imagem, mesmo parecendo ser bastante 'colorida', possui um padrão muito fácil de perceber se analisada pixel a pixel, como ilustrado na imagem a seguir que representa seu histograma da componente R:

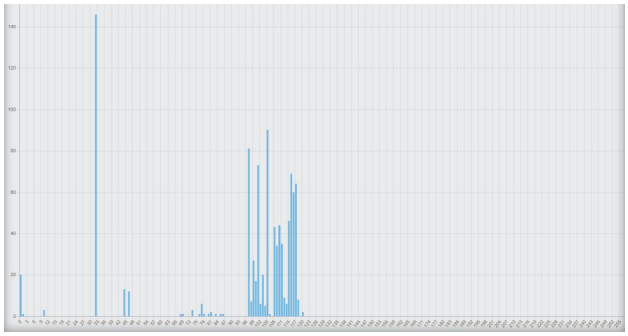

Figura 2: A componente R de uma imagem codificada diretamente

A fragilidade deste método é que todos os pixels ficam totalmente concentrados em uma certa região do gráfico, a primeira metade do histograma, e mais concentrada na área entre 97 e 120, que corresponde às letras, e o valor 32, que corresponde aos espaços (o que justifica a tonalidade mais escura da imagem) e torna ainda mais claro que as imagens foram mapeadas para ASCII.

Ou seja, esse método pode ser decodificado sem nenhuma análise de frequência, bastando converter cada pixel para elemento da tabela ASCII diretamente. Ao final, o texto recebido fará sentido, e poderá ser lido por qualquer um, pois todos os caracteres são úteis para a comunicação.

2.2 Abordagem 2: Tabela diferente com lixo

Não usar a tabela ASCII, torna a mensagem mais difícil de ser decodificada, devido ao fato de cada valor da componente, teoricamente, poder se referir à qualquer caractere. Na implementação 2 deste trabalho, uma imagem será formada por pixels com valores úteis e pixels com valores inúteis, a faixa de valores úteis de cada componente é gerada aleatoriamente partindo de um número aleatório inicial (entre 0 e 128) e o número gerado somado a 128. A cada execução do algoritmo e o programa guarda esses valores de início em um pixel estratégico, que nesse caso é o último pixel. Além disso, os pixels úteis, são mapeados para valores de uma tabela de caracteres única, com ordem imprevisível. A tabela está listada a seguir: [, p, q, 7, #, z, :, Ê, ?, ú, @, j, È, T, D, Z, í, /, á, F, (, Ó, X, U, M, ,, Ç, <, Ô, 5, , y, ó, â, l, Ú, W, [, {, 1, \n, S, ô, s, >, f, 3, ª, P, _, -, a, o, À, L, }, ê, ', Q, R, Á, N, C, ~, k, ., $, ^, G, e, ", Y, 9,), ã, |, g, 2, *, 6, w, i, n, A,], ç, 0, !, t, u, v, %, O, 8, d, H, é, Â, E, h, \, °, è, É, Í, x, 4, b, V, c, B, &, m, à, J, I, +, K, =, Ã, r, ;, `. Nesta tabela, são usados pouco menos de 128 elementos, e o valor de cada caractere é somado ao valor gerado aleatoriamente, que obrigatoriamente estará dentro da faixa de pixels úteis. A função de que define os pixels é a mostrada a seguir: I(i,j) = T2(a+seedR,b+seedG,c+seedB), sendo seed o valor gerado aleatoriamente para cada componente. Os valores úteis preenchem cerca de 50% da imagem, enquanto vários valores inúteis, são gerados aleatoriamente de forma à suas componentes não pertencerem as faixas

úteis. Depois estes pixels são inseridos no meio dos dados de forma aleatória, até completar a imagem. Isso resulta em imagens diferentes para cada execução. que podem estar em qualquer cor de qualquer componente. Segue abaixo 3 imagens que são convertidas para a mesma frase 'Casa Branca', obtidas a partir de 3 execuções diferentes do algoritmo.

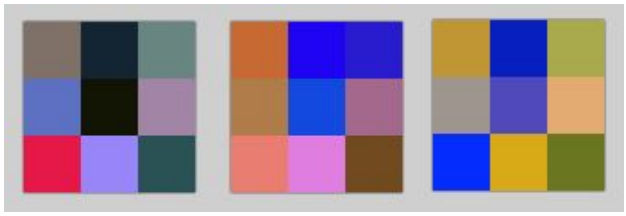

Figura 3: Execuções de um mesmo texto usando algoritmo 2

O último pixel, como já mencionado, guarda o valores de início das faixas I(i,j) = T2(seedR, seedG, seedB), e o primeiro pixel, guarda uma soma fixa ao último, para definir se essa é uma imagem codificada ou uma imagem normal, da seguinte forma: I(0,0) = T2(seedR + somaFixaR, seedG + somaFixaG, seedB + somaFixaB). Mesmo sendo possível haver imagens não codificadas com essa mesma distância entre valores de primeiro e último pixel, a chance é de uma em 16 milhões. O problema, desse método, é que, com o crescimento das entradas, os textos vão apresentando um certo padrão, no histograma. Como ilustrado a seguir, para um texto de volume médio de caracteres, e para outro texto de volume maior.

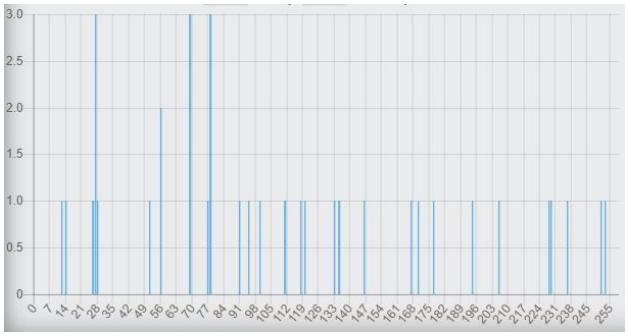

Figura 4: Histograma da componente R de um texto de volume médio de caracteres.

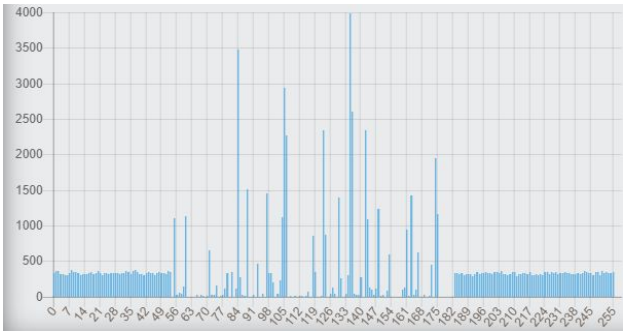

Figura 5: Histograma da componente R de um texto de volume alto de caracteres.

Além de que, possuem histogramas semelhantes, com a diferença sendo apenas a deslocação do início da faixa útil.

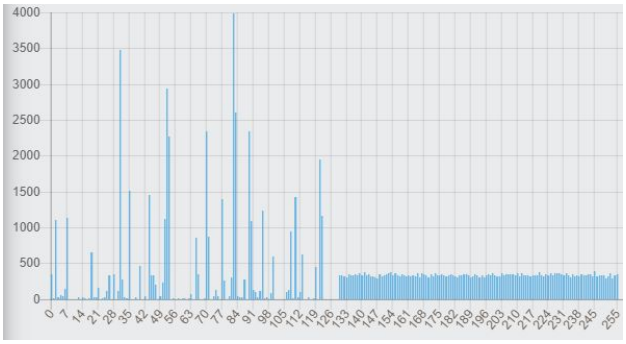

Figura 6: Histograma da componente R do texto da Figura 5, em outra execução do algoritmo 2.

Esse histograma deixa claro qual faixa de componente é útil ou não. Sabendo quais pixels são úteis, basta descobrir qual o tipo de mapeamento. Sem acesso ao processo de codificação, um método útil para decodificar a ser usado é o de análise de frequência, mas isso necessitaria o conhecimento da língua, e de um texto razoavelmente grande. Mas, se alguém tiver acesso ao processo de geração das imagens, e analisar as criações do algoritmo um a um, sabendo o texto e o resultado, fica fácil descobrir qual caractere é mapeado a cada valor,

gerando imagens com valores repetidos ‘a’ por exemplo, e vendo qual valor ele tem se comparado ao início dos valores úteis. A imagem a seguir, é resultado de uma compactação de 9744 a’s seguidos..

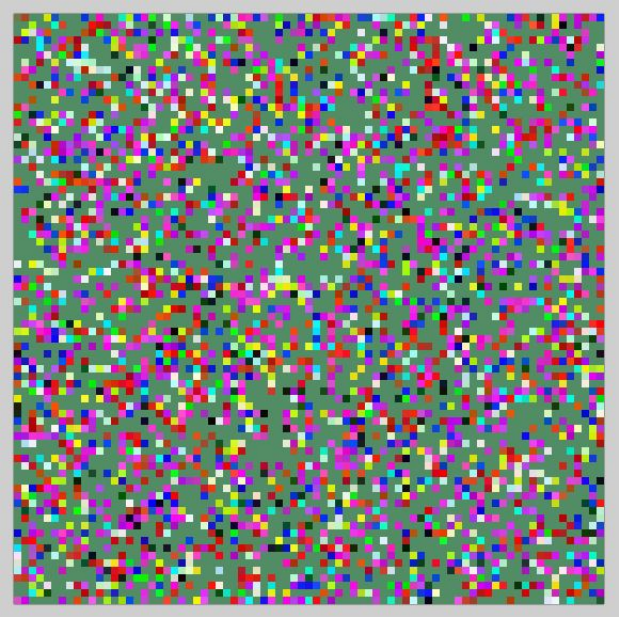

Figura 7: Imagem codificada com alta concentração de a’s.

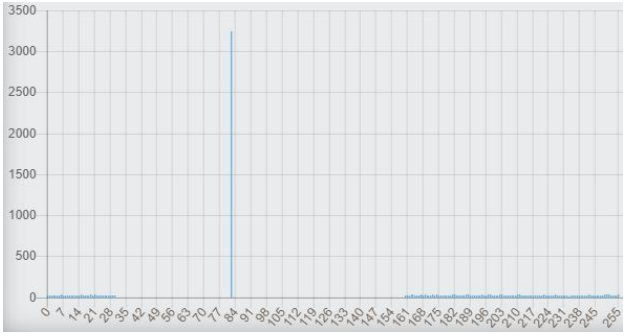

Figura 8: Histograma da componente R da imagem representada pela figura 7.

Nesse caso, o valor que mais se repete está à 51 posições do início dos valores úteis. O que é, exatamente o valor correspondente à letra ‘a’ na tabela.

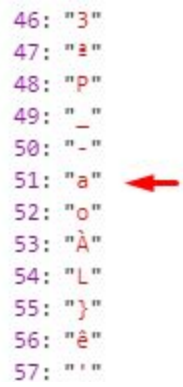

Figura 9: Valor de cada caractere na tabela de caracteres do algoritmo 2.

Essa codificação falha em proteger imagens de outros usuários, de alguém que tem acesso à codificações. E protege os dados da mesma forma que o primeiro método protegeria se a tabela ASCII fosse embaralhada. Porém, para proteger os dados de alguém que não tem acesso ao processo de codificação, esse método seria aproveitável, se os dados aleatórios, tivesse uma distribuição mais parecida com a da faixa útil, sendo impossível reconhecer uma padrão na análise de frequência. A única forma de quebrar esse padrão, seria espalhar os valores, mesmo que iguais, dentro da área útil.

2.3 Abordagem 3: Criptografia de Chave ‘Aleatória’

Uma forma útil de se guardar os valores sem nenhuma forma de serem decifrados, é criptografando eles antes de inserir na imagem. Na terceira implementação deste projeto, será mostrada a criptografia conhecida como ‘Cifra de Viginère’, usando uma chave de 2048 bits, escolhida aleatoriamente, dentro de uma chave maior, baseada no comprimento do texto informado pelo usuário.

A Cifra de Viginère, é um método de criptografia que busca deslocar cada valor do alfabeto n posições, para cada valor n da chave, onde a chave também pertence ao

alfabeto que se deseja criptografar., e então usando a chave repetitivas vezes ao longo da codificação/decodificação[3]. Esta Cifra, tem fraquezas, que se resumem à 2 fatores:

1. Comprimento pequeno da chave.
2. Comprimento de chave conhecido

Usando uma chave de tamanho 1, a criptografia se torna um simples deslocamento de tamanho máximo n, sendo n o tamanho do alfabeto. No nosso caso, o alfabeto é de 0-255, ou seja, com até 256 tentativas, dá pra achar o deslocamento que resulta no texto original. Com uma chave tamanho 2, são 256*256 = 65536 possibilidades, a chave usada na implementação tem tamanho no mínimo 128, ou seja, são 256^128 possibilidades, o que resolve a primeira fraqueza. O segundo problema diz respeito ao tamanho da chave, visto que a chave é utilizada o tanto de vezes que ela for necessária para codificar todo o texto, um mesmo deslocamento, será usado repetidas vezes ao longo da codificação, isso faz com que se usado textos de mesmo caractere, seja possível analisar qual o tamanho da chave, devido à repetição dos dados codificados. E saber o tamanho da chave, pode permitir o atacante, fazer uma análise de frequência na imagem, usando os caracteres separados por n caracteres, sendo n o tamanho da chave. Segue abaixo, exemplos de codificações de textos com múltiplos a's seguidos.

Figura 10: Codificação de Viginère, para caracteres 'a' repetidos, volume pequeno.

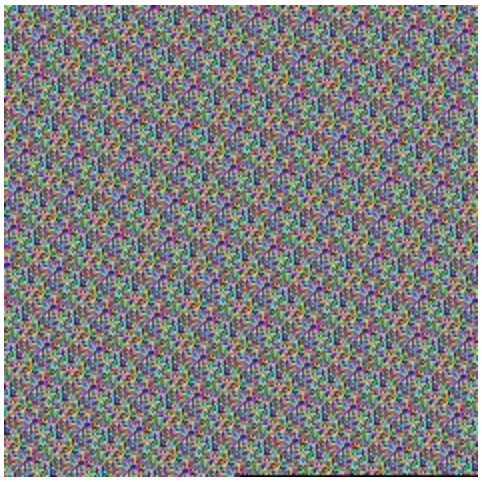

Figura 11: Codificação de Viginère, para caracteres 'a' repetidos volume médio.

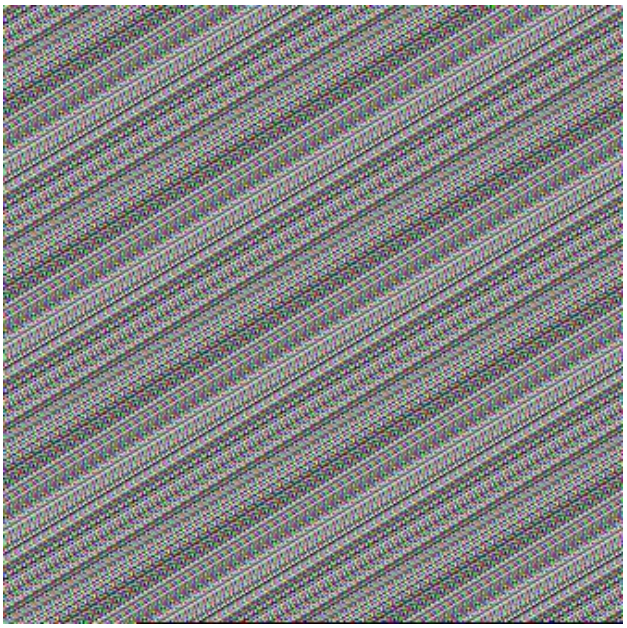

Figura 12: Codificação de Viginère, para caracteres 'a' repetidos, volume grande.

No exemplo, dá pra ser calculado o tamanho da chave, porém, o programa calcula um novo tamanho de chave para cada texto, pela expressão:

var key = key.slice((((text.length+7)**3)%256), ((text.length+7)**3%256)+(((text.length+89)*127)%256)

);

Como esse tamanho é usado somente para esse problema, é impossível usar o dado encontrado para decodificar outras imagens. Também é possível, esconder o tamanho da chave, passando a criptografia duas vezes, seguindo os seguintes passos: “Codificar o texto”, “Copiar a imagem gerada e colar no codificador”, “Codificar novamente, a imagem codificada”, como ilustrado nas figuras 12 e 13:

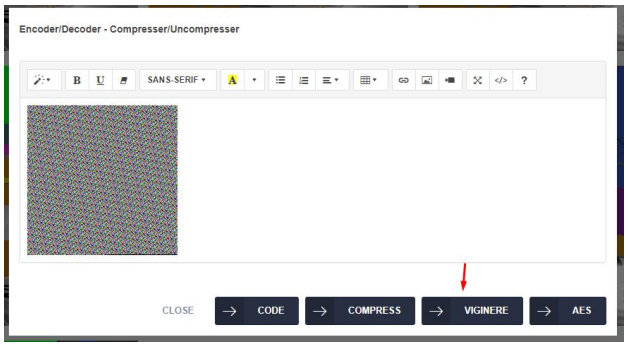

Figura 12: Copiando a imagem para a aplicação, e gerando Viginère novamente..

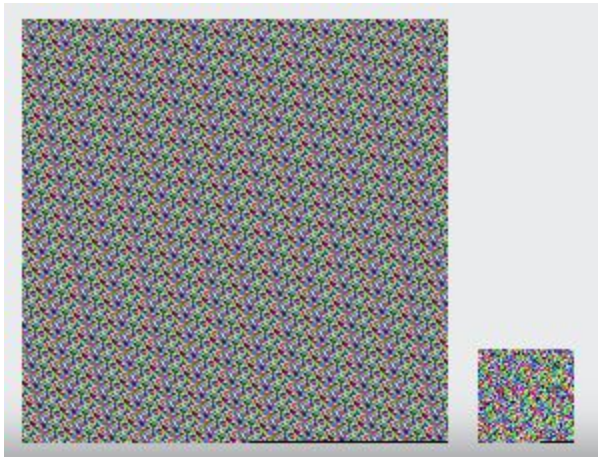

Figura 13: Imagem codificada em Viginère em base64, codificada em Viginère, depois de um base64.

A única informação que seria extraída dessa imagem, é que o texto se repete em tamanhos iguais ao tamanho da chave. Porém, não daria pra descobrir qual a letra, devido a chave ser aleatória. Mesmo que o usuário repita a mesma letra, mudando apenas a quantidade de caracteres, as cores de cada letra, é totalmente alterada, por exemplo

Segue abaixo as cores geradas por 10 e 11 a’s seguidos, respectivamente:

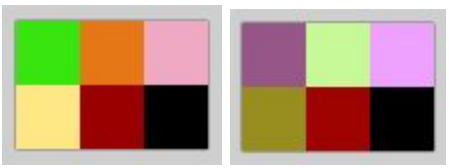

Figura 14: Codificação de Viginère, para caracteres ‘a’ repetidos, diferindo apenas um ‘a’.

Analisando o histograma de um texto grande (5 parágrafos do Lorem Ipsum), nenhum dado relevante pode ser retirado, devido à uma distribuição aleatória dos dados, como ilustrado pela figura 15 e 16.

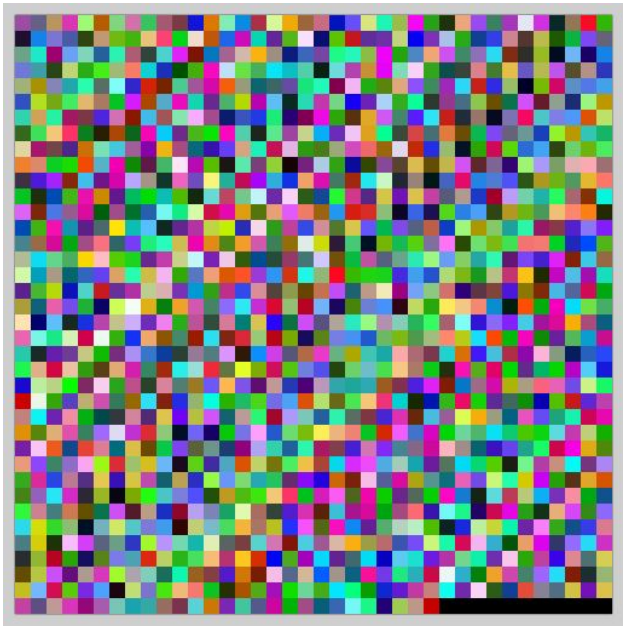

Figura 15: Codificação de Viginère, para um texto qualquer retirado do site “Lorem Ipsum”.

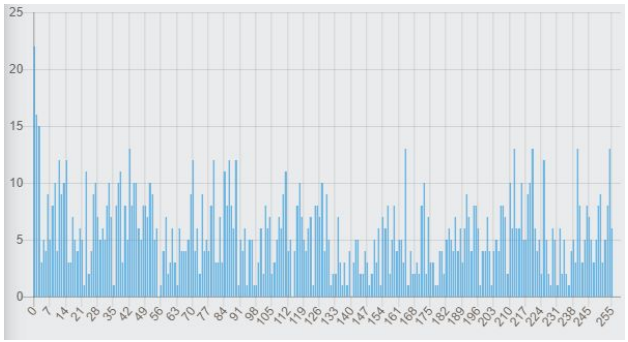

Figura 16: Histograma da componente R, da imagem representada pela figura 15.

2.4 Abordagem 4: Criptografia AES

A última abordagem é baseada no algoritmo de criptografia consolidado na literatura, o algoritmo AES [4].

Essa criptografia é criada a partir de muitos cálculos, os quais não serão explicado neste artigo, mas pode ser lido detalhadamente em um artigo ilustrado[1], ou em um artigo mais matemático[2].

Ela gera dados aleatórios até mesmo para textos com somente uma letra, onde, não é possível identificar este padrão, a partir dos dados criptografados. A seguir, é mostrada a imagem gerada por 59.160 a's:

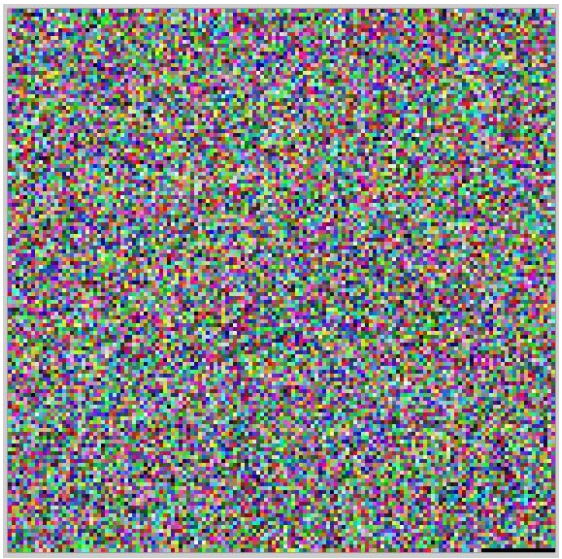

Figura 17: Imagem gerada a partir da codificação de 59.160 a's seguidos em AES..

Pelo histograma a seguir, é possível ver uma distribuição aleatória.

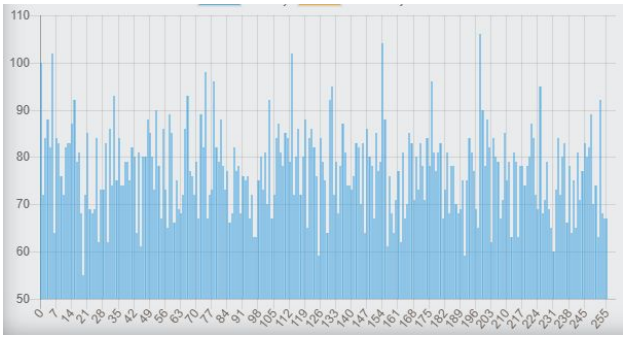

Figura 18: Histograma da componente R, da imagem representada pela figura 17.

A imagem a seguir foi gerada para os mesmo 5 parágrafos do Lorem Ipsum, usado na imagem correspondente a figura 15.

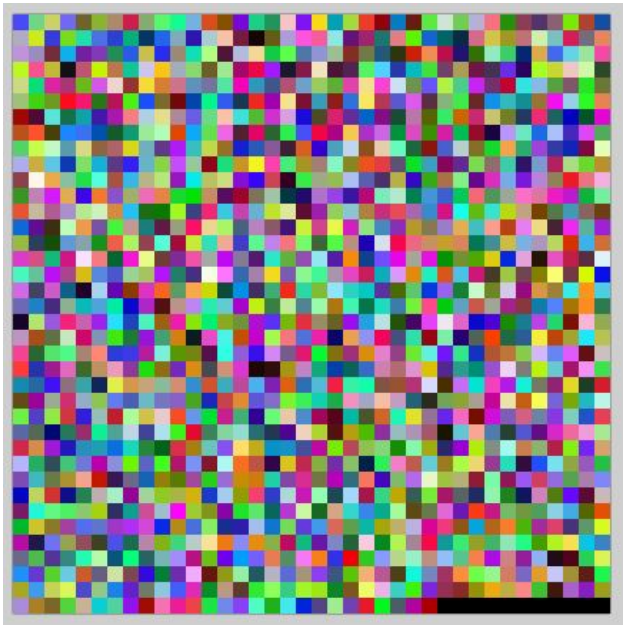

Figura 19: Imagem gerada pela codificação de um texto aleatório em AES.

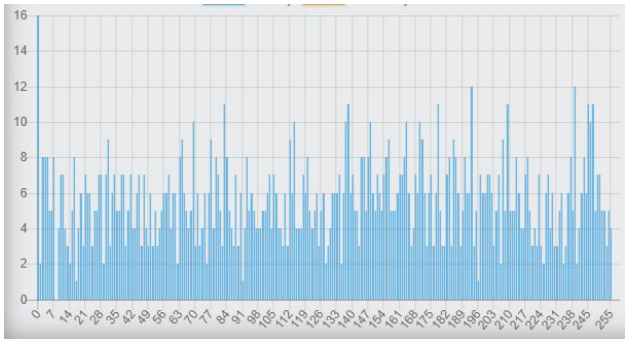

Figura 20: Histograma da componente R, da imagem representada pela figura 19.

Nessa implementação adicionaremos mais uma forma de segurança, a adição de senha, com o objetivo de cada usuário poder proteger seus dados de outros usuários que possuem acesso ao decodificador.

No AES, a chave de criptografia usada tem 32 bytes, quando o usuário escolhe uma senha, o MD5 desta senha, é usado como chave para codificar e decodificar a imagem.

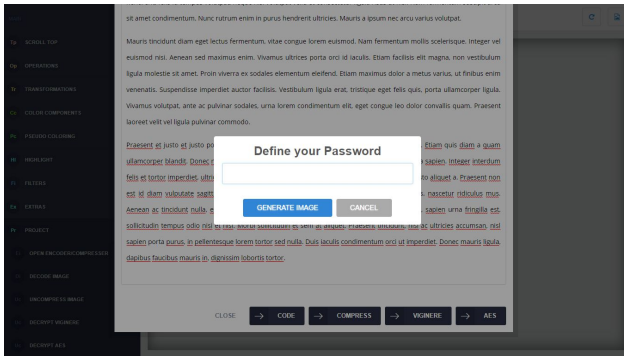

Figura 21: Visão das escolha de senha do usuário durante a codificação em AES.

Ao início da criptografia, 3 caracteres são adicionados ao final do texto inicial recebido, ao decodificar com uma chave fornecida pelo usuário, o programa testa se o resultado possui ao final do texto, os 3 caracteres, e se tiver, mostra os resultados na tela.

3 Resultados e Conclusões

A aplicação desenvolvida para este artigo, tem o intuito de fornecer ao usuário, uma interface para 1: digitar e formatar o texto, inserir imagens, links, headings, tabelas e vídeos. A interface também permite o usuário alterar o código HTML gerado pela sua formatação, como ilustrado na imagem a seguir:

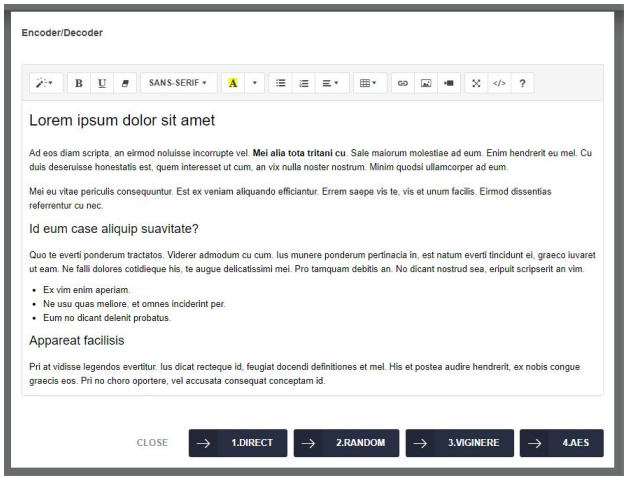

Figura 22: Interface de formatação da aplicação.

Após o usuário formatar o texto, ele poderá gerar a imagem codificada, escolhendo qualquer uma das abordagens, no menu presente na parte inferior da interface. A imagem a seguir, mostra um texto aleatório codificado nas 4 abordagens, em ordem:

Figura 23: Texto aleatório codificado usando as 4 abordagens.

Para ler o texto novamente, pode-se selecionar a imagem, clicando nela, e então ir até a opção do menu para decodificá-la, como mostrado na imagem a seguir:

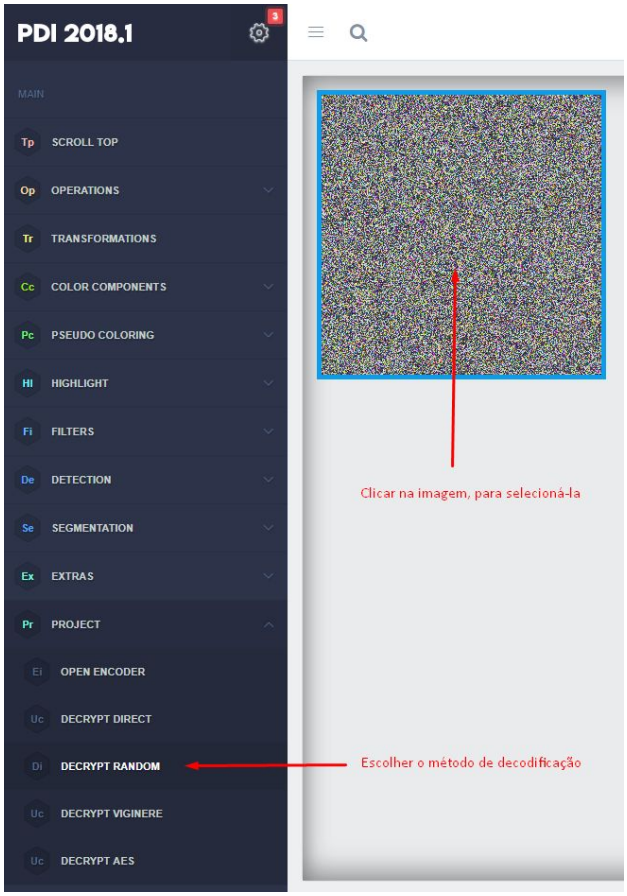

Figura 24: Decodificando imagem.

Caso a decodificação tenha sido realizada com sucesso, a tela do codificar irá abrir com o texto, e uma opção para abrir o editor, com o texto gerado, e outra para fechar a janela, como ilustrado na imagem a seguir:

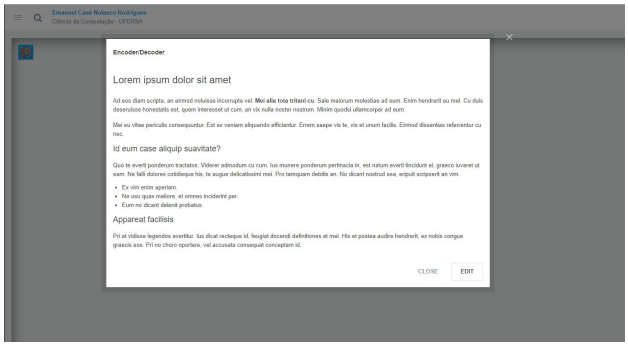

Figura 25: Texto aleatório decodificado usando o decodificador AES.

Caso a codificação falhe, o usuário, receberá uma mensagem dizendo que não foi possível decodificar aquela imagem, como ilustrado na imagem a seguir.

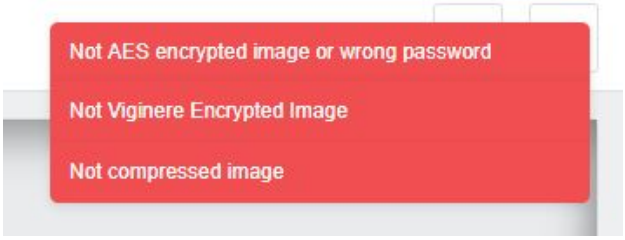

Figura 25: Exemplos de erros recebidos em decodificações falhas.

Ao fim, o processo da codificação/decodificação, segue o seguinte fluxo:

Figura 26: Fluxo de codificação/decodificação usando a aplicação.

Como mostrado, cada uma das abordagens, possui características que são inerentes ao método usado, e tangem à diversos fatores como: complexidade, segurança e velocidade. A tabela a seguir, irá mostrar um pequeno comparativo entre todas as abordagens apresentadas neste artigo:

Nome	Classificação	Compressão	Complexidade	Segurança	Velocidade
Direto	Codificação	Zero	Baixa	Baixa	Rápida
Aleatório	Codificação	Negativa	Média	Média	Lenta
Viginère	Criptografia	Zero/Positiva	Média	Alta	Rápida
AES	Criptografia	Zero	Alta	Muita Alta	Média

Tabela 1: Comparativo entre as abordagen apresentadas.

Referências

[1] Moserware
A Stick Figure Guide to the Advanced Encryption Standard (AES).
http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html, Sep. 2009 .

[2] Federal Information Processing Standards
Announcing the ADVANCED ENCRYPTION STANDARD (AES)
https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf, Nov. 2001.

[3] Vigenère cipher
https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher

[4] University of Wisconsin–Madison.
ALGEBRAIC CRYPTANALYSIS OF AES: AN OVERVIEW
http://www.math.wisc.edu/~boston/nover.pdf.