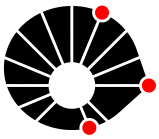
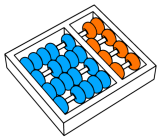




Estudo Comparativo de Técnicas de Criptografia Visual

André Vila Nova Wagner da Costa *Breno Nunes Tavares*
Eduardo Carvalheira Teixeira de Aguiar *Hélio Pedrini*

Relatório Técnico - IC-PFG-23-32
Projeto Final de Graduação
2023 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Estudo Comparativo de Técnicas de Criptografia Visual

André Vila Nova Wagner da Costa* Breno Nunes Tavares†
Eduardo Carvalheira Teixeira de Aguiar‡ Hélio Pedrini§

Resumo

Este trabalho busca reproduzir e comparar técnicas de criptografia visual. O estudo procurou entender as peculiaridades de cada metodologia implementada, o que cada técnica proposta na literatura oferece, além de suas vantagens e desvantagens. Para tal tarefa, foram realizados experimentos utilizando metodologias aritméticas, como o uso da operação XOR e da aritmética modular, além de algumas de suas técnicas subvariantes, que visam melhorar ainda mais seus resultados e/ou desempenhos. A eficácia e a eficiência dos algoritmos foram medidas utilizando métricas como PSNR (*Peak Signal-to-Noise Ratio*), SSIM (*Structural Similarity Index Measure*) e tempo de execução do código.

1 Introdução

Historicamente, a comunicação sempre desempenhou um papel fundamental no desenvolvimento sociocultural e na disseminação de conhecimento. Em contextos casuais, como nas mais básicas conversas do dia-a-dia, ou em contextos mais complexos, como na troca de ideias ou na transmissão de alertas sobre perigos iminentes, a comunicação está presente e é um ponto focal de inúmeras interações humanas, podendo ser realizada através de sinais visuais ou sonoros.

Entretanto, percebeu-se que, em contextos sensíveis, tornou-se crucial que apenas destinatários específicos fossem capazes de entender o significado das mensagens transmitidas. Tal necessidade resultou no surgimento e aprimoramento de técnicas criptográficas, concebidas para proteger informações confidenciais e impedir que partes indesejadas as compreendessem.

Uma dessas técnicas, recentemente em destaque, é a criptografia visual, técnica proposta inicialmente por Naor e Shamir na EUROCRYPT'1994 [6], uma abordagem que se mostra promissora na proteção de informações sensíveis. Ela se baseia na ocultação de informações visuais de forma que a imagem original seja escondida em duas ou mais *shares*, geradas de

*Instituto de Computação, Universidade Estadual de Campinas, 13083-852 Campinas, SP.
†Instituto de Computação, Universidade Estadual de Campinas, 13083-852 Campinas, SP.
‡Instituto de Computação, Universidade Estadual de Campinas, 13083-852 Campinas, SP.
§Instituto de Computação, Universidade Estadual de Campinas, 13083-852 Campinas, SP.

acordo com um algoritmo de criptografia, enquanto, teoricamente, apenas a pessoa que possui todas as *shares* e o método de decodificar a imagem pode recuperar de forma satisfatória a imagem original.

Este trabalho propõe uma análise aprofundada dessa técnica emergente, explorando de forma a complementar suas bases teóricas em conjunto com os resultados práticos que obtemos em experimentos realizados durante a duração do projeto.

Ao final do estudo da criptografia visual, pretende-se não apenas explorar suas bases conceituais, mas também avaliar sua eficácia como ferramenta para a comunicação segura em um mundo cada vez mais digital e interconectado, de modo que seja possível realizarmos, de forma comparativa, uma análise entre os distintos métodos e variáveis que esse campo apresenta, elencando as situações ideais, vantagens e desvantagens de cada um deles.

Nossa pesquisa busca contribuir na compreensão e aplicação dessa tecnologia inovadora, analisando seu potencial impacto na proteção da privacidade e na segurança das informações na era contemporânea, em um mundo cada vez mais digital e global, onde a privacidade é extremamente valorizada.

2 Objetivos

Este estudo visa analisar e comparar diversas técnicas de implementação da criptografia visual, concentrando-se na avaliação da qualidade da imagem reconstruída após o processo de decodificação.

Para isso, serão empregadas métricas que julgamos adequadas de acordo com o escopo do projeto, além de testes de tempo e performance, em que se pretende investigar o tempo necessário para os procedimentos de codificação e decodificação de cada técnica estudada.

3 Metodologia

Neste contexto, exploramos uma variedade de metodologias de criptografia visual, cada uma com suas peculiaridades, limitações, vantagens e desvantagens. A seguir, foi realizada uma descrição do funcionamento dos métodos, destrinchando-se o funcionamento dos algoritmos de codificação e decodificação utilizados em cada um deles.

3.1 Pixel Expansion

O método de criptografia visual denominado *Pixel Expansion*, proposto por Naor e Shamir [6], é o mais simples e limitado entre os algoritmos que visitamos e que iremos cobrir, visto que, foi uma das primeiras metodologias propostas para realizar-se a criptografia visual. No entanto, seu entendimento nos oferece uma base didática fundamental para o entendimento da criptografia visual como um todo e dos métodos mais avançados que foram propostos posteriormente.

Primeiramente, definimos uma imagem I como um conjunto de pixeis, de modo que ela tenha uma altura m e uma largura n. Ao iniciarmos o método, também é necessário definir um fator F que irá multiplicar as dimensões da imagem original.

Após a análise de diversas implementações disponíveis na literatura, concluímos que o fator mais comum para a multiplicação do tamanho da imagem é 2, tanto horizontalmente quanto verticalmente, visto que, em geral, o aumento da imagem não é desejado, mas se mostra necessário para que essa técnica funcione. Desse modo, no algoritmo analisado, a imagem final J terá dimensões $2m \times 2n$, devido ao fator $F = 2$.

Após a implementação desses parâmetros, ainda é preciso definir que padrões utilizaremos para os pixeis que irão compor a imagem final, ou seja, é necessário definir uma lista predeterminada de padrões com dimensões $F \times F$ que será utilizada pelo algoritmo.

- ENCRYPT (Pixel a pixel) [12]:

    1. Inicialmente, é necessário gerar as *shares* que serão utilizadas. Elas são geradas inteiramente brancas e possuem as dimensões $2m \times 2n$, assim como a imagem resultante;
    2. Primeiramente, é necessário selecionar um dos seis padrões disponíveis de forma aleatória. Esse padrão $P1$ será utilizado na *share* 1, e corresponderá ao pixel P que está sendo criptografado;
    3. Caso o pixel da P corresponda a um pixel BRANCO, selecionar o mesmo padrão $P1$ e inseri-lo na posição correspondente na *share* 2, de modo que $P1 = P2$. Nesse método, não há uma forma de obtermos um pixel completamente branco, ou seja, esse caso será uma fonte de erro para a imagem final;
    4. Caso o pixel P corresponda a um pixel PRETO, selecionar para a *share* 2 um padrão $P2$ de modo que $P2$ corresponda ao valor complementar de $P1$, fazendo com que a sobreposição de $P1$ e $P2$ resultem em um pixel completamente preto, equivalente ao original.

- DECRYPT [12]:

    1. Para realizar a decodificação, basta realizarmos a sobreposição das *shares*. No código referenciado por este trabalho, foi utilizado o "&" (operador AND) entre as imagens, finalizando a decodificação.

Os seis padrões utilizados no código empregado para para realizar os testes estão ilustrados na Figura 1.

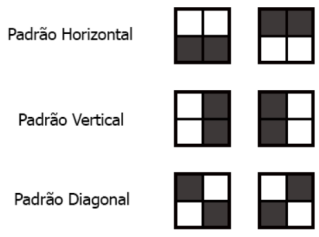


Figura 1: Padrões utilizados no *Pixel Expansion*.

3.2 Algoritmo XOR

Este método propõe o uso da operação lógica de "ou exclusivo" (XOR), em que 1 é retornado caso as entradas sejam diferentes e 0 caso sejam iguais. Por serem pixeis em imagens, os valores de entrada da operação oscilam entre 0 e 255, logo, o XOR realiza a decomposição do número para binário e a comparação dos bits dos valores passados. Por exemplo, se houver a comparação entre 200 (0b11001000) e 100 (0b1100100), obtemos 172 (0b10101100).

Para o funcionamento do método, é necessária a seleção de um número arbitrário N de *shares*, com tamanho idêntico à imagem que será encriptada. As *shares* podem ser geradas aleatoriamente ou corresponderem a imagens pré-definidas, portanto, para que ocorra a encriptação, basta iterar por esta lista de *shares*, realizando, inicialmente, a operação de XOR entre a primeira *share* e a imagem original, gerando uma nova imagem.

Em seguida, basta utilizar o resultado do XOR anterior no lugar da imagem original, obtendo a imagem encriptada. Para realizar a decodificação, é necessário repetir o processo de iteração entre as *shares* e a imagem resultante, utilizando a mesma operação lógica.

- ENCRYPT [13]:

    1. Criar uma lista de imagens que servirão como *shares*, podendo ser geradas aleatoriamente ou não;
    2. Indexar a imagem original no final da lista;
    3. Iterar pela lista das *shares* até a penúltima posição;
    4. Realizar a operação de XOR entre a última imagem da lista e a *share* atual;
    5. Sobrescrever o resultado na última posição da lista.

- DECRYPT [13]:

    1. Iterar pela lista das *shares* até a penúltima posição;
    2. Realizar a operação de XOR entre a última imagem da lista e a *share* atual;
    3. Sobrescrever o resultado na última posição da lista;
    4. A última imagem da lista será a imagem decriptada.

3.2.1 Meaningful Shares

O método XOR com *shares* significativas, ou *meaningful shares*, consiste em utilizar o método proposto anteriormente, entretanto, com uma diferença fundamental: neste caso, é possível fornecer ao programa as imagens que serão utilizadas como bases para gerar as *shares*, denominadas *covers*.

Desse modo, o algoritmo é adaptado para que ele receba, além da imagem secreta, N imagens que serão utilizadas como *covers*. O algoritmo será responsável por alterar, da melhor forma possível, as imagens *covers* que foram fornecidas a ele, de modo que o erro seja distribuído nas *shares*, e não na imagem final.

- ENCRYPT [7]:

1. Fornecer para o método de criptografia o valor inicial, V_i, do pixel que desejamos criptografar (de 0 a 255, para imagens em escala de cinza), uma lista com os *covers* selecionados e as coordenadas (m, n) em que o pixel se encontra;
2. Em seguida, selecionar um valor aleatório P entre 0 e 255 e um *cover* aleatório C proveniente da lista de *covers*;
3. Definir o valor do pixel de saída V_o como o valor inicial V_i fornecido para a função. Em seguida, percorrer em *loop* todos os *covers*, com exceção de C, aplicando a operação de XOR entre V_o e o pixel correspondente às linhas e colunas fornecidas em cada *cover*;
4. Criar uma lista vazia denominada OUT. Percorrer novamente a lista de *covers* em outro laço e preencher OUT com o valor de Vo, caso o *cover* seja igual a C, ou com o valor do resultado da operação de XOR entre o valor do pixel nas coordenadas (m, n) do *cover* e P, caso contrário;
5. Utilizando o vetor OUT, substituir os pixeis das coordenadas (m,n) em cada *cover* correspondente.

Assim, a criptografia das *shares* é realizada para pixeis individuais. No entanto, o código utilizado para realizar os experimentos também conta com um coeficiente β, que pode ser definido pelo usuário e varia de 0 a 1. Este valor define o percentual de pixeis que serão criptografados ou camuflados nas *shares* resultantes. Isso indica que, para cada pixel, antes que sua criptografia seja realizada, é gerado um número aleatório entre 0 e 1; caso esse número seja menor que β, o passo a passo acima é executado, encriptando o pixel. Caso contrário, o pixel é "camuflado", conforme o algoritmo descrito a seguir.

- PIXEL CAMOUFLAGE [7]:

1. Selecionar um pixel S, correspondente ao pixel na imagem secreta original nas coordenadas (m, n);
2. Percorrer os *covers* e realizar a operação XOR entre S e cada pixel P, nas coordenadas (m, n) em seu *cover* correspondente, salvando o resultado em S;
3. Encontrar o *cover* que possui a menor diferença absoluta entre o valor de P e a operação XOR entre P e o resultado S, obtido no passo anterior;
4. Esse *cover* C é o único que terá o valor de seu pixel alterado para o resultado da operação de XOR entre o valor original e S. Quanto aos outros *covers*, o valor do pixel permanecerá o mesmo e apenas será copiado para suas respectivas *shares*.

Por fim, para o processo de decodificação, realiza-se os mesmos passos do método XOR.

- DECRYPT [7]:

1. Iterar pela lista das *shares* até a penúltima posição;
2. Realizar a operação de XOR entre a última imagem da lista e a *share* atual;
3. Sobrescrever o resultado na última posição da lista;
4. A última imagem da lista será a imagem decriptada.

3.3 Aritmética Modular

A metodologia que utiliza aritmética modular se aproxima bastante da solução que utiliza a operação lógica XOR. Assim, aplica-se a mesma lógica de separação em um número arbitrário N de *shares*, aleatórias ou não, de tamanho idêntico à imagem que será encriptada.

A diferença encontra-se nas iterações da encriptação e decodificação, em que, ao invés de aplicar-se o XOR, utiliza-se a operação matemática de módulo.

- ENCRYPT [11]:

    1. Criar uma lista de imagens que servirão como *shares*, podendo ser geradas aleatoriamente ou não;
    2. Indexar a imagem original no final da lista;
    3. Iterar pela lista das *shares* até a penúltima posição;
    4. Somar os valores dos pixeis entre a última imagem da lista e a *share* atual;
    5. Encontrar o resto da divisão do valor obtido por 256 (operação de módulo);
    6. Sobrescrever o resultado na última posição da lista.

- DECRYPT [11]:

    1. Iterar pela lista das *shares* até a penúltima posição;
    2. Subtrair os valores dos pixeis entre a última imagem da lista e a *share* atual;
    3. Somar 256 ao resultado;
    4. Encontrar o resto da divisão do valor obtido por 256 (operação de módulo);
    5. Sobrescrever o resultado na última posição da lista;
    6. A última imagem da lista será a imagem decriptada.

3.4 Criptografia AES

A metodologia de criptografia AES (*Advanced Encryption Standard*) foi criada pelo instituto nacional de padrões e tecnologia dos EUA (NIST) em 2001 [14]. O AES é um algoritmo de chave simétrica, ou seja, utiliza a mesma chave para o processo de codificação e decodificação. Além disso, é composto por um algoritmo de cifra de bloco, especificamente a rede de substituição-permutação, que, como já diz o nome, realiza diversas rodadas de substituição e, posteriormente, permutação no bloco que está sendo cifrado [2, 16].

O tamanho de bloco do algoritmo AES é fixo, sendo de 128 bits, enquanto a chave passada pode alternar entre 128, 192 e 256 bits. Seu tamanho especifica quantas rodadas de substituição e permutação irão ocorrer, 10, 12 e 14, respectivamente, durante o processo de codificação e decodificação.

Esta metodologia pode ser adaptada para ser utilizada em processos de criptografia em imagens, encriptando a imagem utilizando AES e convertendo a chave em *shares* via criptografia visual. Dessa forma, para converter a chave em *shares*, utiliza-se métodos de criptografia como o XOR visto anteriormente [5].

- ENCRYPT [8]:

  1. Definição de uma chave para a criptografia AES;
  2. Transformação da imagem em Base64;
  3. Aplicação da metodologia AES na imagem, utilizando a chave passada;
     (a) Para cifrar a informação, o tamanho de chave passado especifica o número de rodadas de transformação para ser aplicado:
        i. 10 rodadas para chaves de 128 bits;
        ii. 12 rodadas para chaves de 192 bits;
        iii. 14 rodadas para chaves de 256 bits;
     (b) As chaves das rodadas são derivadas da chave de cifragem;
     (c) Adição da chave da rodada inicial:
        i. Os bytes do estado são combinados com bytes da chave da rodada usando XOR;
     (d) Rodadas intermediárias:
        i. Utilizando uma tabela de pesquisa, cada byte é substituído por outro;
        ii. As últimas três linhas do estado são deslocadas ciclicamente um certo número de vezes;
        iii. Combinar os quatro bytes em cada coluna de estado;
        iv. Os bytes do estado são combinados com bytes da chave da rodada usando XOR;
     (e) Rodada final:
        i. Utilizando uma tabela de pesquisa, cada byte é substituído por outro;
        ii. As últimas três linhas do estado são deslocadas ciclicamente um certo número de vezes;
        iii. Os bytes do estado são combinados com bytes da chave da rodada usando XOR;
  4. A imagem ficará encriptada em formato de cifra;
  5. Criar uma *share* em branco de tamanho $(255, h, 1)$, sendo h o tamanho da chave;
  6. Preencher os pixeis desta imagem utilizando a transformação de caractere em Unicode;
  7. Criação de 2 *shares* R e P de tamanho $(255, h, 3)$ aleatórias;
  8. Aplicação de criptografia visual entre a *share* gerada pela chave e as restantes, no nosso caso aplicamos XOR.

- DECRYPT [8]:

  1. Aplicação do método de decodificação da abordagem escolhida nas *shares* R e P;
  2. Transformar os pixeis da imagem resultante de *Unicode* para caractere;

3. Aplicação do método de decodificação do algoritmo AES;
   (a) Utilizando a imagem cifrada e a chave que foi decodificada;
   (b) Realizar o processo inverso da encriptação;
4. Retornar os bytes de Base64 ao estado inicial, obtendo a imagem original.

3.5 Bit Level Decomposition

Este método, uma evolução do *Pixel Expansion* desenvolvido por Naor e Shamir [17], baseia-se na operação lógica AND. Nessa abordagem, o valor 1 é retornado quando ambas as entradas são 1, e 0 caso contrário. Essa técnica é projetada para lidar com imagens em escala de cinza, envolvendo a conversão para binário, aplicação do método de criptografia e, finalmente, reconversão para a escala de cinza original.

Para executar o código, a imagem de entrada I com altura m e largura n é processada. Ao contrário do *Pixel Expansion*, que utiliza um fator de multiplicação, este método emprega, por padrão, o fator 2. Em outras palavras, duplica ambas as dimensões da imagem, resultando em uma imagem final com altura $2m$ e largura $2n$.

A conversão de escala de cinza para binário é realizada por meio de uma matriz tridimensional T, com altura m, largura n e 8 canais de profundidade. Cada coordenada do primeiro canal da matriz recebe o bit menos significativo da imagem I em escala de cinza. Este bit é então descartado, e o processo é repetido para os outros sete canais, criando assim uma representação binária da imagem I, conhecida como *Bit Level Decomposition*. A conversão inversa de binário para escala de cinza é feita calculando os valores de cinza de cada pixel com base na sua representação binária de 8 bits, variando de 0 a 255.

O método utiliza os mesmos seis padrões de quadros 2×2 de pixeis brancos e pretos do *Pixel Expansion*, conforme ilustrado na Figura 1. No entanto, ao contrário de selecionar apenas um par dos modelos apresentados, neste sistema, um padrão é aleatoriamente escolhido oito vezes, uma para cada canal da matriz T. Isso garante que as imagens intermediárias produzidas dificilmente sejam idênticas, mesmo ao executar o programa várias vezes com a mesma imagem I de entrada.

- ENCRYPT [9]:

1. Converter a imagem em escala de cinza para binário;
2. Criar duas imagens binárias intermediárias, $S1$ e $S2$, com o dobro de altura e largura da imagem original e com 8 canais de profundidade;
3. Selecionar um dos padrões gerados de forma aleatória. Esse padrão $P1$ será utilizado no primeiro canal vazio da $S1$;
4. Caso o pixel da imagem que está sendo codificado corresponda a um pixel BRANCO, selecionar o mesmo padrão $P1$ e colocá-lo no canal correspondente na $S2$, de modo que $P1 = P2$. Nesse método, assim como no *Pixel Expansion*, não há uma forma de obtermos um pixel completamente branco, ou seja, esse caso será uma fonte de erro para a imagem final;

5. Caso o pixel da imagem que está sendo codificado corresponda a um pixel PRETO, selecionar para a $S2$ um padrão $P2$ de modo que $P2$ corresponda ao valor complementar de $P1$, resultando em um pixel completamente preto, equivalente ao original;
6. Repetir sete vezes os passos 3, 4 e 5, de forma a preencher cada canal de $S1$ e $S2$;
7. Converter $S1$ e $S2$ para escalas de cinza.

- DECRYPT [9]:

1. Converter $S1$ e $S2$ para binário;
2. Criar uma imagem A, com dimensões iguais às das $S1$ e $S2$ (incluindo os 8 canais de profundidade);
3. Realizar sobreposição de $S1$ em $S2$ utilizando a operação lógica AND e salvar o resultado em A;
4. Converter a imagem A de binário para escala de cinza, finalizando o processo de decodificação.

4 Resultados

Após uma extensa análise e pesquisa dos diversos métodos disponíveis na bibliografia, concluímos que para realizar uma análise comparativa adequada dos métodos seria necessário que fosse estabelecido um fluxo de trabalho, visando padronizar de uma maneira compreensível e didática os testes que realizamos para cada método.

Inicialmente, criamos um arquivo `.ipynb` no *Google Colaboratory* para cada método selecionado. Essa plataforma foi escolhida pois, primeiramente, não nos traria problemas de versionamento das bibliotecas utilizadas, o que poderia vir a acontecer caso os testes fossem realizados em máquinas diferentes. Ademais, o *Colab* [10] fornece ao usuário as mesmas especificações de RAM, disco e outros recursos computacionais em geral (Figura 2), o que facilita a replicação do experimento e diminui a variação causada por diferenças de hardware, o que nos possibilita testar o tempo de execução do programa, além dos testes mais tradicionais utilizando indicadores como o PSNR (*Peak Signal-to-Noise Ratio*).

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   34C    P8     9W /  70W |      0MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

Figura 2: Especificações de GPU do *Google Colaboratory*.

Após a replicação e alguns ajustes nos códigos que selecionamos no Github, concluímos que iríamos testar os métodos para duas imagens, sendo elas o "Baboon" e a "Lenna", ambas disponíveis na internet para teste e com o mesmo formato, 512×512 pixels, e apresentadas na Figura 3.

(a) Baboon (b) Lenna

Figura 3: Imagens utilizadas nos testes dos métodos.

Com todos esses pontos definidos, decidimos que, como forma de comparar os métodos, seria adequado separar os testes em duas categorias: testes de tempo, visando verificar a eficiência e viabilidade do algoritmo, e testes de precisão, em que métricas como o PSNR e o SSIM são utilizados para comparar a imagem original com a imagem gerada após o processo de codificação e decodificação.

A métrica de PSNR, ou *Peak Signal-to-Noise Ratio* (relação sinal-ruído de pico), visa medir a relação entre a máxima energia de um sinal e o ruído que afeta sua representação verdadeira. Desta maneira, devido a natureza desta métrica, ela é comumente utilizada para a comparação entre imagens e foi escolhida para realizar a comparação entre os métodos analisados. Além disso, seu valor normalmente é expressado em uma escala logarítmica, utilizando-se a unidade decibel, devido às amplitudes dinâmicas da maioria dos sinais.

Para o cálculo do PSNR, é necessário a formulação do erro quadrático médio [3]. Tendo duas imagens monocromáticas I e K de tamanho $M \times N$, temos que:

$$\mathrm{MSE} = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \|I(i,j) - K(i,j)\|^2$$

$$\mathrm{PSNR} = 10 \cdot \log_{10} \left(\frac{\mathrm{MAX}_I^2}{\mathrm{MSE}} \right) = 20 \cdot \log_{10} \left(\frac{\mathrm{MAX}_I}{\sqrt{\mathrm{MSE}}} \right)$$

em que MAX indica o valor máximo de sinal existente na imagem.

Já em relação ao SSIM, a medida do índice de similaridade estrutural, é uma medida que foi comumente utilizada para prever a qualidade percebida em imagens e vídeos digitais,

como em transmissões de televisão ou imagens cinematográficas, de modo que ela funcione como uma forma de medir a semelhança entre duas imagens. Esta métrica é totalmente dependente de uma referência, por isso, é necessário passar a imagem original, sem distorções para que ocorra a comparação.

A equação para calcular o SSIM [18] entre duas imagens x e y é dada por:

$$\text{SSIM}(x, y) = l(x, y)^{\alpha} \cdot c(x, y)^{\beta} \cdot s(x, y)^{\gamma}$$

em que $l(x, y)$, $c(x, y)$ e $s(x, y)$ são as medidas de similaridade de luminância, de contraste e de estrutura entre as imagens x e y, respectivamente, enquanto α, β e γ são parâmetros que controlam a importância relativa dos três componentes.

Definindo 1 como o valor para estes três parâmetros, visto que, para este trabalho, as propriedades citadas acima são igualmente importantes, podemos reescrever a fórmula desta maneira:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

em que μ_x e μ_y são as médias de intensidade dos pixeis de x e de y, σ_x e σ_y são os desvios-padrão das intensidades dos pixeis em x e y, e $\sigma_x y$ é a covariância entre x e y, respectivamente. c_1 e c_2 são constantes usadas para evitar problemas matemáticos caso os valores citados sejam muito pequenos (tendendo a zero) e são calculadas através das fórmulas:

$$c_1 = (k_1 \cdot L)^2$$
$$c_2 = (k_2 \cdot L)^2$$

em que L é a faixa dinâmica entre as imagens. Para este trabalho, k_1 é igual a 0.01 e k_2 é igual a 0.03 [4].

É importante salientar que o SSIM é aplicável somente a matrizes bidimensionais. Isso significa que essa medida pode processar apenas uma escala de cores por vez, como binária, em tons de cinza ou outras cores individuais. Portanto, no caso de imagens RGB, que são representadas por matrizes tridimensionais com uma profundidade de 3, o cálculo do SSIM é realizado separadamente para cada escala de cor (vermelho, verde e azul) da imagem. Depois, é calculada a média aritmética dos valores obtidos para fornecer uma avaliação conjunta.

$$\text{SSIM} = \frac{\text{SSIM(Red)}}{3} + \frac{\text{SSIM(Green)}}{3} + \frac{\text{SSIM(Blue)}}{3}$$

A última comparação entre as metodologias analisadas foi justamente medir o tempo em que os algoritmos demoram para encriptar e decriptar. Para isto, basta utilizar a biblioteca *datetime* da linguagem de programação Python para medir o tempo de execução do programa. Devido à volatilidade destes dados devido ao tipo de hardware utilizado e a fatores externos, foram realizadas 10 medições destes valores, extraindo-se a sua média e o desvio-padrão da mesma para evitar que estes fatores afetem os resultados, todas em um mesmo computador, a fim de evitar que a diferença entre hardwares influencie na obtenção desses dados.

Com as métricas estabelecidas, conduzimos as medições para cada técnica e as compilamos na Tabela 1. Adicionalmente, no método de *meaningful shares*, optamos por avaliar essas medidas para distintos valores de beta, reunindo-os em uma tabela específica, conforme apresentado na Tabela 2.

Tabela 1: Resultados das métricas para cada método.

Método	PSNR (Lenna)	PSNR (Baboon)	SSIM (Lena)	SSIM (Baboon)	Tempo Médio de Codificação (s)	Desvio-Padrão de Codificação (s)	Tempo Médio de Decodificação (s)	Desvio-Padrão de Decodificação (s)
Pixel Expansion	28.249	28.131	0.001	0.001	5.149	0.744	1.146	0.354
XOR	100	100	1	1	0.048	0.057	0.032	0.043
Modular Arithmetic	100	100	1	1	0.05	0.04	0.032	0.026
AES	100	100	1	1	0.761	0.061	0.705	0.102
Bit Level Decomposition	27.501	27.488	0.488	0.527	38.28	1.432	8.63	1.013

Tabela 2: Resultados das métricas para diferentes valores de β utilizando *meaningful shares*.

β	PSNR Médio	SSIM Médio	Tempo de Codificação (s)	Tempo de Decodificação (ms)
0	19.608	0.411	14.24	5.246
0.25	13.752	0.15	14.687	4.948
0.5	11.355	0.079	10.497	4.687
0.75	9.817	0.038	6.376	4.965
1	8.688	0.009	6.479	4.965

4.1 Pixel Expansion

O método conhecido como *Pixel Expansion*, embora seja um dos mais simples nas análises realizadas, desempenha um papel fundamental como base para a compreensão de métodos mais complexos. Inicialmente, é essencial destacar que essa técnica opera apenas em imagens em preto e branco, devido à seleção de bits que compõem as *shares*, que são sempre binárias. Isso ocorre devido aos padrões utilizados, que como demonstrado na figura 1, são compostos por grades 2×2 de pixeis pretos e brancos. Um efeito adicional causado pela utilização desses padrões na criptografia é o tamanho da imagem final, alterada de acordo com as dimensões da grade.

Em uma análise mais objetiva do método, em comparação com as outras opções, é notável que ele apresenta um dos maiores tempos tanto de codificação quanto de decodificação. O tempo prolongado de criptografia é resultado de uma técnica de pós-processamento denominada *Extraction*, em que após a sobreposição das *shares*, realiza-se uma operação reversa de criptografia, examinando grades 2×2 da imagem final e determinando se correspondem a pixeis pretos ou brancos na imagem original. Esse pós-processamento torna o método capaz de gerar uma imagem idêntica à original, no entanto, para efeitos de comparação de métodos, os erros apresentados na tabela são os obtidos antes de realizar o pós-processamento.

Ao analisar-se as métricas de erro, é importante evidenciar que o código realiza um redimensionamento da imagem visando facilitar a comparação da imagem original e da imagem recuperada. O baixo valor do PSNR obtido indica que houve uma perda considerável de detalhes após a execução do programa, o que é esperado considerando-se as limitações do método, por exemplo, o fato de que não é possível obtermos padrões que, após a sobreposição das *shares*, correspondam a pixeis completamente brancos. As mesmas limitações também são as responsáveis pela baixa similaridade estrutural obtida, indicada pela métrica SSIM, que apresentou valores próximos a zero para ambos os testes.

Desse modo, considerando-se as inúmeras limitações do método, tais como a baixa fidelidade entre a imagem original e a obtida, funcionamento apenas em imagens preto e brancas e o alto tempo de processamento e pós-processamento, é extremamente improvável que o método do *Pixel Expansion* possua aplicações no cotidiano em que seja necessário recuperar a imagem original sem um erro considerável. No entanto, para aplicações em que o objetivo é, por exemplo, passar uma mensagem em texto ocultada utilizando a criptografia visual, esse método pode ser considerado um candidato, visto que ele apresenta simplicidade e é funcional o suficiente para exercer essa função.

4.2 Algoritmo XOR

A abordagem XOR é uma das mais simples de ser implementada, visto que é necessário apenas aplicar a operação lógica nas *shares* para que a imagem seja encriptada e, posteriormente, decriptada. Consequentemente, isto possibilita que este método seja um dos mais rápidos para realizar ambos os processos de codificação e decodificação, como evidenciado pela tabela de resultados supracitada.

Além de ser possível aplicar esta abordagem em imagens coloridas e monocromáticas, uma vantagem desta abordagem em relação à anterior, é que ela é capaz de gerar um número N de *shares*, assim, possibilitando uma criptografia mais segura. Além disso, com estas *shares* sendo geradas aleatoriamente, a utilização parcial das *shares* no processo de decodificação não revelam informação alguma sobre a imagem original, mantendo um padrão indecifrável, ou seja, somente com a utilização de todas as *shares* geradas que é possível recuperar a imagem original.

4.2.1 Meaningful Shares

O algoritmo XOR com a funcionalidade de *Shares* Significativas implementada apresenta diversas peculiaridades, de modo que seu fluxo de testes foi feito de forma distinta dos outros métodos para evidenciar suas particularidades. Primeiramente, como forma de analisar o funcionamento do método, foi necessário escolher um número N de *covers* que iríamos utilizar, responsáveis por servir de base para as N *Shares* Significativas que iríamos obter. Após testes iniciais, concluímos que $N = 5$ *covers* seria um número suficiente para realizar o teste, visto que, para $N = 2$ *covers*, não são produzidos resultados didáticos e relevantes para nossa pesquisa.

Em seguida, seria necessário definir para quantos valores de β testaríamos o algoritmo. Foi determinado que β seria variado entre os valores 0, 0.25, 0.5, 0.75, 1, visando verificar

sua influência nas métricas e nas imagens geradas.

Após a realização dos testes, como primeira análise, posicionamos as *shares* obtidas para cada coeficiente horizontalmente, obtendo os resultados apresentados nas Figuras 4 a 8.

Figura 4: *Shares* obtidas com $\beta = 0$.

Figura 5: *Shares* obtidas com $\beta = 0.25$.

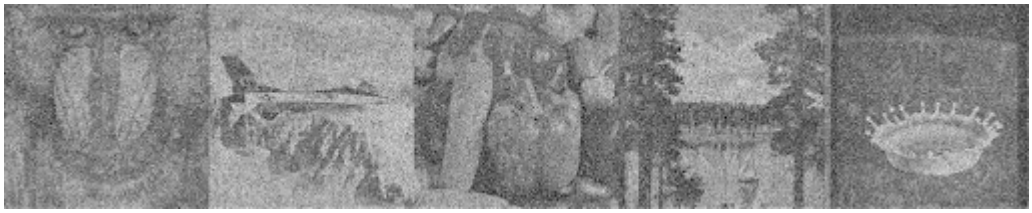


Figura 6: *Shares* obtidas com $\beta = 0.5$.

Figura 7: *Shares* obtidas com $\beta = 0.75$

A partir dessa primeira análise, já é possível notar como o coeficiente β influencia a geração das *shares*. Basicamente, para um β próximo a 0, o algoritmo de camuflagem dos pixeis será utilizado mais vezes do que o algoritmo responsável por criptografar os pixeis, dando origem à *shares* relativamente semelhantes aos *covers* que as geraram. Para β

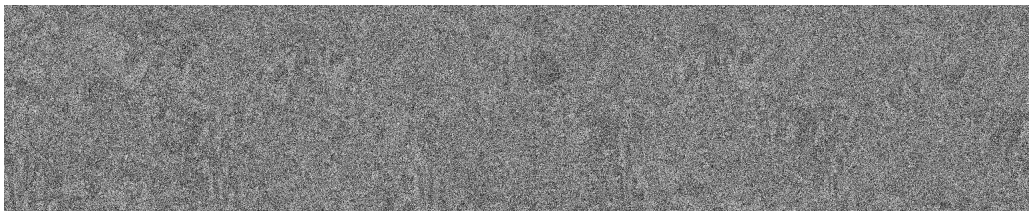


Figura 8: *Shares* obtidas com $\beta = 1.0$.

próximo a 1, o contrário também é válido. Em suma, o coeficiente βserve como uma forma do usuário definir o quão significativo ou não serão suas *shares*.

No entanto, outra característica do método pode ser notada analisando-se as imagens mostradas nas Figuras 9 a 13, em que é representado o processo de decodificação gradual utilizando-se o XOR, como descrito no algoritmo na seção de metodologia.

Figura 9: Decodificação passo-a-passo para $\beta = 0$.

Figura 10: Decodificação passo-a-passo para $\beta = 0.25$

Figura 11: Decodificação passo-a-passo para $\beta = 0.5$.

Através da análise comparativa do processo de decodificação para β distintos, fica claro que, para um valor de β próximo a 0, é possível obter uma imagem semelhante a imagem secreta com apenas uma parcela das *shares* necessárias para realizar a decodificação

Figura 12: Decodificação passo-a-passo para $\beta = 0.75$.

Figura 13: Decodificação passo-a-passo para $\beta = 1$.

completa. No caso de $\beta = 0$, é possível notar que a imagem secreta já aparece de forma consideravelmente legível na penúltima transformação, enquanto para β mais próximos a 1, esse fenômeno é notado de forma bem menos marcante.

Desse modo, é possível concluir que, além de ser responsável por determinar o quão significativas será uma *share*, também pode-se afirmar que β é responsável pela segurança do método. A imagem final nunca será completamente recuperada sem que o usuário possua todas as *shares* para realizar a decodificação, no entanto, dependendo da aplicação, é possível que o efeito da recuperação parcial de uma imagem utilizando-se apenas uma fração das *shares* seja indesejado, comprometendo a segurança do método e o tornando impraticável para determinadas situações.

Partindo-se agora para uma análise técnica das métricas e dos resultados obtidos após a realização dos experimentos, é possível notar que, em relação ao tempo gasto para realizar a criptografia das imagens, o coeficiente β novamente interfere diretamente na quantidade de tempo gasto no processo. Isso ocorre pois o algoritmo de criptografar os pixeis é mais rápido que o algoritmo de camuflar os pixeis, portanto, para casos em que o primeiro é priorizado, o tempo é menor. Em relação ao tempo de decodificação, pode-se considerar que ele é da magnitude de milissegundos, e a variação de β não o altera de forma considerável.

Em relação às métricas de erro obtidas, é notável que, novamente, β é fundamental no entendimento do erro obtido. É importante ressaltar que, diferentemente dos outros métodos, o XOR com *Shares* Significativas distribui o erro entre as *shares*, ou seja, o cálculo do erro é feito entre os *covers* e as *shares* obtidas. Para β mais baixos, como já foi discutido, as *shares* são mais significativas, o que significa que uma menor quantidade de pixeis foi alterada, apresentando um PSNR e um SSIM maior do que quando valores de β mais altos são utilizados.

4.3 Aritmética Modular

Como a abordagem do uso de aritmética modular se aproxima com a solução de XOR, os resultados também são similares, obtendo tempos de codificação e decodificação semelhantes. Ademais, as propriedades de ser capaz de aplicar a metodologia em imagens coloridas e monocromáticas se mantêm, porém, a aplicação em imagens binárias é prejudicada neste caso.

Em relação ao número de *shares* geradas, a solução mantêm a propriedade observada na metodologia que utiliza a operação lógica XOR. Assim como a característica de somente se recuperar a imagem criptografada através do uso de todas as *shares* geradas, inviabilizando o uso de uma parcela das *shares* para obter informações sobre a imagem original.

4.4 Criptografia AES

A abordagem do AES também possui diversas peculiaridades, provenientes do fato que ela corresponde a uma adaptação de um código de criptografia que utiliza uma chave para realizar as operações. Apesar disso, como a imagem é transformada em Base64 para realizar a codificação, é possível aplicar este método em imagens coloridas, binárias ou em escalas de cinza.

A imagem fica encriptada em um cifra, em que, para recuperá-la, é necessário utilizar a chave passada. Por sua vez, a chave é separada em duas *shares*, "R" e "P" (Figura 14). No entanto, este número pode variar de acordo com a metodologia utilizada. Como neste caso foi utilizado o XOR, podem ser utilizadas de 2 a 8 *shares*. A dimensão das imagens oriundas da chave correspondem a $(255, h)$, em que h é o tamanho da chave, e portanto, quanto maior a chave passada, maior será a imagem, e consequentemente, maior o consumo de memória e tempo para realizar os procedimentos.

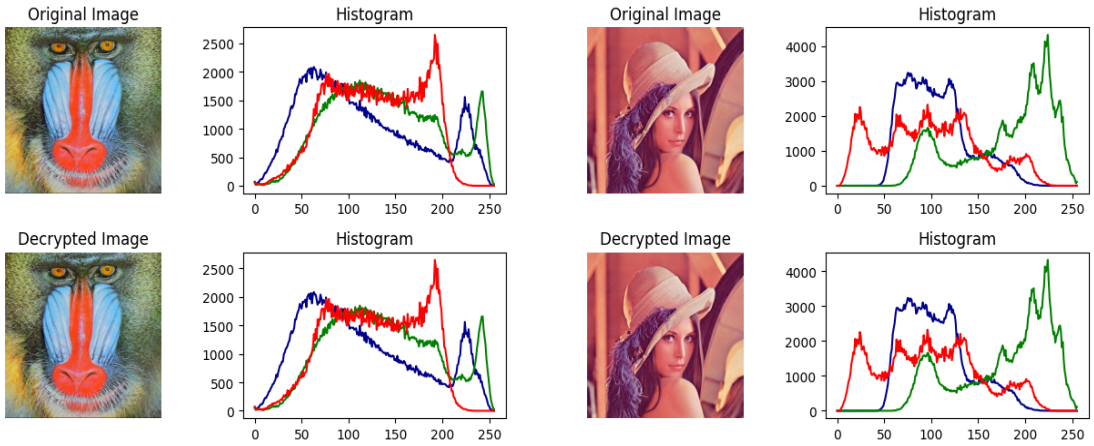
(a) *Share* "R" (b) *Share* "P"

Figura 14: Separação da chave de criptografia em *shares* utilizando criptografia AES.

Analisando-se as métricas deste método, percebe-se que não ocorre divergências entre as imagens finais e iniciais, visto que o PSNR é 100 e o SSIM é 1, indicando que as imagens são idênticas. Além disso, analisando-se o histograma das camadas RGB das imagens iniciais e finais (Figura 15), evidencia-se que de fato não houve presença de ruído após o processo de codificação e decodificação.

Já em relação a métrica temporal, nota-se que sua performance não é a melhor das apresentadas, sendo mais lenta que os métodos aritméticos como o XOR e a operação modular, devido às suas etapas complexas de codificação e decodificação descritas anteriormente. Porém, o valor é consideravelmente baixo, demorando aproximadamente 1,5 segundos para encriptar e decriptar uma imagem, isto pois, em sua concepção, o algoritmo AES visava a alta velocidade e o baixo consumo de RAM [15].

Ademais, como visto pelas *shares* e como esta metodologia é utilizada para encriptar e



(a) Baboon (b) Lenna

Figura 15: Histograma das camadas RGB das imagens iniciais e finais utilizando criptografia AES.

decriptar documentos secretos do governo americano [1], a possível recuperação da imagem original por fontes externas é quase nula.

4.5 Bit Level Decomposition

Este método se assemelha significativamente ao *Pixel Expansion*, representando, essencialmente, uma evolução desse mesmo conceito. Embora seja capaz de processar imagens em escala de cinza, ele ainda realiza o processo de criptografia com imagens binárias, exigindo conversões entre escalas de cinza e binária, e vice-versa.

Assim como o *Pixel Expansion*, esse sistema gera apenas duas *shares* e uma imagem final recuperada com dimensões duplicadas em relação à original, embora seja possível redimensioná-la para as dimensões originais. Essa abordagem aumenta a segurança do código, reduzindo significativamente as chances de obtenção de *shares* idênticas em execuções repetidas com a mesma imagem de entrada.

Ao executar o código, observa-se que as *shares* resultantes carecem de significado visual humano, assemelhando-se a imagens geradas aleatoriamente e sem uma conexão clara entre si. A imagem final, obtida pela sobreposição dessas *shares*, apresenta semelhanças evidentes com a original, facilitando seu reconhecimento. No entanto, algumas falhas são perceptíveis, sendo a principal a diferença de luminosidade, pois a imagem recuperada é consideravelmente mais escura que a original. Embora a perda de detalhes não seja imediatamente notável, é uma falha a ser considerada, pois detalhes mais sutis da figura podem ser comprometidos durante o processo de criptografia.

Ao analisar os valores das métricas utilizadas, torna-se evidente que este método é inferior aos demais. Ele não apenas obteve o menor valor de PSNR, revelando a perda de detalhes das imagens originais, como ainda apresentou o maior tempo de execução, com uma diferença substancial em relação às demais técnicas, tanto na decodificação quanto

na decodificação. Isso sugere que este método não seria prático em situações cotidianas, como a necessidade de criptografia visual em aplicativos para dispositivos móveis, como *smartphones*.

Além disso, essa técnica registrou um baixo valor de SSIM, aproximando-se de 0.5, evidenciando a baixa similaridade estrutural entre as imagens. Embora esse valor seja consideravelmente superior ao obtido pelo *Pixel Expansion*, ainda está longe do ideal, evidenciando que esses métodos são os únicos que não apresentaram imagens com uma alta similaridade estrutural.

5 Conclusões

A análise detalhada dos métodos de criptografia visual revela uma gama diversificada de abordagens com diferentes níveis de eficácia, aplicabilidade e segurança.

Primeiramente, o método do *Pixel Expansion*, embora fundamental para a compreensão de técnicas mais complexas, exibe limitações significativas, como seu funcionamento exclusivo em imagens binárias, além de um alto tempo de processamento e pós-processamento. Sua utilidade é restrita a situações específicas, como ocultar mensagens em texto utilizando criptografia visual. Porém, há o fator da facilidade de decriptar, somente realizando a sobreposição das *shares*.

Já o algoritmo XOR, se destaca pela simplicidade de implementação e velocidade tanto na codificação quanto na decodificação. Sua capacidade de gerar um número variável de *shares* contribui para uma criptografia mais segura, garantindo que a informação original permaneça oculta sem a utilização de todas as *shares*. Este parece ser o algoritmo mais promissor, devido aos resultados e simplicidade.

No caso do XOR com *Shares* Significativas, a influência do coeficiente β é notável não apenas na geração das *shares*, mas também na segurança do método. Valores mais baixos de β resultam em *shares* mais significativas, porém com a possibilidade de uma recuperação parcial da imagem secreta. Essa abordagem pode ser interessante caso queira camuflar as *shares* com imagens personalizadas, deixando menos claras para possíveis atacantes que as imagens são chaves para a decodificação.

A abordagem utilizando aritmética modular compartilha semelhanças com o XOR, porém, sua aplicação em imagens binárias é limitada. A propriedade de necessitar de todas as *shares* para recuperar a imagem original é mantida, preservando-se a segurança do método.

Por outro lado, o AES, adaptado para criptografar imagens, oferece uma camada adicional de segurança ao dividir a chave em *shares*. Ele apresenta imagens idênticas após a decodificação, mas seu tempo de processamento é mais lento em comparação com os métodos mais simples. Este algoritmo pode ser utilizado para o caso em que a segurança seja o foco do usuário.

O método de Bit Level Decomposition, embora se assemelhe ao *Pixel Expansion* em sua lógica básica, revela-se o menos eficiente, mostrando perda de detalhes considerável, tempos prolongados de execução e baixa similaridade estrutural entre as imagens. Por estes motivos, não foi possível encontrar cenários em que seu uso seria recomendado.

Cada método possui suas vantagens e desvantagens, sendo crucial selecionar o mais apropriado de acordo com as necessidades específicas de segurança, velocidade e capacidade de processamento para uma aplicação particular em que a criptografia visual será aplicada.

Referências

[1] Lynn Hathaway. National policy on the use of the advanced encryption standard (AES) to protect national security systems and national security information. *National Security Agency*, 23, 2003.

[2] Howard Heys. Selected Areas in Cryptography. In *6th Annual International Workshop*, volume 1758, page 79, Kingston, Ontario, Canada, August 2000. Springer Science & Business Media.

[3] Quan Huynh-Thu and Mohammed Ghanbari. Scope of validity of PSNR in image/video quality assessment. *Electronics letters*, 44(13):800–801, 2008.

[4] Scikit image Contributors. scikit-image.metrics.structural_similarity, 2023. `https://scikit-image.org/docs/stable/api/skimage.metrics.html#skimage.metrics.structural_similarity`.

[5] Venkata Krishna Pavan Kalubandi, Hemanth Vaddi, Vishnu Ramineni, and Agilandeeswari Loganathan. A novel image encryption algorithm using AES and visual cryptography. In *2nd International Conference on Next Generation Computing Technologies*, pages 808–813, 2016.

[6] Moni Naor and Adi Shamir. Visual Cryptography. In *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques*, volume 950 of *Lecture Notes in Computer Science*, pages 1–12, Perugia, Italy, May 1994. Springer.

[7] Vila Nova, Tavares, and Aguiar. AES Implementation, November 2023. `https://colab.research.google.com/drive/1801Rku4IUginUW22eVprvh7EOwxAK9fN?usp=sharing`.

[8] Vila Nova, Tavares, and Aguiar. AES Implementation, November 2023. `https://colab.research.google.com/drive/1AHnYywgU5nJWp6wu0DXuDyAWfLZ6rYVg?usp=sharing`.

[9] Vila Nova, Tavares, and Aguiar. Bit Level Decomposition Implementation, November 2023. `https://colab.research.google.com/drive/1ZrRYQ6DEvCLpyjx2hzAfIKm5elSPOMw1?usp=sharing`.

[10] Vila Nova, Tavares, and Aguiar. Colab specifications, November 2023. `https://colab.research.google.com/drive/1iyf8VytGqzw-d67KczUvRTk3Vytwd-nW?usp=sharing`.

[11] Vila Nova, Tavares, and Aguiar. Modular Arithmetic Implementation, November 2023. https://colab.research.google.com/drive/164jIsSNEG_t8kLyBT9dX9PMNUaxb-uH1?usp=sharing.

[12] Vila Nova, Tavares, and Aguiar. Pixel Expansion Implementation, November 2023. https://colab.research.google.com/drive/1wpo1g0lo5XJvxrbuzky39z5Tijwjf0zS?usp=sharing.

[13] Vila Nova, Tavares, and Aguiar. XOR Method Implementation, November 2023. https://colab.research.google.com/drive/1RMR2bBN47FWr-Pmhgf_mw32Cv2EmILkv?usp=sharing.

[14] National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES). *Federal information processing standards publication 197*, pages 1–53, 2001.

[15] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson. Performance comparison of the AES submissions. In *Second AES Candidate Conference*, pages 15–34, 1999.

[16] Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson, Tadayoshi Kohno, and Mike Stay. The Twofish team's final comments on AES Selection. *AES round*, 2(1):1–13, 2000.

[17] D Taghaddos and A Latif. Visual cryptography for gray-scale images using bit-level. *Journal of Information Hiding and Multimedia Signal Processing*, 5(1):90–97, 2014.

[18] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers*, volume 2, pages 1398–1402. Ieee, 2003.