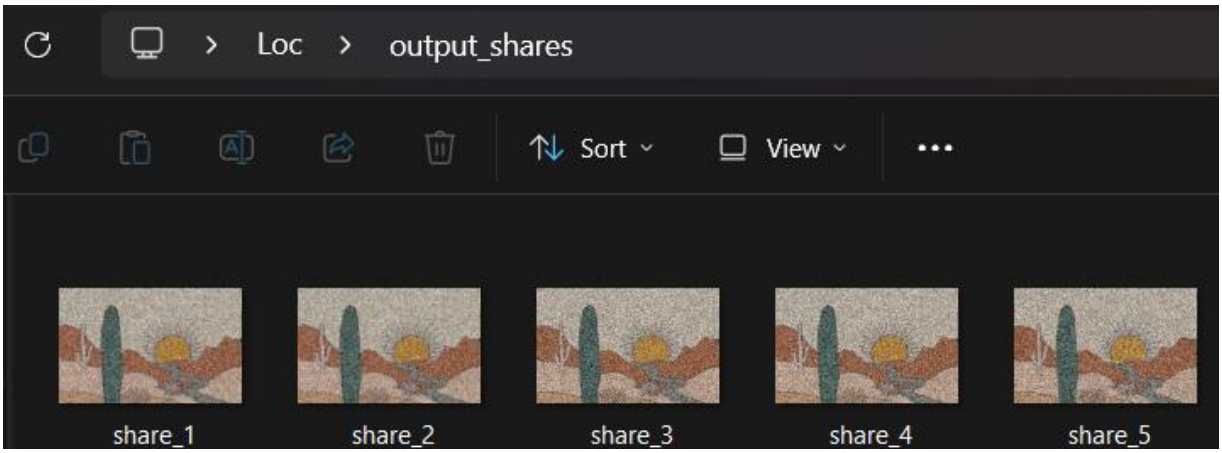
XOR-Based (k,n) Visual Cryptography Scheme With Meaningful Shares

Đoạn code thực hiện việc triển khai phương pháp chia sẻ bí mật (Secret Sharing) với hình ảnh. Sau đây là ví dụ về kết quả chạy code với 2 hình ảnh cùng kích thước:

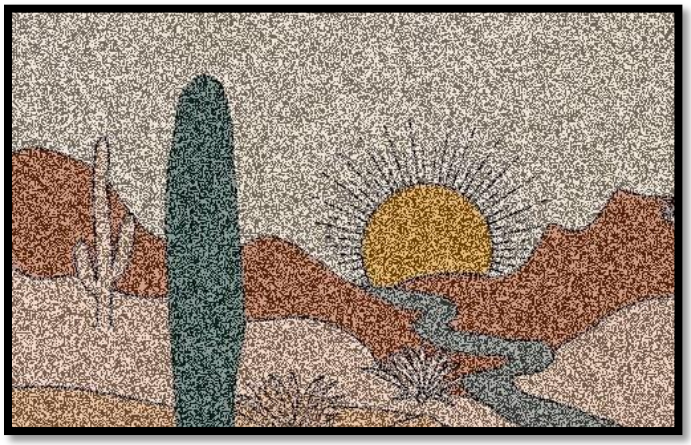
Hình ảnh gốc:

Hình ảnh phũ:

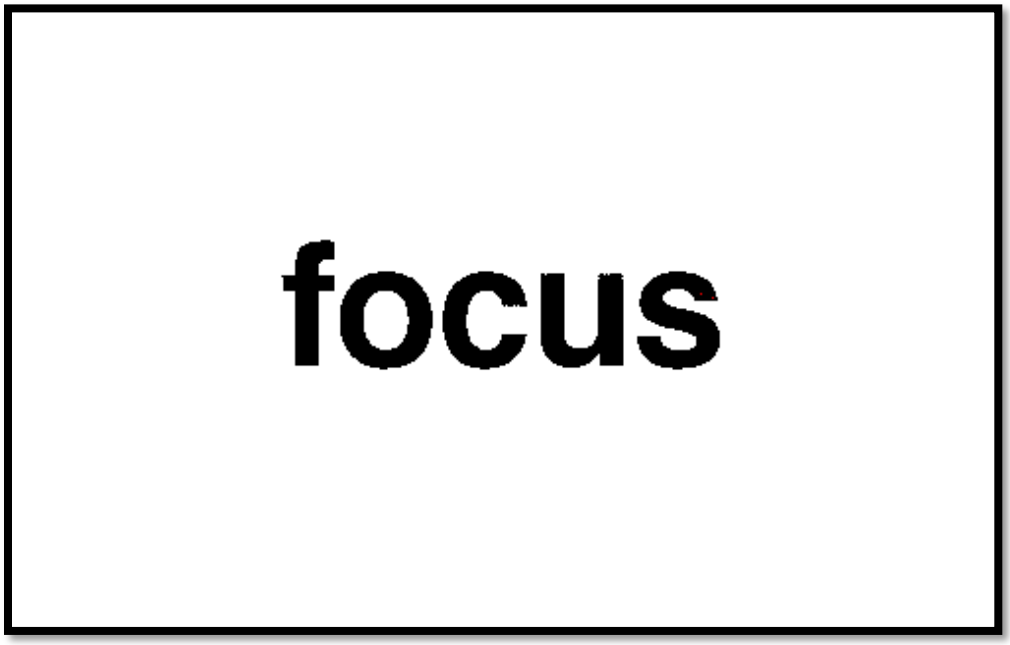
Kết quả sau khi chạy chương trình với lựa chọn tách 5 ảnh share:



Share 1:



Hình ảnh sau khi kết hợp lại 5 ảnh share trên với thuật toán:



GIẢI THÍCH CODE:

```
# Hàm XOR giữa hai ảnh nhị phân (sử dụng từng kênh màu)
def xor_image(image1, image2):
    return cv2.bitwise_xor(image1, image2)
```

Hàm cv2.bitwise_xor(image1, image2) trong OpenCV thực hiện **phép XOR nhị phân** giữa hai ảnh hoặc ma trận.

output(x,y)=image1(x,y)⊕image2(x,y)

```
# Chuyển ảnh thành ảnh nhị phân
def binarize_image(image):
    _, binary_image = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)
    return binary_image
```

Sử dụng hàm `cv2.threshold` của OpenCV để thực hiện **nhị phân hóa** (`binarization`) một ảnh xám (`grayscale image`) thành ảnh nhị phân (`binary image`).

Mỗi pixel trong ảnh xám có giá trị cường độ từ 0 đến 255.

Bất kỳ pixel nào có giá trị lớn hơn hoặc bằng 127 sẽ được đặt thành giá trị 255 (trắng).

Pixel nào có giá trị nhỏ hơn 127 sẽ được đặt thành 0 (đen).

cv2.THRESH_BINARY: Chỉ định kiểu nhị phân hóa (binary thresholding).

```
def generate_random_matrices(h, w):
    M_even = np.random.randint(0, 2, size=(h, w), dtype=np.uint8) * 255
    M_odd = np.random.randint(0, 2, size=(h, w), dtype=np.uint8) * 255
    return M_even, M_odd
```

Tạo hai ma trận nhị phân ngẫu nhiên M_even và M_odd có kích thước bằng ảnh bí mật. Hai ma trận nhị phân kích thước h×w, mỗi phần tử có giá trị 0 hoặc 255.

Ý nghĩa của tham số đầu vào:

- **h**: Chiều cao (số hàng) của ma trận cần tạo.
- **w**: Chiều rộng (số cột) của ma trận cần tạo.

Sinh ma trận ngẫu nhiên kích thước **h×w**, với mỗi phần tử là số nguyên ngẫu nhiên nằm trong khoảng [0, 2) (tức là giá trị chỉ có thể là **0** hoặc **1**).

`dtype=np.uint8`: Đảm bảo rằng ma trận sử dụng kiểu dữ liệu **`unsigned 8-bit integer`** để lưu trữ giá trị (tương thích với ảnh số).

*** 255:** Chuyển giá trị nhị phân từ 0 và 1 => 0 và 255

Hàm **generate_shares** thực hiện việc mã hóa một ảnh bí mật (secret image) thành nhiều chia sẻ bí mật (shares). Các chia sẻ này được kết hợp với một ảnh phủ màu (cover image) để bảo đảm tính thẩm mỹ và tăng cường bảo mật.

Mục tiêu chính của hàm:

1. Chuyển ảnh bí mật thành ảnh nhị phân (bởi XOR chỉ hoạt động hiệu quả với dữ liệu nhị phân).
2. Tạo ra n chia sẻ từ ảnh bí mật bằng cách sử dụng các ma trận ngẫu nhiên M_even và M_odd.
3. Pha trộn các chia sẻ với ảnh phủ để tạo ra kết quả màu sắc hấp dẫn.
4. Đảm bảo rằng chỉ khi có đủ k trong số n chia sẻ, người dùng mới có thể tái tạo lại ảnh bí mật.

```
# Tạo các chia sẻ từ ảnh bí mật và ảnh phủ sử dụng phương pháp XOR cải tiến
def generate_shares(secret_image, cover_image, n, k):
      secret_image = binarize_image(secret_image)  # Chuyển ảnh bí mật thành nhị
phân
      h, w = secret_image.shape
      shares = []
      # Tạo ma trận M_even và M_odd
      M_even, M_odd = generate_random_matrices(h, w)
      # Tách ảnh phủ thành ba kênh (Red, Green, Blue)
      cover_image_bgr = cv2.split(cover_image)  # Tách các kênh của ảnh màu
      # Tạo (n-1) chia sẻ ngẫu nhiên và lấy hàng ngẫu nhiên từ M_even và M_odd
      random_shares = []
```

Tạo **n−1** chia sẻ ngẫu nhiên.
Mỗi lần lặp:

1. Chọn ngẫu nhiên một ma trận (M_even hoặc M_odd) làm nguồn dữ liệu để tạo chia sẻ.
2. Thực hiện phép XOR giữa secret_image và ma trận đã chọn.
3. Kết quả là một chia sẻ ngẫu nhiên được thêm vào danh sách.

```
for i in range(n - 1):
        r = np.random.randint(0, 2)  # Chọn ngẫu nhiên r trong {0, 1}
        if r == 0:
            D = M_even
        else:
            D = M_odd
        share = np.copy(secret_image)
        for i in range(h):
            for j in range(w):
                # Thực hiện XOR giữa các giá trị pixel của ảnh bí mật và ảnh phủ
                share[i, j] = D[i, j] ^ secret_image[i, j]  # XOR với ảnh phủ
        random_shares.append(share)
```

Final_share được tạo bằng cách thực hiện phép XOR giữa ảnh bí mật và tất cả các chia sẻ ngẫu nhiên đã tạo. Việc này đảm bảo rằng chỉ khi có đủ k chia sẻ, ảnh bí mật mới có thể được tái tạo.

```
    # Tạo chia sẻ cuối cùng với phép XOR của tất cả các chia sẻ ngẫu nhiên
    final_share = secret_image.copy()
    for rand_share in random_shares:
        final_share = xor_image(final_share, rand_share)
    random_shares.append(final_share)
```

Với mỗi chia sẻ:

- Pha trộn chia sẻ nhị phân với từng kênh màu của ảnh phủ.
- Kết hợp ba kênh (BGR) lại thành một ảnh màu.
- Thêm ảnh này vào danh sách chia sẻ.

```
    # Pha trộn các chia sẻ với ảnh phủ màu và thêm vào danh sách chia sẻ
    for share in random_shares:
        # Pha trộn từng kênh với chia sẻ
        blended_share_bgr = []
        for i in range(3):  # Xử lý 3 kênh (Blue, Green, Red)
            blended_channel = blend_with_cover(share, cover_image_bgr[i])
            blended_share_bgr.append(blended_channel)

        # Kết hợp lại thành ảnh màu
        blended_share = cv2.merge(blended_share_bgr)
        shares.append(blended_share)
```

```
# Pha trộn chia sẻ với từng kênh màu của ảnh phủ
def blend_with_cover(binary_share, cover_channel):
    return cv2.addWeighted(cover_channel, 0.5, binary_share, 0.5, 0)
```

cover_channel: Đây là một kênh màu (Blue, Green, hoặc Red) của ảnh phủ được lấy bằng hàm cv2.split. Nó là một ma trận 2D với các giá trị pixel trong khoảng từ 0 đến 255.

binary_share: Đây là một chia sẻ nhị phân (binary) của ảnh bí mật, với các giá trị pixel chỉ là 0 hoặc 255.

0.5 (alpha): Trọng số của kênh màu từ ảnh phủ.

0.5 (beta): Trọng số của chia sẻ nhị phân.

0 (gamma): Không thêm bất kỳ giá trị nào vào kết quả, giữ nguyên tổng trọng số.

Công thức:

output (x , y) = 0.5 * cover_channel (x , y) + 0.5 * binary_share (x , y)

Giá trị pixel ở mỗi điểm (x,y) là trung bình của giá trị pixel từ ảnh phủ và chia sẻ nhị phân.

Dựa trên nguyên lý chia sẻ bí mật: Một ảnh bí mật đã được phân tách thành n chia sẻ, và chỉ cần đủ k chia sẻ trong số đó để tái tạo lại ảnh gốc.

Phép XOR: Phép XOR đảm bảo tính thuận nghịch, tức là nếu hai dữ liệu đã được XOR với nhau, chúng có thể được khôi phục bằng cách XOR lại với dữ liệu ban đầu.

Kiểm tra số lượng chia sẻ trong danh sách shares.

```
# Tái tạo ảnh bí mật từ các chia sẻ
def reconstruct_secret(shares, k):
    if len(shares) < k:
        print(f"Lỗi: Cần ít nhất {k} chia sẻ để tái tạo ảnh bí mật.")
```

Mỗi chia sẻ có thể là ảnh màu hoặc ảnh xám. Hàm **binarize_image** được gọi để chuyển từng chia sẻ thành ảnh nhị phân (giá trị pixel chỉ là 0 hoặc 255), giúp chuẩn bị dữ liệu cho phép XOR.

```
# Chuyển các chia sẻ thành ảnh nhị phân
    binary_shares = [binarize_image(share) for share in shares]
```

Ảnh đầu tiên trong danh sách chia sẻ nhị phân được chọn làm ảnh khởi tạo. Kết quả của các phép XOR sẽ dần dần được xây dựng dựa trên ảnh này

```
    reconstructed = binary_shares[0]
```

Bắt đầu từ chia sẻ thứ hai, thực hiện phép XOR giữa ảnh tái tạo hiện tại (reconstructed) với các chia sẻ tiếp theo trong danh sách.

Công thức:

$$\textbf{reconstructed} = \textbf{reconstructed} \oplus \textbf{binary\_shares}[\,i\,]$$

```
    for i in range(1, k):
        reconstructed = xor_image(reconstructed, binary_shares[i])
    return reconstructed
```

Lý do sử dụng XOR:

- Tính chất của XOR: $A \oplus A = 0$ và $A \oplus 0 = A$.
- Khi đủ k chia sẻ hợp lệ được XOR với nhau, toàn bộ nhiễu ngẫu nhiên (từ các ma trận M_even và M_odd) bị triệt tiêu, và ảnh gốc sẽ được tái tạo.

```
# Lưu ảnh chia sẻ vào thư mục
def save_images(images, output_dir, prefix="share"):
>>>
```

Khi tệp được chạy trực tiếp (chạy bằng lệnh python tên_tệp.py), giá trị của "__name__" sẽ là "__main__".

Khi tệp được nhập (import) vào một tệp khác dưới dạng một module, giá trị của __name__ sẽ là tên của module (tên tệp, không bao gồm phần mở rộng .py).

Câu lệnh if __name__ == "__main__" kiểm tra điều kiện:

- Nếu tệp được chạy trực tiếp, khối mã bên trong if __name__ == "__main__": sẽ được thực thi.
- Nếu tệp được nhập vào từ một tệp khác, khối mã đó sẽ bị bỏ qua.

```
if __name__ == "__main__":
```

Đọc ảnh gốc:

```
    secret_image_path = "C:/Users/Loc/Pictures/out/goc.jpg"
    secret_image = cv2.imread(secret_image_path, cv2.IMREAD_GRAYSCALE)
    if secret_image is None:
        print("Lỗi: Không tìm thấy ảnh bí mật.")
        exit()
```

Đọc ảnh phũ:

```
    cover_image_path = "C:/Users/Loc/Pictures/out/phu1.jpg"
    cover_image = cv2.imread(cover_image_path)  # Đọc ảnh màu (BGR)
    if cover_image is None:
        print("Lỗi: Không tìm thấy ảnh phủ.")
        exit()
```

```
    # Số lượng chia sẻ (n) và số chia sẻ cần thiết để tái tạo ảnh (k)
    n = 5 # Số chia sẻ muốn tạo
    k = 5 # Số chia sẻ cần thiết để tái tạo ảnh
```

Quy trình:

1. Đọc ảnh bí mật và ảnh phủ.
2. Gọi hàm generate_shares để tạo chia sẻ.
3. Lưu các chia sẻ bằng save_images.
4. Tái tạo ảnh bí mật từ các chia sẻ bằng reconstruct_secret.

_____END_____