UNIVERSITY OF INFORMATION TECHNOLOGY
FACULTY OF COMPUTER NETWORK AND COMMUNICATION

Báo cáo
Môn: Mật mã học
Cuối kì I (2023 – 2024)

ENCRYPTION AUDIO DATA ON MULTIMEDIA PRODUCT SERVICE PLATFORM

Student: Mai Nguyễn Nam Phương

Student ID Number: 22521164

Student: Nguyễn Nhật Quang

Student ID Number: 22521203

Student: Cao Quốc Thịnh

Student ID Number: 18521436

Class: NT219.O21

University of Information Technology

Lecturer: PhD. Nguyễn Ngọc Tự

Lời cảm ơn

Lời đầu tiên, cho phép tập thể Nhóm 22 đến từ lớp NT219.O21 xin gửi lời cảm ơn và tri ân sâu sắc đến thầy Nguyễn Ngọc Tự - giảng viên môn Mật Mã Học, vì sự tận tâm giúp đỡ, chỉ bảo và hướng dẫn trong quá trình thực hiện dự án. Những buổi thảo luận và sự hướng dẫn của thầy đóng vai trò quan trọng trong việc hoàn thiện đề tài này. Chúng em chân thành cảm ơn thầy vì kiến thức và kinh nghiệm mà thầy đã chia sẻ với chúng em.

Chúng em cũng muốn gửi lời cảm ơn đến tất cả các thành viên trong nhóm đồ án. Sự đóng góp và nỗ lực của mỗi người trong việc tìm kiếm tài liệu, đưa ra ý tưởng và hoàn thiện đề tài đã tạo nên thành công của cuộc . Mặc dù kiến thức của chúng em còn hạn chế và không tránh khỏi những sai sót, nhưng sự đóng góp ý kiến của mọi người đã giúp chúng em hoàn thiện và cải thiện đề tài một cách tốt nhất.

Cuối cùng, vì thời gian và năng lực có hạn nên không thể tránh khỏi sai sót trong khi thực hiện đồ án học tập của chúng em. Rất mong sự góp ý và bổ sung của thầy và các bạn để đề tài chúng em trở nên hoàn thiện hơn. Một lần nữa, chân thành cảm ơn tất cả mọi người đã tham gia và ủng hộ cuộc dự án này.

Contents

I. Tổng quan đề tài:

1. Chủ đề:

- Project topic: Mã hóa âm thanh

- Project title: Mã hóa dữ liệu âm thanh trên nền tảng dịch vụ sản phẩm đa phương tiện

2. Ngữ cảnh vấn đề:

- Lượng dữ liệu âm thanh đang ngày càng tăng lớn, bao gồm các nội dung như phim ảnh, video, podcast, âm nhạc trực tuyến... Điều quan trọng là cần có các giải pháp mã hóa hiệu quả để lưu trữ, truyền tải và bảo mật dữ liệu âm thanh.

- Nhiều dịch vụ số ngày nay cung cấp nội dung đa phương tiện kết hợp video, hình ảnh và âm thanh. Việc xử lý và mã hóa dữ liệu âm thanh trên các nền tảng này là rất cần thiết.

- Nghiên cứu về mã hóa dữ liệu âm thanh giúp cải thiện chất lượng trải nghiệm nghe nhạc, xem phim của người dùng trên các thiết bị di động và internet.

2.1. Lí do chọn đề tài:

- Âm thanh là một phần quan trọng của dịch vụ sản phẩm đa phương tiện. Việc mã hóa dữ liệu âm thanh đảm bảo rằng người dùng có thể truy cập và tận hưởng âm thanh một cách chất lượng và liền mạch trên nền tảng này.

- Mã hóa dữ liệu âm thanh có thể cung cấp cơ chế bảo mật để đảm bảo rằng dữ liệu âm thanh không bị truy cập trái phép hoặc biến đổi trong quá trình truyền tải.

2.2. Mục tiêu:

- Từ những điều đó, ta cần phải xây dựng một hệ thống mã hoá và giải mã để có thể bảo đảm về tính bảo mật, tính toàn vẹn và tính khả dụng của âm thanh khi được phát trực tuyến

3. Vai trò của các bên liên quan:

- Nhà cung cấp dịch vụ (Service Provider): Thực hiện các biện pháp bảo mật, cung cấp cơ chế mã hóa và đảm bảo tính khả dụng và chất lượng âm thanh cho người dùng.

- Tác giả (Author): Đăng tải các bản nhạc, file âm thanh.

- Người dùng (User): Những người đăng kí/ đăng nhập, mua bản quyền và sử dụng dịch vụ để có thể nghe các nội dung được phát.

- Hạ tầng (Infrastructure): Cung cấp nền tích cho dịch vụ lưu trữ, truyền tải dữ liệu mã hóa âm thanh an toàn, ổn định.

- Kẻ tấn công (Attacker): Cố gắng xâm phạm hệ thống để truy cập trái phép vào nội dung bản quyền.

4. Giải pháp:

- Xây dựng nền tảng cung cấp dịch vụ nghe nhạc trực tuyến cho người dùng

- Mã hoá dữ liệu và lưu trữ

- Giải mã và cung cấp cho nền tảng giúp cung cấp đến người dùng

II. Bối cảnh:

1. Double DNA encoding:

- Double DNA encoding là quá trình mã hóa đôi sử dụng phép cộng DNA.

- Trong quá trình này, dữ liệu nhị phân được chuyển đổi thành dạng mã DNA theo một quy tắc nhất định, thông thường là chuyển mã 02 bit thành một trong 4 nucleic acid A, C, G, T.

- Sau khi chuyển đổi sang dạng mã DNA, phép cộng DNA được áp dụng giữa chuỗi DNA của dữ liệu và hai chuỗi mã DNA khác.

- Phép cộng DNA tuân theo bảng phép cộng DNA, trong đó quy định kết quả của phép cộng giữa các cặp nucleic acid.

- Kết quả sau cùng của double DNA encoding là một chuỗi DNA thu được sau phép cộng DNA thứ hai.

- Chuỗi kết quả này sau đó được chuyển đổi trở lại dạng nhị phân để áp dụng các bước tiếp theo trong thuật toán mã hóa.

2. Chaotic map:

- Chaotic map là bản đồ hỗn loạn, được sử dụng để tạo ra các chuỗi ngẫu nhiên nhưng mang tính hỗn loạn cao.

- Chaotic map cung cấp tính phi tuyến và nhạy cảm với điều kiện ban đầu. Chúng thường được sử dụng trong các thuật toán mã hóa để tạo ra sự lẫn lộn và lan truyền.

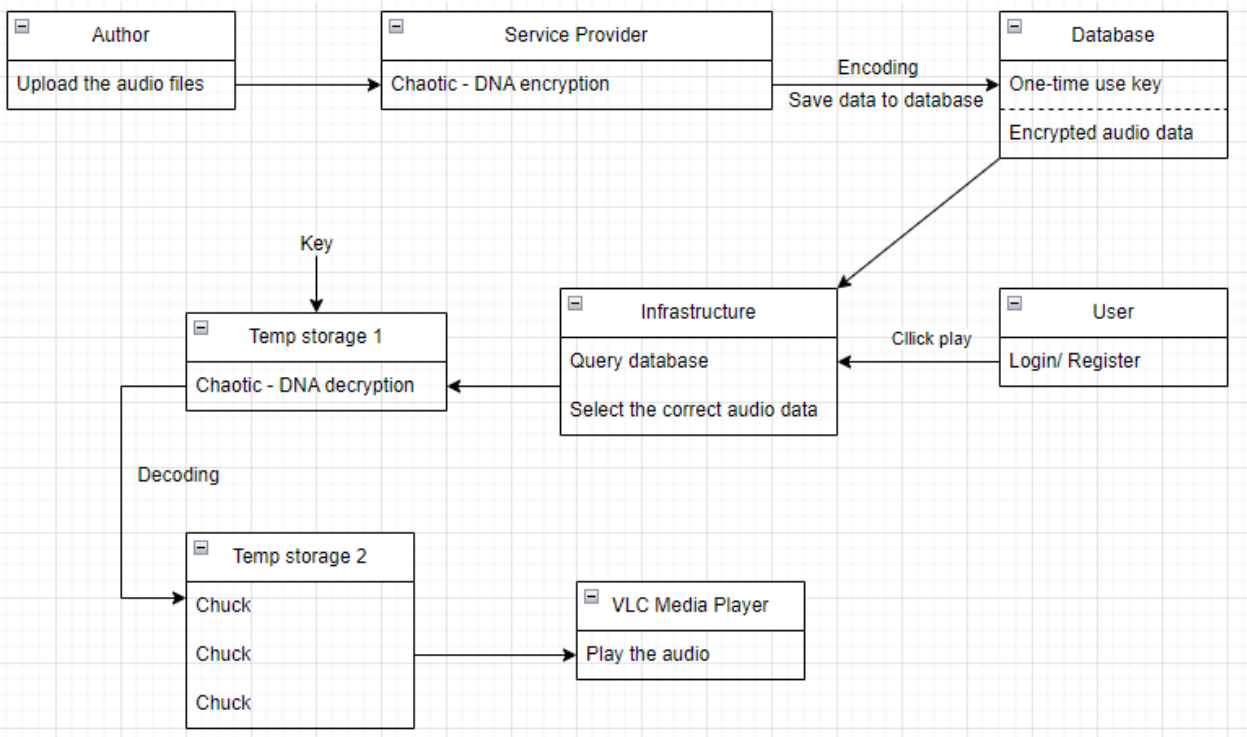
- Các chaotic map phổ biến bao gồm Logistic map, Sine map, Tent map. Tuy nhiên chúng có một số nhược điểm như khoảng hỗn loạn nhỏ, cửa sổ ổn định.

- Sử dụng hai loại chaotic map là Sine-Cosine map (SC map) và Logistic Sine Cosine map (LSC map). Đây là hai bản đồ một chiều kết hợp, có khoảng hỗn loạn lớn và khắc phục các nhược điểm của các bản đồ truyền thống.

- Chaotic map được sử dụng trong thuật toán để tạo ra các chuỗi ngẫu nhiên phục vụ các bước pha truy xuất, lan truyền dữ liệu.

III. Hướng giải quyết và code:

1. Solution Architecture:



- Tác giả tải file bài nhạc lên hệ thống.

- Hệ thống sẽ mã hóa file âm thanh đó, đồng thời lúc này xuất ra hai file “One-time use key” và file âm thanh đã được mã hóa được lưu vào database.

- Khi người dùng muốn nghe nhạc phải đăng nhập vào trang web với thông tin cá nhân đã được đăng kí.

- Người dùng nhấn chọn bài hát muốn nghe.

- Hệ thống sẽ truy vấn và chọn dữ liệu bài hát đã được mã hóa đúng với yêu cầu của người dùng từ cơ sở dữ liệu, đưa ra bộ nhớ tạm 1.

- File bài hát này sẽ kết hợp với “One-time use key” để được giải mã theo từng chuck âm thanh và đưa vào bộ nhớ tạm 2.

- Giao diện VLC media player sẽ được hiển thị làm giao diện chính để phát nhạc cho người dùng theo từng chuck âm thanh được tải lên.

2. Encryption:

- Nhập và ghi dữ liệu âm thanh:

```
# Hàm nhập dữ liệu âm thanh từ file
def Input(filename):
  # Đọc dữ liệu âm thanh với kiểu dữ liệu là int16
  data, sample_rate = librosa.load(filename, sr=None, dtype=np.float32)

  # Chuyển đổi dữ liệu âm thanh thành kiểu int16
  data_int16 = (data * np.iinfo(np.int16).max).astype(np.int16)

  return data_int16, sample_rate

# Hàm ghi dữ liệu âm thanh ra file
def Output(filename, sample_rate, data):
  # Mở file âm thanh để ghi
  with open(filename, "wb") as f:
    # Ghi dữ liệu âm thanh
    scipy.io.wavfile.write(f, sample_rate, data)
```

- Khởi tạo khóa và Vector:

```
# Tạo random ra hai mảng ShareKey và IntialVector với 128 bits
def Generation():
  ShareKey = [random.randint(0, 1) for i in range(128)]
  IntialVector = [random.randint(0, 1) for i in range(128)]
  return "".join(str(bit) for bit in ShareKey), "".join(str(bit) for bit in IntialVector)

# Hàm chuẩn hoá từ dạng binary sang dạng hex
def Hex(bin_string):
  return format(int(bin_string, 2), "02x")
```

- Tạo ra chuỗi ngẫu nhiên bằng chaotic map:

```
# Tạo ra Sine Cosine Sequence
def SineCosineChaoticMap(x0, m, n):
  # Khởi tạo mảng kết quả
  SineCosineSequence = np.zeros(n)

  # Gán giá trị hạt giống ban đầu
  SineCosineSequence[0] = x0

  # Tính các giá trị tiếp theo của map
  for i in range(1, n):
    A = np.sin(-m * SineCosineSequence[i - 1] + pow(SineCosineSequence[i - 1], 3) - m * np.sin(SineCosineSequence[i - 1]))
    B = np.cos(-m * SineCosineSequence[i - 1] + pow(SineCosineSequence[i - 1], 3) - m * np.sin(SineCosineSequence[i - 1]))
    SineCosineSequence[i] = abs(abs(A) - abs(B))

  return SineCosineSequence

# Tạo ra Logistic Sine Cosine Sequence
def LogisticSineCosine(x0, r, n):
  # Khởi tạo mảng kết quả
  LogisticSineCosineSequence = np.zeros(n)

  # Gán giá trị hạt giống ban đầu
  LogisticSineCosineSequence[0] = x0

  # Tạo biến pi
  pi = math.pi

  # Tính các giá trị tiếp theo của map
  for i in range(1, n):
    LogisticSineCosineSequence[i] = np.cos(pi * (4 * r * LogisticSineCosineSequence[i - 1] * (1 - LogisticSineCosineSequence[i-1]) + (1 - r) * np

  return LogisticSineCosineSequence
```

- Permutation:

```
def Permutation(Audio, IV3, SineCosineSequence):
  Length = len(Audio)
  rem = Length % 16 # used for calculating the total number of the variables SequenceSize
  div = int(Length / 16) # used for calculating the remaining number

  # the 16 different dynamic sequence for the permutation process
  InitialSequence1 = Initial_Seq(IV3, SineCosineSequence, 16)
  InitialSequence2 = Initial_Seq(IV3, SineCosineSequence, rem)

  audio_pos = 0
  output = np.array([], dtype = np.int16)
  Permutated_Val = np.zeros(16, dtype = np.int16)

  for i in range(div):
    # Rotate the variable InitialSequence1 by I3 * (i + 1) % 16 and store in variable Sequence
    Sequence = np.roll(InitialSequence1, IV3 * (i + 1) % 16)
    Audio_16_val = Audio[audio_pos : audio_pos + 16]

    # Take 16 values of the audio data, and then permute these values according to the Sequence variable
    for j in range(16):
      Permutated_Val[j] = Audio_16_val[Sequence[j]]
    output = np.concatenate((output, Permutated_Val))
    audio_pos = audio_pos + 16

  Audio_16_val = Audio[audio_pos:audio_pos + rem]
  Permutated_Val1 = np.zeros(rem, dtype = np.int16)

  for i in range(rem):
    Permutated_Val1[i] = Audio_16_val[InitialSequence2[i]]
  output = np.concatenate((output, Permutated_Val1))

  return output
```

- Diffusion:

```python
def int16_array_to_binary16(array):
    # Chuyển đổi mảng int16 sang chuỗi nhị phân 16 bit
    binary_array = [np.binary_repr(x, width=16) for x in array]
    return binary_array

def Binary_to_DNA_Seq(rule, binary_arr):
  encoding_rules = {
    0: ['A', 'C', 'T', 'G'],
    1: ['A', 'G', 'T', 'C'],
    2: ['T', 'C', 'A', 'G'],
    3: ['T', 'G', 'A', 'C'],
    4: ['C', 'A', 'G', 'T'],
    5: ['C', 'T', 'G', 'A'],
    6: ['G', 'A', 'C', 'T'],
    7: ['G', 'T', 'C', 'A']
  }

  # Chuyển đổi binary_arr thành chuỗi DNA
  dna_seq_arr = []
  for binary_str in binary_arr:
    dna_seq = ''
    for i in range(0, len(binary_str), 2):
      index = int(binary_str[i:i+2], 2)
      dna_seq += encoding_rules[rule][index]
    dna_seq_arr.append(dna_seq)

  return dna_seq_arr
```

```python
def DNA_addition(dnaseq_arr_1, dnaseq_arr_2):
    # Tạo bảng cộng
    addition_table = {
        'A': ['A', 'C', 'G', 'T'],
        'C': ['C', 'G', 'T', 'A'],
        'G': ['G', 'T', 'A', 'C'],
        'T': ['T', 'A', 'C', 'G']
    }

    # Thực hiện phép cộng
    add_dna_arr = []
    for dna1, dna2 in zip(dnaseq_arr_1, dnaseq_arr_2):
        result = ''
        for base1, base2 in zip(dna1, dna2):
            # Chuyển đổi base2 thành số tương ứng
            tmp2 = 'ACGT'.index(base2)

            # Thực hiện cộng và thêm vào kết quả
            result += addition_table[base1][tmp2]
        add_dna_arr.append(result)

    return add_dna_arr
```

```python
def DNA_Seq_to_Binary(rule, dna_seq_arr):
    # Tạo bảng giải mã
    decoding_rules = {
        0: {'A': '00', 'C': '01', 'T': '10', 'G': '11'},
        1: {'A': '00', 'G': '01', 'T': '10', 'C': '11'},
        2: {'T': '00', 'C': '01', 'A': '10', 'G': '11'},
        3: {'T': '00', 'G': '01', 'A': '10', 'C': '11'},
        4: {'C': '00', 'A': '01', 'G': '10', 'T': '11'},
        5: {'C': '00', 'T': '01', 'G': '10', 'A': '11'},
        6: {'G': '00', 'A': '01', 'C': '10', 'T': '11'},
        7: {'G': '00', 'T': '01', 'C': '10', 'A': '11'}
    }

    # Chuyển đổi chuỗi DNA thành chuỗi nhị phân
    binary_arr = []
    for dna_seq in dna_seq_arr:
        binary_str = ''
        for dna in dna_seq:
            binary_str += decoding_rules[rule][dna]
        binary_arr.append(binary_str)

    return binary_arr
```

```python
def dna_apply(Key, binary_audio, binary_chaosValue1, binary_chaosValue2):
  rule = Key % 8
  dnaseq_audio = Binary_to_DNA_Seq(rule, binary_audio)
  dnaseq_val1 = Binary_to_DNA_Seq(rule, binary_chaosValue1)
  dnaseq_val2 = Binary_to_DNA_Seq(rule, binary_chaosValue2)

  AddValue1 = DNA_addition(dnaseq_audio, dnaseq_val1)
  Result = DNA_addition(AddValue1, dnaseq_val2)

  dnaseq_output = DNA_Seq_to_Binary(rule, Result)
  return dnaseq_output
```

- Tạo SecretKey và InitialVector:

```python
def generate_encryption_key(Length, pathShareKey, pathIntiialVector):
    # Tạo ShareKey (SecretKey) và InitialVector một cách ngẫu nhiên
    ShareKey, InitialVector = Generation()

    with open(pathShareKey, 'wb') as f:
        f.write(bytes(ShareKey, 'utf-8'))

    with open(pathIntiialVector, 'wb') as f:
        f.write(bytes(InitialVector, 'utf-8'))

    InitialVector_hex = Hex(InitialVector)

    I0 = int(InitialVector_hex[0:8], 16)
    I1 = int(InitialVector_hex[8:16], 16)
    I2 = int(InitialVector_hex[16:24], 16)
    I3 = int(InitialVector_hex[24:32], 16)

    IP_SC = 1 / (np.floor(int(ShareKey, 2)) + 1)
    CP_SC = 2.2 + ((I2 ^ I3) % 5)
    IP_LSC = 1 / (np.floor(int(InitialVector, 2)) + 1)
    CP_LSC = 1 / (np.floor(I3) + 1)

    val = I0 % 64
    IntermediateKey1 = ShareKey[val:val + 32]
    d = (2 * math.floor(np.floor(int(InitialVector, 2)) / 2)) + 1
    Tmp_Key = left_rotate(ShareKey, d)
    val1 = I1 % 64
    IntermediateKey2 = Tmp_Key[val1:val1 + 32]
```

- Encryption:

```python
def encrypt_song(path, pathShareKey, pathIntiialVector):
    # Đọc file âm thanh
    Audio, sample_rate = Input(path)
    Length = len(Audio)

    # Tạo mảng rỗng kiểu int16
    CipherVoice = np.zeros(Length, dtype=np.int16)

    # Tạo key cho mỗi bài hát
    InitialVector, IntermediateKey1, IntermediateKey2, SineCosineSequence, LogisticSineCosineSequence , I0, I1, I2, I3 = generate_encryption_key(

    # PERMUTATION PHASE
    Permutated_Audio = Permutation(Audio, I3, SineCosineSequence)

    # DIFFUSION PHASE
    IntermediateKey1_int = int(IntermediateKey1, 2)
    IntermediateKey2_int = int(IntermediateKey2, 2)

    CM1 = ((IntermediateKey1_int * SineCosineSequence) / pow(2, 16)).astype(np.int16)
    CM2 = ((IntermediateKey2_int * LogisticSineCosineSequence) / pow(2, 16)).astype(np.int16)

    binary_audio = int16_array_to_binary16(Permutated_Audio)
    binary_val1 = int16_array_to_binary16(CM1)
    binary_val2 = int16_array_to_binary16(CM2)

    DNA_data = dna_apply(IntermediateKey2_int, binary_audio, binary_val1, binary_val2)
```

3. Decryption:

- Đọc dữ liệu đầu vào:

```
# Hàm nhập dữ liệu âm thanh từ file
def Input(filename):
  # Đọc dữ liệu âm thanh với kiểu dữ liệu là int16
  data, sample_rate = librosa.load(filename, sr=None, dtype=np.float32)

  # Chuyển đổi dữ liệu âm thanh thành kiểu int16
  data_int16 = (data * np.iinfo(np.int16).max).astype(np.int16)

  return data_int16, sample_rate

# Hàm ghi dữ liệu âm thanh ra file
def Output(filename, sample_rate, data):
  # Mở file âm thanh để ghi
  with open(filename, "wb") as f:
    # Ghi dữ liệu âm thanh
    scipy.io.wavfile.write(f, sample_rate, data)

# Đọc vào hai mảng ShareKey và IntialVector với 128 bits từ file
def ReadFromFile(pathShareKey, pathIntiialVector):
  with open(pathShareKey, 'rb') as f:
    ShareKey = f.read().decode('utf-8')

  with open(pathIntiialVector, 'rb') as f:
    IntialVector = f.read().decode('utf-8')

  return ShareKey, IntialVector
```

- Thuật toán khởi tạo:

```
def generate_encryption_key(Length, pathShareKey, pathIntiialVector):
    # Tạo ShareKey (SecretKey) và InitialVector một cách ngẫu nhiên
    ShareKey, InitialVector = ReadFromFile(pathShareKey, pathIntiialVector)
    InitialVector_hex = Hex(InitialVector)

    I0 = int(InitialVector_hex[0:8], 16)
    I1 = int(InitialVector_hex[8:16], 16)
    I2 = int(InitialVector_hex[16:24], 16)
    I3 = int(InitialVector_hex[24:32], 16)

    IP_SC = 1 / (np.floor(int(ShareKey, 2)) + 1)
    CP_SC = 2.2 + ((I2 ^ I3) % 5)
    IP_LSC = 1 / (np.floor(int(InitialVector, 2)) + 1)
    CP_LSC = 1 / (np.floor(I3) + 1)

    val = I0 % 64
    IntermediateKey1 = ShareKey[val:val + 32]
    d = (2 * math.floor(np.floor(int(InitialVector, 2)) / 2)) + 1
    Tmp_Key = left_rotate(ShareKey, d)
    val1 = I1 % 64
    IntermediateKey2 = Tmp_Key[val1:val1 + 32]

    # CHAOTIC MAP GENERATION
    SineCosineSequence = SineCosineChaoticMap(IP_SC, CP_SC, Length)
    LogisticSineCosineSequence = LogisticSineCosine(IP_LSC, CP_LSC, Length)

    return InitialVector, IntermediateKey1, IntermediateKey2, SineCosineSequence, LogisticSineCosineSequence, I0, I1, I2, I3
```

- Tạo ra chuỗi ngẫu nhiên bằng Chaotis map:

```
# Tạo ra Sine Cosine Sequence
def SineCosineChaoticMap(x0, m, n):
  # Khởi tạo mảng kết quả
  SineCosineSequence = np.zeros(n)

  # Gán giá trị hạt giống ban đầu
  SineCosineSequence[0] = x0

  # Tính các giá trị tiếp theo của map
  for i in range(1, n):
    A = np.sin(-m * SineCosineSequence[i - 1] + pow(SineCosineSequence[i - 1], 3) - m * np.sin(SineCosineSequence[i - 1]))
    B = np.cos(-m * SineCosineSequence[i - 1] + pow(SineCosineSequence[i - 1], 3) - m * np.sin(SineCosineSequence[i - 1]))
    SineCosineSequence[i] = abs(abs(A) - abs(B))

  return SineCosineSequence

# Tạo ra Logistic Sine Cosine Sequence
def LogisticSineCosine(x0, r, n):
  # Khởi tạo mảng kết quả
  LogisticSineCosineSequence = np.zeros(n)

  # Gán giá trị hạt giống ban đầu
  LogisticSineCosineSequence[0] = x0

  # Tạo biến pi
  pi = math.pi

  # Tính các giá trị tiếp theo của map
  for i in range(1, n):
    LogisticSineCosineSequence[i] = np.cos(pi * (4 * r * LogisticSineCosineSequence[i - 1] * (1 - LogisticSineCosineSequenc

  return LogisticSineCosineSequence
```

- Reverse_Permutation:

```
def Reverse_Permutation(Permutated_Audio, IV3, SineCosineSequence):
    Length = len(Permutated_Audio)
    rem = Length % 16
    div = int(Length / 16)

    InitialSequence1 = Initial_Seq(IV3, SineCosineSequence, 16)
    InitialSequence2 = Initial_Seq(IV3, SineCosineSequence, rem)

    audio_pos = 0
    output = np.array([], dtype=np.int16)
    Audio_Val = np.zeros(16, dtype=np.int16)

    for i in range(div):
        Sequence = np.roll(InitialSequence1, IV3 * (i + 1) % 16)
        Permutated_Audio_16_val = Permutated_Audio[audio_pos: audio_pos + 16]

        for j in range(16):
            Audio_Val[Sequence[j]] = Permutated_Audio_16_val[j]

        output = np.concatenate((output, Audio_Val))
        audio_pos = audio_pos + 16

    Permutated_Audio_16_val = Permutated_Audio[audio_pos: audio_pos + rem]
    Audio_Val1 = np.zeros(rem, dtype=np.int16)

    for i in range(rem):
        Audio_Val1[InitialSequence2[i]] = Permutated_Audio_16_val[i]

    output = np.concatenate((output, Audio_Val1))

    return output
```

- Diffusion:

```
def DNA_Seq_to_Binary(rule, dna_seq_arr):
    # Tạo bảng giải mã
    decoding_rules = {
        0: {'A': '00', 'C': '01', 'T': '10', 'G': '11'},
        1: {'A': '00', 'G': '01', 'T': '10', 'C': '11'},
        2: {'T': '00', 'C': '01', 'A': '10', 'G': '11'},
        3: {'T': '00', 'G': '01', 'A': '10', 'C': '11'},
        4: {'C': '00', 'A': '01', 'G': '10', 'T': '11'},
        5: {'C': '00', 'T': '01', 'G': '10', 'A': '11'},
        6: {'G': '00', 'A': '01', 'C': '10', 'T': '11'},
        7: {'G': '00', 'T': '01', 'C': '10', 'A': '11'}
    }

    # Chuyển đổi chuỗi DNA thành chuỗi nhị phân
    binary_arr = []
    for dna_seq in dna_seq_arr:
        binary_str = ''
        for dna in dna_seq:
            binary_str += decoding_rules[rule][dna]
        binary_arr.append(binary_str)

    return binary_arr
```

```
def dna_apply(Key, binary_cipher, binary_chaosValue1, binary_chaosValue2):
  rule = Key % 8

  dnaseq_cipher = Binary_to_DNA_Seq(rule, binary_cipher)
  dnaseq_val1 = Binary_to_DNA_Seq(rule, binary_chaosValue1)
  dnaseq_val2 = Binary_to_DNA_Seq(rule, binary_chaosValue2)

  AddValue1 = DNA_subtract(dnaseq_cipher, dnaseq_val2)
  Result = DNA_subtract(AddValue1, dnaseq_val1)

  dnaseq_output = DNA_Seq_to_Binary(rule, Result)
  return dnaseq_output
```

- Decryption:

```
def decrypt_song(path, pathShareKey, pathIntiialVector):
    # Đọc file âm thanh
    CipherVoice, sample_rate = Input(path)

    Length = len(CipherVoice)

    for i in range(Length):
      if (CipherVoice[i] > 0):
        CipherVoice[i] = CipherVoice[i] + 1
      elif (CipherVoice[i] < 0):
        CipherVoice[i] = CipherVoice[i] - 1

    #Tạo mảng rỗng kiểu int16
    PlainVoice = np.zeros(Length, dtype=np.int16)

    # Tạo key cho mỗi bài hát
    InitialVector, IntermediateKey1, IntermediateKey2, SineCosineSequence, LogisticSineCosineSequence , I0, I1, I2, I3 = generate_encryption_key(

    # DECVOICE GENERATION

    I01 = np.zeros(Length, dtype=np.int16)
    I02 = np.zeros(Length, dtype=np.int16)

    for i in range(Length):
      I01[i] = np.int32((I1 * SineCosineSequence[i]))
      I02[i] = np.int32((I2 * LogisticSineCosineSequence[i]))

    DecVoice = np.zeros(Length, dtype=np.int16)
```

```
for i in range(Length):
  val = (I0 * (i + 1)) % 2
  if val == 0:
    DecVoice[i] = np.int16((CipherVoice[i] - I01[i] * (i + 1)) % pow(2,16))
  else:
    DecVoice[i] = np.int16((CipherVoice[i] - I02[i] * (i + 1)) % pow(2,16))

# DIFFUSION PHASE

IntermediateKey1_int = int(IntermediateKey1, 2)
IntermediateKey2_int = int(IntermediateKey2, 2)

CM1 = ((IntermediateKey1_int * SineCosineSequence) / pow(2, 16)).astype(np.int16)
CM2 = ((IntermediateKey2_int * LogisticSineCosineSequence) / pow(2, 16)).astype(np.int16)

binary_cipher = int16_array_to_binary16(DecVoice)
binary_val1 = int16_array_to_binary16(CM1)
binary_val2 = int16_array_to_binary16(CM2)

DNA_data = dna_apply(IntermediateKey2_int, binary_cipher, binary_val1, binary_val2)

Permutated_Audio = np.zeros(Length, dtype = np.int16)

for i in range(Length):
  #Permutated_Audio[i] = np.int16(int(DNA_data[i], 2))
  Permutated_Audio[i] = np.array(int(DNA_data[i], 2), dtype=np.uint16)

PlainVoice = Reverse_Permutation(Permutated_Audio, I3, SineCosineSequence)

return PlainVoice, sample_rate, InitialVector
```

IV. Thực nghiệm và kết quả:

- Video demo trong file zip

V. Tổng kết và công việc tương lai:

1. Tổng kết:

- Tạo ra nền tảng đơn giản cung cấp dịch vụ nghe nhạc cho người dùng

- Mã hóa và giải mã thành công, thu về biểu đồ như mong đợi

- Đảm bảo dữ liệu được đưa đến người dùng như mong muốn, không thể bị đánh cắp bởi attacker

- Toàn bộ dữ liệu trước khi được đưa vào lưu trữ đều được mã hoá

2. Công việc tương lai:

- Nghiên cứu và ứng dụng các thuật toán mã hóa mới như mã hóa không mất dữ liệu, mã hóa trích xuất đặc trưng và nhận diện giọng nói để nâng cao chất lượng và độ an toàn.

- Phát triển các giải pháp mã hóa dữ liệu âm thanh phù hợp với các định dạng file khác nhau như mp3, wav, m4a.

- Tích hợp mã hóa âm thanh vào các nền tảng phổ biến như YouTube, Spotify, SoundCloud để tối ưu hóa trải nghiệm người dùng.

- Nghiên cứu các giải pháp mã hóa âm thanh trực tuyến và trên thiết bị di động nhằm bảo vệ quyền riêng tư và bản quyền.

- Xây dự hệ thống mang tính trực tuyến

- Cung cấp thêm các dịch vụ cần thiết cho các bên liên quan

- Nâng cấp thuật toán để tối ưu thời gian thực thi