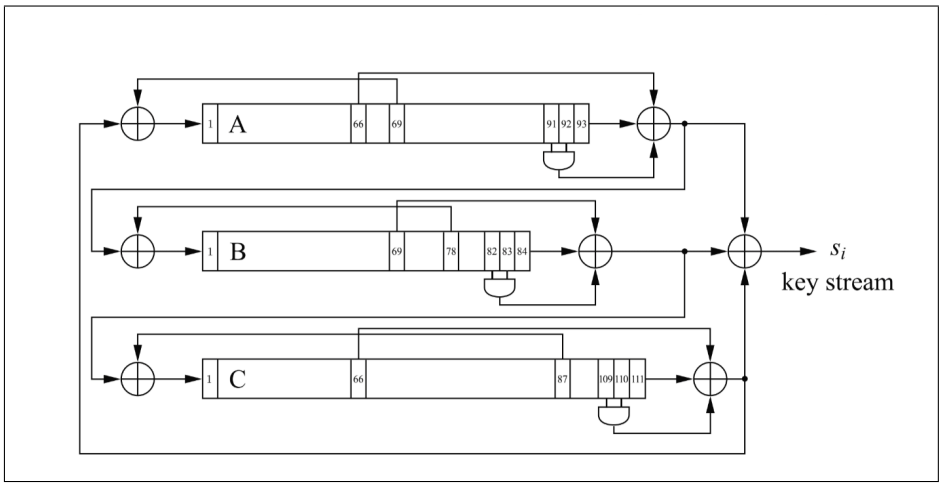
BÁO CÁO BÀI TẬP 2
HỆ MÃ DÒNG TRIVIUM

Đàm Ngọc Khánh - 20205207

Ngày 25 tháng 4 năm 2023

1 Mô tả hệ mã Trivium



2 Mã nguồn thực hiện

Mã nguồn yêu cầu đầu vào nhập 1 thông điệp cần mã hóa, IV và key được khởi tạo là chuỗi bit với đỗ dài 80.

- Hàm sinh dòng khóa nhận đầu vào IV, Key và độ dài của dòng khóa mong muốn trả về dãy dòng khóa dạng chuỗi bit
- Hàm mã hóa nhận đầu vào IV, Key và thông điệp cần mã hóa, trả về chuỗi các byte
- Hàm giải mã nhận đầu vào IV, Key, chuỗi các byte đã được mã hóa, trả về bản rõ sau khi giải mã

2.1 Hàm sinh dòng khóa

2.1.1 Mã nguồn

```python
def generate_trivium(iv, key, n):
# Khởi tạo trạng thái ban đầu của 288 bit
    stream_key = []
    state = [0] * 288
    for i in range(80):
        state[i] = iv[i]
        state[i+93] = key[i]
    state[285] = state[286] = state[287] = 1
# Vòng lặp sinh dòng khóa
    for i in range(1152 + n) :
        r1 = state[65] ^ state[92] ^ (state[90] & state[91])
        r2 = state[176] ^ state[161] ^ (state[174] & state[175])
        r3 = state[287] ^ state[242] ^ (state[285] & state[286])
# Nếu chạy tới lần thứ 1153 trở đi thì thêm vào key stream
        if(i >= 1152):
            stream_key.append(r1 ^ r2 ^ r3)

        state[92] = r1 ^ state[170]
        state[176] = r2 ^ state[263]
        state[287] = r3 ^ state[68]
        # Dich 1 vị trí các bit
        state.insert(0, state.pop())
    # Trả về dòng khóa dưới dạng chuỗi bit
    return ''.join(map(str, stream_key))
```

2.1.2 Giải thích

- Hàm nhận vào tham số IV, Key có độ dài 80 bit được khởi tạo trước đó và độ dài dòng khóa mong muốn. Đầu tiên, khởi tạo mảng state trạng thái ban đầu của 288 bit của hệ mã trivium = 0.

- Dùng vòng lặp 80 lần để gán các giá trị IV và Key lần lượt vào 80 bít đầu của thanh A, B ứng với vị trí 0 -> 79 cho IV và 93 -> 173 cho Key trong mảng trạng thái state. Cuối cùng gán 3 bit cuối của thanh C = 1 tương ứng với bit thứ 285, 286, 287 của mảng state
- Sau khi hoàn thành trạng thái ban đầu của hệ mã với IV, Key đầu vào. Ta chạy vòng lặp 1152 + n lần bởi sau 1152 lần để mã đủ ngẫu nhiên và phụ thuộc hoàn toàn vào IV và Key. ta sẽ lấy dòng bit khi chỉ số i >= 1152.

2.2 Hàm mã hóa

2.2.1 Mã nguồn

```python
# Hàm mã hóa
def encrypt_trivium(iv, key, plaint_text) :
    #Gọi hàm sinh dòng khóa
    key_stream = generate_trivium(iv, key, len(plaint_text)*8)
    #Chuyển dòng khóa dạng binary về string
    segments = [key_stream[i:i+8] for i in range(0, len(key_stream), 8)]
    key_stream_str = ''.join([chr(int(''.join(map(str, segment)), 2)) for segment in segments])
    #Xor 2 string gồm string muốn mã hóa và string stream key
    cipher_str = "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(key_stream_str, plaint_text)])
    #Trả về dạng chuỗi các byte
    return "".join([hex(ord(c)).replace("0x", "").zfill(2) for c in cipher_str])
```

2.2.2 Giải thích

- Gọi hàm sinh dòng khóa với độ dài = 8 * độ dài của thông điểm (1 kí tự 1 byte = 8 bit)
- Chuyển dòng khóa dạng bit về dạng string rồi xor string key stream với thông điệp rồi thực hiện chuyển đổi về chuỗi các byte và trả về chuỗi các byte đó

2.3 Hàm giải mã

2.3.1 Mã nguồn

```python
#Hàm giải mã
def decrypt_trivium(iv, key, cipher_text) :
    #Chuyển chuỗi byte về dạng string
    cipher_str = bytes.fromhex(cipher_text).decode('latin-1')
    #Sinh dòng khóa
    key_stream = generate_trivium(iv, key, len(cipher_text)*4)
    #Chuyển dòng khóa dạng binary về dạng string
    segments = [key_stream[i:i+8] for i in range(0, len(key_stream), 8)]
    key_stream_str = ''.join([chr(int(''.join(map(str, segment)), 2)) for segment in segments])
    #Xor bản mã với khóa để thu được bản rõ và trả về bản rõ
    return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(key_stream_str, cipher_str)])
```

2.3.2 Giải thích

- Chuyển bản mã từ chuỗi các byte về dạng string

- Sinh khóa từ IV, key đầu vào rồi chuyển về dạng string
- Xor 2 string gồm string dòng khóa và string bản mã ta sẽ được bản rõ rồi trả về bản rõ