Nguyễn Tuấn Thành – 20205027

Trivium Implemention And Writeups

‘Trivium is a synchronous stream cipher designed to generate up to 2^64 bits of key stream from an 80-bit secret key and an 80-bit initial value (IV). As for most stream ciphers, this process consists of two phases: first the internal state of the cipher is initialized using the key and the IV, then the state is repeatedly updated and used to generate key stream bits. ’

1. Secret key and IV setup

My trivium implemention is written in python 3. The 80-bit IV and 80-bit secret key are randomly generated by ‘secrets’ library.

My get_random_bits function:

```
105  def get_random_bits(length):
106      randbits = secrets.randbits(length)
107      randstring = '{0:080b}'.format(randbits)
108      return bytearray(map(int ,randstring))
109
```

‘The algorithm is initialized by loading an 80-bit key and an 80-bit IV into the 288-bit initial state, and setting all remaining bits to 0, except for s 286 , s287, and s288 . Then, the state is rotated over 4 full cycles, in the same way as explained above, but without generating key stream bits. ’

Pseudo code for the algorithm:

(s 1 , s2 , . . . , s93) ← (K1, . . . , K80 , 0, . . . , 0)
(s 94, s95 , . . . , s177) ← (IV1 , . . . , IV80 , 0, . . . , 0)
(s 178 , s279 , . . . , s288) ← (0, . . . , 0, 1, 1, 1)
for i = 1 to 4 · 288 do
    t1 ← s66 + s91 · s92 + s93 + s171
    t2 ← s162 + s175 · s176 + s177 + s264
    t3 ← s243 + s286 · s287 + s288 + s69
    (s1 , s2 , . . . , s93) ← (t3, s1 , . . . , s92)
    (s94 , s95 , . . . , s177) ← (t1 , s94, . . . , s176)
    (s178 , s279 , . . . , s288) ← (t2 , s178 , . . . , s287)
end for

My Key and IV setup implemention:

```
 9  class Trivium:
10      def __init__(self, key, iv):
11          """in the beginning we need to transform the key as well as the IV.
12          Afterwards we initialize the state."""
13          self.state = None
14          self.counter = 0
15          self.key = key
16          self.iv = iv
17          '''
18          <Key and IV setup>
19          288-bit Initialize state
20              (s 1 , s2 , . . . , s93 ) <- (K1, . . . , K80 , 0, . . . , 0)
21              (s 94, s95 , . . . , s177) <- (IV1 , . . . , IV80 , 0, . . . , 0)
22              (s 178 , s279 , . . . , s288) <- (0, . . . , 0, 1, 1, 1)
23          '''
24          # bit 1 -> 93
25          init_list = list(map(int, list(self.key)))
26          init_list += list(repeat(0, 13))
27          # bit 94 -> 177
28          init_list += list(map(int, list(self.iv)))
29          init_list += list(repeat(0, 4))
30          # bit 178 -> 288
31          init_list += list(repeat(0, 108))
32          init_list += list([1, 1, 1])
33          self.state = deque(init_list)
34
35          # Do 4 full cycles, drop output
36          for i in range(4 * 288):
37              self._gen_keystream()
```

2. Key stream generation

Pseudo-code:

for i = 1 to N do
    t1 ← s66 + s93
    t2 ← s162 + s177
    t3 ← s243 + s288
    zi ← t1 + t2 + t3
    t1 ← t1 + s91 · s92 + s171
    t2 ← t2 + s175 · s176 + s264
    t3 ← t3 + s286 · s287 + s69
    (s1 , s2 , . . . , s93) ← (t3, s1 , . . . , s92)
    (s94 , s95 , . . . , s177) ← (t1 , s94, . . . , s176)
    (s178 , s279 , . . . , s288) ← (t2 , s178 , . . . , s287)
end for

My key stream generation implemention code:

```
45      def _gen_keystream(self):
46          '''
47          <Key stream generation>
48          for i = 1 to N do
49              t1 <- s66 + s93
50              t2 <- s162 + s177
51              t3 <- s243 + s288
52              zi <- t1 + t2 + t3
53              t1 <- t1 + s91 · s92 + s171
54              t2 <- t2 + s175 · s176 + s264
55              t3 <- t3 + s286 · s287 + s69
56              (s1 , s2 , . . . , s93 ) <- (t3, s1 , . . . , s92)
57              (s94 , s95 , . . . , s177 ) <- (t1 , s94, . . . , s176 )
58              (s178 , s279 , . . . , s288) <- (t2 , s178 , . . . , s287 )
59          end for
60          '''
61          t_1 = self.state[65] ^ self.state[92]
62          t_2 = self.state[161] ^ self.state[176]
63          t_3 = self.state[242] ^ self.state[287]
64
65          z = t_1 ^ t_2 ^ t_3
66
67          t_1 = t_1 ^ self.state[90] & self.state[91] ^ self.state[170]
68          t_2 = t_2 ^ self.state[174] & self.state[175] ^ self.state[263]
69          t_3 = t_3 ^ self.state[285] & self.state[286] ^ self.state[68]
70
71          self.state.rotate() #1 positive rotation
72
73          self.state[0] = t_3
74          self.state[93] = t_1
75          self.state[177] = t_2
76
77          return z
```

3. Encrypt and Decrypt function:

Encrypt func:

```
114  def encrypt(input, output):
115      key = get_random_bits(80)
116      iv = get_random_bits(80)
117      plain = get_bytes_from_file(input)
118      print("[+] Plain: ", plain)
119      trivium = Trivium(key, iv)
120      keystream = trivium.keystream_1(len(plain) * 8)
121      print("[+] IV in hex:  {}".format(bits_to_hex(iv)))
122      print("[+] Key in hex: {}".format(bits_to_hex(key)))
123      print("[-] Keystream in hex: {}".format(keystream))
124      cipher = trivium.encrypt(plain, keystream)
125      print("[-] Cipher: {}".format(cipher.hex()))
126      print(cipher)
127      with open(output, "wb") as output_file:
128          # 80 first bits of the output file is iv
129          output_file.write(iv)
130          output_file.write(cipher)
```

After encrypt, the program writes 80-bit of IV and cipher to the output file.

Decrypt func:

```
132  def decrypt(input, output, key):
133      with open(input, "rb") as input_file:
134          #80 first bits of the input file is iv
135          iv = bytearray(input_file.read(80))
136          #the rest is cipher
137          cipher = bytes(input_file.read())
138      print("[+] Cipher in bytes: ", cipher)
139      trivium = Trivium(key, iv)
140      keystream = trivium.keystream_1(len(cipher) * 8)
141      print("[+] IV in hex:  {}".format(bits_to_hex(iv)))
142      print("[+] Key in hex: {}".format(bits_to_hex(key)))
143      print("[-] Keystream in hex: {}".format(keystream))
144      plain = trivium.decrypt(cipher, keystream)
145      print("[-] Plain: {}".format(plain))
146      if (output):
147          with open(output, "wb") as output_file:
148              output_file.write(plain)
```

80 first bits of the cipher file is IV. Decrypt func takes secret key from user input and generate keystream in order to decrypt the ciphertext.

END