Introduction to Cryptography: Homework 2

Alvin Lin

January 2018 - May 2018

Exercise 1

Assume the initialization vector and the key of Trivium each consist of 80 all-zero bits. Compute the first 70 bits $s_1, \ldots, s_{70}$ during the warm-up phase of Trivium. Note that these are only internal bits which are not used for encryption since the warm-up phase lasts for 1152 clock cycles.
$s_1, \ldots, s_{70} =$ `01100000000000000000000000000000000000000000000000000000000000000000000000000110`
Code used:

```
def trivium_cycle(a, b, c):
  assert(len(a) == 93)
  assert(len(b) == 84)
  assert(len(c) == 111)
  assert(type(a) is deque)
  assert(type(b) is deque)
  assert(type(c) is deque)

  a_out = a[65] ^ a[92] ^ (a[90] & a[91])
  b_out = b[68] ^ b[83] ^ (b[81] & b[82])
  c_out = c[65] ^ c[110] ^ (c[108] & c[109])

  a_in = a[68] ^ c_out
  b_in = b[77] ^ a_out
  c_in = c[86] ^ b_out

  a.rotate(1)
  b.rotate(1)
  c.rotate(1)

  a[0] = a_in
  b[0] = b_in
  c[0] = c_in

  return a_out ^ b_out ^ c_out
```

Exercise 2

A linear feedback shift register (LFSR) is used to generate a keystream for a shift cipher. The LSFR has five bits $(s_4 s_3 s_2 s_1 s_0)$, the feedback bit is given by the formula $s_3 + s_0 (\mod 2)$, and the sequence of s_0 values forms the keystream. The LFSR is initialized with the $(s_4 s_3 s_2 s_1 s_0) = (11011)$. How many keystream bits will be generated before the keystream starts repeating? What is the sequence of keystream bits?

Clock	s_4	s_3	s_2	s_1	$s_0 = k$
0	1	1	0	1	1
1	0	1	1	0	1
2	0	0	1	1	0
3	0	0	0	1	1
4	1	0	0	0	1
5	1	1	0	0	0
6	1	1	1	0	0
7	1	1	1	1	0
8	1	1	1	1	1
9	0	1	1	1	1
10	0	0	1	1	1
11	1	0	0	1	1
12	1	1	0	0	1
13	0	1	1	0	0
14	1	0	1	1	0
15	0	1	0	1	1
16	0	0	1	0	1
17	1	0	0	1	1

10 bits will be generated before the keystream begins to enter a repeating sequence. This LSFR has a keystream sequence of 110011.

Exercise 3

Alex and Blake are encrypting messages using RC4. You, Harry the Hacker, are eavesdropping on their communications. Each plaintext message is a sequence of characters; each character is represented as an 8-bit binary number using the ASCII character encoding. Alex and Blake are using the same key to encrypt every message. Because RC4 does not define how to incorporate a nonce into the keystream generator algorithm, Alex and Blake are using this (insecure!) scheme: Generate the keystream using the (fixed) key, then add (mod 256) the nonce to each byte of the keystream. You happen to know that when Alex sent the plaintext `BARACKOBAMA` with a nonce of 1, the ciphertext was:
`01000011 00011011 00010010 00110000 11111000 10100111 10001110 11101001 00010100 00011101 01100100`
You now observe Blake send the following ciphertext with a nonce of 2: `01000110 00010100 00001111 00110011 11110000 10101001 10010110 11111110 00000011 00011100 01110110`
What is the plaintext of Blake's message? Explain how you found the plaintext. Your answer must be a narrative description, not code or pseudocode.

The first thing we need to do is figure out the keystream. We can convert the known plaintext into its ASCII representation in bits. Because of the properties of XOR, if $a \oplus b = c$, then it is also true that $a \oplus c = b$. Thus, can XOR the bits of the known plaintext with the bits of the known ciphertext output to get the keystream nonce combination that was used to encrypt it.

Plaintext ASCII	B	A	R	A	C	K	O	B	A	M	A
Plaintext bits	01000010	01000001	01010010	01000001	01000011	01001011	01001111	01000010	01000001	01001101	01000001
Ciphertext bits	01000011	00011011	00010010	00110000	11111000	10100111	10001110	11101001	00010100	00011101	01100100
Keystream + Nonce	00000001	01011010	01000000	01110001	10111011	11101100	11000001	10101011	01010101	01010000	00100101

Since we know that a nonce of 1 was added to each byte for this specific message, we can subtract that from each byte of the keystream to get the fixed key. We don't need to do this since we know the second

message was encrypted using a nonce of two. The keystream used to encrypt the second message can be obtained merely by adding 1 (mod 256) to the first keystream.

Fixed Key + 1	00000001	01011010	01000000	01110001	10111011	11101100	11000001	10101011	01010101	01010000	00100101
Fixed Key + 2	00000010	01011011	01000001	01110010	10111100	11101101	11000010	10101100	01010110	01010001	00100110

We can now decrypt the second ciphertext using this new keystream. We can XOR each bit of the new keystream with each bit of the second ciphertext to get each bit of the plaintext.

Keystream	00000010	01011011	01000001	01110010	10111100	11101101	11000010	10101100	01010110	01010001	00100110
Ciphertext bits	01000110	00010100	00001111	00110011	11110000	10101001	10010110	11111110	00000011	00011100	01110110
Plaintext bits	01000100	01001111	01001110	01000001	01001100	01000100	01010100	01010010	01010101	01001101	01010000

Once we have the bits of the plaintext, we can simply convert each byte back into ASCII to get the readable plaintext.

Plaintext bits	01000100	01001111	01001110	01000001	01001100	01000100	01010100	01010010	01010101	01001101	01010000
Plaintext ASCII	D	O	N	A	L	D	T	R	U	M	P

Blake's message was `DONALDTRUMP`.

If you have any questions, comments, or concerns, please contact me at alvin@omgimanerd.tech