ECE 4802, Project 2

Calvin Figuereo-Supraner

November 8 2016

Versioning

```
$ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.2) 5.4.0 20160609
$ python3 --version
Python 3.5.2
$ julia --version
julia version 0.4.5
```

Usage

```
$ ./q1c.jl
$ ./q2b.jl
$ gcc q2c.c && ./a.out
$ gcc q3.c && ./a.out
$ ./q4.py
```

Packages

The program `q4.py` requires the packages `pycrypto` and `joblib`, installed as below.

```
$ pip3 install pycrypto==2.6.1 joblib==0.10.3
```

Problem 1

1a

XOR the plaintext and ciphertext to obtain the LFSR sequence. The sequence length is 7, so the degree is $L = 3$.

```
pt    1001001 0011011 0110010 0100110
ct    1011110 0001100 0100101 0110001
-------------------------------------
lfsr  0010111 0010111 0010111 0010111
```

1b

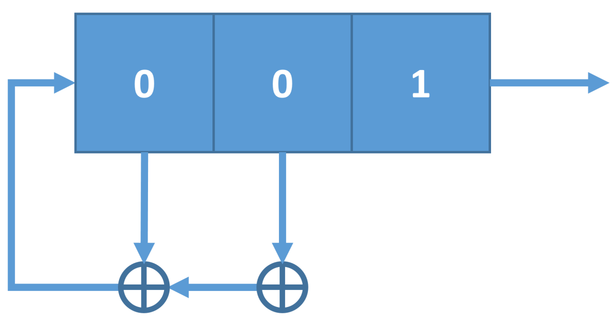
The first $L = 3$ bits of the LFSR sequence are the initialization vector $\{0, 0, 1\}$.

1c

Find the LFSR coefficients by solving the linear system below, created from the first 6 bits of the LFSR sequence. The script `q1c.jl` returns $\{1, 1, 0\}$. The LSFR is also pictured below.

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

```
$ ./q1c.jl
[1.0,1.0,0.0]
```



1d

The key is the feedback coefficients of the LFSR. The initial content of the LFSR can't be used as a key, because you need to "warmup" the LFSR before the output has good statistical properties.

Problem 2

2a

XOR the plaintext and ciphertext headers to obtain the LFSR behavior. The first $m = 6$ bits are the initialization vector: $\{1, 1, 1, 1, 1, 1\}$.

```
W    P    I
101100111101000
J    5    A
010011111100000
---------------
111111000001000
```

2b

Find the LFSR coefficients by solving the linear system below, created from the first 12 bits of the LFSR sequence. The script `q2b.jl` returns $\{1, 1, 0, 0, 0, 0\}$.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_o \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

```
$ ./q2b.jl
[1.0,1.0,0.0,0.0,0.0,0.0]
```

2c

The program `q2c.c` outputs "WPIWOMBAT" as the plaintext.

2d

Wombats live in Australia and Tasmania.

2e

This was a known-plaintext attack.

Problem 3

3a

The program `q3.c` implements DES S-box S_2 using the function `out_S2()`. Verification is below.

```
$ gcc q3.c && ./a.out
problem 3a, verify DES S_2
x       out_S2(x)
4       8
8       6
24      12
56      9
...
```

3b

The program `q3.c` also computes the SAC for S_2, for $i = 0 \dots 5$. The SAC is fulfilled by S_2.

```
$ gcc q3.c && ./a.out
...
problem 3b, compute SAC
i       SAC_coef:
0       20      16      16      20
1       28      44      56      52
2       44      40      40      44
3       36      44      28      48
4       36      52      36      36
5       44      36      44      44
```

Problem 4

The script `q4.py` implements an exhaustive key search for DES. Verification is below.

```
$ ./q4.py
$ cat answer.txt
0x00000000c2445aee
```

Problem 5 - Bonus

The script `q4.py` can solve Problem 5 by changing line 11 to "`keyspace = 1 << 64`". However, it failed to find a key within a reasonable time frame. As a benchmark, the script takes 1.682 seconds to exhaust a keyspace of 2^{20}. We can find the upper and lower bound by assuming 1 or 31 leading zero bits respectively.

$$t_{exhaust} < 2^{63}\left(\frac{1.682\text{s}}{2^{20}}\right) = 468840\text{yr}$$
$$t_{exhaust} > 2^{33}\left(\frac{1.682\text{s}}{2^{20}}\right) = 3\text{h}49\text{m}$$

Source Code

q1c.jl

```
#!/usr/bin/env julia

b = [0; 1; 1]
A = [0 0 1;
     0 1 0;
     1 0 1]
println(abs(A\b))
```

q2b.jl

```
#!/usr/bin/env julia

b = [0; 0; 0; 0; 0; 1]
A = [1 1 1 1 1 1;
     1 1 1 1 1 0;
     1 1 1 1 0 0;
     1 1 1 0 0 0;
     1 1 0 0 0 0;
     1 0 0 0 0 0]
println(abs(A\b))
```

q2c.c

```
#include <stdio.h>
#include <stdint.h>

#define MSG_BITLEN 45   // 45 bits in the message
#define MSG_SYMLEN 9    // 9 symbols in the message
#define SYM_BITLEN 5    // 5 bits per symbol

#define LFSR_DEGREE 6
#define LFSR_INIT 0x3f

const uint8_t bit_rev[32] = {
    0x00, 0x10, 0x08, 0x18, 0x04, 0x14, 0x0c, 0x1c,
    0x02, 0x12, 0x0a, 0x1a, 0x06, 0x16, 0x0e, 0x1e,
    0x01, 0x11, 0x09, 0x19, 0x05, 0x15, 0x0d, 0x1d,
    0x03, 0x13, 0x0b, 0x1b, 0x07, 0x17, 0x0f, 0x1f
};

uint8_t stob(char s)
{
    return (s >= 'A') ? s - 65: s - 22;
}

char btos(uint8_t b)
{
    return (b < 26) ? b + 65: b + 22;
}

int main(void)
{
    /* Generate a sufficiently-long LFSR sequence */
    uint64_t lfsr = LFSR_INIT; // sequence
    uint64_t bit;
    uint8_t state = LFSR_INIT;
    int offset = LFSR_DEGREE;
    do {
        bit = (state ^ (state >> 1)) & 1; // coefs: {1, 1, 0, 0, 0, 0}
        state = (state >> 1) | (bit << (LFSR_DEGREE-1));
        lfsr += (bit << offset++);
    } while (offset < MSG_BITLEN);

    /* Convert the ciphertext into plaintext */
    char ct_s[MSG_SYMLEN] = "J5A0EDJ2B";
    char pt_s[MSG_SYMLEN];
    uint64_t ct_b = 0;
    uint64_t pt_b;
    for (int i=0; i<MSG_SYMLEN; i++) {
        ct_b += (uint64_t) bit_rev[stob(ct_s[i])] << i*SYM_BITLEN;
        pt_b = ct_b ^ lfsr; // sub-optimal :(
        pt_s[i] = btos(bit_rev[pt_b >> (i*SYM_BITLEN) & 0x1f]);
        printf("%c", pt_s[i]);
    }
    puts("");

    return 0;
}
```

q3c.c

```
#include <stdio.h>
#include <stdlib.h>

#define LBIT  0b00100000
#define MBITS 0b00011110
#define RBIT  0b00000001

#define OUTER2(x) ((x & LBIT) >> 4 + (x & RBIT))
#define INNER4(x) ((x & MBITS) >> 1)

static const char S2[4][16] =
{
    {15,  1,  8, 14,  6, 11,  3,  4,  9,  7,  2, 13, 12,  0,  5, 10},
    { 3, 13,  4,  7, 15,  2,  8, 14, 12,  0,  1, 10,  6,  9, 11,  5},
    { 0, 14,  7, 11, 10,  4, 13,  1,  5,  8, 12,  6,  9,  3,  2, 15},
    {13,  8, 10,  1,  3, 15,  4,  2, 11,  6,  7, 12,  0,  5, 14,  9}
};

char out_S2(char x)
{
    return S2[OUTER2(x)][INNER4(x)];
}

int main(void)
{
    printf("problem 3a, verify DES S_2\n");

    printf("x\tout_S2(x)\n");
    printf("%d\t%d\n",  4, out_S2( 4));
    printf("%d\t%d\n",  8, out_S2( 8));
    printf("%d\t%d\n", 24, out_S2(24));
    printf("%d\t%d\n", 56, out_S2(56));

    printf("problem 3b, compute SAC\n");

    int i, j, k, sum, SAC_coef[6][4] = {0};

    /* fill SAC_coef */
    for (i = 0; i < 64; i++) {
        for (j = 0; j < 6; j++) {
            sum = out_S2(i) ^ out_S2(i ^ (1 << j));
                for (k = 0; k < 4; k++)
                    SAC_coef[j][k] += (sum >> k) & 1;
        }
    }

    /* print SAC_coef */
    printf("i\tSAC_coef:\n");
    for (i = 0; i < 6; i++) {
        printf("%d\t", i);
        for (j = 0; j < 4; j++) {
            printf("%d\t", SAC_coef[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

q4.py

```
#!/usr/bin/env python3

from Crypto.Cipher import DES
from joblib import Parallel, delayed
import binascii
import struct

pt = binascii.unhexlify('48656c6c6f212121')
ct = binascii.unhexlify('d52bd481f21e25a1')

keyspace = 1 << 32
n_cpus = 8
chunk_size = int(keyspace/n_cpus)

def check(cpu):
    for k in range(chunk_size * cpu, chunk_size * (cpu+1)):
        if DES.new(struct.pack('>Q',k), DES.MODE_ECB).encrypt(pt) == ct:
            return k

keys = Parallel(n_jobs=n_cpus)(delayed(check)(cpu) for cpu in range(n_cpus))

for k in keys:
    if k is not None:
        with open('answer.txt', 'w') as f:
            f.write(str(hex(k))+'\n')
        break
```