

COLLEGE OF COMPUTER AND INFORMATION SCIENCE
NORTHEASTERN UNIVERSITY
CSG252: CRYPTOGRAPHY AND COMMUNICATION SECURITY
FALL 2005
PROBLEM SET 3 - SOLUTIONS
PREPARED BY TIM MORGAN

Problem 1

(a) [5 points]:

N_L table for $\pi_{S'}$:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
1	8	10	6	8	10	8	8	6	4	6	6	8	10	8	4	10
2	8	10	8	10	6	8	6	8	6	8	10	4	4	6	8	10
3	8	8	10	10	8	12	10	6	6	6	8	8	10	6	12	8
4	8	10	8	6	8	10	8	6	10	4	10	8	6	8	6	4
5	8	12	6	6	10	10	8	12	6	10	8	8	8	8	10	6
6	8	8	12	8	10	10	6	10	8	8	8	12	6	6	6	10
7	8	6	6	8	12	6	10	8	8	6	6	8	4	6	10	8
8	8	10	10	8	8	6	6	8	10	8	4	6	10	4	8	6
9	8	8	8	12	10	10	6	10	10	6	6	6	8	12	8	8
A	8	12	10	10	6	6	12	8	8	8	6	10	6	10	8	8
B	8	6	12	6	8	6	8	10	4	6	8	6	8	10	8	6
C	8	8	10	10	12	8	10	6	8	12	10	6	8	8	6	6
D	8	6	8	6	6	12	10	8	8	10	4	6	6	8	6	8
E	8	6	6	12	6	8	8	10	6	8	8	10	8	6	6	4
F	8	8	8	8	8	8	12	12	10	6	10	6	10	6	6	10

(b) [8 points]:

Trace a path through the SPN starting with X_{16} , through S_4^1 , then out through V_{13}^1 to S_1^2 , through V_1^2 to S_1^3 and finally out of that sbox via V_1^3 and V_3^3 to U_1^4 and U_9^4 respectively. For the three active sboxes, observe that if the variables T_1 , T_2 , and T_3 are defined as:

$$\begin{aligned}
 T_1 &= U_{16}^1 \oplus V_{13}^1 \\
 T_2 &= U_4^2 \oplus V_1^2 \\
 T_3 &= U_1^3 \oplus V_1^3 \oplus V_3^3
 \end{aligned}$$

Then the bias of each of these, based on our N_L table, is:

$$\begin{aligned}
T_1 &= \frac{4-8}{16} = -\frac{1}{4} \\
T_2 &= \frac{4-8}{16} = -\frac{1}{4} \\
T_3 &= \frac{4-8}{16} = -\frac{1}{4}
\end{aligned}$$

And with the piling-up lemma, we can calculate the bias of:

$$bias(T_1 \oplus T_2 \oplus T_3) = (2^{3-1}) \cdot (-\frac{1}{4})^3 = -\frac{1}{16}$$

Simplifying the summation of these variables gives:

$$\begin{aligned}
T_1 \oplus T_2 \oplus T_3 &= U_{16}^1 \oplus V_{13}^1 \oplus U_4^2 \oplus V_1^2 \oplus U_1^3 \oplus V_1^3 \oplus V_3^3 \\
&= K_{16}^1 \oplus X_{16} \oplus K_4^2 \oplus U_4^2 \oplus U_4^2 \oplus K_1^3 \oplus U_1^3 \oplus U_1^3 \oplus K_1^4 \oplus U_1^4 \oplus K_9^4 \oplus U_9^4 \\
&= K_{16}^1 \oplus K_4^2 \oplus K_1^3 \oplus K_1^3 \oplus K_9^4 \oplus X_{16} \oplus U_1^4 \oplus U_9^4
\end{aligned}$$

Since the bias of this whole expression is $-\frac{1}{16}$, and the unknown key bits are fixed for a given key, the bias of $(X_{16} \oplus U_1^4 \oplus U_9^4)$ is $\pm\frac{1}{16}$.

(c) [12 points]:

(Taken from standard text solutions)

```

Algorithm: LINEARATTACK( $\mathcal{T}, T, \pi_{S'}^{-1}$ )
for  $(L_1, L_2) \leftarrow (0, 0)$  to  $(F, F)$ 
  do  $Count[L_1, L_2] \leftarrow 0$ 
  for each  $(x, y) \in \mathcal{T}$ 
    for  $(L_1, L_2) \leftarrow (0, 0)$  to  $(F, F)$ 
      do
         $\begin{cases} v_{(1)}^4 \leftarrow L_1 \oplus y_{(1)} \\ v_{(3)}^4 \leftarrow L_2 \oplus y_{(3)} \\ u_{(1)}^4 \leftarrow \pi_{S'}^{-1}(v_{(1)}^4) \\ u_{(3)}^4 \leftarrow \pi_{S'}^{-1}(v_{(3)}^4) \\ z \leftarrow x_{16} \oplus u_1^4 \oplus u_9^4 \\ \text{if } z = 0 \\ \text{then } Count[L_1, L_2] \leftarrow Count[L_1, L_2] + 1 \end{cases}$ 
     $max \leftarrow -1$ 
    for  $(L_1, L_2) \leftarrow (0, 0)$  to  $(F, F)$ 
       $Count[L_1, L_2] \leftarrow |Count[L_1, L_2] - T/2|$ 
      do
         $\begin{cases} \text{if } Count[L_1, L_2] > max \\ \text{then } \begin{cases} max \leftarrow Count[L_1, L_2] \\ maxkey \leftarrow (L_1, L_2) \end{cases} \end{cases}$ 
    output  $(maxkey)$ 

```

(d) [25 points]:

An implementation in C by Tim Morgan is included below. This attack typically requires the expected number of plaintext/ciphertext pairs (1000-1500), but it was found that against certain random pairs that the algorithm converged to the correct key in as few as 92 pairs.

```

/*
 * A linear cryptanalytic attack against the modified 3.1 cryptosystem.
 * Implemented in C. Expects two arguments: the number of pairs to read
 * from a binary file, followed by the name of the binary file to read
 * from. This program expects the input file to contain one binary
 * plaintext followed by it's corresponding binary ciphertext, repeated
 * for all pairs.
 */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <unistd.h>
#include <errno.h>
#include <math.h>

int MAX_KEY = 256;
unsigned char inverse_sbox[16] = {15, 3, 2, 6, 1, 9, 5, 11, 0, 14, 8,
                                13, 4, 7, 10, 12};

unsigned int readPairs(int fh, unsigned short* pt,
                      unsigned short* ct,
                      unsigned int length);
void buildTallies(unsigned int* tallies,
                 unsigned short* pt,
                 unsigned short* ct,
                 unsigned int num_pairs);
bool testKey(unsigned char, short pt, short ct);
unsigned char findBestKey(unsigned int* tallies,
                          unsigned int num_pairs);

int main(int argc, char** argv)
{
    unsigned int num_pairs;
    unsigned char best_key;
    unsigned short* plaintext;
    unsigned short* ciphertext;
    unsigned int* tallies;
    int fh;

    if(argc < 3)
    {
        printf("ERROR: 2 arguments required.\n");
        exit(1);
    }

    num_pairs = atoi(argv[1]);
    plaintext=(unsigned short*)malloc(sizeof(unsigned short)*num_pairs);
    ciphertext=(unsigned short*)malloc(sizeof(unsigned short)*num_pairs);
    tallies = (unsigned int*)malloc(sizeof(unsigned int)*MAX_KEY);

```

```

if(plaintext == NULL || ciphertext == NULL || tallies == NULL)
{
    printf("ERROR: could allocate enough memory.\n");
    exit(2);
}

fh = open(argv[2], O_RDONLY);
if(fh == -1)
{
    printf("ERROR: could not open file.\n");
    exit(3);
}

if(readPairs(fh, plaintext, ciphertext, num_pairs) < num_pairs)
{
    printf("ERROR: could not read %d pairs from file.\n", num_pairs);
    exit(4);
}
close(fh);

buildTallies(tallies, plaintext, ciphertext, num_pairs);
best_key = findBestKey(tallies, num_pairs);
printf("Most likely key bits:\n K5(1) = %X, K5(3) = %X\n",
       (best_key >> 4) & 0x0F, best_key & 0x0F);

return 0;
}

void buildTallies(unsigned int* tallies,
                 unsigned short* pt,
                 unsigned short* ct,
                 unsigned int num_pairs)
{
    int i,j;

    for(j=0; j < MAX_KEY; j++)
        tallies[j] = 0;

    for(i=0; i < num_pairs; i++)
        for(j=0; j < MAX_KEY; j++)
            tallies[j] += (1 ^ testKey(j, pt[i], ct[i]));
}

unsigned int readPairs(int fh,
                      unsigned short* pt,
                      unsigned short* ct,
                      unsigned int length)
{
    unsigned int i=0;
    unsigned int retlen=4;
    int tmp[1];
    int total=0;

```

```

for(i=0; (i < length) && (retlen == 4); i++)
{
    retlen = read(fh, tmp, 4);
    total += retlen;
    pt[i] = *tmp & 0x0000FFFF;
    ct[i] = (*tmp >> 16) & 0x0000FFFF;
}

if(total < 0)
{
    printf("errno: %d", errno);
    total = 0;
}

return (unsigned int)(total/4);
}

bool testKey(unsigned char k5, short pt, short ct)
{
    unsigned char x16, y_1, y_3;
    unsigned char k5_1, k5_3;
    unsigned char v4_1, v4_3;
    unsigned char u4_1, u4_3;
    unsigned char ret_val;

    x16 = pt & 1;
    y_1 = (ct >> 12) & 0x0F;
    y_3 = (ct >> 4) & 0x0F;
    k5_1 = (k5 >> 4) & 0x0F;
    k5_3 = k5 & 0x0F;

    v4_1 = k5_1 ^ y_1;
    v4_3 = k5_3 ^ y_3;
    u4_1 = inverse_sbox[v4_1];
    u4_3 = inverse_sbox[v4_3];

    ret_val = x16;
    ret_val ^= (u4_1 >> 3) & 1;
    ret_val ^= (u4_3 >> 3) & 1;

    return ret_val;
}

unsigned char findBestKey(unsigned int* tallies, unsigned int num_pairs)
{
    int i;
    unsigned char best;
    unsigned int new_diff, best_diff = 0;

    for(i = 0; i<MAX_KEY; i++)
    {
        new_diff = abs(tallies[i]-(int)(num_pairs/2));
        if (best_diff < new_diff)
        {

```

```
        best_diff = new_diff;
        best = i;
    }
}

return best;
}
```