YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CPSC 467a: Cryptography and Computer Security *Handout #10*

Felipe Saint-Jean *October 10, 2005*

Solutions to Problem Set 2

In the problems below, "textbook" refers to *Introduction to Cryptography with Coding Theory: Second Edition* by Trappe and Washington..

Problem 7: Simplified CFB Mode

Textbook, problem 4.9.9.

Solution:

part a

The encryption is

$$C_j = P_j \oplus L_{32}(E_K(X_j))$$

with

$$X_{j+1} = R_{32}(X_j)||C_j$$

Therefore to decrypt we need to generate the X_j sequence and a P_j based on known values. At step j we know C_j so the X_j generation stays the same:

$$X_{j+1} = R_{32}(X_j)||C_j$$

Because X_1 is given we can generate all X_j. Using the xor properties we can solve for P_j and get the decryption algorithm for P_j as

$$P_j = C_j \oplus L_{32}(E_k(X_j))$$

part b

Let's start by unfolding the decryption algorithm

$$\tilde{P}_1 = \tilde{C}_1 \oplus L_{32}(E_k(X_1)) \quad (1)$$
$$\tilde{X}_2 = R_{32}(X_1)||\tilde{C}_1 \quad (2)$$
$$\tilde{P}_2 = C_2 \oplus L_{32}(E_k(\tilde{X}_2)) \quad (3)$$
$$\tilde{X}_3 = R_{32}(\tilde{X}_2)||C_2 \qquad = \tilde{C}_1||C_2 \quad (4)$$
$$\tilde{P}_3 = C_3 \oplus L_{32}(E_k(\tilde{X}_3)) \quad (5)$$
$$X_4 = R_{32}(\tilde{X}_3)||C_3 \qquad = C_2||C_3 \quad (6)$$
$$P_4 = C_4 \oplus L_{32}(E_k(X_4)) \quad (7)$$

The key point to notice that X_4 and P_4 are correct even though $\tilde{X}_3$ is broken $R_{32}(\tilde{X}_3) = C_2$ is correct because C_2 is correct. $\tilde{P}_3$ is broken because $\tilde{X}_3 \neq X_3$ therefore $E_K(\tilde{X}_3) \neq E_K(X_3)$.

Problem 8: DES Brute Force Speedup

Textbook, problem 4.9.11.

Solution:

In the general case to search for a 56 bit key we need to try all 2^{56} possibilities. If $C_1 = E_{K^*}(M_1)$ and $C_2 = E_{K^*}(\bar{M_1})$, where K^* is the key we are looking for, then in the p-th attempt to find the key if $E_{K_p}(M_1) = C_1$ then we know the key is K_p. If $\bar{C_2} = E_{K_p}(M_1)$ then we know that the key is $\bar{K_p}$ because of the complementation property. This allows an attacker to search only in half of the complete key space since he has only to look in the "uncomplemented" half. An easy way to do that is to fix say the first bit to 0. Then every key with the first bit 1 is the complement of a key with first bit 0 reducing the search space to a half.

Problem 9: Birthday Paradox Calculation

Write a computer program to compute p_n, the probability that at least two people in a random collection of n people have the same birthday. Ignore leap years and assume the probability of a person's birthday falling on any given day is exactly $1/365$, independent of everyone else in the set. Your program should work for n in the range $[1, 365]$. Using your program, find the smallest value of n for which $p_n \geq 1/2$ and for which $p_n \geq 3/4$.

Solution:

The simplest way to think the probability is by actually thinking of the complement. What is the probability of not having two people with the same birthday in a room with n persons. That probability is

$$1 - p_n = 1 \cdot \frac{365-1}{365} \cdot \frac{365-2}{365} ... \cdot \frac{365-n+1}{365} = \prod_{i=0}^{n-1} \frac{365-i}{365}$$

The intuitive argument is that the first person can have his birthday any day without colliding, so his probability is $\frac{365}{365} = 1$. The second guy can have his birthday any day except the first guy's birthday so his probability of not colliding is $\frac{365-1}{365}$ and so on.

Then the probability of two or more persons having their birthday the same day is

$$p_n = 1 - \prod_{i=0}^{n-1} \frac{365-i}{365}$$

because it is the complementary event. For $n = 23$ the probability is slightly higher than $\frac{1}{2}$. For $n = 32$ it is just more than $\frac{3}{4}$.

n	$prob$
22	0.475695
23	0.507297
31	0.730455
32	0.753348

Program p9.c

```
#include <stdio.h>
double prob(int n){
    // This computes the probability that
    // there is no two persons with the same Bday
    int i;
```

```
    double prob = 1;
    for (i=0;i<n;i++){
        prob *= (365.0-i)/365.0;
    }
    return 1.0 - prob;
}


int main(int argc, char **argv){

    int n;
    n=1;
    for(n=1;prob(n)<0.5;n++);
    printf("for n=%d prob=%f\n",n,prob(n));

    for(n=1;prob(n)<0.75;n++);
    printf("for n=%d prob=%f\n",n,prob(n));
}
```

Problem 10: Simplified DES Implementation

Textbook, problem 4.10.1.

Solution:

part a

See code at the end.

part b

For the key 011001011 and plain text 011100100110 the encryptions are

C_1	100110011000
C_2	011000000010
C_3	000010111111
C_4	111111111100

part c

A weak key is a key in which $\forall M E_K(E_K(M)) = M$. To prove there are no weak keys we need to find a message M_K for each key K s.t. $E_K(E_K(M_K)) \neq M_k$. Even though finding a single M s.t. $E_K(E_K(M)) \neq M$ proves the statement above, it is a stronger statement and it might not be true (although it is possible to do so).

part d

By swapping L and T after the forth stage two E_K functions applied one after the other so encryption and decryption become exactly the same function except for the key ordering. So any key s.t.

$$K_1 = K_4 \quad (8)$$

$$K_2 = K_3 \quad (9)$$

$$(10)$$

will be a weak key. Thus 000000000 and 111111111 are weak keys with the given key generating scheme.

Program p10.c

```
#include <stdio.h>

#define assert(x) if(!(x)) fprintf(stderr,"Error in line %d file %s\n", __LINE__,__FILE__)

#define eval_box(b,x) b[(x)>>3][(x) & 0x7]


char *binString(int n, unsigned int mask, char *buff){
    int i;
    int top;
    for (top=31;!((1<<top) &mask);top--);
    top++;
    for (i=0;i<top;i++){
        buff[top-i-1] =(1<<i)&n?'1':'0';
    }
    buff[top]=0;
    return buff;

}


unsigned char s1[2][8] = {{5,2,1,6,1,4,7,0},
         {1,4,6,2,0,7,5,3}};

unsigned char s2[2][8] = {{4,0,6,5,7,1,3,2},
        {5,3,0,7,6,2,1,4 }};



unsigned char expand(unsigned char c){
    assert(c<64);
    return ((c & 0x30) << 2)| ((c & 0x4) << 3) | ((c & 0xc)<<1)|((0x9 & c) >> 1 ) | (c & 0x3);
}


unsigned char keyRound(unsigned int K, int round){
    unsigned int mask = (1<<(9-round+1))-1;
    unsigned char ret;
    if (round > 1)
        ret =  ((K & mask) << (round-2) | (K & (~mask))>>(9-round+2));
    else
        ret = K>>1;
    //printf("Key %X round %d\n",ret,round);
    return ret;
//    return  (K & mask) << (round-2); //| (K & (~mask))>>(9-round+1));
}
unsigned char f(unsigned char R, unsigned char K){
    unsigned char iv;
    iv = expand(R) ^ K;
    return (eval_box(s1,iv>>4) << 3) | eval_box(s2,iv & 0xf);
}

unsigned int encript(unsigned int pt, unsigned int K, int rounds){
    int i;
    unsigned char L,R,tmp;
    L = 0x3f & (pt >> 6);
    R = pt & 0x3f;
    for (i=1;i<=rounds;i++){
//        printf("1:L=%x R=%x key = %x\n",L,R,keyRound(K,i));
        tmp = (L ^ f(R,keyRound(K,i)));
        L = R;
        R=tmp;
//        printf("2:L=%x R=%x\n",L,R);
    }
    return (L<<6) | R;
```

```
}

unsigned int decript(unsigned int ct, unsigned int K, int rounds){
    int i;
    unsigned char L,R,tmp;
    // Right is L and Left is Right
    R = 0x3f & (ct >> 6);
    L = ct & 0x3f;
    for (i=rounds;i>=1;i--){
//        printf("1:L=%x R=%x\n",L,R);
        tmp = (L ^ f(R,keyRound(K,i)));
        L = R;
        R=tmp;
//        printf("2:L=%x R=%x\n",L,R);
    }
    return (R<<6) | L;
}
int readInt(){
    char buff[255];
    int i=0;
    char c;
    while(((c = getchar()) != '\n') && i++<255){
        buff[i-1]=c;
    }
    buff[i]=0;
    return atoi(buff);
}

unsigned int readBin(int bits){
    char c;
    int i;
    unsigned int ret = 0;
    for (i=bits-1;i>=0;i--){
        c = getchar();
        ret  = ret | (c == '1'?1<<i:0);

    }
    while (getchar()!='\n');
    return ret;
}

int main(int argc, char **argv){
    int weak_flag = 0;
    unsigned int i;
    unsigned int key = 0x99;
    unsigned int pt = 0xbee;
    unsigned int ct = 0x00;
    char buff[32];
    int rounds;
    printf("Key?");
    key = readBin(9);
    printf("Plaintext?");
    pt = readBin(12);
    printf("Key     %s\n",binString(key,0x1ff,buff));
    // part b
    for (i=1;i<=4;i++){
        printf("\nRounds %d\n",i);
        printf("Plaint text %X = %s\n",pt,binString(pt ,0xfff,buff));
        ct = encript(pt, key,i);
        printf("Cipher text %X = %s\n",ct,binString(ct ,0xfff,buff));
        pt = decript(ct, key,i);
        printf("Plaint text %X = %s\n",pt,binString(pt ,0xfff,buff));
    }

    // part c
    weak_flag = 0;
    for (i=0;i<(1<<9);i++){
        int weak = 0;
```

```
        for (pt=0;pt<(1<<12);pt++){
            if (pt != encript( encript(pt, i ,4), i ,4))
                break;

            if (pt == ((1<<12)-1))
                weak++;
        }
    }
    if (weak_flag)
            printf("Found a weak key");
    else
            printf("No weak key found");
}
```