2022 암호분석경진대회

3번 문제

사용자의 패스워드 정보는 시스템이 해킹되어 유출되는 경우에도 안전성을 보장하기 위해 시스템 내에 변환되어 저장되게 된다. 따라서 변환된 패스워드 정보로부터 패스워드를 찾아내기 위해서는 전수조사를 통해 모든 경우의 수를 확인해 봐야 한다. 하지만 모든 경우의 수를 계산하는 것은 매우 많은 시간이 걸리기 때문에 이를 고속화하는 연구가 활발히 진행되고 있다.

[문제]

아래에는 특정 패스워드 변환 기법에 대한 크래킹에 사용되는 레퍼런스 코드가 제시되어 있다. 해당 코드를 기반으로 고속으로 패스워드를 크래킹할 수 있는 코드를 8-비트 저전력 프로세서 상에서 구현하시오.

```
#pragma GCC optimize ("-O3")

// type definition
typedef unsigned char    u8;
typedef unsigned short   u16;

// rotation left function
#define ROL(X, Y) ((X<<Y) | (X>> (8-Y)))      // optimization
#define ROL_4BIT(X, Y) ((X<<Y) | (X>> (4-Y)))    // optimization

#define T1(B,C,D,E) (B & C) | (D & E)
#define T2(F,G,H)    (F ^ G ^ H)
#define DATA_SIZE 4
#define SIZE_TEXT 32

u8 init_constant[8]  = {0xAA, 0x11, 0xBB, 0x22, 0xCC, 0x33, 0xDD, 0x44};
u8 init_constant2[4] = {0x12, 0x34, 0x56, 0x78};

// https://eprint.iacr.org/2017/622.pdf (GIFT block cipher)
u32 s_box[16] = {0x1, 0xa, 0x4, 0xc, 0x6, 0xf, 0x3, 0x9, 0x2, 0xd, 0xb, 0x7, 0x5, 0x0, 0x8, 0xe};

void byte_permutation(u8* inout){
  u8 tmp;

  tmp = inout[7];
  inout[7] = inout[6];
  inout[6] = inout[5];
  inout[5] = inout[4];
  inout[4] = inout[3];
  inout[3] = inout[2];
  inout[2] = inout[1];
  inout[1] = inout[0];
  inout[0] = tmp;

}
```

```
void Func1(u8* in, u8* out){
  u32 i=0;
  u8 tmp1, tmp2;
  u8 func1_tmp[8] = {0,};

  //constant initialization
  for(i=0;i<8;i++){
    func1_tmp[i] = init_constant[i];
  }

  for (i = 0; i < 16; i++) {
    //computation
    tmp1 = T1(func1_tmp[1], func1_tmp[2], func1_tmp[3], func1_tmp[4]);
    tmp2 = T2(func1_tmp[5], func1_tmp[6], func1_tmp[7]);

    func1_tmp[0] = func1_tmp[0] + tmp1 + tmp2;

    func1_tmp[0] = func1_tmp[0] + in[i%4];

    func1_tmp[0] = ROL(func1_tmp[0],1);

    //permutation
    byte_permutation(func1_tmp);
  }

  for(i=0;i<4;i++){
    out[i] = func1_tmp[i] ^ func1_tmp[i+4];
  }
}

void byte_to_bit(u8* in, u8* out, u32 size_bit) {
  u32 size_byte = size_bit / 8;
  u32 i, j;
  u8 bit_selector = 1;

  for (i = 0; i < size_byte; i++) {
    bit_selector = 1;

    for (j = 0; j < 8; j++) {
      if (in[i] & bit_selector){
        out[i * 8 + j] = 1;
      }else{
        out[i * 8 + j] = 0;
      }
      bit_selector = bit_selector << 1;
    }
```

```
  }
}

void bit_to_byte(u8* in, u8* out, u32 size_bit) {
  u32 size_byte = size_bit / 8;
  u32 i, j;

  for (i = 0; i < size_byte; i++) {
    out[i] = 0;
    for (j = 0; j < 8; j++) {
      out[i] = out[i] + (in[i * 8 + j] << j);
    }

  }
}

void PERMUTATE_FUNC(u8* in, u8* out) {
  u8 text_tmp[SIZE_TEXT];
  u32 i, j;

  for (i = 0; i < 4; i++) {
    for (j = 0; j < 8; j++) {
      text_tmp[i * 8 + j] = in[i + j * 4];
    }
  }

  for (i = 0; i < SIZE_TEXT; i++) {
    out[i] = text_tmp[i];
  }
}

void Func2(u8* in, u8* out){
  u8 func2_tmp[4] = {0,};
  u8 func2_key[4] = {0,};

  u8 func2_bit_tmp1[32] = {0,};
  u8 func2_bit_tmp2[32] = {0,};

  u8 tmp1, tmp2;
  u8 cnt;
  int i,j=0;

  func2_key[0] = in[0]^init_constant2[0];
  func2_key[1] = in[1]^init_constant2[1];
  func2_key[2] = in[0]^init_constant2[2];
  func2_key[3] = in[1]^init_constant2[3];
```

```
  func2_tmp[0] = in[2];
  func2_tmp[1] = in[3];
  func2_tmp[2] = in[2];
  func2_tmp[3] = in[3];

  for(i=0;i<16;i++){
    cnt = i;

    for(j=0;j<4;j++){
      func2_key[j] = func2_key[j]^cnt;
      tmp1 = s_box[ (func2_tmp[j]&0xF) ];
      tmp2 = (s_box[ (( func2_tmp[j]>>4 ) & 0xF) ]) & 0xF;

      func2_tmp[j] = tmp1 + (tmp2<<4);
      func2_tmp[j] = func2_tmp[j] ^ func2_key[j];
    }
    byte_to_bit(func2_tmp,func2_bit_tmp1, DATA_SIZE * 8);
    PERMUTATE_FUNC(func2_bit_tmp1,func2_bit_tmp2);
    bit_to_byte(func2_bit_tmp2, func2_tmp, DATA_SIZE * 8);
  }

  out[0]=func2_tmp[0];
  out[1]=func2_tmp[1];
  out[2]=func2_tmp[2];
  out[3]=func2_tmp[3];
}

u8 Matching(u8* in1, u8* in2){
  int i;
  u8 flag=1;
  for (i=0;i<8;i++){
    if(in1[i]!=in2[i]){
      flag = 0;
      return flag;
    }
  }
  return flag;
}

void int_to_char(u32 in, u8* out){
  int i=0;
  out[0]=in%10;
  out[1]=(in%100)/10;
  out[2]=(in%1000)/100;
  out[3]=(in%10000)/1000;
  out[4]=(in%100000)/10000;
  out[5]=(in%1000000)/100000;
```

```
  out[6]=(in%10000000)/1000000;
  out[7]=(in%100000000)/10000000;

  for(i=0;i<8;i++){
    out[i] = out[i] + 48;
  }
}

u8 Cracking(u32 init_int, u8* password, u8* output, u8* answer){
  u8 check;
  int_to_char(init_int,password);
  Func1(password,    output);
  Func2(&password[4], &output[4]);
  check=Matching(output, answer);
  return check;
}

void setup() {
  Serial.begin(9600); // open the serial port at 9600 bps:
  pinMode(LED_BUILTIN, OUTPUT);

  u8 password[8]= {0,};
  u8 output[8] = {0,};
  u8 answer_test[8] = {0x26, 0xce, 0x59, 0x80, 0x98, 0x73, 0x64, 0x16};
  u32 init_int = 12345678;
  u32 i, j=0;
  u8 check=0;

  Serial.println("-----------------");
  Serial.println("    TEST VECTOR ");
  Serial.println("-----------------");

  //TEST VECTOR
  check = Cracking(init_int, password, output, answer_test);

  if(check){
    Serial.println(">> CORRECT");
  }else{
    Serial.println(">> WRONG");
  }

  // Warning: This is example. This test vector will be changed in evaluation.
  // password range: 00000000 ~ 99999999
  u8 answer_bench[8] = {0x2b, 0x46, 0x9d, 0x42, 0x7b, 0x8f, 0x11, 0x9e};

  Serial.println("-----------------");
  Serial.println("     BENCHMARK ");
```

```
  Serial.println("------------------");

  u32 time1;
  u32 time2;
  time1 = millis();

  init_int=0;
  for(i=0;i<99999999;i++){
    check = Cracking(init_int, password, output, answer_bench);
    if(check){
      Serial.print("Answer is ");
      Serial.println(init_int);
      break;
    }

    for(j=0;j<8;j++){        output[j] = 0;      }
    init_int++;
  }
  time2 = millis();
  Serial.print(">> ");
  Serial.println((time2-time1));
  Serial.println("------------------");
}

void loop() {

}
```

[주의사항]

1) 구현 타겟 플랫폼은 아두이노 우노 (8-비트 AVR 프로세서)이며 상세한 정보는 웹사이트를 참고하도록 한다. https://store.arduino.cc/usa/arduino-uno-rev3

2) 아래 벤치마크 코드에서 빨간 글자 부분을 제외하고는 모두 수정이 가능하다. 즉 레퍼런스 코드의 setup() 함수 내에서는 아래의 검은 글자 부분만 수정이 가능하다 (for문의 경우에도 내부 파라미터 수정 및 해당 위치가 아닌 다른 곳 (예를들어 Cracking 함수내)에 위치시키는 것도 허용한다. 현재 구현된 if문 형식의 Correctness 체크 구문의 경우에도 원하는 형식으로 변경가능하다. 단 출력되는 값은 레퍼런스 코드와 동일해야 한다.). **단 Cracking 함수의 경우 입력 인자와 출력 인자의 수와 형식에 대한 변경은 허용하지 않는다 (Cracking 함수 내부 수정은 허용한다.)**. setup()함수 외의 함수는 자유롭게 수정가능하다. 또한 개발 용이성을 위한 새로운 함수 추가도 허용한다.

```
  time1 = millis();
  init_int=0;
  for(i=0;i<99999999;i++){
    check = Cracking(init_int, password, output, answer_test);
    if(check){
      Serial.print("Answer is ");
      Serial.println(init_int);
```

```
        break;
    }

    for(j=0;j<8;j++){        output[j] = 0;      }
    init_int++;
  }
  time2 = millis();
```

3) 인라인 어셈블리 혹은 어셈블리 코드로 프로그래밍하는 경우에도 Cracking 함수의 내부 연산과 함수만을 수정하도록 한다.

4) 레퍼런스 코드 내의 변환 함수의 경우 암호학적으로 완전한 함수가 아닌 경량화된 함수이기에 충돌쌍이 발생할 수도 있다. 이 경우 충돌쌍 또한 정확한 값을 찾은 것으로 간주한다.

5) 프로그래밍은 Arduino IDE를 사용하도록 한다. 결과물은 (*.ino) 형식만을 허용한다.
https://www.arduino.cc/en/main/software

6) 결과물은 다음 2종을 포함한다.
- 아두이노 코드 (테스트 벡터 확인 과정, 벤치마크 과정)
- 문서 (구현 기법 상세, 테스트 벡터 확인 결과, 벤치마크 결과)

6) 평가방법은 다음과 같다.
- 테스트 벡터 통과 (30점, 절대 평가)
- 문서화 (30점, 절대평가)
- 벤치마크 결과 (40점, 상대평가): 벤치마크는 새로운 테스트 벡터를 실제 평가에서 적용한다. 즉 현재 코드에는 표기하지 않은 값을 통해 평가한다. 비밀번호 값은 8자리 숫자만을 받으며 값의 범위는 00000000 ~ 99999999 이다.