1.Diffie Hellman Algorithm

앨리스와 밥이 공개된 통신망에서 디피-헬만 키 교환을 하기 위해서는 다음과 같은 절차를 거친다.

1. 앨리스가 소수 p, 그리고 1부터 $p-1$까지의 정수 g를 선택하여 사전에 밥과 공유한다.
2. 앨리스가 정수 a를 선택한다. 이 정수는 외부에 공개되지 않으며, 밥 또한 알 수 없다.
3. 앨리스가 $A = g^a \bmod p$, 즉 g^a를 p로 나눈 나머지를 계산한다.
4. 밥이 마찬가지로 정수 b를 선택하여 $B = g^b \bmod p$를 계산한다.
5. 앨리스와 밥이 서로에게 A와 B를 전송한다.
6. 앨리스가 $B^a \bmod p$를, 밥이 $A^b \bmod p$를 계산한다.

마지막 단계에서 $B^a = (g^b)^a = g^{ab}$, $A^b = (g^a)^b = g^{ab}$ 이며 따라서 앨리스와 밥은 $g^{ab} \bmod p$라는 공통의 비밀 키를 공유하게 된다.

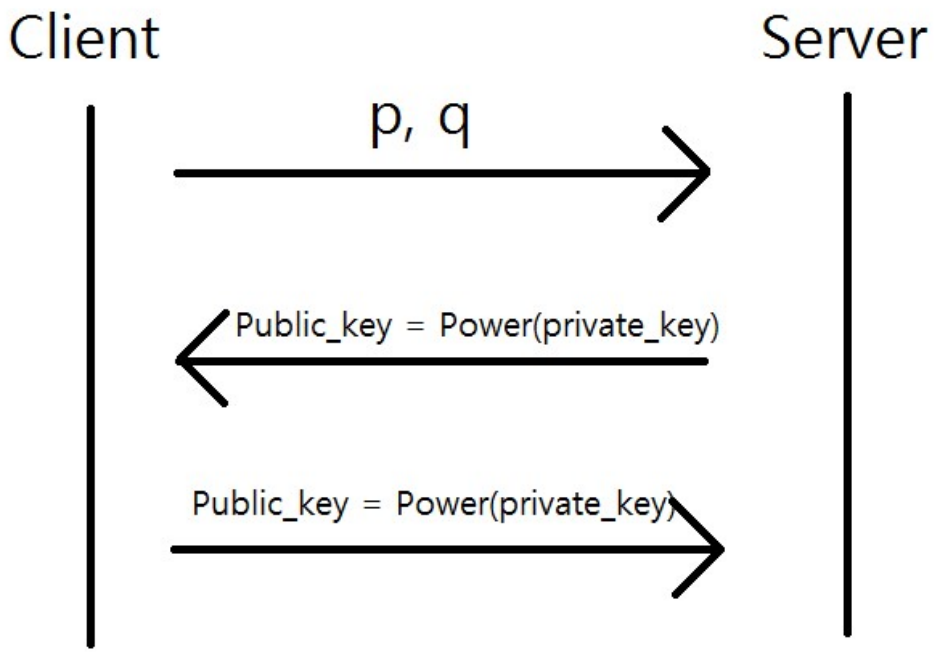
출처 : https://ko.wikipedia.org/wiki/%EB%94%94%ED%94%BC-%ED%97%AC%EB%A7%8C_%ED%82%A4_%EA%B5%90%ED%99%98

Client(C#) 과 Server(C) 사이에서 통신이 이루어진다.

프로그램 구성)

1. 소수 p와 정수 g를 Client(C#) -> Server(C) 로 보내준다.
2. Server(C)는 소수 p 와 정수 g를 받은 뒤 저장을 하고 자신이 갖고 있던 Private_key를 통해 Public_key를 생성한다. 그 뒤 Private는 자신만 갖고 있으며 Public_Key를 Client(C#) 쪽으로 보내준다.
3. Client(C#)은 Server(C)에서 Public_Key를 받는다. 그 뒤 자신의 Private_Key를 통해 Public_Key를 생성한다. 그뒤 Public_Key를 Server(C) 쪽으로 보내준다.

4. Server(C)는 자신의 Private_Key와 상대 Client(C#)의 Public_Key를 통해 최종적인 Last_key를 생성한다.
5. Client(C#)는 자신의 Private_Key와 상대 Server(C)의 Public_Key를 통해 최종적인 Last_key를 생성한다.
6. 여기서 Server(C) 와 Client(C#)이 갖고 있는 Last_Key는 같다.



2. Base64 인코딩

Base 64 (베이스 육십사)란 8비트 이진 데이터(예를 들어 실행 파일이나, ZIP 파일 등)를 문자 코드에 영향을 받지 않는 공통 ASCII 영역의 문자들로만 이루어진 일련의 문자열로 바꾸는 인코딩 방식을 가리키는 개념

3. C#(Client) 소스코드

```
if (!fi.Exists) //파일 존재하지 않을시
{
    Console.WriteLine("파일 비존재!");
    int getValueLength = 0;
    setbyte = Encoding.Default.GetBytes(random_prime.ToString());
    socket.Send(setbyte, 0, setbyte.Length, SocketFlags.None);   //prime 보냄
    Console.WriteLine("Prime value : " + Encoding.Default.GetString(setbyte) + " & " + setbyte.Length);
    Console.Write(">>");


    setbyte = Encoding.Default.GetBytes(random_integer.ToString());      //integer 보냄
    socket.Send(setbyte, 0, setbyte.Length, SocketFlags.None);
    Console.WriteLine("Integer value : "+Encoding.Default.GetString(setbyte) + " & " + setbyte.Length);

    socket.Receive(getbyte, 0, getbyte.Length, SocketFlags.None);   //상대방의 공개키 가져옴
    getValueLength = byteArrayDefrag(getbyte);
    your_Public_key = Convert.ToUInt64(Encoding.Default.GetString(getbyte, 0, getValueLength + 1));
    Console.WriteLine("Your public key : " + your_Public_key + " & " + getValueLength);
    //Console.WriteLine(your_Public_key);
    myPublic_key = power(random_integer, myPrivacy_key, random_prime);
    setbyte = Encoding.Default.GetBytes(myPublic_key.ToString());      //나의 공개키 보냄
    socket.Send(setbyte, 0, setbyte.Length, SocketFlags.None);
    Console.WriteLine("my public key : " + Encoding.Default.GetString(setbyte) + " & " + setbyte.Length);


    last_key = power(your_Public_key, myPrivacy_key, random_prime);


    System.IO.File.WriteAllText(FilePath, last_key.ToString(), Encoding.Default);
    Console.WriteLine("최종키 is " + last_key);


    count = 1;

}
```

password.txt 파일이 존재하지 않을 시 상호 키를 갖고 있어야 한다. 이전에 했었던 diffie-hellman 알고리즘을 통한 키를 생성하는 과정이다.

```
else    //파일 존재할때
{
    TextReader tr = fi.OpenText();
    string abc = tr.ReadToEnd();
    Console.WriteLine("파일 존재!");
    byte[] cba = Encoding.Default.GetBytes(abc);
    string result = Convert.ToBase64String(cba);        //Base64인코딩을 통한 데이터 전송
    setbyte = Encoding.Default.GetBytes(result);
    Console.WriteLine("여기 값은 무엇인가: " + result);

    socket.Send(setbyte, 0, setbyte.Length, SocketFlags.None);
    Console.WriteLine(Encoding.Default.GetString(setbyte));
    count++;
}
```

파일이 존재할시 password.txt에서 값을 읽어와 base64암호화를 하여 데이터를 보낸다.

4. C(Server) 소스코드

C언어에서는 Base64인코딩을 지원해주는 라이브러리가 없으므로 실제 구현된 코드를 사용하였다.

```
static char base64_table[] = {
    'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
    'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z',
    'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
    'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
    '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '+', '/', '\0'
};

static char base64_pad = '='
// Base64 Encoding
unsigned char* base64_encode(const unsigned char* str, int length, int* ret_length)
{
    const unsigned char* current = str;
    int i = 0;
    unsigned char* result = (unsigned char*)malloc(((length + 3 - length % 3) * 4 / 3 + 1) * sizeof(char));

    while (length > 2)
    { /* keep going until we have less than 24 bits */
        result[i++] = base64_table[current[0] >> 2];
        result[i++] = base64_table[((current[0] & 0x03) << 4) + (current[1] >> 4)];
        result[i++] = base64_table[((current[1] & 0x0f) << 2) + (current[2] >> 6)];
        result[i++] = base64_table[current[2] & 0x3f];
```

```
        current += 3;
        length -= 3; /* we just handle 3 octets of data */
    }

    /* now deal with the tail end of things */
    if (length != 0)
    {
        result[i++] = base64_table[current[0] >> 2];
        if (length > 1)
        {
            result[i++] = base64_table[((current[0] & 0x03) << 4) + (current[1] >> 4)];
            result[i++] = base64_table[(current[1] & 0x0f) << 2];
            result[i++] = base64_pad;
        }
        else
        {
            result[i++] = base64_table[(current[0] & 0x03) << 4];
            result[i++] = base64_pad;
            result[i++] = base64_pad;
        }
    }
    if (ret_length)
    {
        *ret_length = i;
    }
    result[i] = '\0'
    return result;
}
// Base64 Decoding
unsigned char* base64_decode(const unsigned char* str, int length, int* ret_length)
{
    const unsigned char* current = str;
    int ch, i = 0, j = 0, k;
    /* this sucks for threaded environments */
    static short reverse_table[256];
    static int table_built;
    unsigned char* result;

    if (++table_built == 1)
    {
        char* chp;
        for (ch = 0; ch < 256; ch++)
        {
            chp = strchr(base64_table, ch);
            if (chp)
            {
                reverse_table[ch] = chp - base64_table;
```

```
            }
            else
            {
                reverse_table[ch] = -1;
            }
        }
    }

    result = (unsigned char*)malloc(length + 1);
    if (result == NULL)
    {
        return NULL;
    }

    /* run through the whole string, converting as we go */
    while ((ch = *current++) != '\0')
    {
        if (ch == base64_pad) break

        /* When Base64 gets POSTed, all pluses are interpreted as spaces.
This line changes them back.  It's not exactly the Base64 spec,
but it is completely compatible with it (the spec says that
spaces are invalid).  This will also save many people considerable
headache.  - Turadg Aleahmad <turadg@wise.berkeley.edu>
*/

        if (ch == ' ') ch = '+'

        ch = reverse_table[ch];
        if (ch < 0) continue

        switch (i % 4)
        {
            case 0:
                result[j] = ch << 2;
                break
            case 1:
                result[j++] |= ch >> 4;
                result[j] = (ch & 0x0f) << 4;
                break
            case 2:
                result[j++] |= ch >> 2;
                result[j] = (ch & 0x03) << 6;
                break
            case 3:
                result[j++] |= ch;
                break
```

```
        }
        i++;
    }

    k = j;
    /* mop things up if we ended on a boundary */
    if (ch == base64_pad)
    {
        switch (i % 4)
        {
            case 0:
            case 1:
                free(result);
                return NULL;
            case 2:
                k++;
            case 3:
                result[k++] = 0;
        }
    }
    if (ret_length)
    {
        *ret_length = j;
    }
    result[k] = '\0'
    return result;
}
```

```
if(access(filename,mode)!=0){
  srand(time(NULL));
myPrivacy_key = rand()%100;
//소수값인 p를 받아 온다.
read(client_sock, recvline, MAXLINE);
random_prime = atoll(recvline);
printf("your prime : %lld\n\n",random_prime);

//정수값인 g를 받아 온다.
memset(recvline,0,sizeof(recvline));
read(client_sock, recvline, MAXLINE);
random_integer = atoll(recvline);
printf("your integer : %lld\n\n",random_integer);

myPublic_key = power(random_integer,myPrivacy_key,random_prime);
printf("myPrivacy_key is %lld, my Public_keyis %lld\n",myPrivacy_key,myPublic_key );
//나의 공개키를 보낸다.
 sprintf(sendline,"%lld",myPublic_key);
 size = strlen(sendline);
write(client_sock,sendline,strlen(sendline));
//memset(sendline,0,MAXLINE);

//상대방의 공개키를 받아 온다.
memset(recvline,0,sizeof(recvline));
read(client_sock, recvline, MAXLINE);
your_Public_key = atoll(recvline);
printf("your publickey : %lld\n\n",your_Public_key);
last_key = power(your_Public_key,myPrivacy_key,random_prime);
printf("최종: %lld\n",last_key);



fp = fopen(filename,"a");
fprintf(fp,"%lld",last_key);
fclose(fp);
}
```

파일이 존재하지 않을 경우 키를 생성하는 과정이다.
atoll 함수는 char[] 형을 int형으로 바꿔주는 함수이다.

```
else
{
  printf("키 파일 이미 존재 \n");
  printf("키를 기다린다\n");
  memset(recvline,0,sizeof(recvline));
  read(client_sock, recvline, MAXLINE);
  memcpy(your_base64_lastKey,recvline,sizeof(recvline));
  //your_base64_lastKey = atoll(recvline);

  printf("키를 받았으며 비교한다\n");
  fp = fopen(filename,"r");
  fgets(key_compare,MAXLINE,fp);
  last_key = atoll(key_compare);
  //base64로 전송온다. 그래서 lld 가 아닌 %s로 받아와야 한다.
  printf("텍스트에 저장된 값 : %lld, 받아온 키 값: %s \n",last_key,your_base64_lastKey);
  sprintf(compare_key,"%lld",last_key);
  compare_key2 = base64_encode((unsigned char *)compare_key, strlen(compare_key), &encoding_size);
  printf("디코딩된 키값 : %s\n\n",compare_key);

  if(strcmp(compare_key2,your_base64_lastKey)){
    printf("%d\n\n",strcmp(compare_key2,your_base64_lastKey));
    printf("키 값이 다르다\n");
    printf("나의 키는 %s, 너의 키는 %s \n",compare_key2,your_base64_lastKey);
    printf("---------연결 종료-------------\n");

  }
  else{
    printf("키 값이 같다\n");
    printf("%d\n\n",strcmp(compare_key2,your_base64_lastKey));
    printf("나의 키는 %s, 너의 키는 %s \n",compare_key2,your_base64_lastKey);
  }
  fclose(fp);
}
```

password.txt가 존재한다면 키를 기다리구 난후 base64인코딩 된 값을 받아온다. Client 에서 받은 값을 Base64 Decoding 해야하나 Decoding 시 크기가 필요함으로 Server쪽에서 가지고 있던 키를 인코딩하여 비교를 하였다.
비교후 값이 같다면 넘어가고 같지 않다면 종료를 해야하나 종료부분은 아직 만들지 않았다.