SEED 알고리즘 기반의 윈도우즈 안드로이드 호환 파일 보호 시스템 개발

서광덕[O] 정동현 김환욱 노영주
한국산업기술대학교 컴퓨터공학과
{tab006[O], love3gozj, unlesslove, yrho}@kpu.ac..kr

Development of SEED Algorithm-based File Security System for Windows and/or Android Devices

Kwang-Deok Seo[0] Dong-Heon Jung hwan-wook Kim young-Joo Noh
Dept. of Computer Engineering, Korea Polytechnic University

요 약

IT기기의 발전으로 다량의 개인정보들이 생성되고 이동하고 사라진다. 이에 따라 개인정보 보호의 중요성이 커지며 기존의 단순한 파일 숨김 또는 파일 잠금 형태의 보호 시스템이 아닌 데이터를 암호화하여 보안성을 높이는 시스템이 필요하다. 우리가 개발한 시스템은 윈도우즈에서는 물론 안드로이드 기기에서도 사용할 수 있으며, 윈도우즈 PC와 안드로이드 스마트폰이 통신하며 암·복호화를 가능하게 하여 활용성을 높이려 노력하였고, PC버전의 경우 대용량데이터 암·복호화 및 파일 안전 삭제 기능을 추가 하여 구현 하여 PC의 특수성도 별도로 고려하였다.

1. 서론

최근 많은 논란을 가져온 어떤 커뮤니티 대기업의 6만 명이상의 개인정보유출 사고, 얼마 지나지 않아 대형오픈마켓의 회원정보 유출 사고가 잇따랐다. 그 후 인터넷 사용자들은 개인 정보 보호에 대한 경각심을 불어 일으켰다[1][2].

하지만 인터넷 사용자들은 웹서버에 회원가입을 위해 기입한 자신의 신상정보만을 중요하게 생각 하며 기업이 보안에 좀 더 신경 쓸 것을 당부 하지만 정작 자신이 사용하는 PC와 스마트폰은 안전할거라는 생각을 한다. 하지만 개인의 컴퓨터의 경우 악성코드나 바이러스에 취약하며 그로 인한 네트워크 보안 또한 간간한 해킹 툴을 이용하여 무용지물이 되어 버린다.

한때 컴퓨터에 많은 지식이 없는 초등학생들이 재미 삼아서 간단한 해킹 프로그램을 이용하여 상대방의 컴퓨터에 악성 코드를 심고 악성 코드를 실행시켜 상대방의 컴퓨터를 접속 하여 컴퓨터를 제어하고 개인의 데이터를 조작하고 삭제하고 인터넷에 퍼트리는 등의 범죄를 저질렀다. 만약 많은 사용자들이 사용하거나 쉽게 다른 사람들이 쉽게 PC에 접근 할 수 있는 환경의 경우엔 더더욱 파일 자체의 보호도 필요하다[4].

이렇게 네트워크상의 보안뿐만 파일 자체의 보호를 위해서는 현재 널리 쓰이고 있는 파일 숨김 및 파일 잠금 프로그램이 있지만 이런 형태의 경우 데이터 자체는 온전히 남아 있기 때문에 위험에 노출될 가능성이 있다.

(그림 1) 개인정보 취약점[3]

우리가 개발한 시스템은 이러한 점을 염두에 두고 데이터의 자체에 암호화를 하여 보안성을 높임으로써 해결할 수 있다. 또한 대용량 암호화 기능과 삭제된 파일을 복구 시 데이터의 정보를 보호 하는 파

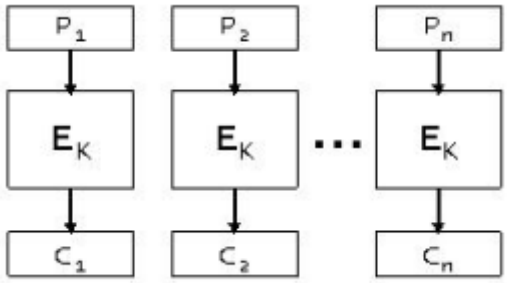
일 안전 삭제 기능을 포함하고 있다.

제2의 컴퓨터라 불리는 스마트 폰, 태블릿PC 등에서도 PC와 같은 방법으로 데이터를 암호화를 구현 하였으며, 한 사용자가 PC와 스마트폰을 연동하여 적시적소에서 필요에 의해 PC의 데이터를 암·복호화 할 수 있다는 기존의 프로그램과 다른 이점을 가지고 있다.

2. 설계

2.1 SEED 설계

한국인터넷 진흥원(이하 Kisa)에서 제공되는 SEED알고리즘은 여러 가지의 운영모드를 사용할 수 있다[5]. 현재 시스템의서 사용하는 SEED알고리즘의 사용하는 알고리즘의 운영모드는 ECB 평문 블록을 독립적으로 암호화 하는 운영모드이며 알고리즘 정확한 사용 환경은 32bit 운영체제 환경에 128bit 블록 알고리즘 이다.



(그림 2) ECB 모드 블록 암호화[6]

2.1 암호화

Kisa에서 제공되는 SEED알고리즘의 경우 128bit 데이터와 128bit의 key값을 이용한 암호화 작업만을 수행하고 데이터의 전후 처리 알고리즘 작업을 제공하지 않기 때문에 데이터의 처리에 중점을 두어 설계했다.

최초 파일을 읽어 들여 데이터의 사이즈를 구하고 16byte씩 읽어 들여 암호화를 수행 한다. 이때 문제가 발생하는데 데이터의 크기가 16byte 배수가 되지 않는다면 크기에 맞게 데이터를 채워주어야 한다.

```
평문 데이터(48bits) : 4F 52 49 54 48 4D
적용 결과(128bits) : 4F 52 49 54 48 4D 00 00 00 00 00 00 00 00 00 00
```

(그림 3) 데이터 패딩

처리 방법으로는 마지막 16byte 데이터에 뒤쪽 공간을 "0"을 추가하여 배열의 크기를 맞추는 작업을 수행한 후 에 암호화를 진행하며 이후 복호화 과정에서 추가된 "0"비트를 제거하여 데이터를 완벽히 복원에 낸다.

```
public void EnChipher_BigByte(String Address , int key)
{
        SeedCipher  en  = new SeedCipher();
            try{
                e_key[0] = (byte) (key & 0x000000FF);
                for(int i=1; i<=15; i++){
                        e_key[i] = (byte) ((key >> (i+8)) & 0x000000FF);
                }

                TextView content = (TextView)findViewById(R.id.content);
                in = new FileInputStream(Address);
                BufferedInputStream input =  new BufferedInputStream(in);

                e_buffer = new byte[input.available()];
                t_buffer = new byte[input.available()];

        while(result > -1)
                result = input.read(e_buffer);

        t_buffer = en.encrypt(e_buffer,e_key);
```

(그림 4) 안드로이드 암호화 소스

```
while(1)
{
   count = fread(Ciphertext,1,16, Rp);
   if(count!=16)
   {
      Art=16-count;
      for(a=0;a<Art;a++)
      {
         memset(Ciphertext+count,NULL,Art);
      }
      if(count==0)
         break;
      count=count+Art;
   }
   SeedRoundKey(pdwRoundKey,Key);
   SeedEncrypt(Ciphertext,pdwRoundKey);
}
```

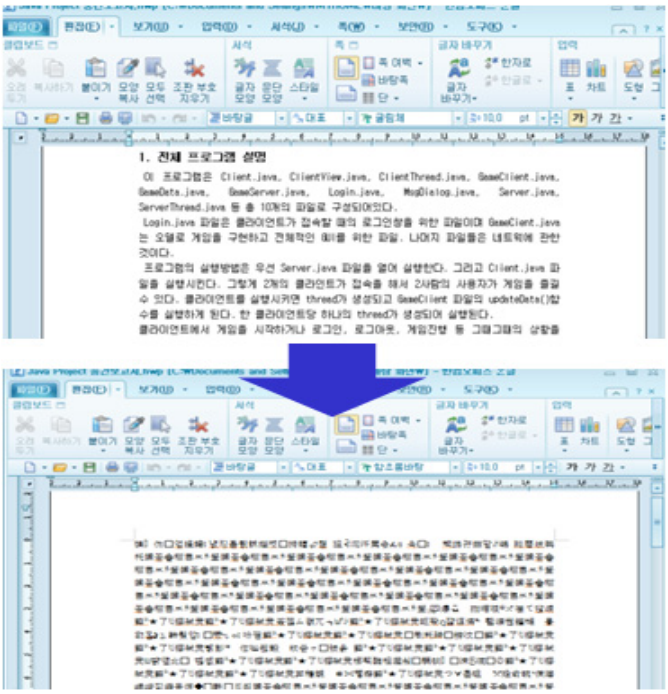
(그림 5) MFC 암호화 소스

암호화된 파일은 데이터의 끝에 암호화에 사용된 키 값과 암호화된 파일이라는 명시를 주기위해서 "Cipher"라는 비트를 추가적으로 넣어 준다. 이렇게 암호화된 파일은 임시파일로 저장한 뒤 원본파일의 파일이름과 확장자를 추가하여 파일을 생성 한다.

```
Rp = fopen(NewAddress1, "a+");
SeedRoundKey(pdwRoundKey,Key);
SeedEncrypt(Key,pdwRoundKey);
fwrite(Key,1,16, Rp);
fwrite("Cipher",1,16, Rp);
```

(그림 6) 암호화 키 및 플래그 적용

암호화된 텍스트 파일을 보면 암호화가 되어있는 것을 확인 할 수 있다.



(그림 7) 암호화 적용 후

2.2 파일 안전 삭제

삭제된 파일은 쉽게 복원 프로그램에 의해서 다시 열어 볼 수가 있다. 그러기에 파일 삭제 시 알고리즘을 이용하여 삭제하기 전에 파일의 데이터를 임의로 덮어씌운 뒤 파일을 삭제하여 복원하여도 데이터의 정보를 숨 길수가 있다.

```
Op = fopen(Address,"w");
for(Cnt = 0; Cnt < FileSize; Cnt++)
{
   fwrite("0",1,1, Op);
   DownCnt = DownCnt+1;
}
fclose(Op);
```

(그림 8)삭제 알고리즘

2.3 통신

PC버전의 MFC기반 서버에서 android기반의 스마트폰간의 통신은 TCP/IP이용한 통신이며, 가장먼저 해결되어야 하는 문제는 C++ 서버와 JAVA 클라이언트간의 데이터의 원활한 데이터의 전송이 되어야 한다. 이를 위해 데이터 변환을 한다.

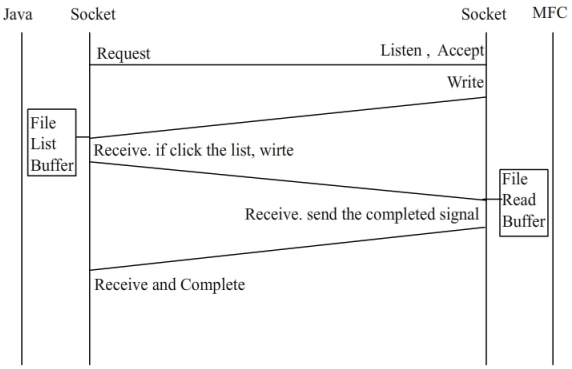
```
CString sendString;
BSTR bstr;
int len;
while(pos != NULL)
{
   sendString = m_filelist.GetNext(pos);
   bstr = sendString.AllocSysString();
   char* sizeBuf = new char[sendString.GetAllocLength()];
   len = WideCharToMultiByte(CP_ACP, 0, bstr, -1, sizeBuf, 0, NULL, NULL);
   delete [] sizeBuf;

   char* sendBuf = new char[len];
   WideCharToMultiByte(CP_ACP, 0, bstr, -1, sendBuf, len, NULL, NULL);
   m_Data->Send(sendBuf, len);

   delete [] sendBuf;
   SysFreeString(bstr);
}
```

(그림 9) 멀티바이트로 데이터 변환

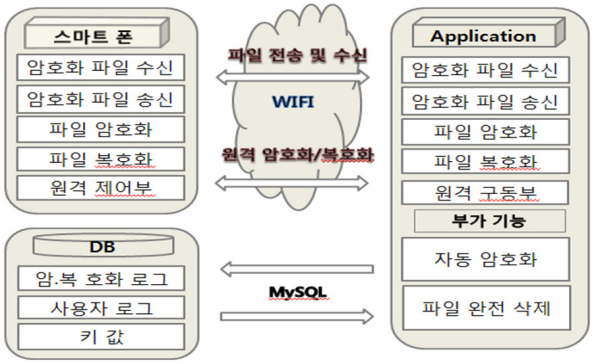
대기 중인 서버에 스마트폰을 이용하여 접속 하면 서버는 지정된 폴더내의 파일 개수와 이름을 읽어들여 해당 스마트폰에 리스트를 전송한다. 리스트를 건네받은 스마트폰에서는 암호화할 파일을 선택하면 다시 해당파일 이름을 서버로 전송 한다. 이후 서버에서는 해당 파일을 암호화 하고 완료 메시지를 전송한다.



(그림 10) 통신 흐름도

3. 전체 구성도

전체적인 시스템의 구성도 이다. 위에서 언급하지 않았지만 암호화 와 관련된 모른 기록들을 DB에 저장하도록 설계되어있다. My-SQL을 이용하여 한 개의 테이블 안에 저장하며 4개의 속성 “날짜”,“시간”,“파일명”,“암호화 구분 키” 으로 저장 한다.



(그림 11) 전체 구성도

4. 결론 및 향후 연구과제

지금까진 암호화 알고리즘을 이용하여 PC와 스마트폰의 데이터를 암호화하는 시스템은 없었다. 보안의 우수성만을 본다면 단연 다른 시스템들보다 뛰어나다. 그리고 다른 기기들의 이식도 가능하다.

다만 앞에서 언급했듯이 자주 사용하고 자주 수정되는 데이터들의 암호화하는 기존의 시스템들보다 오히려 번거로움을 가져올 수 있다. 보안이 중요 하긴 하지만 편리성 역시 사용자들에게는 중요한 문제이기 때문이다.

편리성과 보안성의 모두 만족할 수 있는 시스템이 되지 못한 것이 가장 큰 아쉬움이라 할 수 있겠다.

참고 문헌

[1]"사상 최악"…3천500만 네이트-싸이 개인정보 '털렸다.
http://news.inews24.com/php/news_view.php?g_serial=592356&g_menu=020300&rrf=nv
[2]"옥션, 고객정보 유출 어디까지?(종합)
"http://www.mt.co.kr/view/mtview.php?type=1&no=2008020517571980887&outlink=1
[3]"애인 사진… 친구 전화번호… 나만의 사생활 고스란히 노출
"http://news.dongascience.com/PHP/NewsView.php?kisaid=20120906100000000305&classcode=01
[4]"[u클린]해킹 대중화 시대..초등생도 '해킹도사'"
http://www.mt.co.kr/view/mtview.php?type=1&no=2005060911282342206&outlink=1
[5]"다양한 프로토콜에서 SEED이용 가이드라인"
http://www.kisa.or.kr/public/major/policyList.jsp
[6]"SEED소스 코드 매뉴얼"
http://www.kisa.or.kr/public/major/policyList.jsp