보안과 암호화 기법

(Security and Cryptograph Techniques)

대부분의 클라이언트/서버 어플리케이션에서는 ID 와 패스워드를 묻는 최소한의 보안을 하고 있다. 그렇지만, 데이터 암호화 등의 다소 높은 수준의 보안이 요구되는 경우도 많다.
이러한 주제를 가지고 정말로 많은 양의 문서와 책들이 쓰여져 왔고, 그리고 많은 연구도 진행 중이다.
그렇지만, 여기에서 다루고자 하는 것은 그렇게 이론적이고 어려운 암호화 기법에 대한 것이 아니다. 실제로 델파이 어플리케이션을 사용하면서 간단하게 이용할 수 있는 암호화 기법에 대해서 알아보고, 이를 실제로 구현하여 적용하는 방법에 대해서 알아볼 것이다. 그리고, 현재 실용화되어 사용되고 있는 표준 암호화 알고리즘의 종류와 이들을 지원하는 컴포넌트를 소개하고 이를 사용하는 방법에 대해서 알아보도록 한다.

패스워드와 XOR 암호화

보통의 경우 어플리케이션에 패스워드를 이용해서 보안을 하는 경우가 많다. 그런데, 이렇게 사용하는 패스워드를 .INI 파일이나 레지스트리, 일반 텍스트 파일 등에 기록할 경우 아무나 쉽게 패스워드를 간파할 수 있게 된다. 이럴 때에 기본적인 암호화 기법이 필요한데, 가장 쉬우면서도 유용하게 쓸 수 있는 것이 XOR 암호화 기법이다.
이 암호화 기법은 개념적으로도 매우 쉽고, 알고리즘이 간단하기 때문에 수행속도도 매우 빠른 것이 장점이다. 단점은 키를 알게 되면 쉽게 깨질 수 있다는 것이다.
XOR 암호화 기법을 적용하기 위해서는 키가 되는 숫자를 이용하게 된다. 이를 키 번호(키 number)라고 하며 이를 이용해서 메시지를 암호화 하거나, 해독한다. 사용 방법은 암호화하고자 하는 문자나 숫자를 키 번호와 XOR 연산을 행해서 나오는 값을 저장하고, 이를 해독할 때에는 역시 키 번호로 XOR 연산을 하면 된다.
그러면, 실제로 이를 구현하는 코드를 살펴보자.

```
procedure XORBuffer(Buffer: TMemoryStream; 키: String);
var
  Len, Place: Word;
  Character: Char;
  Size: Integer;
begin
  Buffer.Seek(0, 0);
```

```
  Len := Length(키);
  Place := 1;
  Size := Buffer.Size;
  while Size > 0 do
  begin
    Buffer.Read(Character, 1);
    Character := Char(Byte(Character) xor Byte(키[Place]));
    Buffer.Seek(-1, 1);
    Buffer.Write(Character, 1);
    Inc(Place);
    if Place > Len then Place := 1;
    Dec(Size);
  end;
end;
```

키 번호를 문자열로 받아서 그 길이를 이용해서 반복하도록 하고, 암호화가 될 문자열은 버퍼에 담아서 이를 키 번호와 xor 연산을 함으로써 암호화 하는 것이다.
여기에서는 TMemoryStream 클래스를 사용했는데, 문자열을 사용하지 않은 이유는 xor 연산의 결과가 0 일 경우 PChar 문자열의 마지막 문자임을 나타내게 되므로, 이러한 문자열의 규칙에 위배되지 않기 위해서 스트림을 이용하였다. 그렇지만, 이렇게 암호화한 내용을 .INI 파일이나 레지스트리에 저장하고자 하면 문자열로의 변환 과정이 필요하다.

UUEncoding 과 UUDecoding

유닉스는 8 비트의 데이터 중 7 비트를 사용하는 통신 시스템을 가진 경우가 많은데, 이를 해결하기 위한 방법으로 알려진 것이 UUEncoding, UUDecoding 이라는 것이다. 이 방법을 사용하면 데이터를 '!' 문자인 ASCII 33 부터 'z'인 ASCII 122 까지의 문자로 변환하는 것으로, 이를 이용하면 3 개의 문자를 4 개의 바이트에 담게 된다. 이후 4 개의 바이트는 마스크를 거쳐서 32 를 더해서 결과 값으로 출력된다. 이 방법의 단점은 암호화된 데이터가 원래의 이진 데이터 보다 1/4 정도 커진다는 것이지만, 크게 문제가 되지는 않는다/
UUEncoding 을 이용한 암호화 프로시저의 코드는 다음과 같다.

```
procedure UUEncodeBuffer(InBuffer, OutBuffer: PChar; InSize: Word; OutSize: Word);
var
  Chars: Array[0..3] of Byte;
  Hunk: Array[0..2] of Byte;
```

```
  i: Integer;
begin
  OutSize := 0;
  repeat
    FillChar(Chars, 4, 0);
    if InSize > 3 then
    begin
      Move(InBuffer^, Hunk, 3);
      Inc(InBuffer, 3);
      Dec(InSize, 3);
    end
    else
    begin
      Move(InBuffer^, Hunk, InSize);
      Inc(InBuffer, InSize);
      Dec(InSize, InSize);
    end;
    Chars[0] := Hunk[0] shr 2;
    Chars[1] := (Hunk[0] shl 4) + (Hunk[1] shr 4);
    Chars[2] := (Hunk[1] shl 2) + (Hunk[2] shr 6);
    Chars[3] := Hunk[2] and $3F;
    for i := 0 to 3 do Chars[i] := (Chars[i] and $3F) + 32;
    Move(Chars, OutBuffer^, 4);
    Inc(OutBuffer, 4);
    Inc(OutSize, 4);
  until   InSize <= 0;
end;
```

쉽게 말하자면 암호화할 데이터를 InBuffer, 데이터의 크기를 InSize 에 담아오면 암호화를 거쳐서 암호화된 데이터를 OutBuffer, 크기를 OutSize 로 반환하는 프로시저이다.
이때 3 바이트를 4 바이트로 (3 바이트에 8 비트를 6 비트씩 4 바이트로) 저장한 후, 이를 $3F 로 마스크 한 후, 32 를 더해서 OutBuffer 에 결과값으로 변환하는 작업을 한다.
이런 UUEncoding 알고리즘을 이용한 암호화 데이터를 풀 때에는 다음과 같은 UUDecoding 프로시저를 사용한다.

```
procedure UUDecodeBuffer(InBuffer, OutBuffer: PChar; InSize: Word; OutSize: Word);
```

```
var
  Chars: Arrays[0..3] of Byte;
  Hunk: Arrays[0..2] of Byte;
  i: Integer;
begin
  OutSize := 0;
  repeat
    FillChar(Hunk, 3, 0);
    if InSize >= 4 then
    begin
      Move(InBuffer^, Chars, 4);
      Inc(InBuffer, 4);
      Dec(InSize, 4);
    end
    else
    begin
      Move(InBuffer^, Chars, InSize);
      Inc(InBuffer, InSize);
      Dec(InSize, InSize);
    end;
    for i := 0 to 3 do
    begin
      if Chars[i] = Ord(``) then Chars[i] := Ord(` `);
      Chars[i] := Chars[i] – 32;
    end;
    Hunk[0] := (Chars[0] shl 2) + (Chars[1] shr 4);
    Hunk[1] := (Chars[1] shl 4) + (Chars[2] shr 2);
    Hunk[2] := (Chars[2] shl 6) + Chars[3];
    Move(Hunk, OutBuffer^, 3);
    Inc(OutBuffer, 3);
    Inc(OutSize, 3);
  until InSize <= 0;
end;
```

이런 방식으로 UUEncoding, UUDecoding 을 사용하면 .INI 파일이나 레지스트리에 데이터를 저장할 수 있다. 앞에서 설명한 xor 암호화 기법을 같이 혼용해서 일단 데이터를 xor

로 암호화한 뒤 이를 UUEncoding 알고리즘을 이용해서 변환한 뒤 이를 저장하고, 해독할 때에는 UUDecoding 알고리즘을 이용해서 푼 뒤, 이를 다시 xor 을 이용해서 원래의 값을 얻어내게 된다.

암호화 컴포넌트와 표준 알고리즘

암호화라는 것은 앞에서 예를 들어 설명한 것과 같이 특정 알고리즘이나 계산식 등을 통해 데이터를 변경하고, 변경된 데이터를 원할 때 다시 원래의 데이터로 변환할 수 있는 것을 모두 포함한 것이다.

이러한 암호화 알고리즘에는 크게 나누어 private-키와 publick-키 알고리즘으로 구분할 수 있다. Private 키 알고리즘을 사용할 때에는 사용자가 암호화와 해독 과정을 모두 같은 키로 할 수 있게 된다. 데이터가 같은 위치에 있을 경우에는 이것이 문제가 되지 않지만, 데이터가 전송되어야 할 경우에는 문제가 될 수 있다. 예를 들어, 보안이 요구되는 메시지를 전송한다고 할 때 전송하는 사람이 private 키를 사용해서 메시지를 암호화한 뒤에 메시지를 보냈다고 하자, 그러면 이를 받는 사람은 암호화하는데 사용한 private 키를 이용해서 메시지를 해독해야 한다. 그런데, 문제는 이렇게 해독할 열쇠가 되는 private 키를 어떻게 안전하게 얻을 수 있는가 하는 문제이다.

이 문제를 해결하기 위해 1976 년 처음 개발된 방법이 바로 public-키 암호화 기법이다. 이 방법은 데이터를 암호화하고 해독하는데 2 개의 연관되었지만 다른 키를 사용하는 것이다. 이때 암호화 키를 public 키라고 하는데, 이렇게 암호화를 하는데 사용된 키가 해독하는데 사용되지 않기 때문에 키가 발견될 염려가 없다. 이때 데이터를 해독할 때에는 private 키가 사용된다. 이제 암호화된 메시지를 전송하고자 하면, 전송하고자 하는 사람이 public 키를 보내고 이를 이용해서 메시지를 암호화한다. 그리고, 암호화된 메시지를 받은 쪽에서는 자신의 private 키를 사용해서 해독하면 된다.

Private-키 암호화는 쉽게 말해서 데이터에 대한 패스워드를 알고 있는 사람들끼리만 이 패스워드를 이용한 해쉥 함수를 처리함으로써 암호화를 하는 방법이고, public-키 암호화는 패스워드를 걸지 않고도 일반적인 데이터에 대한 public-키를 생성해서 암호화를 하는 방법이다.

Public-키 암호화 알고리즘은 private-키 알고리즘에 비해 암호화와 해독을 할 때 1,000 배가 더 걸린다. 또한, Public-키 암호는 private-키 암호에 비해 같은 정도의 보안 레벨을 제공하기 위해서는 10 배는 더 길어야 한다.

표준화된 암호화 알고리즘으로는 매우 여러 가지가 있지만 널리 쓰이는 것으로는 DES, MD5, RC5, SHA 등이 있다. 이들은 모두 public-키 암호화 기법으로 이들 중에서 DES 와 SHA 에 대해서 알아보자. 그에 비해 앞에서 설명한 XOR 연산자를 이용한 암호화는 일종의 private-키 암호화 기법이라고 말할 수 있다.

- DES

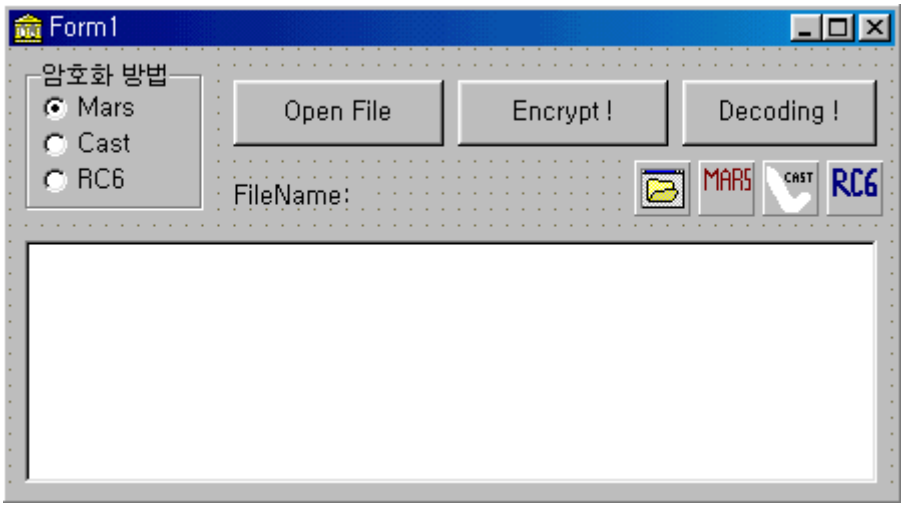
DES 는 Data Encryption Standard 의 약자로 암호학계에서 20 년간의 테스트를 거친 알고리즘이기 때문에 특별한 약점이 없는 표준적인 알고리즘이다. DES 는 암호화와 해독을 위해서 64 비트 블록에 대해서 56 비트의 키를 사용한다. 최근의 경향으로 보아 56 비트의 키는 다소 짧다는 의견이 대두되고 있지만, 보안의 정도가 아주 높아야 하는 경우를 제외하고는 사용하는데 큰 무리가 없다.

- SHA

SHA 는 Secure Hash Algorithm 의 약자로 NIST 와 NSA 에서 개발된 암호화 표준이다. 마이크로소프트에서의 code signing 에서도 MD5 와 함께 표준으로 사용되고 있다. SHA 알고리즘은 160 비트의 해쉬를 만들어내기 때문에, 128 비트를 이용하는 MD5 에 비해 보안의 정도가 높다고 할 수 있다.

- 암호화 컴포넌트의 활용

인터넷에 보면 여러가지 암호화 컴포넌트가 프리웨어로 공개되고 있다. 그중에서도 표준 알고리즘인 DES, SHA 를 비롯한 여러가지 알고리즘을 구현한 컴포넌트 들을 DSP(Delphi Super Page)나 델파이 델리(Delphi Deli) 등에서 구할 수 있다. CD-ROM 과 함께 제공되는 40 장 디렉토리에 보면, Crypt 라는 서브 디렉토리가 있는데 여기에 앞으로 차세대 암호화 알고리즘의 표준으로 제안되고 있는 Cast, RC6, Mars 알고리즘을 컴포넌트화한 유닛의 소스코드가 포함되어 있다. 이들 알고리즘에 대해서 더 자세히 알고 싶으면 Mars 는 http://www.reaserch.ibm.com/security/mars/, Cast 는 http://www.entrust.com/, RC6 는 http://theory.lcs.mit.edu/~rivest/ 홈 페이지를 방문하기 바란다. 이들 컴포넌트는 모두 Duff Neill 에 의해 제작되었다. 이 컴포넌트 들을 이용하여 예제 프로그램을 작성할 것이므로 이들을 컴포넌트 팔레트에 설치하기 바란다.
그 밖에도 David Barton 이 RC5, Blowfish, rmd160, RC5, Misty, IDEA, Skipjack 등의 암호화 기법을 구현한 컴포넌트를 공개하였는데 이를 Others 서브 디렉토리에 같이 제공하고 있다. David Barton 은 델파이를 이용한 암호화 컴포넌트 페이지를 운영하고 있는데, 관심있는 독자들은 http://web.ukonline.co.uk/david.w32/delphi.html 주소에서 운영하는 그의 홈 페이지를 방문해보기 바란다.
그러면, 이 컴포넌트 들을 이용하여 파일을 암호화하고 해독하는 방법을 알아보자.
먼저 폼 위에 라디오 그룹과 라벨 컴포넌트 2 개, 버튼 3 개를 얹고 TOpenDialog, TMars, TCast, TRC6 컴포넌트와 메모 컴포넌트를 다음과 같이 폼에 하나씩 추가한다.



그리고, 각 컴포넌트의 프로퍼티를 앞의 그림에 맞도록 설정한다. 여기서 Open File 버튼을 클릭하면 메모 컴포넌트에 읽어올 파일을 대화상자에서 선택하도록 한다. 그리고, Encrypt 버튼을 클릭하면 라디오 그룹에서 선택한 암호화 방법으로 암호화하고 파일의 확장자가 .enc 인 파일로 저장한다. Decoding 버튼을 클릭하면 확장자가 .enc 인 암호화된 파일의 내용을 해독해서 확장자가 .dec 인 파일로 저장한다.
이때 이런 과정의 내용을 메모 컴포넌트에서 계속 볼 수 있도록 한다.
TRC6, TMars, TCast 컴포넌트의 사용 방법은 완전히 동일하다. 키가 될 문자열을 Key 프로퍼티에 설정하고, InputFile 프로퍼티와 OutputFile 프로퍼티에 각각 읽어올 파일 이름과 암호화나 해독 과정을 거쳐 생성될 출력 파일 이름을 지정한 뒤에 암호화할 경우에는 EncipherFile, 해독할 경우에는 DecipherFile 메소드를 호출하면 된다.
먼저 파일을 읽어오는 Button1 의 OnClick 이벤트 핸들러를 다음과 같이 작성한다.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if OpenDialog1.Execute then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
    Label2.Caption := OpenDialog1.FileName;
  end;
end;
```

여기서 미리 '텍스트 파일'이나 '모든 파일'을 선택할 수 있도록 OpenDialog1 객체의 Filter 프로퍼티를 오브젝트 인스펙터에서 설정하도록 한다. 그리고, Label2 는 암호화와 해독이 진행되는 원래 파일의 이름을 보여주도록 하는 것이다.
암호화와 해독을 진행하도록 하는 Button2 와 Button3 의 OnClick 이벤트 핸들러는 다음과

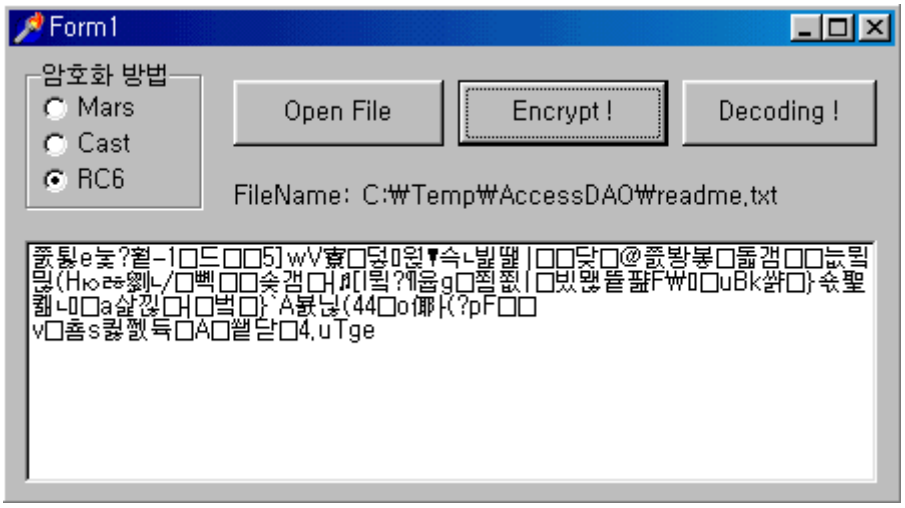
같이 작성한다.

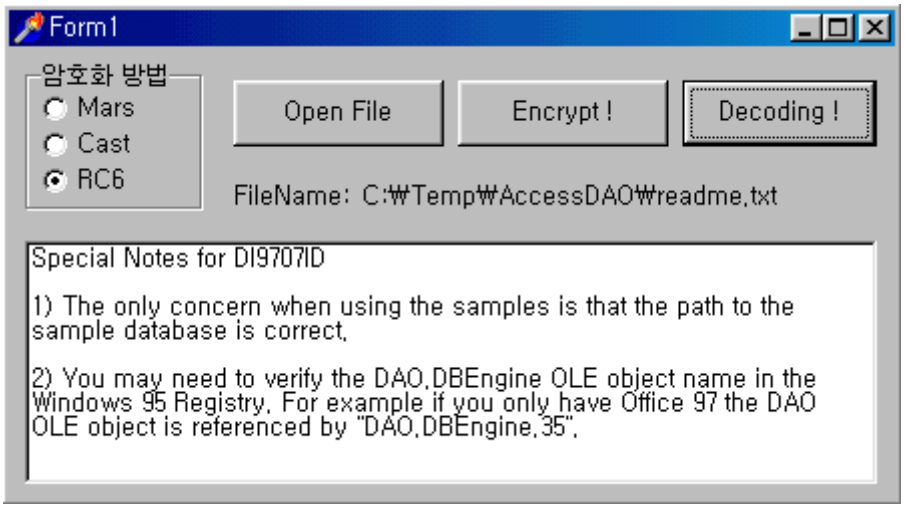
```
procedure TForm1.Button2Click(Sender: TObject);
begin
  case RadioGroup1.ItemIndex of
    0:
      begin
        Mars1.Key := 'Sample Key';
        Mars1.InputFile := Label2.Caption;
        Mars1.OutputFile := ChangeFileExt(Label2.Caption, '.enc');
        Mars1.EncipherFile;
      end;
    1:
      begin
        Cast1.Key := 'Sample Key';
        Cast1.InputFile := Label2.Caption;
        Cast1.OutputFile := ChangeFileExt(Label2.Caption, '.enc');
        Cast1.EncipherFile;
      end;
    2:
      begin
        RC61.Key := 'Sample Key';
        RC61.InputFile := Label2.Caption;
        RC61.OutputFile := ChangeFileExt(Label2.Caption, '.enc');
        RC61.EncipherFile;
      end;
  end;
  Memo1.Lines.LoadFromFile(ChangeFileExt(Label2.Caption, '.enc'));
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  case RadioGroup1.ItemIndex of
    0:
      begin
        Mars1.Key := 'Sample Key';
```

```
        Mars1.InputFile := ChangeFileExt(Label2.Caption, '.enc');
        Mars1.OutputFile := ChangeFileExt(Label2.Caption, '.dec');
        Mars1.DecipherFile;
      end;
    1:
      begin
        Cast1.Key := 'Sample Key';
        Cast1.InputFile := ChangeFileExt(Label2.Caption, '.enc');
        Cast1.OutputFile := ChangeFileExt(Label2.Caption, '.dec');
        Cast1.DecipherFile;
      end;
    2:
      begin
        RC61.Key := 'Sample Key';
        RC61.InputFile := ChangeFileExt(Label2.Caption, '.enc');
        RC61.OutputFile := ChangeFileExt(Label2.Caption, '.dec');
        RC61.DecipherFile;
      end;
  end;
  Memo1.Lines.LoadFromFile(ChangeFileExt(Label2.Caption, '.dec'));
end;
```

이들 컴포넌트의 사용 방법에 대해서는 앞에서도 간단히 설명하였으므로 자세한 설명은 생략한다. 참고로 ChangeFileExt 함수는 파일 이름의 확장자를 2 번째 파라미터에 지정한 문자열로 변경하기 때문에 읽어온 파일의 확장자를 .enc 와 .dec 로 변경하여 지정할 수 있다.

컴파일을 하고, 이를 실행한 뒤에 Open File 버튼을 클릭하여 암호화할 대상 파일을 선택하도록 하자. 그리고, 'Encrypt !' 버튼을 클릭하면 암호화 과정을 거쳐 .enc 파일이 생성되면서 이 파일을 메모에 읽어 온다. 아마도 다음과 같이 알 수 없는 문자열이 보일 것이다. 이제 다시 'Decoding !' 버튼을 클릭하면 해독 과정을 거쳐 .dec 파일이 생성되면서 이 파일을 메모에 읽어 온다. 이 과정을 거치면 원래 텍스트 파일과 같은 내용으로 복원되는 것을 확인할 수 있을 것이다.

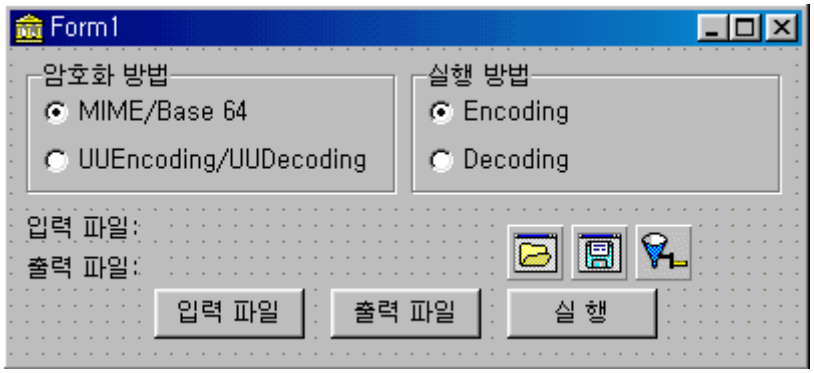




NetMaster 컴포넌트의 활용

39 장에서도 이미 설명한 바 있지만, MIME/Base 64 암호화나 UUEncoding/UUDecoding 암호화는 TNMUUProcessor 컴포넌트를 이용해서 수행할 수 있다. 앞서 UUEncoding 과 UUDecoding 을 하기 위한 루틴을 소개한 바 있으나, TNMUUProcessor 컴포넌트를 이용하여 암호화를 수행하는 예제를 하나 작성해보도록 하자.

먼저 폼에 TRadioGroup 컴포넌트 2 개와 버튼 3 개, 그리고 TLable 컴포넌트 4 개와 TOpenDialog, TSaveDialog, TNMUUProcessor 컴포넌트를 하나씩 올려 놓고 다음과 같이 디자인한다.



'입력 파일'과 '출력 파일' 버튼을 클릭하면 Label3, Label4 에 입력과 출력으로 사용될 파일의 이름이 표시되도록 다음과 같이 Button2, Button3 의 OnClick 이벤트 핸들러를 다음과 같이 작성한다.

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  if OpenDialog1.Execute then
    Label3.Caption := OpenDialog1.FileName;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  if SaveDialog1.Execute then
    Label4.Caption := SaveDialog1.FileName;
end;
```

그리고, '실 행' 버튼을 클릭하면 라디오 그룹에서 선택한 방법으로 인코딩 또는 디코딩을 하도록 다음과 같이 Button1 의 OnClick 이벤트 핸들러를 작성하면 된다.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  InStream, OutStream: TFileStream;
begin
  InStream := TFileStream.Create(Label3.Caption, fmOpenRead);
  OutStream := TFileStream.Create(Label4.Caption, fmCreate);
  try
    case RadioGroup1.ItemIndex of
```

```
        0: NMUUProcessor1.Method := uuMIME;
        1: NMUUProcessor1.Method := uuCode;
      end;
      NMUUProcessor1.InputStream := InStream;
      NMUUProcessor1.OutputStream := OutStream;
      case RadioGroup2.ItemIndex of
        0: NMUUProcessor1.Encode;
        1: NMUUProcessor1.Decode;
      end;
    finally
      InStream.Free;
      OutStream.Free;
    end;
end;
```

이제 프로그램을 컴파일하고 실행한 뒤에 해당되는 파일들의 이름을 지정하고, '실 행' 버튼을 클릭하면 암호화된 파일이 생성되거나, 암호화된 파일이 해독될 것이다.

인터넷 보안 표준 (Internet Security Standards)

인터넷이 일반화 되면서, 전자 상거래도 활발하게 적용되고 있다. 이러한 인터넷에서도 보안은 매우 큰 문제라고 할 수 있다. 이를 위해 암호화와 같은 보안 기법이 절실히 요구된다.
마이크로소프트와 네트스케이프 브라우저에서 사용하는 보안 프로토콜은 public-키와 private-키 암호화 방법의 장점을 모두 수용하고 있다. 가장 널리 쓰이는 보안 프로토콜이 바로 SSL(Secure Sockets Layer)이다. SSL 은 네비게이터의 첫번째 버전과 IE 3.0 부터 채용하고 있다. 1995 년에 마이크로소프트는 새로운 보안 프로토콜을 제안했는데, 이것이 바로 PCT(Private Communications Technology)이다. 이것 역시 IE 3.0 에서부터 채용되었다. 그밖에 IETF(Internet Engineering Task Force)에서는 SSL 을 바탕으로 새로운 인터넷 표준 보안 프로토콜을 제안했는데 이것이 TLS(Transport Layer Security)이다.

- SSL(Secure Sockets Layer)

네비게이터 3.0 과 4.0 의 좌하단 코너에는 보안 아이콘이 나타난다. 네비게이터 3.0 에서는 아이콘의 형태가 체인의 형태이고, 4.0 에서는 자물쇠 형태이다. 끊어진 체인이 연결되거나, 자물쇠가 잠긴 형태로 바뀌면 이것은 서버에 접근할 때 보안 세션으로 들어간다는 것을 의

미한다.
브라우저가 보안이된 웹 페이지에 처음 접속할 때에 서버는 'hello request' 메시지를 전송한다. 보안 세션을 시작하기 위해서는 브라우저가 'client hello,'라고 불리는 메시지로 반응해야 하며, 서버는 'server hello.'로 응답해야 한다. 이런 초기 과정에서 브라우저와 서버는 handshake 프로토콜을 이용해서 보안 정보를 주고 받는다. 이 과정이 SSL 의 첫번째 파트이다. 클라이언트의 'hello' 메시지는 세션 ID 라는 숫자를 가지고 있는데, 이를 이용해서 브라우저와 서버 간의 세션을 나타낸다. 또한, 메시지는 서버에게 어떤 암호화 알고리즘을 쓸 것인지, 그리고 SSL 의 버전과 브라우저가 지원하는 압축 방법 등을 알려준다. 마지막으로 브라우저가 생성해낸 난수를 포함한다. 서버 'hello' 메시지는 브라우저에 의해 제공되는 것들 중에서 압축 방법과 암호화 알고리즘, 적절한 SSL 버전과 다른 난수, 가능한 세션 ID 등을 선택해서 반응하게 딘다.
이 단계에서 클라이언트와 서버는 디지털 certificate 를 서로 교환하게 되는데, 이를 통해 서로의 내용을 확인하게 된다. 서버의 certificate 에는 handshake 프로토콜에서 선택된 public-키 암호화 알고리즘에 적합한 public 키를 포함하며, 이 키가 짧은 시간 이용된다. 그렇지만, 실제적인 트랜잭션은 private-키 암호화 기법을 이용해서 암호화된다.
이를 구현하기 위해서 서버와 클라이언트는 브라우저에 의해 생성된 하나의 private 키를 가지고, 서버로의 전송을 위한 마스터 키로 public 키만을 사용하는 것이 아니라 브라우저가 premaster secret 키를 대신 보낸다. 이미 정의된 프로토콜에 따라 서버는 premaster secret 키를 이용하여 실제 마스터 키를 결정한다. 이렇게 함으로써 실제 마스터 키를 전송할 필요가 없게 되며, 이런 프로세스가 완료되면 브라우저와 서버는 마스터 키의 복사본을 가지게 된다.

- 인터넷 익스플로러 보안 (Internet Explorer Security)

IE 3.0 은 SSL 과 PCT 를 모두 지원한다. PCT 는 SSL 과 마찬가지로 private 키를 암호화할 때 public 키 암호를 사용하는 방식을 이용한다. SSL 과 PCT 의 가장 큰 차이점은 바로 handshake 프로토콜 단계에 있다. PCT 는 호환 가능한 프로토콜과 접속하기 위해 보다 적은 수의 메시지를 요구하며, 더 많은 암호화 알고리즘을 지원한다. 또한, 인증(authentication)과 암호화에 서로 다른 키를 사용함으로써 보안의 정도가 높다.
IE 4.0 에서는 이 밖에도 자체적인 보안 구역(security zone)을 이용하여 사용자가 자신의 브라우저의 보안 레벨을 조절할 수 있도록 하고 있다. 각각의 보안 구역에 특정 행동만 할 수 있도록 보안 레벨을 부여할 수 있다. 예를 들어, 회사의 인트라넷 사이트를 trusted zone 으로 설정하면 특별한 암호화를 거치지 않고 이 사이트와 통신할 수 있다. 반대로 처음으로 방문하는 인터넷 사이트를 untrusted zone 을 설정하면, 항상 서버에게 브라우저가 정보를 보내기 전에 SSL 인증을 할 것을 요구하게 된다.

- 인터넷 보안의 미래 – TLS (Transport Layer Security)

TLS 프로토콜은 SSL 에 기초한 프로토콜로 앞으로 표준이 될 가능성이 높다.
참고로 신용카드 회사를 중심으로 새로운 보안 표준을 개발하고 있는데, 이것이 SET(Secure Electronic Transaction) 표준이다. SET 는 TLS 와 같은 프로토콜과 함께 사용할 수 있으며, 인증과 신뢰성에 초점을 맞춘 프로토콜이다

정 리 (Summary)

이번 장에서는 가장 기본적인 보안을 유지하기 위해 간단한 암호화 방법에서부터 현재 표준적으로 사용되고 있는 몇 가지 알고리즘을 소개하고, 차세대 암호화 표준 알고리즘으로 제안되고 있는 알고리즘을 구현한 컴포넌트를 이용하여 간단한 예제를 작성해 보았다. 그리고, 인터넷에서 제안되고 있는 몇 가지 암호화에 관한 프로토콜을 소개하였다.
최근에 이런 암호화와 보안의 중요성은 날이 갈수록 강조되고 있다. 물론 제대로 된 보안을 위해서는 하드웨어적인 보안과 설비 등이 필요하고 고려해야 될 요소가 많지만, 의외로 앞에서 설명한 간단한 방법으로도 웬만한 수준의 보안은 유지할 수 있으므로 조금만 더 신경을 써서 어플리케이션을 마무리 할 것을 권하는 바이다.