Journal of Korea Multimedia Society Vol. 9, No. 10, October 2006(pp. 1371-1379)

패딩 문자열 길이 정보를 이용한 패딩 알고리즘 설계

장 승 주†

요 약

본 논문에서는 여러 문자열 단위로 입력되는 문자열을 하나의 문자열로 구성하기 위하여 문자열 길이 정보를 이용한 패딩 알고리즘을 제안한다. 기존의 패딩 알고리즘은 단순히 공백 문자를 삽입함으로써 실제 문자열과 패딩 문자를 구분하지 못하는 문제점을 가지고 있다. 이러한 문제점을 해결하기 위하여 본 논문에서는 패딩하는 문자열 길이를 패딩 값으로 구성한다. 이렇게 함으로써 단순히 공백 문자나 "00"을 패딩하는 경우보다 문자열과 패딩 문자를 구분하는 것이 훨씬 용이하고, 정확히 동작된다. 본 논문에서 제안하는 패딩 알고리즘은 데이터 암호화 및 복호화 알고리즘에 사용가능하다.

Design of a Padding Algorithm Using the Pad Character Length

Seung-Ju Jang†

ABSTRACT

This paper suggests the padding algorithm using padding character length to concatenate more than one string without side-effect. Most existing padding algorithms padding null character in the empty location could not discriminate the real string from the padded character. To overcome this problem, in this paper, the padded character contains pad character length information. This mechanism is working better than NULL or "00" padding cases. The suggested padding algorithm could be effective for data encryption and decryption algorithms.

Key words: Padding Algorithm(패딩 알고리즘), Pad Character Length(패딩 문자열 길이 정보), Encryption(암호화), Decryption(복호화), File Concatenation(파일 연결)

1. 서 론

1980년 초반 이후, 개인용 컴퓨터인 PC가 널리 보급되면서 이와 함께 IT분야가 급격히 발전해왔다. 그리고 현재는 컴퓨터로 거의 대부분의 정보들을 처리하는 단계에까지 이르렀다. 또한 이와 더불어 방대한 정보를 축약하는 압축 기술이나 중요한 데이터에 대한 보안의 중요성이 대두하게 되었다. 요즈음 컴퓨터를 사용하는 사용자는 데이터 압축 기능과 보안 기능을 필수적으로 사용하고 있다. 압축 알고리즘과 데이터 보안 알고리즘에서는 블록 단위 데이터 처리가 필요하다. 이때 블록단위의 데이터 처리를 위해서 패딩 알고리즘을 사용한다. 패딩이란 "채워 넣는다"는 뜻으로 본 논문에서는 블록 단위 암호화 알고리즘에서 패딩 알고리즘에 대해 다루기로 한다. 현재 블록 단위 데이터 처리에서 패딩 알고리즘에 대한 연구가 이루어지고 있으며, 패딩 알고리즘을 적용한 데이터의 안정성 역시 이슈가 되고 있다.

패딩 알고리즘이 주로 사용되는 데이터 암호 알고리즘은 64 비트 블럭 암호 기법을 사용하는 DES, DES3, DESX, RC5, BLOWFISH, CAST128, IDEA, SAFER, RC2 등이 있으며, 128 비트 블럭 암호 기법으로 SEED, CRYPTON, RIJNDAEL, CAST256, RC6, TWOFISH, MARS, SERPENT 등을 사용한다

※ 교신저자(Corresponding Author) : 장승주, 주소 : 부산광역시 부산진구 엄광로 995(614-714), 전화 : 051)890-1710, FAX : 051)890-1619, E-mail : sjjang@deu.ac.kr

접수일 : 2006년 3월 28일, 완료일 : 2006년 8월 24일

† 정회원, 동의대학교 컴퓨터공학과 부교수

[1-3]. 그러나 현재 패딩 알고리즘은 파일의 끝 부분이 어디인지를 인식 시켜주기 위해서 사용되고 있다. 끝에 NULL문자나 패딩 문자열 갯수를 넣는다거나 파일 또는 문자열의 끝이라고 인식하도록 특정 문자를 넣고 나머지 바이트들을 NULL로 채우는 등 문자열의 끝을 인식한다.

또한 기존 패딩의 가장 원시적인 방법은 임의의 데이타를 삽입하여 평문이 8바이트가 되도록 하는 것이다. 가장 단순한 방법으로 데이터를 단순히 0으로만 채웠다고 가정하자. 그러면 일단 암호화는 되겠지만 평문에 0이 계속되는 구간이 있다면 문제가 발생할 것이다. 이러한 문제를 해결하기 위하여 복호화를 할 때 얼마나 많은 바이트가 패딩되었는지 알려주는 구별자를 평문에 표시하는 패딩 스킴을 사용하여 패딩 문자의 위치 구별 문제를 해결할 수 있다. 패딩 스킴은 공격하는 사람에게 암호문의 약점이 될 수 있는 여분의 정보를 주지 않는다.

본 논문에서는 기존 패딩 알고리즘의 문제점을 해결하기 위하여 문자열 길이 정보를 이용한 패딩 알고리즘을 설계한다. 본 논문에서 제안하는 문자열 패딩 알고리즘은 파일의 암호화, 복호화에 주로 사용된다. 데이터 암호, 복호 시에 데이터 쓰기, 읽기 각각에 대해서 패딩 알고리즘을 제안한다. 먼저 데이터 쓰기를 할 경우 패딩 알고리즘에서 입력되는 값은 두 가지이다. 이 값은 기록하고자 하는 파일의 전체 바이트 크기 값과 실제 데이터가 저장된 버퍼 공간의 주소 값이다. 출력되는 값은 패딩 처리된 후 새로운 버퍼의 주소 값에 반환된다. 파일 내에 쓰기를 할 경우 알고리즘은 패딩 시킬 총 바이트 수를 결정한다. 이 값이 결정되고 나면 패딩이 시도될 첫 번째 패키지의 패딩 구역에는 패딩되는 총 바이트 수를 기록한다. 두 번째 패키지의 적용 시 0번째 바이트에는 패딩 체크 문자를 기록하고, 나머지 1바이트부터 7바이트 위치에는 정해진 패딩 문자열을 기록한다.

본 논문에서 제안하는 패딩 알고리즘은 두개 이상의 파일을 연결하여 사용하는 환경에서 기존의 패딩 알고리즘은 두 번째 파일 데이터 부분에 대한 정확한 구분이 되지 않는 문제점을 가지고 있다. 이러한 문제를 해결하기 위하여 본 논문에서 패딩 알고리즘을 제안한다.

본 논문에서 제안하는 패딩 알고리즘은 Mpeg-4나 암호/복호화 알고리즘 등에 사용가능하다. 본 논문은 2장에서 관련 연구, 3장에서 논문에서 제안하는 패딩 알고리즘 설계 내용을 언급한다. 4장에서는 구현 및 분석을 언급한다. 그리고 5장에서는 결론을 맺는다.

2. 관련 연구

일반적인 경우의 암호화, 복호화 알고리즘은 블록 단위로 암호화를 하면서 자체적인 패딩 모드를 가지고 있다. 블록 암호화 알고리즘은 주어진 평문을 정해진 길이의 블록(64 혹은 128 비트)으로 나누어 암호화를 수행하는 알고리즘이다. 이 블록 암호화 알고리즘은 여러 가지 '모드'로 구현되며 '패딩(덧붙이기)'이 필요하다. 일반적으로 블록 암호화 알고리즘은 출력을 생성하기 위해서, 지정된 입력에 대해서 실행하는 조작 (또는 일련의 조작)을 하게 된다. 변환에는 항상 암호화 알고리즘 명 (DES 등)이 포함되어 있으며 피드백 모드와 패딩 방식을 사용 한다 [4-7].

일반적으로 보안 알고리즘에서 패딩의 기능은 필수적으로 사용되고 있다. 패딩 알고리즘은 대칭키 암호 및 비대칭 암호에서 많이 사용되는 개념이다. Triple-DES는 8바이트에 대해서만 동작한다. 그러나 정확히 8바이트 단위로 데이터가 입력 되지 않는 경우가 있다. 이때 부족한 문자열에 대해서 패딩을 해야 한다. 패딩의 가장 원시적인 방법은 임의의 데이타를 삽입하여 평문이 8바이트가 되도록 하는 것이다. 가장 단순한 방법으로 데이터를 단순히 0으로만 채웠다고 가정하자. 그러면 일단 암호화는 되겠지만 평문에 0이 계속되는 구간이 있다면 문제가 발생할 것이다. 이러한 문제를 해결하기 위하여 복호화를 할 때 얼마나 많은 바이트가 패딩되었는지 알려주는 구별자를 평문에 표시하는 패딩 스킴을 사용하여 패딩 문자의 위치 구별 문제를 해결할 수 있다. 패딩 스킴은 공격하는 사람에게 암호문의 약점이 될 수 있는 여분의 정보를 주지 않는다 [8-11].

공개키 암호화의 경우 가장 널리 사용되고 있는 알고리즘 중의 하나는 RSA 이다. RSA 알고리즘은 다음과 같은 문제점을 가지고 있다. 공격자가 직접적으로 제공한 원값 (raw value) 을 RSA를 사용해 복호화하거나 서명해서는 안된다. 정확히 동일한 원값을 여러번 복호화하거나 서명해서는 안된다. 이러한 문제는 임의의 패딩 문자열을 추가함으로써 해결될

수 있다[12-14].

DES-CBC 알고리즘은 평문이 6 mod 8 octet이 되도록 패딩을 수행한다. 이때 패딩 octet의 길이를 나타내는 Pad Length Octet 정보를 추가한다. 기존의 패딩 방식은 다음의 표 1과 같이 4가지 부류로 나뉜다.

표 1. 기존의 패딩 방식

Padding Algorithm	Padding 방식에 대한 설명
standard padding	일반적인 패딩 방식으로서 처리할 블록의 남는 공간에 임의의 문자를 채워 넣은 후 맨 끝에 padding 한 개수를 집어넣어서 패딩 된 데이터임을 표시하는 방식
space padding	처리할 블록의 남는 공간을 공백으로 채우는 방식
null padding	처리할 블록의 남는 공간을 null로 채우는 방식
one-and-zeroes padding	end-character-padding 이라고도 불리며 처리할 데이터 블록의 마지막에 원문의 끝임을 나타내는 문자를 넣은 후 남은공간을 임의의 문자로 채우는 방식

2.1 기존 데이터 처리 방식의 문제점

기존 패딩 방식으로 유닉스에서 많이 사용하는 명령어인 redirection 기능을 이용한 파일 연결의 경우에 대한 문제점을 살펴보면 다음 그림 1과 같다.

위의 그림 1에서 (a)는 일반적인 데이터 파일내의 각 바이트 위치에서 데이터를 보여 준다. "a" 파일은 2 패키지 크기(1 패키지는 각각의 8바이트 단위를 나타냄)인데, 두 번째 패키지내의 데이터 위치가 14번째 바이트에서 끝난다. "b" 파일도 2 패키지 크기인데 두 번째 패키지 위치가 12번째에서 끝난다. 그림 1의 (a)에서 두 개의 파일 데이터를 연결한 파일 모양이 "c"이다. 그런데 그림 1의 (b)에서처럼 파일 데이터를 구성하여 "c"파일을 읽을 경우 데이터를 정확히 읽을 수 없는 문제가 발생한다.

만약 1 패키지 단위로 암호화 시킨 후에 UNIX 명령어에서 double redirection을 이용하여 두 개의 파일을 서로 연결시켜 또 하나의 데이터 파일을 만든다면 복호화 시 블록 단위를 잘못 인식하여 정상적인 데이터 파일로 복구될 수 없게 된다.

예를 들어 그림 1의 "a"파일은 "A B C D E F G H", "I J K L M N \n"을 각각 암호화 시켰으며, "b"

파일 a (num%8!=0)

index	0	1	2	3	4	5	6	7
내용	A	B	C	D	E	F	G	H
index	8	9	10	11	12	13	14	15
내용	I	J	K	L	M	N	\n	

파일 b

index	0	1	2	3	4	5	6	7
내용	O	P	Q	R	S	T	U	V
index	8	9	10	11	12	13	14	15
내용	W	X	Y	Z	\n			

(a)

index	0	1	2	3	4	5	6	7
내용	A	B	C	D	E	F	G	H
index	8	9	10	11	12	13	14	15
내용	I	J	K	L	M	N	\n	O
index	16	17	18	19	20	21	22	23
내용	P	Q	R	S	T	U	V	W
index	24	25	26	27	28	29	30	31
내용	X	Y	Z	\n				

(b)

그림 1. 유닉스의 redirection 매카니즘을 이용한 파일 연결: (a)일반적인 데이터 파일 "a"와 "b", (b) 일반적인 데이터 파일 "a"와 "b" 파일을 연결한 "c" 파일의 모양

파일은 "O P Q R S T U V", "W X Y Z \n"을 각각 암호화 시켰다. 그러나 이 두 파일을 UNIX 명령어에서 double redirection을 이용하여 합쳐 "c"파일을 만든 후 그 내용을 복호화하는 경우 "A B C D E F G H" 부분은 암호화되기 전과 암호화된 후의 블록안 내용의 상태가 동일하므로 정상적으로 복호화 되지만, "I J K L M N \n O"부분의 복호화를 수행 할 경우 마지막 문자인 "O" 부분은 블록 암호화 시 사용되지 않은 부분이므로 복호화 착오 현상이 발생하여 임의의 다른 문자로 복호화 된다. 또한 그 다음 패키지인 "P Q R S T U V W"의 복호화 역시 암호화 시켰을 때의 블록의 내용인 "O P Q R S T U V"와는 달라졌으므로 제대로 복호화 되지 않는다.

2.2 기존 패딩 방식의 문제점

패딩은 블록 암호화 시 비트수를 맞춰주기 위해서 특정 비트를 삽입하는 것을 말한다. 이때 가장 중요시 되는 것은 패딩하고자 하는 위치에 어떤 데이타를 넣어줄 것이며, 이것을 어떻게 읽을 것인가 하는 것이다. 다음 그림 2는 기존의 패딩 방식을 이용하여 데이터 처리를 한 결과를 보여준다.

그림 2는 정상적인 패딩 방법 중 두 가지 예를 나타낸 것이다. 그러나 이 두 가지 패딩 방법에는 허점이 있다. (a) 데이터 파일의 경우 00이라는 char가 출현할 가능성이 크므로 위와 같은 패턴이 일어나는 파일이 존재한다면 (a)는 정상적으로 복호화가 되지 않는다. 또한 (b)의 경우는 원문의 끝을 나타내기 위한 끝 char로 "*"이 사용되었지만, ASCII 코드는 256자밖에 되지 않으므로 끝 char가 나타날 가능성은 1/256이다. 그러므로 (b)의 경우 또한 정확한 복호화가 불가능하다고 볼 수 있다.

그러므로 패딩을 정확하게 하기 위해서는 어떤 bit들을 패딩 문자들로 채워 줄 것 인지가 가장 큰 문제가 되며, 일반적인 데이타에서 패딩 한 것과 유사한 데이터가 존재하는 경우가 발생한다면 패딩인지 아닌지를 구별하는 것이 중요하다.

3. 패딩 알고리즘 설계

3.1 쓰기에서 패딩 알고리즘 설계

데이터 쓰기를 수행하고자 할 경우에 패딩 알고리즘은 데이터를 저장 매체에 기록을 하고자 할 경우에 적용 가능하다. 다음은 본 논문에서 제안하는 쓰기 연산에서 패딩 알고리즘이다.

그림 3에서 먼저 데이터 쓰기를 할 경우 패딩 알고리즘에서 입력되는 값은 두 가지 이다. 이 값은 기록하고자 하는 파일의 전체 바이트 크기 값과 실제 데이터가 저장된 버퍼 공간의 주소 값이다. 출력되는 값은 패딩 처리된 후 새로운 버퍼의 주소 값이 반환된다. 파일 내에 쓰기를 할 경우 패딩 알고리즘은 패딩 시킬 총 바이트 수를 결정한다. 이 값이 결정되고 나면 첫 번째 패딩 위치에는 패딩되는 총 바이트 수를 기록한다. 두 번째 패딩 라인 적용 시 패딩 라인의 처음부터 위치에는 정해진 패딩 문자열을 기록한다.

index	0	1	2	3	4	5	6	7
내용	A	B	C	D	E	F	G	H
index	8	9	10	11	12	13	14	15
내용	I	J	K	L	M	N	\n	00
index	16	17	18	19	20	21	22	23
내용	00	00	00	00	00	00	00	9

(a)

index	0	1	2	3	4	5	6	7
내용	A	B	C	D	E	F	G	H
index	8	9	10	11	12	13	14	15
내용	I	J	K	L	M	N	\n	*
index	16	17	18	19	20	21	22	23
내용	00	00	00	00	00	00	00	00

(b)

그림 2. 기존의 패딩 방식을 이용한 데이터 처리의 예: (a) 파일 d (standard padding 기법 : 패딩 문자 = 00), (b) 파일 e (end-character-zeroes padding 기법 : 끝 문자 = *, 패딩 문자 = 00)(음영 처리 부분 : 패딩된 값들을 나타냄)

```
padding_for_write_op(int total_bytes, buf)
input
   total_bytes : 기록하고자 하는 파일 전체 크기
   buf : 실제 데이터가 저장된 버퍼
output
   패딩 처리된 새로운 버퍼 주소 값
begin
   패딩 시킬 바이트 수를 결정;
   첫 번째 패키지의 패딩 시켜야 할 부분들에는 패딩
   되는 바이트 값을 기록;
   두 번째 패키지 패딩 적용 시 0바이트 위치에 패딩
   체크 문자열을 넣고
      나머지 바이트들(1~7)에는 패딩 되는 바이트
      값을 기록;
end
```

그림 3. 데이터 쓰기 시 패딩 알고리즘

나머지 위치에는 패딩되는 총 바이트 수를 기록한다. 패딩 알고리즘을 암호 알고리즘에 적용한 예는 다음 그림 4와 같다.

그림 4의 ①번 단계에서 패딩 시킬 문자의 갯수를 결정한 후, ②번 단계에서 2번째 패키지의 모자라는 부분을 문자 갯수 만큼 저장한다. ③번 단계에서 3번째 패키지의 첫 번째 바이트에 패딩을 구별하기 위한 임의의 약속된 문자 "Z"를 삽입하고 ④번 단계에서 남은 부분을 ②번 단계처럼 문자 갯수를 저장하여 패딩을 끝낸다. 이때 기존의 방식의 문제점과 마찬가지로 Z라는 문자열이 나타날 가능성에 대한 부분을 방지하기 위하여 패딩된 문자열의 길이 정보를 추가하여 이러한 문제를 극복하였다.

① 패딩 시킬 byte 수를 결정 (아래에서는 8-7+8=9 byte)

Index	0	1	2	3	4	5	6	7
내 용	A	B	C	D	E	F	G	H
Index	8	9	10	11	12	13	14	15
내 용	I	J	K	L	M	N	\n	1byte
Index	16	17	18	19	20	21	22	23
내 용	2byte	3byte	4byte	5byte	6byte	7byte	8byte	9byte

② 패딩 시킬 첫 번째 라인만 byte 수의 값으로 패딩 시킴

Index	0	1	2	3	4	5	6	7
내 용	A	B	C	D	E	F	G	H
Index	8	9	10	11	12	13	14	15
내 용	I	J	K	L	M	N	\n	09
Index	16	17	18	19	20	21	22	23

③ 패딩 시킬 두 번째 라인의 첫 번째 byte(index 16)에 read 시 패딩임을 체크가 가능하도록 임의의 약속된 문자(지금은 Z라고 정의함)를 삽입

Index	0	1	2	3	4	5	6	7
내 용	A	B	C	D	E	F	G	H
Index	8	9	10	11	12	13	14	15
내 용	I	J	K	L	M	N	\n	09
Index	16	17	18	19	20	21	22	23
내 용	Z							

④ 남은 나머지 byte들을 패딩 byte 값을 사용하여 패딩 시킴

Index	0	1	2	3	4	5	6	7
내 용	A	B	C	D	E	F	G	H
Index	8	9	10	11	12	13	14	15
내 용	I	J	K	L	M	N	\n	09
Index	16	17	18	19	20	21	22	23
내 용	Z	09	09	09	09	09	09	09

그림 4. 문자열 정보를 이용한 쓰기 패딩 알고리즘의 적용 예

```
padding_for_read_op(int total_bytes, buf)
input
    total_bytes : 읽고자 파일의 전체 크기
    buf : 실제 데이터가 저장된 버퍼 공간
output
    패딩 처리를 제거한 데이터가 저장된 버퍼 주소값
begin
    복호된 데이터가 저장된 buffer 내의 데이터를 8개 바이트씩 나눔
    2번째 패키지부터 각 패키지의 0 번째 문자가 패딩 체크 문자인지 체크
    if padding check char then
            7번째 문자에서 패딩된 문자열 수(바이트 수)를 읽어옴
            if 8 < 패딩된 문자열 수 < 16 then
                    padding_sp = 패키지의 끝 index - 패딩 문자열 수
                    for(padding_sp ~ 패딩 문자열 수)
                            if(패딩된 문자 == 패키지의 끝 문자)
                                    padding_num++
                    if padding_num == 패딩 문자열 수 - 1
                            padding 구역을 삭제함
                    end if
            end if
    end if
end
```

그림 5. 데이터 읽기시 패딩 알고리즘

3.2 읽기에서 문자열 정보를 이용한 패딩 알고리즘 설계

문자열 패딩 알고리즘을 이용하여 기록을 한 데이터에 대해서 읽기를 할 경우에 문자열 정보를 이용한 패딩 알고리즘의 동작은 다음 그림 5와 같다.

그림 5는 데이터 읽기 시 패딩 알고리즘을 나타낸다. 입력되는 값은 읽고자하는 파일의 전체 크기, 실제 데이터를 읽어낼 버퍼 공간 주소이다. 출력되는 값은 패딩 처리를 제거한 데이터가 저장된 버퍼 주소값이다. 먼저 복호된 데이터가 저장된 버퍼 내의 데이터를 8 바이트씩 나눈다. 2번째 패키지부터 0 번째 문자가 패딩 체크 문자(여기서는 "Z")인지 아닌지를 확인한다. 만약 이것이 패딩 체크 문자이면 7번째 문자에서 패딩된 문자열 갯수를 가지고 와서 합당한 패딩 구역 내에 들어있는지 확인한다. 만약 패딩된 문자열이 8바이트보다 작거나 같은 경우 또는 16바이트 보다 큰 경우 패딩 구역의 크기에 위배되므로 오류이다. 만약 합당하다면, 이 7번째 문자는 패딩된 문자의 갯수와 패딩된 문자일 가능성이 크다. 패딩 시작점을 계산하여 패딩 시작점부터 현재 패키지의 끝까지 패딩된 문자의 일치성을 탐색한다. 만약 패딩된 문자-1의 일치성을 보인다면 패딩 구역이 확실하고 따라서 이 구역을 제거한다. 그림 6은 데이터 읽기 시 패딩 알고리즘을 이용한 예제를 나타낸다.

그림 6의 읽기 패딩 알고리즘을 간략히 설명하면 다음과 같다. 우선 한 패키지씩 나눈 후 첫 번째(index 0)byte에서 패딩이 있었음을 나타내는 문자가 존재하는지 찾고, 패딩 체크 문자가 존재하는 라인의 마지막 byte가 정해진 구역 내의 문자라면 일단 패딩으로 예상하고 탐색하여 패딩이 정확하다면 삭제를 시킨다. 파일의 끝까지 읽기 패딩 알고리즘을 반복 수행하면 원하는 정보만 남게되고, 시스템에는 남은 버퍼의 사이즈와 버퍼의 값을 반환해 주면 읽기 패딩은 끝난다.

4. 구현 및 분석

본 논문에서 제안한 패딩 알고리즘을 시스템에 구현한 결과에 대해서 분석한다. 실제 구현한 시스템 환경은 3.1과 같다. 본 논문에서 제안한 패딩 알고리즘을 적용한 환경은 blowfish 알고리즘을 대상으로 하였다. blowfish 알고리즘[15]을 대상으로 구현한 결과 기존 패딩 알고리즘은 2장에서 지적한 바와 같이 UNIX의 redirection 명령어($ cat a >> b)와 같은 경우에 데이터 암호, 복호 시에 문제가 발생하는 것을 확인할 수 있었다. 그러나 본 논문에서 제안한 알고리즘을 blowfish 에 구현한 다음 UNIX 의 redirection 명령어($ cat a >> b) 를 실행한 결과 정상

① 첫 번째(index 0)byte에서 패딩이 있었음을 나타내는 문자(위에서 Z로 정의)가 존재하는지 찾는다.

Index	0	1	2	3	4	5	6	7
내 용	A	B	C	D	E	F	G	H
Index	8	9	10	11	12	13	14	15
내용	I	J	K	L	M	N	\n	09
Index	16	17	18	19	20	21	22	23
내용	Z	09	09	09	09	09	09	09

② 패딩 체크 문자가 존재하는 라인의 마지막 byte가 (09~15)안에 속하는지 비교하고 속한다면 padding byte일 가능성이 많으므로 패딩으로 예상되는 구역을 탐색한다.

Index	0	1	2	3	4	5	6	7
내 용	A	B	C	D	E	F	G	H
Index	8	9	10	11	12	13	14	15
내 용	I	J	K	L	M	N	\n	09
Index	16	17	18	19	20	21	22	23
내 용	Z	09	09	09	09	09	09	09

(주 padding byte가 09이므로 끝에서 9개의 byte들을 마지막 byte와 비교하여 같은 갯수를 얻는다. [padding byte값-1]의 일치도를 보이면 패딩이라고 정의한다. 위의 예에서는 8개의 byte가 09이므로 [9-1=8]이 되고, 패딩이라고 볼 수 있다.)

③ 2번의 과정에서 찾은 수만큼 삭제를 시킨다.

Index	0	1	2	3	4	5	6	7
내 용	A	B	C	D	E	F	G	H
Index	8	9	10	11	12	13	14	15
내 용	I	J	K	L	M	N	\n	
Index	16	17	18	19	20	21	22	23
내 용								

④ 위의 [1~4]의 과정을 반복하면 앞의 패딩을 제거한 정보들만 남게 되는데, 그것을 제거해 주고, 버퍼의 사이즈를 맞춰 줘야한다. 메모리를 새로운 변수로 할당받아 위의 패딩을 제거한 총 문자들 만큼의 길이로 새로운 버퍼를 할당 받아 원래의 버퍼와 교환해주면 된다.
(만약 종료 문자['\0']가 필요하다면 종료 문자를 넣는 코드도 삽입해준다.)

그림 6. 문자열 정보를 이용한 읽기 패딩 알고리즘의 적용 예

적으로 동작함을 확인할 수 있었다.

기존의 암호화 알고리즘을 사용한 경우의 UNIX cat 명령어를 실행했을 때의 화면은 그림 8.과 같다. 본 논문에서 제안하는 패딩 알고리즘을 적용하여 UNIX 명령어를 실행한 경우의 화면은 다음 그림 8.와 같다.

그림 8의 경우는 패딩 알고리즘에서 정상적으로 파일에 첨부를 하면서 패딩 문자열 처리가 이루어지지 않음을 보여준다. 그림 9의 경우는 패딩 알고리즘에서 정상적으로 파일에 첨부를 하면서 패딩 문자열 처리가 이루어지는 것을 보여준다. 위 구현 결과와 같이 본 논문에서 제시한 패딩 알고리즘이 정상적으로 작동함을 확인할 수 있었다.

5. 결 론

본 논문에서는 여러 문자열 단위로 입력되는 문자열을 하나의 문자열로 구성하기 위하여 패딩을 하는 알고리즘을 제안했다. 기존의 패딩 방식은 단순히 공백 문자를 삽입함으로써 실제 문자열과 패딩 문자를 구분하지 못하는 문제점을 가지고 있다. 이러한 문제점을 해결하기 위하여 본 논문에서는 패딩하는 문자열 길이를 패딩 값으로 구성한다. 이렇게 함으로서 단순히 공백 문자나 "00"을 패딩하는 경우보다 문자열과 패딩 문자를 구분하는 것이 훨씬 용이하고, 정확히 동작된다.

본 논문에서 제안하는 문자열 패딩 알고리즘은 파

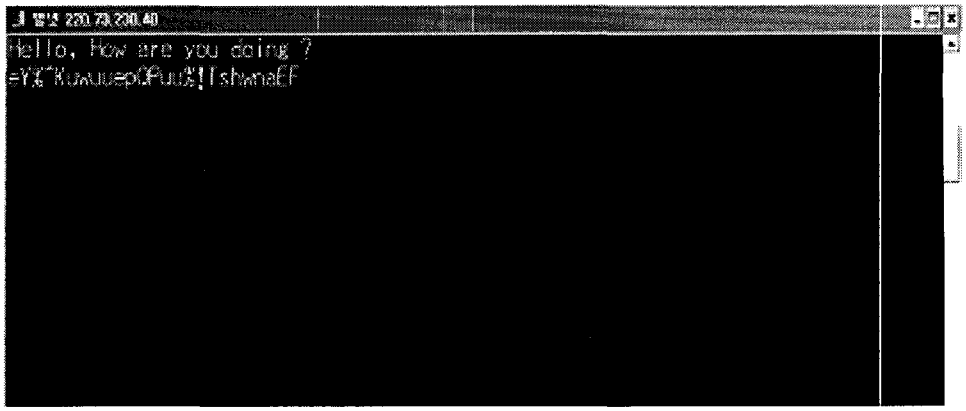

그림 7. 기존의 패딩 알고리즘을 이용한 UNIX cat 명령어 실행후의 결과 파일

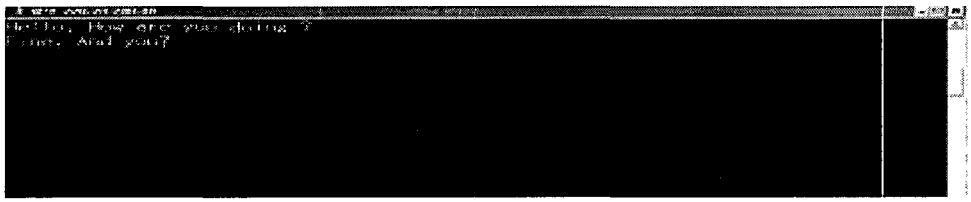

그림 8. 제안한 패딩 알고리즘을 이용한 UNIX cat 명령어 실행후의 결과 파일

일의 암호화, 복호화에 주로 사용된다. 데이터 쓰기, 읽기 각각에 대해서 패딩 알고리즘을 제안한다. 먼저 데이터 쓰기를 할 경우 패딩 알고리즘에서 입력되는 값은 두 가지이다. 이 값은 기록하고자 하는 파일의 전체 바이트 크기 값과 실제 데이터가 저장된 버퍼 공간의 주소 값이다. 출력되는 값은 패딩 처리된 후 새로운 버퍼의 주소 값이 반환된다.

다음으로 데이터 읽기에 대한 패딩 알고리즘이다. 먼저 복호된 데이터가 저장된 버퍼 내의 데이터를 8 바이트씩 나눈다. 2번째 패키지부터 0번째 문자가 패딩 체크 문자이면 7번째 문자에서 패딩된 문자열 개수를 받아와 합당한 패딩 구역 내에 들어있는지 확인 후 패딩 시작점을 계산하여 패딩 시작점부터 현재 패키지의 끝까지 패딩된 문자의 일치성을 탐색한다.

본 논문에서 제시한 알고리즘을 UNIX 운영체제에 적용하여 구현한 결과 패딩 알고리즘이 정상적으로 동작함을 확인할 수 있었다. 본 논문에서 제안한 패딩 알고리즘은 두 개 이상의 파일을 연결하여 사용하는 환경에 초점을 맞추어 개발되었다. 따라서 앞으로 일반적인 상황에도 맞는 다양한 데이터 포맷에 대한 패딩 실험을 통하여 보다 일반화된 형태의 연구는 계속 진행할 예정이다.

참 고 문 헌

[1] Rohit Garg, Chris Y. Chung, Donglok Kim, and Yongmin Kim, "Boundary Macroblock Padding in MPEG-4 Video Decoding Using a Graphics Coprocessor," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, VOL. 12, NO. 8, pp. 1102-1115, AUGUST 2002.

[2] Chris McNab, *Network Security Assessment*, O'REILLY, 2003.

[3] M.Rhee, *Cryptography and Secure Communication*, McGraw-Hill Telecom, 1994.

[4] 윤한성, *정보보안과 암호화*, 21세기사, 2003.

[5] John Black and Hector Urtubia, "Side Channel Attacks on Symmetric Encryption Schemes : The Case for Authenticated Encryption," *11th Proceedings of Usenix Security*, Vol. 11, No 1, pp. 220-233, 2002.

[6] Michael Welschenhach , *C와 C++로 구현하는 암호화 알고리즘*, 인포북, 2003.

[7] Christoph Heer, Muenchen, Kay Migge, and Muenchen, "VLSI hardware accelerator for

the MPEG-4 padding algorithm (Paper #: 3655-14)," *SPIE Proceedings*, Vol. 3655, 1999.

[8] Jone E. Hershey, *Cryptography Demystified*, McGraw-Hill Telecom, 2003.

[9] Bruce Schneier, *Applied Cryptography : Protocols, Algorithms and Source Code in C*, John Wiley & Sons, 1996.

[10] Niels Ferguson and Bruce Schneier, *Practical Cryptography*, John Wiley & Sons, 2003.

[11] William Stallings, *Cryptography and Network Security PRINCIPLES AND PRACTICES*, Third Edition, Prentice Hall, 2003.

[12] Blake Dournaee, *XML Security*, chapter 2 p25-p29, InfoBook, 2002.

[13] Keith Lockstone, CIX : http://www.cix.co.uk/~klockstone/bookends.htm, *Random Prefixes and Suffixes*, November 1995,

[14] Christopher J. Alberts, *Managing Information Security Risks*, Addison Wesley, 2002.

[15] http://www.schneier.com/blowfish.html.

장 승 주

1985년 부산대학교 계산통계학과(전산학) 학사

1991년 부산대학교 계산통계학과(전산학) 석사

1996년 부산대학교 컴퓨터공학과 박사

1987년~1996년 한국전자통신연구원 시스템 S/W연구실

1993년~1996년 부산대학교 시간강사

2000년~2002년 University of Missouri at Kansas City, visiting professor

1996년~현재 동의대학교 컴퓨터공학과 부교수

관심분야 : 운영체제, 임베디드 시스템 운영체제, 분산시스템, 시스템 보안