Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.26, NO.6, Dec. 2016 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2016.26.6.1539

의사 난수 생성 방식을 이용한 EncFS의 취약점 개선 연구

정 원 석,† 정 재 열, 정 익 래‡
고려대학교 정보보호대학원

The Vulnerability Improvement Research Using Pseudo-Random Number Generator Scheme in EncFS

Won-Seok Jeong,† Jaeyeol Jeong, Ik Rae Jeong‡
Graduate School of Information Security, Korea University

요 약

현대 사회에서 스토리지를 필요로 하는 애플리케이션이 많아졌다. 그 중에서 특히 핀테크의 발달은 스토리지의 보안 필요성을 크게 증대시켰다. 스토리지에는 사용자의 민감 정보가 저장되어 있는데 이 정보들이 노출되면 사용자들이 금전적인 피해를 입게 된다. 따라서 이러한 사고를 막기 위해 스토리지 암호화를 수행해야 하는데 스토리지 암호화를 수행하는 애플리케이션 중 하나로 EncFS가 있다. EncFS는 IV를 이용하여 암호화를 수행하는데 IV 값은 EncFS 단위 블록마다 유일해야 한다. 하지만 동일한 IV 값이 생성되는 경우들이 존재한다. 이를 이용해 암호화 과정을 조작하여 평문과 암호문 간의 관계를 알 수 있는 취약점이 존재한다. 본 논문에서는 해당 취약점이 IND-CPA에 안전하지 않음을 보이고 이를 개선하는 기법을 제안한다.

ABSTRACT

In modern society, the number of applications, which needs storage, is increased. Among them, the advance of FinTech increased the importance of storage encryption. FinTech storage, storing sensitive information, should be kept secure. Unless the storage is kept, many users will be damaged monetarily. To prevent this problem, we should encrypt the storage. A EncFS, which is one of the most popular storage encryption application, uses different IVs for each block to provide higher levels of security in the encryption algorithm. However, there is a vulnerability related to the usage of same IVs. In this paper, we propose a technique that decrypts the ciphertexts without knowing the secret key by using the vulnerability. Moreover, we show that the EncFS is not secure under IND-CPA model and propose a new scheme which is secure under IND-CPA model.

Keywords: EncFS, AES, CBC, CFB, Pseudo-Random Number Generator

I. 서 론

현대 사회에서 무선 인터넷과 와이파이(Wi-Fi)의 사용이 확대되면서 하루에 생성되는 데이터의 양도 과거에 비해 급격하게 증가했다. 이로 인해 고객에게 편의를 제공하는 서비스인 메신저 앱이나 클라우드 데이터베이스에서 데이터를 저장하는 스토리지의 수요가 크게 늘었다. 특히, 최근 들어 핀테크(Fintech) 기술의 도입으로 어디에서나 금융 서비스를 이용할 수 있게 되었다. 이러한 서비스들을 이용하는 사용자들의 카드 정보나 결제 비밀번호와 같은 민감 정보들이 스토리지에 저장되기 때문에 그 중

Received(11. 02. 2016), Modified(12. 13. 2016), Accepted(12. 13. 2016)
† 주저자, ms.windows.dev@gmail.com
‡ 교신저자, irjeong@korea.ac.kr(Corresponding author)

요성이 더 커졌다. 왜냐하면 이 민감 정보들이 외부로 노출되면 사용자에게 금전적으로 막대한 피해를 줄 수 있기 때문이다. 따라서 이를 보호하기 위해서는 해당 스토리지의 암호화 과정이 필수적이다.

NIST에서는 이러한 데이터 스토리지를 암호화하기 위한 지침[1]을 작성했고 여기에서는 스토리지 암호화 기술을 전체 디스크 암호화(Full Disk Encryption), 볼륨 암호화(Volume Encryption), 가상 디스크 암호화(Virtual Disk Encryption), 파일 및 폴더 암호화(File·Folder Encryption)의 4가지로 구분했다. 여기에서 전체 디스크 암호화 기술과 볼륨 암호화 기술은 사용하는 볼륨이 하나일 경우에 매우 유사하게 동작한다. 이 기술들이 적용된 대표적인 애플리케이션(application)으로는 Windows 운영체제에 내장된 BitLocker[2], Mac OS에 내장된 FileVault 2[3] 등이 있다. 그리고 가상 디스크 암호화 기술과 파일·폴더 암호화 기술은 나머지 2가지 기술들과는 다르게 개발자들이 구현하기에 상대적으로 쉽기 때문에 다양한 애플리케이션들이 있다. 그 중 대표적인 애플리케이션으로는 EncFS[4], AxCrypt[5] 등이 있다.

이 중 EncFS는 리눅스(linux) 운영체제 환경에서 데이터가 저장된 폴더를 암호화하는 애플리케이션이다. EncFS는 eCryptfs[6,7]와 같은 기존 애플리케이션과 다르게 시스템 커널(system kernel) 영역이 아니라 사용자 영역에서 FUSE(File system in USErspace)[8]라는 프레임워크를 이용해 파일 및 폴더 암호화를 수행한다. 이를 통해 커널 영역에 접근할 수 있는 루트(root) 권한을 가지지 않은 일반 사용자들도 파일 및 폴더 암호화 요청을 할 수 있다.

EncFS는 AES 암호 알고리즘을 이용하며 블록 암호화와 스트림(stream) 암호화 방식을 혼용하는데 기존의 블록 암호의 블록 크기와는 다른 EncFS에서 자체적으로 설정한 블록 크기를 만족하는 데이터를 CBC(Cipher Block Chaining)[9] 모드를 이용해 암호화하고, 블록 크기를 다 채우지 못하는 맨 마지막 블록에 대해서는 CFB(Cipher FeedBack)[9] 모드를 사용해 암호화한다. 즉, 맨 마지막에 존재하는 하나의 블록 크기를 만족하지 못하는 블록과 그 블록 앞에 존재하는 하나의 블록 크기를 만족하는 블록들의 암호화 모드를 다르게 사용한다. 이 과정에서 block IV라는 각 블록마다 고유한 값을 가지는 IV를 생성하여 사용한다.

그런데 동일한 block IV가 생성되는 경우가 있어 이를 통해 블록 암호화 과정과 스트림 암호화 과정을 조작하여 일부 평문 정보를 얻을 수 있고 평문과 암호문 사이의 관계가 노출되는 취약점이 존재한다. 본 논문에서는 EncFS의 암호화 기법의 취약점을 설명하고 Bellare 등이 정의한 모델[10]을 이용해 IND-CPA에 안전하지 않음을 보인다. 그리고 의사 난수 생성기(Pseudo-Random Number Generator)[11, 12]를 이용해 취약점을 개선하는 기법을 제안하고 앞서 정의된 안전성 모델에 안전함을 보인다. 본 논문에서는 생성되는 block IV에 난수성을 주기 위해 의사 난수 생성기를 사용했다. 하지만 해시 함수와 같은 방법을 이용해 유일한 block IV를 생성해도 무방하다.

본 논문의 구성은 2장에서 제안하는 기법들의 배경지식에 대해 설명한다. 3장에서는 EncFS의 암호화 방식에 대해 설명하며, EncFS 내의 취약점과 이것의 안전성을 4장에서 분석하고 5장에서 이 취약점을 개선하기 위한 기법을 제안하고 이 기법이 IND-CPA 공격에 대해 안전함을 보인다. 그리고 6장에서는 기존 기법과의 성능 비교를 통해 이 기법의 효율성을 분석한다.

1.1 관련 연구

EncFS[4] 2003년 Van. Gough에 의해 개발된 파일 및 폴더 암호화 어플리케이션으로 2003년 인터넷을 통해 정식 배포되었으며 현재 1.9.1 버전까지 개발되었다. 본 논문에서 제시하는 취약점은 2014년 1월에 Taylor Holtby의 EncFS Security Audit[13]에서 동일한 IV가 사용될 수 있다는 문제가 처음 제기되었으며 이 경우에 CBC나 CFB 모드로 안전성을 만족시킬 수 없다고 했지만 이 취약점으로 인해 어떤 문제가 발생할 것인지에 대한 언급은 없었다. 이 취약점을 해결하기 위해 초기에는 XTS 모드를 이용해 이를 해결해야 한다는 주장이 제기되었지만 이 주장도 2014년 4월 Thomas Ptacek과 Erin Ptacek[14]에 의해 반박되었다. 이를 해결하기 위해 V. Gough는 파일의 inode를 이용하여 EncFS의 Reverse encrypt 모드를 통해 이를 개선했지만 이 모드는 기본 암호화 모드가 아니기 때문에 이를 인지하지 못한 사용자는 기본 모드를 사용할 것이다. 그리고 해당 모드에서 inode를 이용해 생성한 IV는 SHA1을 이용해 암호화하여 사

용하기 때문에 이에 대한 안전성 분석이 필요하다. 이 취약점이 밝혀진 직후, V. Gough는 EncFS 위키[15]에 이를 EncFS의 1.x 버전이 아닌 2.x 이후의 버전에서 개선할 문제로 분류한 상태이며 현재 배포된 1.9.1 버전까지도 기본 암호화 모드에서 취약점이 존재한다. 따라서 본 논문에서는 기본 암호화 모드에서의 동일한 IV가 생성되는 취약점을 개선했으며 IND-CPA에서 이것이 안전함을 보였다.

eCryptFS[6,7] eCryptFS는 EncFS[4]처럼 리눅스 운영체제에서 사용되는 파일 및 폴더 암호화 어플리케이션이다. 암호화된 파일 시스템을 제공하기 위해 기존 파일 시스템 위에 추가되는 구조이며 암호화 과정에서 필요한 메타데이터가 암호화되는 파일의 헤더 부분에 저장된다. eCryptFS는 EncFS와 가장 많이 비교된다. 같은 목적을 가진 어플리케이션이면서 동일한 환경에서 동작하는 것이 공통점이라고 할 수 있으며 eCryptFS와 같은 기존 어플리케이션들은 시스템 커널 영역에서 암호화 작업을 수행하지만 EncFS는 사용자 영역에서 수행하기 때문이다. 본 논문에서는 EncFS가 사용자 영역에서 암호화 작업을 수행하는 어플리케이션이라는 점에서 공격자가 관리자 권한을 요구하는 커널 영역까지 접근하는 어려움 없이 현존하는 취약점을 이용해 공격할 수 있기 때문에 EncFS의 취약점에 대한 안전성 분석을 진행했다.

II. 배경지식

2.1 의사 난수 생성기

의사 난수 생성기(Pseudo-Random Number Generator)는 입력으로 시드 값을 받아 효율적으로 난수를 생성하기 위해 소프트웨어로 구현된 난수 생성기이다. P. Lacharme 등[12]은 리눅스 운영체제에서의 의사 난수 생성기에 대해 세부적으로 서술 및 분석하고 이를 더 안전하게 구현하기 위한 방법을 제시했다. 여기에서는 의사 난수 생성기를 크게 2가지 방식으로 분류했는데 하나는 결정적인 알고리즘을 사용하면서 내부 상태에 대한 정보 없이는 어떤 난수열이 발생할지 예측할 수 없도록 한 것이고 다른 하나는 '임의성'을 나타내는 비트 값인 엔트로피를 시드 값으로 사용한 의사 난수 생성기라고 한다. 리눅스 운영체제 내부에서의 의사 난수 생성기는 엔트로피를 시드 값을 사용하는 종류이며 본 논문에서의 의사 난수 생성기도 OpenSSL에서 구현된 것[11, 12]으로 리눅스 운영체제의 의사 난수 생성기와 같은 분류에 속한다.

2.2 파일 및 폴더 암호화

NIST의 지침[1]에 따르면 파일 및 폴더 암호화는 스토리지에 저장된 파일이나 폴더를 암호화하는 것을 뜻하며 비밀번호를 입력하는 등의 행위를 통해 암호화된 데이터에 접근할 수 있는 권한을 얻어야만 한다. 파일 및 폴더 암호화 제품은 드라이버나 애플리케이션, 혹은 서비스 등의 다양한 형태로 구현이 가능하기 때문에 파일 시스템 암호화 기법 중에서 제일 널리 사용되고 있다. 파일 및 폴더 암호화 제품은 하나의 파일을 한 번에 암/복호화하도록 설계되었다. 그렇기 때문에 파일이나 폴더의 크기에 따라 시스템 성능 저하에 미치는 영향이 다를 수 있다. 하지만, 파일명이나 파일 메타데이타(metadata), 폴더 내부의 구조 등의 기밀성은 유지하기 힘들기 때문에 공격자가 필요로 하는 정보들이 노출될 수 있으므로 이것이 파일 및 폴더 암호 제품의 취약점이 될 수 있다.

2.3 안전성 모델

본 논문에서 제안하는 기법의 안전성을 분석하기 위해 IND-CPA 모델을 사용한다. IND-CPA는 암호문의 구별 불가능성 개념이 합쳐진 선택 평문 공격 모델로서 다음과 같이 정의된다.

IND-CPA[8] 먼저, 공격자는 일련의 평문 쌍들 $(M_1^0, M_1^1), ..., (M_q^0, M_q^1)$을 선택하여 Challenger에게 전송한다. 여기서 각각의 평문 쌍 (M_n^0, M_n^1)끼리는 동일한 길이를 가진다. 그리고 공격자는 이 평문 쌍들에 대한 암호문은 $C_1, ..., C_q$을 전송받는데 이 $C = \{C_1, ..., C_q\}$는 $M^0 = \{M_1^0, ..., M_q^0\}$ 또는 $M^1 = \{M_1^1, ..., M_q^1\}$를 암호화한 결과 $C = E_K(M^b)$가 되고 공격자는 암호문 C가 $b=0$ 또는 $b=1$인지 추측해야 한다. 이 과정은 Fig. 1과 같다.

Bellare 등의 안전성 정의[9]에서는 공격자가 어떤 평문을 암호화했는지 추측할 수 있는 확률 p에 따라 IND-CPA에 대한 공격자의 Advantage $A(p)$는

$$A(p) = 2p - 1. \quad (2\text{-}1)$$

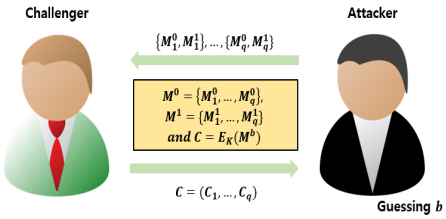

Fig. 1. Challenge process in IND-CPA

과 같고 식 (2-1)에서 $p=\frac{1}{2}$이면 $A(p)=0$이다. 이 경우에는 공격자가 평문과 암호문 사이의 관계를 전혀 알 수 없다. 그리고 $p>\frac{1}{2}$ or $p<\frac{1}{2}$이면 $A(p)>0$ or $A(p)<0$이고 이 경우에는 공격자가 평문과 암호문 사이의 관계를 알 수 있다고 정의된다. 따라서 어떤 암호화 기법이 IND-CPA에 안전하려면 $p=\frac{1}{2}$이 되도록 기법을 설계해야 한다.

2.4 표기법

본 논문에서 기법 설명을 위해 사용할 표기법에 대해 아래 Table 1에서 정의한다. Table 1에서 평문 메시지는 총 3개 단위로 구분된다. 먼저 i는 메시지 번호, 즉, 하나의 파일을 나타낸다. 그리고 j는 i번 메시지에서 하나의 EncFS 단위 블록 1024bytes만큼을 나타낸다. 그리고 k는 하나의 EncFS 단위 블록 내에서 AES 암호화를 수행하는 하나의 128bits 블록을 나타낸다. 예를 들어 설명하면 $M_{1,0,2}$는 1번 파일의 첫 EncFS 단위 블록의 3번째 AES 암호화 블록(257bits~384bits)을 나타낸다. 그리고 M_i^t는 IND-CPA 모델에서 사용하는 기호로 t는 0 또는 1로 한정되며 공격자가 Challenger에게 전송하는 메시지 쌍의 인덱스를 뜻한다. 즉, (M_1^0, M_1^1)은 공격자가 Challenger에게 전송하는 1번 평문 쌍이며 두 평문은 서로 다른 평문이 된다. C_i, $C_{i,j}$, $C_{i,j,k}$도 각각 M_i, $M_{i,j}$, $M_{i,j,k}$과 동일한 인덱스를 나타내며 평문이 아니라 암호문 블록을 뜻한다.

Table 1. Notation in proposed scheme

Notation	Description
M_i, $M_{i,j}$, $M_{i,j,k}$, M_i^t	i-th plaintext message, j-th unit block used in EncFS(1024bytes), k-th unit block used in AES (128bits), A message used in IND-CPA game, where $t \in \{0,1\}$
C_i, $C_{i,j}$, $C_{i,j,k}$	A ciphertext for M_i A ciphertext for $M_{i,j}$ A ciphertext for $M_{i,j,k}$
K	An encryption key of EncFS
p	Guessing probability of Attacker in IND-CPA game
$A(p)$	An advantage value for attacker where probability p is given
BN	Unit block number of EncFS
LBN	Last unit block number of EncFS
SBN	EncFS unit block number having same seed value with the last block
$P(s)$	PRNG(Pseudo-Random Number Generator), where s is seed value
R_n	n-th random number generated by $P(s)$

III. EncFS의 암호화 방식

3.1 EncFS의 암호화 방법 및 단위

EncFS는 Openssl의 EVP API를 이용하도록 구현된 애플리케이션이다. EncFS에서 제공하는 암호화 관련 옵션은 아래 Table 2와 같다. 대칭키 블록 암호화 방식인 AES(Advanced Encryption Standard)[16]와 Blowfish를 암호 알고리즘으로 이용한다. 가장 많이 이용하는 Standard 모드에서는 AES만을, 그리고 사용자가 암호화 옵션을 선택할 수 있는 Expert 모드에서는 AES와 Blowfish 중 하나를 선택하여 사용 가능하다. 키 길이는 128, 192, 256bits를 지원하고 키 생성 함수로는 PBKDF2를 사용한다. 암호화 블록의 크기는 64~4096bytes를 제공하는데 일반적으로 사용하는 Standard 모드에서는 1024bytes로 고정된 블록 길이를 제공한다.

EncFS가 암호화하려는 파일들을 앞서 언급한 고정된 EncFS 블록 크기로 나눈 몫이 블록 암호화를 수행하는 블록의 개수가 된다. 그리고 하나의 블록

Table 2. Encryption option of EncFS

	Standard	Paranoia	Expert.
Cipher	AES	AES	AES / Blowfish
Key length	192bits	256bits	128 / 192 / 256 bits
Key Generation Function	PBKDF2	PBKDF2	PBKDF2
Block Length	1024 bytes	1024 bytes	64~4096 bytes
Salt Length	160bits	160bits	-

크기를 만족 못하는 나머지 블록은 스트림 암호화를 수행한다. 예를 들어 Table 3과 같이 3300bytes 크기의 파일을 1024bytes 블록 크기로 암호화를 수행한다고 가정하면 block #1, #2, #3의 3개의 완전한 블록이 생성되고 228bytes의 완전하지 못한 블록인 block #4가 생긴다.

그리고 파일마다 암호화 과정을 위해 각 블록마다 고유한 block IV를 가질 수 있도록 한다. file IV는 OpenSSL EVP API의 RAND_pseudo_bytes 함수를 이용해 생성된 64bits 길이의 난수이다. EncFS는 file IV를 암호화되는 블록의 블록 번호와 XOR(eXclusive OR) 연산을 수행한 뒤 사용자의 비밀번호를 PBKDF2의 입력으로 이용해 생성한 사용자 키를 가지고 HMAC(Hashed Message Authentication Code)[17]을 거쳐 block IV로 이용한다. 이 과정은 Fig. 2와 같이 진행된다.

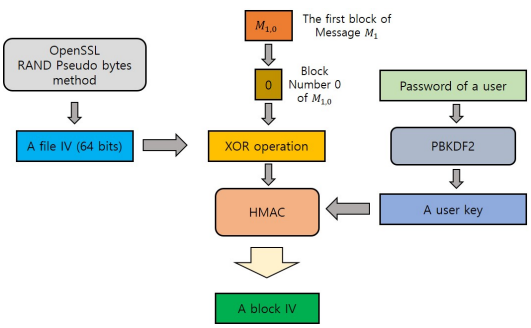

Fig. 2. Generation process of block IV

3.2 EncFS에서의 블록 암호화(CBC)

EncFS에서 하나의 완전한 블록은 AES 또는 Blowfish 알고리즘의 CBC 모드를 이용해 암호화된다. 특별히 수행되는 bit 혹은 byte 단위의 혼합 또는 확산을 위한 과정 없이 평문이 128bits씩 즉, 16bytes씩 암호화된다. 즉, 64개의 암호화 단위 블록이 생긴다. 입력으로 주어지는 평문을 $M_{i,j,k}$라 할 때, 이 평문은 LBN개의 1024bytes 단위 EncFS 블록을 가지고 각각의 EncFS 블록은 64개의 AES 암호화 단위 128bits 블록을 가지게 된다. 이 때, 블록 암호화 과정은 아래 Table 4 및 Fig. 3와 같이 이루어진다.

Table 3. Size of EncFS encryption block

block #1	block #2	block #3	block #4
1024bytes (CBC)	1024bytes (CBC)	1024bytes (CBC)	228bytes (CFB)

Table 4. Block encryption of EncFS

Input: Message block $M_{i,j,k}$
Output : Encrypted block $C_{i,j,k}$
1. for $(j=1; j \le LBN-1; j++)$
1.1. $blockIV = fileIV \oplus j$
1.2. for $(k=1; k \le 64; k++)$
1.2.1. $C_{i,j,k} = E_K(M_{i,j,k} \oplus blockIV)$
1.2.2. $blockIV = C_{i,j,k}$

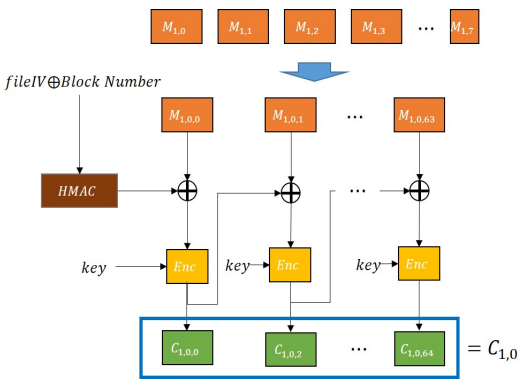

Fig. 3. Block encryption process of EncFS

3.3 EncFS에서의 스트림 암호화(CFB)

나머지 블록은 AES 또는 Blowfish 알고리즘의 CFB 모드를 이용해 암호화되는데 이 과정에서는 ShuffleBytes와 FlipBytes라는 알고리즘도 사용된다. ShuffleBytes는 주어진 데이터 배열에서 이전

인덱스(index) 배열 값과 현재 인덱스의 배열 값을 XOR 연산을 수행하여 그 결과 값을 다시 현재 인덱스 배열 값으로 저장한다. ShuffleBytes의 역 과정으로는 UnShuffleBytes가 있으며 ShuffleBytes의 역순으로 동작한다. 그리고 FlipBytes는 데이터 배열의 인덱스 순서를 역순으로 바꿔준다. 즉, (1 2 3 4 5 6)의 순서가 (6 5 4 3 2 1)로 바뀐다. FlipBytes의 역 과정인 UnFlipBytes는 FlipBytes의 결과로 나온 배열을 입력으로 하여 다시 순서를 원래대로 바꾸어주는 과정이다. 데이터 배열을 $D[i]$라고 한다면 알고리즘은 다음 Table 5, 6, 7과 같고 동작 과정은 Fig. 4, 5, 6와 같다.

Table 5. Algorithm of ShuffleBytes

Input: Data array D, $array\_size$
Output : Data array D'
1. for $(i=0; i<array\_size-1; ++i)$ 1.1. $D'[i]=D[i]\oplus D[i-1]$

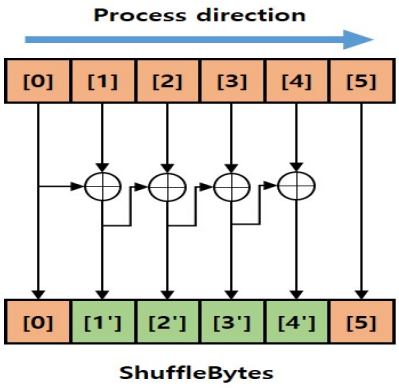

Fig. 4. Process of ShuffleBytes

Table 6. Algorithm of UnShuffleBytes

Input: Data array D', $array\_size$
Output : Data array D
1. for $(i=array\_size-1; i; --i)$ 1.1. if$(i=1)$ 1.1.1. $D[i]=D'[i]\oplus D[i-1]$ 1.1. *else* 1.1.1. $D[i]=D'[i]\oplus D'[i-1]$

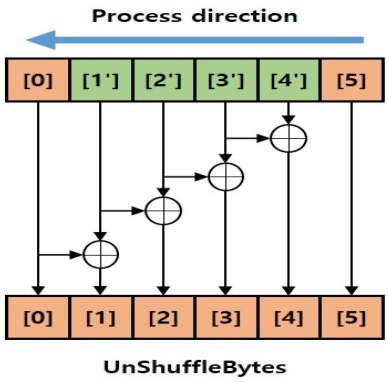

Fig. 5. Process of UnShuffleBytes

Table 7. Algorithm of FlipBytes

Input: Data array D, $array\_size$
Output : Data array D'
1. for $(i=0; i<array\_size/2; ++i)$ 1.1. *reverse* $D[i]$

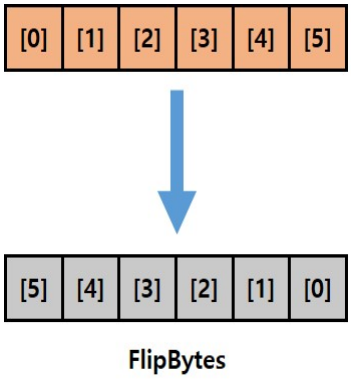

Fig. 6. Process of FlipBytes

위의 알고리즘들을 이용하여 EncFS에서 스트림 암호화를 수행하는 과정은 먼저 원본 데이터에 대해서 ShuffleBytes를 수행한 뒤, CFB 암호화를 한 번 수행한다. 여기서 사용되는 시드 값은 file IV와 마지막 블록 번호를 XOR한 값이다. 이 값을 HMAC에 입력하여 얻은 block IV를 이용해 CFB 연산에 사용한다. 그 결과로 발생한 암호화된 데이터를 FlipBytes를 먼저 수행한 뒤, 다시 ShuffleBytes를 수행한 데이터에 대해서 CFB 암호화를 수행한다. 이 과정에서의 시드 값은 이전에

사용한 시드 값에 1을 더한 값을 사용한다. 따라서 block IV값은 먼저 수행한 CFB 암호화 과정과 다른 값이 된다. 최종적으로 두 번째 CFB 암호화 과정의 결과 값이 EncFS의 스트림 암호화 과정의 최종 결과 값이 된다. 이 알고리즘은 Table 8과 같고 동작 과정은 Fig. 7과 같다.

Table 8. Stream encryption of EncFS

Input: Last message block $M_{i,j}$
Output : Encrypted block $C''_{i,j}$
1. $M'_{i,j} = ShuffleBytes(M_{i,j})$
2. $C_{i,j} = E_K(HMAC(fileIV \oplus LBN)) \oplus M'_{i,j}$
3. $C'_{i,j} = ShuffleBytes(FlipBytes(C_{i,j}))$
4. $C''_{i,j} = E_K(HMAC((fileIV \oplus LBN)+1)) \oplus C'_{i,j}$

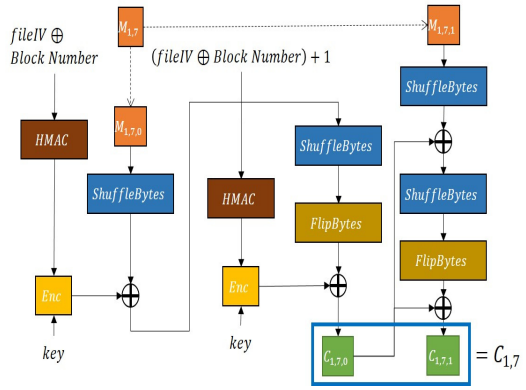

Fig. 7. Stream encryption process of EncFS

IV. EncFS의 취약점 및 안전성 분석

4.1 EncFS의 취약점

앞서 언급한 것처럼 EncFS에서는 마지막 데이터 블록과 나머지 데이터 블록들의 암호화하는 방식이 다르다. 이 과정에서 암호화를 위해 사용되는 IV를 만들기 위한 시드 값은 항상 다르게 사용되어야 한다. 하지만, 블록 암호화 과정과 스트림 암호화 과정에서 같은 시드 값이 사용되는 경우가 많다. 여기에서는 그 중 하나의 예시를 보이도록 한다. 실제 사용되는 시드 값이 64bits이지만, 편의를 위해 8bits 시드 값을 정의하고 이 값을 01010100이라고 가정한다. 그러면 평문 블록이 8개인 경우에서 취약점을 이용한 공격이 가능하다. 이 과정은 아래와 같다.

데이터 블록 $M_{1,0}\|M_{1,1}\|M_{1,2}\|M_{1,3}\|M_{1,4}\|M_{1,5}\|M_{1,6}\|M_{1,7}$이 있다. 각 블록에 들어가는 시드 값은 Table 9와 같이 주어지게 된다.

여기에서 0번 블록과 7번 블록의 2차 스트림 암호화에서 사용되는 시드 값이 동일하다. EncFS에서 암호화되는 폴더 내부의 모든 파일에서는 동일한 키가 사용되기 때문에 위의 두 경우에 대해서 동일한 block IV가 생성된다. 여기서 평문 블록 $M_{1,7}$의 첫 128bits 블록인 $M_{1,7,0}$을 얻을 수 있는데 그 과정은 다음과 같다.

먼저, 주어진 평문 M_1을 암호화한다. 그러면 0번 블록에서 첫 128bits 암호화 블록인 $M_{1,0,0}$의 모든 비트가 0이고, $M_{1,0,0}$을 암호화한 $C_{1,0,0}$은

$$\begin{aligned} C_{1,0,0} &= E_K(M_{1,0,0} \oplus HMAC(fileIV)) \\ &= E_k(0 \oplus HMAC(fileIV)) \\ &= E_k(HMAC(fileIV)) \end{aligned} \quad (4\text{-}1)$$

과 같이 계산할 수 있고 $M_{1,7,0}$을 암호화한 $C_{1,7,0}''$은

$$M_{1,7,0}' = ShuffleBytes(M_{1,7,0}), \quad (4\text{-}2)$$

$$C_{1,7,0} = E_k(HMAC(fileIV \oplus 7)) \oplus M_{1,7,0}', \quad (4\text{-}3)$$

$$C_{1,7,0}' = ShuffleBytes(FlipBytes(C_{1,7,0})), \quad (4\text{-}4)$$

$$C_{1,7,0}'' = E_k(HMAC((fileIV \oplus 7)+1) \oplus C_{1,7,0}', \quad (4\text{-}5)$$

$$C_{1,0,0} \oplus C_{1,7,0}'' = C_{1,7,0}', \quad (4\text{-}6)$$

$$C_{1,7,0} = UnFlipBytes(UnShuffleBytes(C_{1,7,0}')) \quad (4\text{-}7)$$

과 같이 계산할 수 있다. 여기서 0번 블록의 시드

Table 9. Example of Same Seed Value table

Block Number	Encryption Mode	Seed Value
0	**Block**	**01010100**
1	Block	01010101
2	Block	01010110
3	Block	01010111
4	Block	01010000
5	Block	01010001
6	Block	01010010
7	Stream 1st	01010011
7	**Stream 2nd**	**01010100**

값과 7번 블록의 시드 값이 서로 일치하게 되므로 $HMAC(fileIV) = HMAC((fileIV \oplus 7)+1)$이 성립한다. 따라서 식 (4-6)과 같은 결과를 얻을 수 있다. 그리고 ShuffleBytes의 역 연산과정인 UnShuffleBytes는 데이터 블록들 사이의 단순 XOR 연산으로 이루어지고 FilpBytes 역시 단순히 데이터 블록의 순서를 역순으로 바꾸는 과정이기 때문에 역 과정인 UnFilpBytes 역시 공격자가 데이터 블록의 순서를 역으로 바꿔줌으로써 수행된다. 따라서 이 두 과정은 적은 연산량으로 공격자에 의해 수행될 수 있다. 따라서 식 (4-7)과 같이 $C_{1,7,0}{}'$로부터 $C_{1,7,0}$을 공격자가 얻을 수 있다.

이제 M_1을 수정하여 $M_{2,7,0}=0$으로 하고 나머지 데이터를 M_1과 관련 없는 임의의 데이터로 만든 평문 $M_2 = M_{2,0}\|M_{2,1}\|M_{2,2}\|M_{2,3}\|M_{2,4}\|M_{2,5}\|M_{2,6}\|M_{2,7}\|M_{2,8}$에 대해서 한 번 더 암호화를 수행한다. 즉, 0~7번 블록은 블록 암호화가 수행되고 마지막 8번 블록만 스트림 암호화가 수행된다. 그러면 7번 블록이 메시지 수정 이후에 블록 암호화가 되므로 다음과 같은 식이 성립한다.

$$\begin{aligned} C_{2,7,0} &= E_K(M_{2,7,0} \oplus HMAC(fileIV \oplus 7)) \\ &= E_K(0 \oplus HMAC(fileIV \oplus 7)) \\ &= E_K(HMAC(fileIV \oplus 7)) \,. \end{aligned} \tag{4-8}$$

여기서 얻은 $C_{2,7,0}$을 $C_{1,7,0}{}''$로부터 얻은 $C_{1,7,0} = E_K(HMAC(fileIV \oplus 7)) \oplus M_{1,7,0}{}'$에 XOR 연산을 수행하면 다음과 같은 식을 얻을 수 있다.

$$\begin{aligned} C_{2,7,0} \oplus C_{1,7,0} &= E_K(HMAC(fileIV \oplus 7)) \oplus \\ &\quad E_K(HMAC(fileIV \oplus 7)) \oplus M_{1,7,0}{}' \\ &= M_{1,7,0}{}' \,, \end{aligned} \tag{4-9}$$

$$M_{1,7,0} = UnShuffleBytes(M_{1,7,0}{}') \,. \tag{4-10}$$

즉, 공격자가 선택한 평문 암호문 쌍을 통해 둘 사이의 관계에 대한 정보를 얻을 수 있다는 취약점이 존재함을 알 수 있으며 위의 예에서는 $SBN=0$, $LBN=7$이라고 할 수 있다. 이제 앞서 언급한 취약점으로 인해 EncFS가 IND-CPA에 안전하지 않음을 다음 절에서 구체적으로 설명한다.

4.2 EncFS의 안전성 분석

이 절에서는 EncFS에서 IND-CPA를 이용한 공격 시나리오를 일반화한다. 공격자는 Challenger에게 $(M_1^0, M_1^1), (M_2^0, M_2^1)$를 전송한다. 4.1절에서 정의했던 평문 M_1과 M_2를 이용해 $M_1^0 = M_1$, $M_2^0 = M_2$로 하고 M_1^1과 M_2^1는 각각 M_1, M_2와 동일한 길이의 평문으로 전송해야 한다. Bellare 등의 안전성 정의[10]에서 M_1^0과 M_2^0의 길이가 동일해야 한다는 조건이 없기 때문에 이러한 전송이 가능하다. 그러면 Challenger는 $M^0 = \{M_1^0, M_2^0\}$와 $M^1 = \{M_1^1, M_2^1\}$ 중에서 하나를 선택하여 암호문 $C = (C_1, C_2)$로 전송한다. 여기에서 $C = E_K(M^b)$이므로 $b=0$일 때 4.1절의 취약점을 이용한 평문 정보 $M_{1,LBN,0}$을 도출하는 과정을 진행한다. 이 과정은 다음과 같다.

$$M_{1,SBN,0} = 0 \,, \tag{4-11}$$

$$\begin{aligned} C_{1,SBN,0} &= E_K(0 \oplus HMAC(fileIV \oplus SBN)) \\ &= E_K(HMAC(fileIV \oplus SBN)) \\ &= E_K(HMAC((fileIV \oplus LBN)+1)) \,, \end{aligned} \tag{4-12}$$

$$C_{1,LBN,0}{}'' = E_K(HMAC((fileIV \oplus LBN)+1)) \oplus C_{1,LBN,0}{}' \,, \tag{4-13}$$

$$C_{1,SBN,0} \oplus C_{1,LBN,0}{}'' = C_{1,LBN,0}{}' \,, \tag{4-14}$$

$$C_{1,LBN,0} = UnFlipBytes(UnShuffleBytes(C_{1,LBN,0}{}')) \,. \tag{4-15}$$

이제 $M_{2,LBN,0}=0$임을 이용해 $C_{2,LBN,0} = E_K(HMAC(fileIV \oplus LBN))$을 얻을 수 있고 식 (4-15)에서 얻은 $C_{1,LBN,0}$과 XOR 연산을 수행하면

$$\begin{aligned} C_{2,LBN,0} \oplus C_{1,LBN,0} &= E_K(HMAC(fileIV \oplus LBN)) \\ &\quad \oplus E_K(HMAC(fileIV \oplus LBN)) \\ &\quad \oplus M_{1,LBN,0}{}' \\ &= M_{1,LBN,0}{}' \,, \end{aligned} \tag{4-16}$$

$$M_{1,LBN,0} = UnShuffleBytes(M_{1,LBN,0}{}') \tag{4-17}$$

과 같이 온전한 평문 블록 $M_{1,LBN,0}$을 얻을 수 있게 된다.

하지만, M_2가 아닌 다른 평문을 암호화한 암호문의 경우, $M_{2,LBN,0} \neq 0$이므로 $C_{2,LBN,0} \neq E_k($ $HMAC(fileIV \oplus LBN))$이라 $C_{1,LBN,0}$과 XOR 연산을 하면 식 (4-16), (4-17)을 통해 $M_{1,LBN,0}$을 얻을 수 없다. 따라서 M^0와 M^1이 구분 가능해지기 때문에 암호문을 추측할 수 있는 확률 p가 $p > \frac{1}{2}$ or $p < \frac{1}{2}$를 만족하게 된다. 따라서 $A(p) > 0$ or $A(p) < 0$이 되고 EncFS의 암호화 기법은 IND-CPA에 안전하지 않음을 알 수 있다.

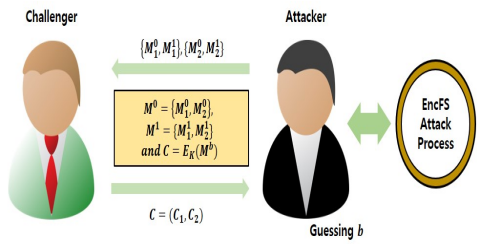

Fig. 8. Attack Scenario on EncFS

V. 제안하는 기법

4.1절에서 언급한 것처럼 EncFS의 암호화 과정은 동일한 시드 값의 생성으로 인해 취약점이 존재했다. 그리고 안전성 분석 시 IND-CPA를 만족하지 못함을 알 수 있다. 따라서 이를 개선하기 위한 방법을 제시한다.

해당 기법에서는 block IV를 유일한 값으로 만들어주는 것을 목적으로 하여 기존 기법에서 HMAC의 입력 값이었던 $fileIV \oplus BN$을 의사 난수 생성기 [11, 12]를 통해 생성된 값으로 바꿔주고 그 결과로 block IV가 유일한 값이 생성되도록 한다.

5.1 제안하는 기법의 특징 및 구성

5.1.1 난수를 이용한 다른 IV값 생성

의사 난수 생성기[11,12]를 이용해 각 블록마다 필요한 난수들을 생성한다. 먼저 file IV를 각 블록 번호와 XOR하여 의사 난수 생성기의 시드 값으로 한다. 그러면 여기서 생성된 각각의 난수들은 다른 값을 가질 것이다. 이를 식으로 나타내면 아래와 같다.

$$R_n = P(fileIV \oplus n), \quad (5\text{-}1)$$

$$R = \{R_1, R_2, \ldots, R_{LBN}, R_{LBN+1}\}. \quad (5\text{-}2)$$

식 (5-1)의 난수는 (5-2)에서처럼 블록 개수 (LBN)보다 하나 많은 총 $LBN+1$개를 생성한다. $LBN+1$번째 난수의 경우 LBN에 1을 더한 값을 블록 번호로 하여 생성하고 나머지 블록의 두 번째 CFB 암호화 과정에서 사용한다. 이 난수들을 Table 9에서의 예시를 이용해 나타내면 아래 Table 10과 같다.

위와 같이 $R_1 = 10110100$, $R_2 = 01100010$, …, $R_8 = 10001110$, $R_9 = 11101000$이 되기 때문에 기존에 같았던 0번 블록과 7번 블록의 시드 값도 달라지고 결과적으로 다른 난수 값이 생성되어 block IV도 다른 값을 가지게 된다.

Table 10. Example of Random Value table

Block Number	Encryption Mode	Seed Value	Random Value
0	**Block**	**01010100**	**10110100**
1	Block	01010101	01100010
2	Block	01010110	11010000
3	Block	01010111	00101110
4	Block	01010000	01110011
5	Block	01010001	10010101
6	Block	01010010	00011100
7	Stream 1st	01010011	10001110
7	**Stream 2nd**	**01011100**	**11101000**

5.1.2 블록 암호화

기존의 블록 암호화 과정의 CBC모드에서 사용되는 block IV는 $HMAC(fileIV \oplus BN)$이다. 따라서 시드 값인 $fileIV \oplus BN$을 $R_n \subset \{R_1, R_2, \ldots, R_{LBN-1}\}$으로 바꿔준다. 따라서 암호화 과정은 다음

Table 11. Proposed block encryption scheme

Input: Message block $M_{i,j,k}$
Output : Encrypted block $C_{i,j,k}$
1. for $(j = 1; j \leq LBN-1; j{+}{+})$
1.1. $blockIV = R_j$
1.2. for $(k = 1; k \leq 64; k{+}{+})$
1.2.1. $C_{i,j,k} = E_K(M_{i,j,k} \oplus blockIV)$
1.2.2. $blockIV = C_{i,j,k}$

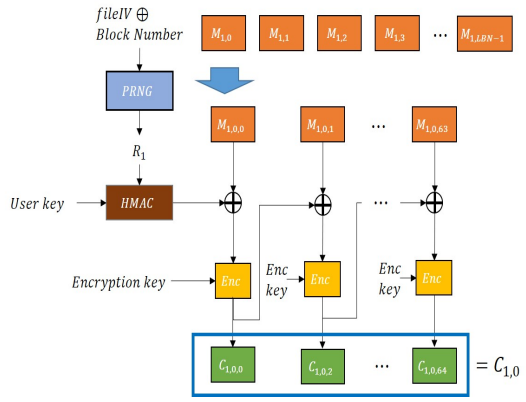

Fig. 9. Process of scheme in Table 11

Table 11 및 Fig. 9와 같이 진행된다.

5.1.3 스트림 암호화

스트림 암호화도 블록 암호화 과정과 동일하게 변경된다. 이 경우의 암호화 과정은 아래 Table 12에서 설명하고 Fig. 10과 같이 표현할 수 있다.

Table 12 Proposed Stream Encryption Scheme

Input: Last message block $M_{i,j}$
Output : Encrypted block $C''_{i,j}$
1. $M' = ShuffleBytes(M_{i,j})$ 2. $C_{i,j} = E_K(HMAC(R_{LBN})) \oplus M'_{i,j}$ 3. $C'_{i,j} = ShuffleBytes(FlipBytes(C_{i,j}))$ 4. $C''_{i,j} = E_K(HMAC(R_{LBN+1})) \oplus C'_{i,j}$

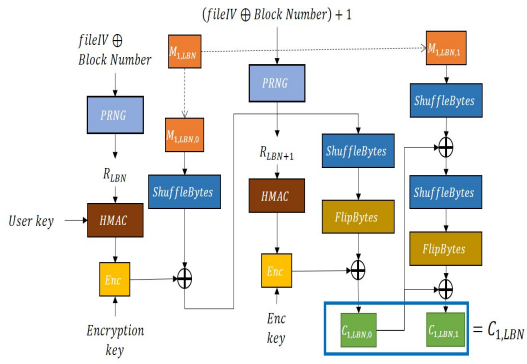

Fig. 10. Process of scheme in Table 12

5.2 제안된 기법의 안전성 분석

본 논문에서는 제안한 기법의 안전성을 IND-CPA 모델을 이용해 분석한다. 공격자는 Challenger에게 $(M_1^0, M_1^1), (M_2^0, M_2^1)$를 전송한다. 그리고 Challenger는 $M^0 = \{M_1^0, M_2^0\}$와 $M^1 = \{M_1^1, M_2^1\}$ 중 하나를 선택해 $C = E_K(M^b)$과 같이 암호화하고 그 암호문 $C = (C_1, C_2)$를 공격자에게 전송한다. 그리고 공격자는 C_1, C_2에 대해서 취약점을 이용해 구분할 수 있는지 확인한다.

공격자가 전송받은 암호문을 가지고 4.1절에서의 취약점을 이용하여 평문 정보 $M_{1,LBN,0}$을 도출해낼 때 제안하는 기법은 아래 수식과 같이 진행된다.

$$R_{SBN} \neq R_{LBN+1}, \tag{5-3}$$

$$HMAC(R_{SBN}) \neq HMAC(R_{LBN+1}), \tag{5-4}$$

$$C_{1,SBN,0} \oplus C_{1,LBN,0}'' = E_K(HMAC(R_{SBN})) \oplus E_K(HMAC(R_{LBN-1})) \oplus C_{1,LBN,0}', \tag{5-5}$$

$$C_{1,SBN,0} \oplus C_{1,LBN,0}'' \neq C_{1,LBN,0}. \tag{5-6}$$

Table 11의 예시와 같이 모든 블록의 R_n이 다른 값을 가지기 때문에 . 따라서 $C_{1,LBN,0}$의 값을 정확히 알 수 없어 식 (4-16), (4-17)을 통한 평문 정보 $M_{1,LBN,0}$을 얻어낼 수 없다. 따라서 공격자가 암호문을 추측할 수 있는 확률 p가 $p = \frac{1}{2}$을 만족하기 때문에 제안한 기법이 IND-CPA에 안전함을 알 수 있다.

VI. 기존 기법과의 성능 비교 분석

이번 장에서는 EncFS에서 사용하는 기법과 제안하는 기법의 차이점을 실험을 통해 분석했다. 두 기법에서 가장 큰 차이점은 block IV를 생성하는 과정이다. 따라서 두 과정의 실행 시간을 비교하였다. 실험은 Intel Core i7-4790 3.60GHz x64 기반 프로세서, 8GB DDR3 메모리 환경에서 각각 256MB, 512MB, 1GB, 2GB, 4GB 데이터를 1KB 블록 단위로 암호화할 때 생성되는 블록들의 개수만큼 block IV를 생성하는 방식으로 진행했다. 결과는 다음 Table 13, Fig. 11과 같다.

Table 13. Evaluation of Schemes

	256MB (sec)	512MB (sec)	1GB (sec)	2GB (sec)	4GB (sec)
EncFS	0.41	0.81	1.63	3.17	6.16
Proposed	0.42	0.83	1.66	3.27	6.46

Fig. 11. Evaluation Graph of schemes

위의 표와 그림을 분석하면 각각의 데이터 크기에 따라 block IV가 생성되는 시간의 추세는 두 기법이 큰 차이가 없이 진행된다. 기존 기법과 제안하는 기법의 시간 차이는 각각 0.01초, 0.02초, 0.03초, 0.1초, 0.3초 순으로 점차 증가하고 있어 더 큰 데이터를 암호화하는 경우에 둘의 차이가 커질 것으로 예상된다. 하지만, 그 차이는 1초 이내로 큰 영향을 미치지 않음을 알 수 있다. 따라서 제안하는 기법을 사용해도 기존 기법과 비슷한 수행 시간에 더 높은 안전성을 기대할 수 있다.

VII. 결 론

본 논문에서는 스토리지 보안을 위한 파일 및 폴더 암호화 프로그램인 EncFS의 특징과 T. Hornby가 제시한 취약점을 분석하고 현존하는 취약점을 이용해 IND-CPA에 안전하지 않음을 확인했다. 비록 이 취약점으로 동일한 IV가 생성될 가능성이 낮을지라도 암호화 기법은 모든 경우에 대해서 안전해야 한다. 그리고 이 취약점을 개선하기 위해 의사 난수 생성기를 이용하여 각 블록마다 다른 block IV를 생성하기 위한 값을 부여함으로써 IND-CPA에 안전한 기법을 설계했고 이 기법이 기존 공격에 안전함을 분석했다. 향후에는 동일한 시드 값이 생성되어도 block IV가 동일해지지 않는 방법에 대해 연구할 것이다.

References

[1] K. Scarfone, M. Souppaya, M. Sexton, "Guide to Storage Encryption Technologies for End User Devices," NIST Special Publication 800-111, pp. '3-1' - '3-9', NIST, Nov. 2007.

[2] Microsoft, "BitLocker: About BitLocker," Available: https://technet.microsoft.com/en-us/library/cc732774(v=ws.11).aspx.

[3] Apple, "OS X: About FileVault 2," Available: https://support.apple.com/ko-kr/HT204837.

[4] V. Gough, "EncFS Encrypted Filesystem," Available: https://github.com/vgough/encfs/blob/master/README.md.

[5] AxCrypt DB "AxCrypt," Available: http://www.axcrypt.net/wp-content/uploads/dlm_uploads/2016/06/AxCryptVersion2AlgorithmsandFileFormat.pdf.

[6] T. Hicks and D. Kirkland, "eCryptfs," Available: http://ecryptfs.org/about.html.

[7] M. A. Halcrow, "eCryptfs: An Enterprise-class Encrypted Filesystem for Linux," In Proceedings of the 2005 Linux Symposium, vol. 1, pp 201-218, Jul. 2005.

[8] M. Szeredi, "File System in User Space," Available: https://github.com/libfuse/libfuse/blob/master/README.md.

[9] M. Dworkin, "Recommendation for Block Cipher Modes of Operation. Methods and Techniques," NIST Special Publication 800-38A, pp. 10-13, NIST, Dec. 2001.

[10] M. Bellare and P. Rogaway, "Introduction to modern cryptography," In UCSD CSE 207 Course Notes, Available : http://digidownload.libero.it/persiahp/crittografia/2005_Introduction_to_Modern_Cryptography.pdf, pp. 102-109, May. 2005.

[11] OpenSSL, "Manual page of RAND add()", Available: https://wiki.openssl.org/index.php/Random_Numbers#Generators.

〔12〕 P. Lacharme, A. Rock, and V. Strubel, "The linux pseudorandom number generator revisited," IACR ePrint Archive, 2012-251, May. 2012.
〔13〕 T. Hornby, "EncFS Security Audit," Available: https://defuse.ca/audits/encfs.htm, Feb, 2014.
〔14〕 T. Ptacek and E. Ptacek, "You Don't Want XTS," Available: https://sockpuppet.org/blog/2014/04/30/you-dont-want-xts/, Apr. 2014.
〔15〕 V. Gough, "EncFS Issues," Available: https://github.com/vgough/encfs/issues/10, Aug. 2014.
〔16〕 J. Daeman and V. Rijmen, The Design of Rijndael. AES - the advanced encryption standard, 1st Ed, Springer-Verlag, pp. 31-50, 2002.
〔17〕 H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," IETF RFC 2104, Feb. 1997.

〈저자소개〉

정 원 석 (Won-Seok Jeong) 학생회원
2015년 2월: 한양대학교 컴퓨터공학과 졸업
2015년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 프라이버시향상기술(PET), 데이터베이스 보안

정 재 열 (Jae Yeol Jeong) 학생회원
2010년 8월: 고려대학교 정보수학과 졸업
2013년 8월: 고려대학교 정보보호대학원 석사 졸업
2013년 9월~현재: 고려대학교 정보보호대학원 박사과정
〈관심분야〉 프라이버시향상기술(PET), 데이터베이스 보안, 생체인식

정 익 래 (Ik Rae Jeong) 종신회원
1998년 2월: 고려대학교 전산학과 학사 졸업
2000년 2월: 고려대학교 전산학과 석사 졸업
2004년 8월: 고려대학교 정보보호대학원 박사 졸업
2006년 6월~2008년 2월: 한국전자통신연구원 암호기술연구팀 선임연구원
2008년 3월~2011년 8월: 고려대학교 정보경영공학전문대학원 조교수
2011년 9월~현재: 고려대학교 정보보호대학원 부교수
〈관심분야〉 프라이버시향상기술(PET), 데이터베이스 보안, 암호 이론