내장형 시스템에서 암호화 파일 시스템을 위한 효율적인 암복호화 기법

(An Efficient Encryption/Decryption Approach to Improve the Performance of Cryptographic File System in Embedded System)

허 준 영 † 박 재 민 †† 조 유 근 †††
(Junyoung Heo) (Jaemin Park) (Yookun Cho)

요 약 내장형 시스템은 민감한 데이타를 다루고 저장하기 때문에 정보를 암호화하여 보호하는 암호화 파일 시스템이 필요하지만, 암호화 파일 시스템 적용은 성능 저하가 크기 때문에 내장형 시스템에는 널리 적용되지 못하였다. 기존의 암호화 파일 시스템은 시스템 구조상 불필요한 성능저하를 가져온다. 본 논문에서 제안하는 ISEA(Indexed and Separated Encryption Approach)는 이러한 불필요한 성능 저하를 제거하고, 시스템이 효율적으로 암복호화를 지원하는 새로운 암복호화 기법이다. ISEA는 암호화와 복호화를 페이지 캐쉬를 기준으로 서로 다른 계층에서 수행한다. 즉, 암호화는 페이지 캐쉬 하위 계층에서 수행하고 복호화는 페이지 캐쉬 상위 계층에서 수행한다. 복호화한 내용은 페이지 캐쉬에 저장하여 후속 I/O 요구에 사용할 수 있게 한다. 또한, ISEA는 페이지를 암호화 블록 단위로 나누어 관리하는 페이지 인덱싱 기법을 제공한다. 페이지 인덱싱은 필요에 따라 페이지를 부분적으로 복호화하여 불필요한 복호화 연산을 제거한다. 페이지 캐쉬 탐색 성공률과 읽기/쓰기 사이즈를 종합한 성능 평가에서 ISEA는 효율적인 성능 개선을 보여준다.

키워드 : 시스템 성능, 페이지 캐쉬, 암호화 파일 시스템, 내장형 시스템, 운영 체제

Abstract Since modern embedded systems need to access, manipulate or store sensitive information, it requires being equipped with cryptographic file systems. However, cryptographic file systems result in poor performance so that they have not been widely adapted to embedded systems. Most cryptographic file systems degrade the performance unnecessarily because of system architecture. This paper proposes ISEA (Indexed and Separated Encryption Approach) that supports for encryption/decryption in system architecture and removes redundant performance loss. ISEA carries out encryption and decryption at different layers according to page cache layer. Encryption is carried out at lower layer than page cache layer while decryption at upper layer. ISEA stores the decrypted data in page cache so that it can be reused in followed I/O request without decryption. ISEA provides page-indexing which divides page cache into cipher blocks and manages it by a block. It decrypts pages partially so that it can eliminate unnecessary decryption. In synthesized experiment of read/write with various cache hit rates, it gives results suggesting that ISEA has improved the performance of encryption file system efficiently.

Key words : System Performance, Page Cache, Encryption File System, Embedded System, Operating System

· 본 연구는 2005년도 협동연구과제와 2단계 BK21 사업의 지원비로 수행하였습니다.

† 학생회원 : 서울대학교 컴퓨터공학부
jyheo@os.snu.ac.kr
†† 비 회 원 : 삼성전자 소프트웨어연구소 연구원
jm.j.park@samsung.com
††† 종신회원 : 서울대학교 컴퓨터공학부 교수
ykcho@snu.ac.kr

논문접수 : 2007년 2월 26일
심사완료 : 2007년 11월 26일

정보과학회논문지 : 시스템 및 이론 제35권 제2호(2008.2)

1. 서 론

산업은 지식 집약적으로 발전하고, 데이타의 가치는 지속적으로 증대되기 때문에, 데이타 보안의 필요성은 더욱 커지고 있다. 내장형 시스템은 물리적으로 기기를 습득할 수 있는 취약성을 가지기 때문에, 다양한 공격에 노출될 수 있는 추가적인 위험을 내재하고 있다. 게다가, 내장형 시스템은 제한적인 자원을 가지기 때문에 데이타를 안전하게 보호하기 위해서 추가적인 사항을 고려해야 한다[1].

데이타 보안의 일반적인 방법은 데이타를 암호화하여 보호하는 것이다. 효율적이고 체계적인 암호화를 적용하기 위해서 다양한 암호화 파일 시스템이 제안되었다[2-9]. 하지만, 암호화 파일 시스템의 적용은 내장형 시스템의 성능을 크게 저하시키므로, 암호화 파일시스템은 내장형 시스템에서 널리 활용되지 못하였다. 암호화 파일 시스템이 내장형 시스템에 적용되기 위해서 해결해야 할 우선적인 과제는 암복호화에 따른 성능저하이다. 본 논문에서는 암복호화에 따른 성능 저하를 최소화하기 위해서 새로운 페이지 캐쉬 암복호화 기법(ISEA-Indexed and Separated Encryption Approach)을 제안한다.

1.1 연구동기

그림 1은 사용자 또는 시스템 계층에서 암복호화를 제공하는 암호화 파일 시스템 계층 구조를 보여준다. 이러한 시스템은 페이지 캐쉬 상위 계층에서 암복호화 연산을 수행한다. 상위 계층에서 암복호화를 수행하기 때문에, 데이타를 암호화하여 페이지 캐쉬에 저장한다. 이러한 시스템은 복호화된 데이타 캐쉬를 사용하지 못하는 단점을 가진다[10]. 즉, 동일한 위치의 데이타를 반복적으로 읽고 쓴다면, 반복적으로 암복호화 연산을 수행해야 한다.

그림 2는 디스크 드라이버 또는 하드웨어 계층에서

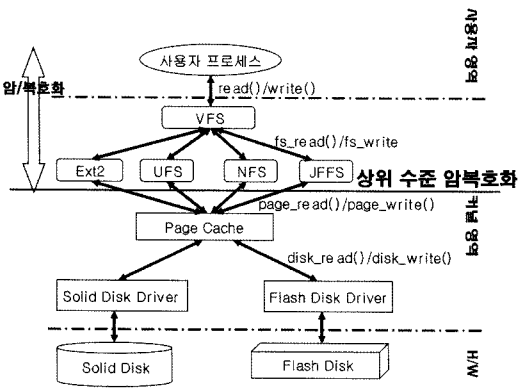


그림 1 상위 계층 암호화 파일 시스템

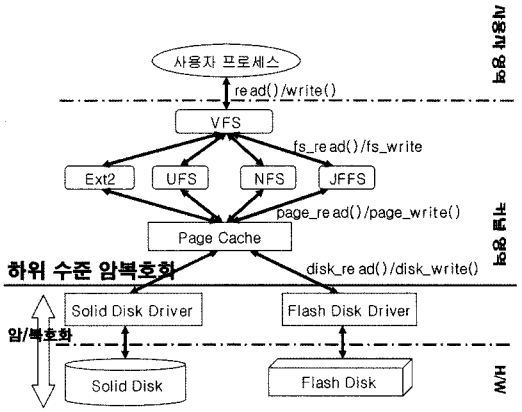


그림 2 하위 계층 암호화 파일 시스템

암복호화를 제공하는 암호화 파일 시스템 계층 구조를 보여준다. 이러한 시스템은 데이타를 디스크에 쓰거나 읽을 때 암복호화 연산을 수행한다. 일반적으로 메모리 I/O 단위가 디스크 I/O 단위와 일치하지 않기 때문에 메모리 I/O를 수행하기 위해 많은 수의 디스크 I/O를 수행해야 한다.[1] 또한, 대부분의 시스템은 성능을 향상시키기 위해 미리 읽기(Read Ahead-디스크 I/O의 횟수를 줄이기 위해서 미리 예측하여 연속된 페이지를 읽는 기능)를 적용하여 디스크 I/O를 수행한다[11]. 따라서 디스크 I/O를 통해 전송된 모든 데이타가 사용자에게 쓰이는 것은 아니기 때문에 디스크에서 페이지 캐쉬로 전송되는 모든 데이타를 복호화하는 것은 앞선 복호화로 인하여 불필요한 성능 저하를 가져올 수 있다.

1.2 ISEA 시스템

그림 3은 ISEA 시스템의 흐름을 보여준다. ISEA는 복호화된 페이지 캐쉬를 사용하기 위해 암복호화 적용

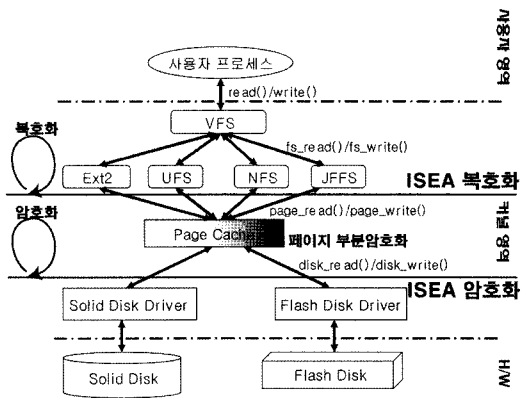


그림 3 ISEA 암호화 파일 시스템

1) 일반적인 리눅스 시스템에서 메모리 I/O 단위는 4,096바이트로 디스크 I/O단위인 512바이트에 비해서 8배 크다.

계층을 분리하였다. 또한 필요에 따른 부분적인 암복호화를 적용하기 위해서 페이지 인덱싱 기법을 적용하였다.

ISEA는 읽기과정에서 페이지에서 읽고 쓰이는 블록만 부분적으로 복호화하여 불필요한 복호화 연산을 최소화하였다. 또한 쓰기 과정에서 최종적으로 변경된 블록을 디스크에 쓸 때 암호화하기 때문에 불필요한 암호화를 제거할 수 있다.

이후 본 논문의 구성은 다음과 같다. 2절에서 기존의 연구방법들을 소개하고, 3절에서 ISEA에 대해서 자세하게 설명하고, 4절에서 ISEA의 설계 및 구현 과정을 설명하고, 5절에서 ISEA의 성능 평가를 보여준다. 마지막으로 6절에서 ISEA의 특징과 함께 논문의 결론을 맺는다.

2. 관련 연구

대부분의 암호화 파일 시스템은 암호화와 사용자 인증을 통하여 데이타를 보호한다. 하지만 암호화 적용 단위, 암호화 계층, 인증 방법, 사용자 투명성 제공 등 적용 방법에는 차이가 있다. 이러한 접근 방법은 크게 저장 장치 암호화, 장치 드라이버 암호화, 파일 시스템 암호화, 사용자 계층 암호화의 네 가지 방식으로 나눌 수 있다.

2.1 저장 장치 암호화

저장장치 암호화는 디스크에 저장된 데이타를 보호하기 위해서 디스크 자체적으로 암호화하는 기법이다[12,13].

일반적으로 OS가 동작해야 데이타 보호 기술이 동작하지만, 이 기법은 OS가 실행되지 않아도 하드웨어가 직접적으로 데이타를 암호화하거나 복호화할 수 있다. 일반적으로 하드웨어가 직접적으로 암호화 연산을 수행하기 때문에 좋은 성능을 보인다. 반면, 디스크 또는 파티션 단위로 암호화하기 때문에 암호화 적용 단위(granularity)가 크고 하드웨어의 도움이 필요하다는 단점을 가진다.

2.2 장치 드라이버 암호화

장치 드라이버 암호화는 파일 시스템 하위 계층에서 장치 드라이버가 직접적으로 암호화하는 기법이다[7,8,14,15]. 장치 드라이버는 상위의 파일 시스템에 대한 지식이 없기 때문에 스왑 파티션이나 디바이스를 직접적으로 암호화할 수 있다. 반면 파일 시스템 구조를 사용하지 못하기 때문에 선택적 암호화가 불가능하다는 단점을 가진다.

2.3 파일 시스템 암호화

파일 시스템 암호화는 파일 시스템이 암호화 기능과 키 관리를 제공하는 기법이다[2-6].

암호화 파일 시스템은 VFS 하위 계층에 독립적인 파일 시스템으로 존재하거나 다른 파일시스템의 상위 계층에 존재하면서 스택처럼 작용할 수도 있다. 대부분의 암호화 파일 시스템은 파일 또는 디렉터리 단위로 개별적인 암복호화가 가능하다.

2.4 사용자 계층 암호화

crypt[7], Gnu PG[17]와 같은 라이브러리를 사용해서 사용자 응용에서 직접적으로 데이타를 암호화할 수 있다. 이러한 기법을 적용하면, 데이타에 접근할 때 수동으로 파일을 암호화하고 복호화하는 과정을 명시해야 한다. 다양한 사용자 응용이 각각 다른 인터페이스와 키 관리, 암호화 알고리즘을 적용하기 때문에 다양한 안전성을 보장해줄 수 있다. 하지만, 이 기법은 많은 사용자 간섭이 필요하고 인증과 키관리에 대한 어려움 때문에 여러 응용간에 일관성을 유지하기 힘들다. 또한 사용자 공간의 암복호화 작업과 반복적인 커널과의 상호교류에 따른 성능저하를 가져온다.

3. ISEA(Indexed and Separated Encryption Approach)

ISEA는 암호화 파일 시스템에 적용 가능한 새로운 암복호화 적용방법이다. ISEA는 성능 향상을 위해 시스템이 직접적으로 페이지 캐쉬 계층에서 암복호화를 지원한다. ISEA의 주된 구성은 다음과 같다.

첫째, 페이지 캐쉬에 복호화된 데이타를 저장하기 위해 암호화/복호화 적용 계층을 분리하였다.

둘째, 중복되고 불필요한 암복호화를 제거하기 위해 페이지 인덱싱 기법을 적용하였다. 페이지 인덱싱이란, 필요에 따라서 페이지를 부분적으로 암복호화하는 기법이다.

3.1 암복호화 계층 분리

ISEA는 암호화 연산을 적용하는 계층과 복호화 연산을 적용하는 계층을 분리하였다. ISEA에서는 페이지 캐쉬 상위계층에서 복호화 연산을 수행하고 하위 계층에서 암호화 연산을 수행한다. 상위 계층에서 복호화 수행 시 그 결과는 페이지 캐쉬에 저장된다. 또한 디스크에 데이타를 저장할 때 하위 계층에서 암호화를 수행하기 때문에 반복되고 불필요한 암호화 연산을 제거할 수 있다. 이러한 계층 분리를 통해서 ISEA는 페이지 캐쉬에 복호화된 데이타를 저장할 수 있다. 이 경우, 페이지 캐쉬는 디스크 I/O 버퍼 캐쉬 뿐만 아니라 암복호화 연산에 대한 캐쉬 역할도 수행할 수 있다. 반면, 상위 계층 뿐만 아니라 하위 계층에서 복호화 연산을 수행해도 페이지 캐쉬에 복호화된 데이타가 존재한다. 하지만, 페이지 캐쉬의 모든 데이타를 사용자가 실제로 사용한다는 보장을 할 수 없다. 반면, 하위 계층에서 복호화한다면 복호화하는 데이타가 실제로 사용되는 데이타인지, 사용될 것이라 예측된 데이타인지, I/O단위를 맞추기 위한

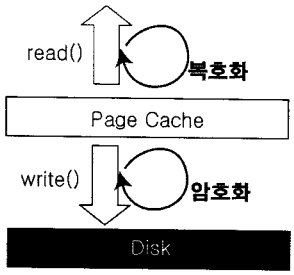


그림 4 암복호화 계층 분리

부수적인 데이타인지 구분할 수 없다. 따라서 ISEA는 실제로 사용되는 데이타만 복호화하기 위해 상위 계층에서 복호화한다. 그림 4는 ISEA에서 이러한 계층 분리를 보여준다.

3.2 페이지 인덱싱 기법

디스크 I/O는 시스템 병목현상의 주된 요인으로 작용해왔다[18]. CPU 속도와 디스크 하드웨어의 속도 차이가 크기 때문에, 디스크 I/O의 횟수 감소는 시스템 성능 향상에 많은 영향을 끼칠 수 있다. 이를 위해 대부분의 시스템들은 미리 읽기, 지연된 쓰기 등의 알고리즘을 적용하고 있다. 미리 읽기는 디스크 I/O 횟수를 줄여 성능 향상을 가져왔지만, 부정확한 예측으로 인하여 사용되지 않을 데이타 I/O를 증가시켰다. 사용되지 않을 데이타를 복호화하는 것은 불필요한 성능저하를 가져온다. 이를 해결하기 위해 ISEA는 페이지 인덱싱 기법을 적용한다.

그림 5는 페이지를 암호화 단위 세그먼트[2]로 나누어 부분적으로 암호화된 페이지를 보여준다. 페이지 인덱싱 기법을 적용하면 페이지를 접근할 때 전체를 복호화하는 것이 아니라, 페이지에서 사용되는 데이타만 필요에 따라서 복호화한다. 인덱스 테이블을 통해서 암호화 세그먼트 단위로 페이지를 관리하기 때문에 부분적인 암복호화가 가능하다. 하위 계층 복호화 시스템은 디스크 블록 단위로 복호화하기 때문에 암호화 적용단위는 디스크 블록 단위에 의존적이다. 반면, 시스템은 페이지 단위로 I/O 요청을 하기 때문에 페이지 단위에 의존한다. ISEA의 암호화 단위는 시스템이나 하드웨어에 의존하지 않고 적용된 암호화 알고리즘의 세그먼트에 의존적이다. 암호화 세그먼트[3]는 암복호화가 적용되는 최소 단위이므로 페이지 내에서 복호화할 수 있는 최소 단위가 된다. 따라서, 페이지를 세그먼트 단위로 나누어 복호화하는 것은 불필요한 데이타 복호화를 최소화할 수 있다. 이를 위해 암호화된 데이타 페이지는 추가적인 인덱스 테이블을 가지고 있다. 테이블의 엔트리들은 암호

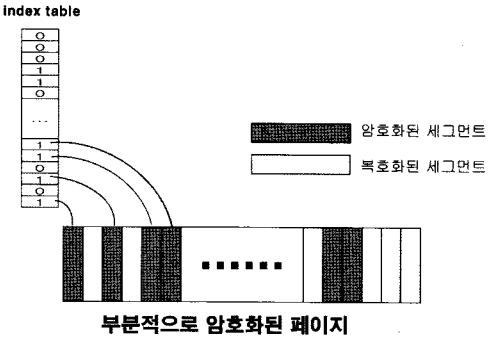


그림 5 페이지 인덱싱을 통한 부분 암호화

화 세그먼트단위로 페이지 내부의 암호화 여부를 표시한다. 엔트리 값이 1이면, 그 세그먼트는 암호화되어 있는 것을 의미하며 엔트리 값이 0이면, 복호화되어 있는 것을 의미한다. ISEA에서는 페이지 데이타를 접근할 때, 인덱스 테이블을 살펴본다. 각 세그먼트에 해당되는 엔트리를 살펴보고 암복호화의 필요성을 판단한다. 한 번 복호화된 세그먼트는 인덱스 테이블의 엔트리를 0으로 설정하여 반복된 접근 시 복호화연산을 하지 않는다. 페이지 크기가 4,096바이트라면 이러한 인덱스 테이블을 관리하기 위해서 추가적으로 32바이트의 인덱스 테이블이 필요하다. 반면 데이타를 암호화하는 과정은 간단하다. 데이타를 디스크에 저장할 때 페이지 캐쉬 인덱스 테이블을 살펴보고 엔트리 값이 0이면 암호화하여 디스크에 저장한다. 이 때 엔트리값은 수정하지 않고 후속 IO에 재사용할 수 있도록 한다.

4. 설계 및 구현

4.1 읽기

알고리즘 1은 ISEA에서 암호화된 데이타를 읽는 과정을 보여준다. 우선, 읽으려 하는 데이타의 페이지 인덱스를 계산한다(알고리즘 1의 1번). 계산된 인덱스를 바탕으로 데이타 페이지가 페이지 캐쉬에 있는 지 확인한다. 만약 페이지 캐쉬에 해당 페이지가 없으면 페이지 캐쉬 엔트리를 생성하고 페이지 크기만큼 디스크 읽기를 수행한다(알고리즘 1의 2번-8번). 이제 페이지 캐쉬에 요청한 데이타가 존재하지만, 부분적으로 암호화되어 있다. 사용자가 요청한 데이타의 길이는 다양하기 때문에 암호화 블록 단위의 길이와 일치하지 않을 수 있다. 따라서 복호화하기 위해 사용자 요청을 암호화 블록 단위로 정렬한다(알고리즘 1의 9번).

그림 6은 ISEA에서 암호화된 데이타를 읽기 위한 부분 복호화와 그에 따른 인덱스 관리를 보여준다. 만약 페이지의 22 오프셋부터 84 오프셋까지 총 62바이트를 읽으려 한다면, 오프셋 22-84을 포함하는 블록들의 인

2) 디스크 블록과의 혼동을 피하기 위해 암호화 블록에 대해 세그먼트라는 용어를 쓰겠다.

3) AES의 암호화 세그먼트 단위는 16바이트이다.

알고리즘 1 ISEA에서 데이타 복호화하여 읽기

```
READ(file, userBuf, from, to)
1: index[] := OffToIndex(file, from, to)
2: for i := 0 to LengthOf(index[]) do
3:   page := FindGetPage(PageCache, index[i])
4:   if page = NULL then
5:     page := AllocNewPage(pageCache, index[i])
6:     DiskRead(file, page)
7:   end if
8: end for
9: start := Align(from), end := Align(to)
10: for i := start to end) do
11:   if pageCache.indexTable[i] = 1 then
12:     Decrypt(pageCache, i, BLOCKCIPHERSIZE)
13:     pageCache.indexTable[i] := 0
14:     i += BLOCKCIPHERSIZE
15:   end if
16: end for
17: return CopyToUser(userBuf, pageCache, from, to)
```

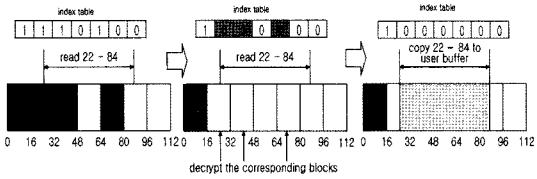


그림 6 ISEA 부분 복호화에 따른 인덱스 관리

덱스 테이블을 참조한다. 만약 인덱스가 1(암호화 상태)이라면 해당되는 블록을 복호화시키고 인덱스를 0으로 설정(복호화 상태)한다(알고리즘 1의 10번-16번). 마지막으로 페이지 캐쉬의 데이타를 사용자 공간에 복사해 주고 그 양을 리턴한다(알고리즘 1의 17번).

4.2 쓰기

알고리즘 2는 ISEA에서 암호화된 데이타 쓰기 과정을 보여준다. 우선 쓰려 하는 데이타의 페이지 인덱스를 계산한다(알고리즘 2의 1번). 계산된 인덱스를 바탕으로 쓰기 위한 준비 작업을 한다. PrepareWrite는 인덱스에 해당되는 페이지를 페이지 캐쉬에 위치시킨다. 만약, 디스크에 존재하는 데이타의 페이지라면 디스크 읽기를 수행한다(알고리즘 2의 2번-4번). 사용자가 요청한 데이타의 길이는 다양하기 때문에 암호화 블록 단위의 길이와 일치하지 않을 수 있다. 따라서 복호화하기 위한 사용자 요청을 암호화 블록 단위로 정렬한다(알고리즘 2의 5번).

그림 7은 ISEA에서 데이타를 쓰기 위한 부분 복호화와 그에 따른 인덱스 관리를 보여준다. 쓰기 과정에서

알고리즘 2 ISEA에서 데이타 암호화하여 쓰기

```
WRITE(file, userBuf, from, to)
1: index[] := OffToIndex(file, from, to)
2: for i := 0 to LengthOf(index[]) do
3:   PrepareWrite(file, index[i])
4: end for
5: start := Align(from), end := Align(to)
6: if pageCache.indexTable[start] = 1 then
7:   Decrypt(pageCache, start, BLOCKCIPHERSIZE)
8:   pageCache.indexTable[start] := 0
9: end if
10: if pageCache.indexTable[end] = 1 then
11:   Decrypt(pageCache, end, BLOCKCIPHERSIZE)
12:   pageCache.indexTable[end] := 0
13: end if
14: for i := start + BLOCKCIPHERSIZE to end - BLOCKCIPHERSIZE
    do
15:   pageCache.indexTable[i] := 0
16:   i += BLOCKCIPHERSIZE
17: end for
18: CopyFromUser(userBuf, pageCache, from, to)
19: ReadFromPageCache(tempBuf, start, end)
20: Encrypt(tempBuf, start, end)
21: return CommitWrite(userBuf, pageCache, from, to)
```

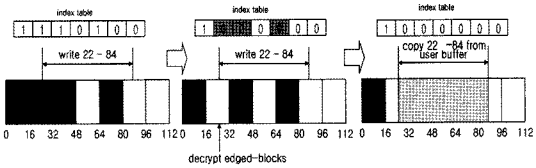


그림 7 ISEA 쓰기에 따른 인덱스 관리

복호화되어 있는 데이타는 암호화해야 하지만, 암호화되어 있는 데이타는 다시 암호화할 필요가 없다. 따라서 이에 대한 인덱스 관리가 필요하다. 만약 페이지의 22 오프셋부터 84오프셋까지 총 62바이트를 쓰려 한다면, 최외곽 블록들(오프셋 16-32, 64-80 블록)을 복호화한다. 이미 80-96 블록은 복호화되어 있으므로(알고리즘 2의 10번-13번), 여기서는 오프셋 16-32을 포함하는 블록만 복호화한다(알고리즘 2의 6번-9번). 복호화된 블록과 사용자 데이타가 쓰여질 블록의 인덱스를 0으로 설정한다(알고리즘 2의 14번-17번). 그 후, 사용자 데이타를 페이지 캐쉬에 복사한다(알고리즘 2의 18번). 디스크에 데이타를 저장하는 과정은 하위 계층 암호화 파일 시스템의 쓰기와 같다(19-21).

5. 실험 및 평가

모든 실험은 하이버스에서 제작된 X-Hyper270B 임

표 1 실험 환경

CPU	Intel Bulverde PXA270(520 MHz)
메인 메모리	SDRAM 64Mbytes
저장 장치	NAND Flash
파일 시스템	YAFFS2 [20]
암호 알고리즘	AES 16바이트 키

베디드 보드[19]에서 이루어졌다. 실험 환경은 표 1과 같으며 AES암호화로 인하여 240.7(Kbytes/초)의 추가 비용이 소모된다. 시스템에서 읽기/쓰기는 페이지 단위로 이루어지므로 연산의 최대 크기를 4킬로바이트로 설정하였다. 모든 성능 측정은 5,000회 이상 수행하였고 그 평균값을 측정하였다.

상위 계층 암호화 파일 시스템(upper layer)은 사용자 암호화 시스템, 암호화 파일 시스템등과 같은 버퍼 캐쉬 상위 계층에서 암호화하는 파일 시스템[2-6,17]을 지칭한다. 하위 계층 암호화 파일 시스템(lower layer)은 디스크 드라이버 암호화, 저장 장치 암호화 시스템과 같이 버퍼 캐쉬 하위 계층에서 암호화 하는 파일 시스템[7,8,12-15]을 지칭한다.

기존의 파일 시스템은 구현상의 차이와 암호화 알고리즘의 차이로 인하여 각각 다른 성능을 보여준다. 따라서, 암복호화 기법의 성능을 비교하기 위하여 기존의 파일 시스템을 사용하는 것은 어렵다. 따라서 기존의 암호화 파일 시스템들을 상위 계층, 하위 계층 시스템으로 분류하여 두 시스템을 제안한 시스템(ISEA)와 동일한 조건으로 구현하였다.

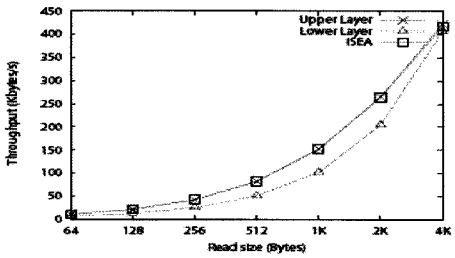
이후, 편의상 상위 계층 암호화 시스템을 상위시스템, 하위 계층 암호화 시스템을 하위시스템으로 지칭하겠다.

5.1 읽기 성능

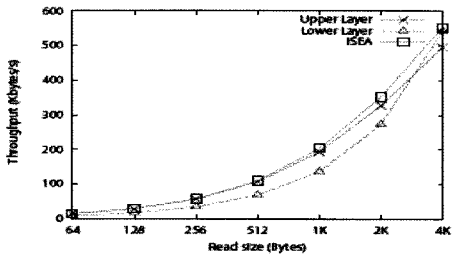
미리 생성된 1000개의 파일에 대해서 페이지 탐색 성공률과 읽기 사이즈를 조정해가며 읽기를 수행하였다. 그림 8은 각 시스템의 읽기 성능을 보여준다. x축은 로그 스케일로 읽기 사이즈를 나타내며 y축은 그에 따른 처리량을 보여준다. 그림 8(a),(b),(c),(d)에서 ISEA는 가장 좋은 성능을 보였으며, (e)에서는 하위 시스템보다는 좋지 않은 성능을 보이나 상위 시스템보다는 월등히 좋은 성능을 보여주었다.

5.1.1 캐쉬의 영향

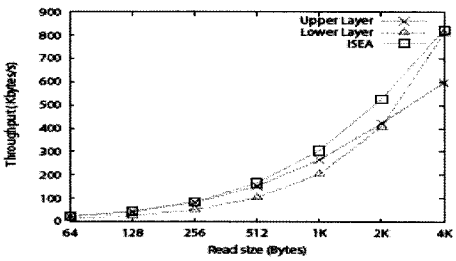
그림 8에서 페이지 캐쉬 탐색 성공률이 커짐에 따라 상위 시스템과 ISEA의 격차가 점점 벌어지는 것을 볼 수 있다. 이는 ISEA가 복호화한 페이지 캐쉬를 직접적으로 사용하기 때문이다. 그림 (a)에서처럼 페이지 캐쉬의 영향을 받지 않는다면 복호화하는 데이타의 양이 동일하기 때문에 ISEA와 상위 시스템은 비슷한 성능을 보인다. 하지만, 페이지 캐쉬 탐색에 성공하면, ISEA는



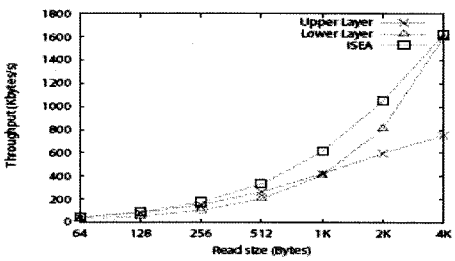
(a) 캐쉬 탐색 성공률 = 0



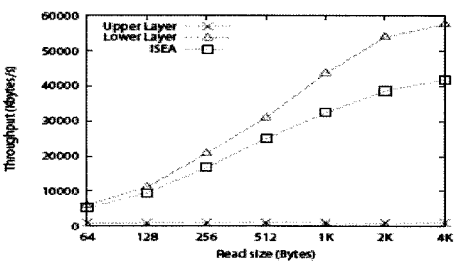
(b) 캐쉬 탐색 성공률 = 0.25



(c) 캐쉬 탐색 성공률 = 0.50



(d) 캐쉬 탐색 성공률 = 0.75



(e) 캐쉬 탐색 성공률 = 1.0

그림 8 캐쉬 성공률에 따른 읽기 성능

데이타를 복호화하지 않고 읽기를 수행하지만, 상위 시스템은 데이타를 반복적으로 복호화해야 한다. 만약 사용자 프로세스가 데이타 블록 하나만 읽으려 하고 데이타 블록이 이미 페이지 캐쉬에 존재한다면, ISEA는 사용자 프로세스에 복사만 하면 되지만, 상위 시스템은 복사하기 전에 복호화해야 한다. 따라서, 페이지 캐쉬에서 복호화하는 데이타의 양이 늘어날수록 두 시스템 성능 격차는 커진다.

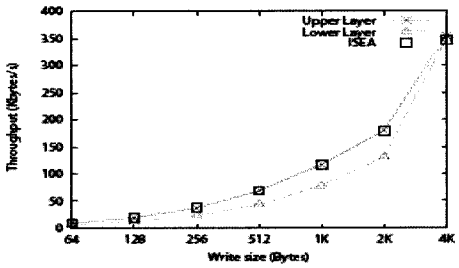
5.1.2 읽기 사이즈의 영향

그림 8에서 읽기 사이즈가 작으면 ISEA와 하위시스템의 성능 격차가 크지만, 4K바이트가 되면 같아짐을 볼 수 있다. 이는 하위 시스템은 4K바이트 단위로 미리 복호화하기 때문이다. 만약 한 페이지가 1,2,3,4블록으로 구성되고, 1 블록을 읽으려 한다면 상위 시스템과 ISEA는 블록 하나만 복호화하며 되지만, 하위 시스템은 한 블록만 요구하더라도 페이지를 구성하는 모든 블록을 복호화해야 한다. 커널은 페이지 단위로 읽기를 요청하기 때문에 하위 시스템은 페이지 전체를 복호화하기 때문이다. 이로 인하여 그림 8에서 읽기 사이즈가 4K바이트보다 작으면, ISEA가 하위 시스템에 비해 좋은 성능을 보인다. 반면 그림 (e)에 하위 시스템이 ISEA보다 좋은 성능을 보인다. 요청한 데이타 모두 페이지 캐쉬에 존재하므로 하위 시스템은 추가비용이 필요하지 않지만, ISEA는 페이지 인덱스 테이블을 관리하기 위해 추가 비용을 소모하기 때문이다.

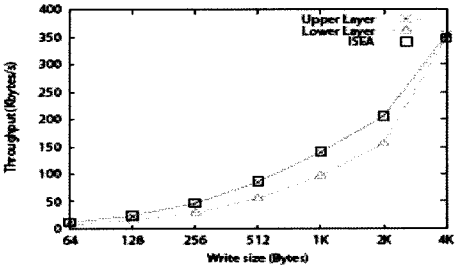
5.2 쓰기 성능

미리 생성된 1000개의 파일에 대해서 페이지 탐색 성공률과 한번에 쓰는 데이타 사이즈를 조정해가며 쓰기를 수행하였다. 그림 9는 각 시스템의 쓰기 성능을 보여준다. x축은 로그 스케일로 쓰기 사이즈를 나타내며 y축은 그에 따른 처리량을 보여준다. 그림 9(a),(b),(c),(d)에서 ISEA와 상위 시스템이 좋은 성능을 보이며, (e)에서는 세 시스템 비슷한 성능을 보여준다.

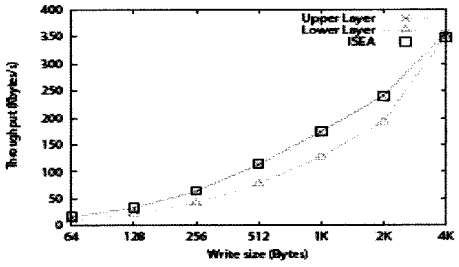
파일 쓰기 사이즈는 쓰기 성능에 크게 영향을 미친다. 만약 1번 블록을 암호화하여 저장하려 하고 한 페이지를 1,2,3,4 블록이 구성한다면, 상위 시스템과 ISEA는 비슷한 과정으로 암호화한다. 1을 쓰기 위해서 1,2,3,4의 데이타 블록을 디스크로부터 읽어 들여 페이지 캐쉬 엔트리를 구성한다. 이후 변경된 데이타 1을 쓰고 암호화하여 저장한다. 상위 시스템은 데이타 1을 암호화하여 페이지 캐쉬에 저장하고 ISEA는 데이타 1을 암호화하여 디스크에 저장한다. 즉 암호화하는 위치만 다를 뿐 전체적인 과정은 비슷하다 하지만, 하위 시스템의 경우 블록 1,2,3,4를 페이지 캐쉬에 읽어올 때, 복사할 뿐만 아니라 복호화해야 한다. 하위 시스템은 상위의 파일 시스템에 대한 지식이 없기 때문에 쓰기를 위한 디스크



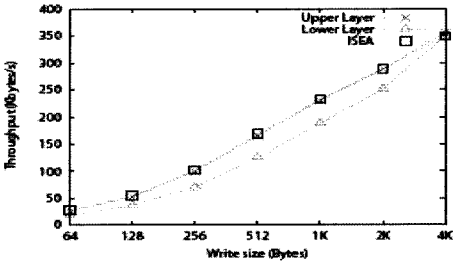
(a) 캐쉬 탐색 성공률 = 0



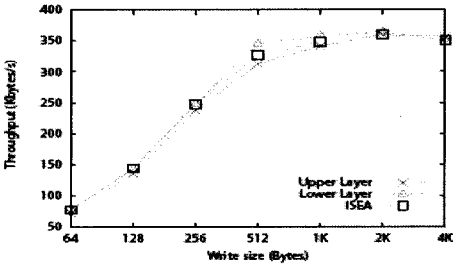
(b) 캐쉬 탐색 성공률 = 0.25



(c) 캐쉬 탐색 성공률 = 0.50



(d) 캐쉬 탐색 성공률 = 0.75



(e) 캐쉬 탐색 성공률 = 1.0

그림 9 캐쉬 탐색 성공률에 따른 쓰기 성능

그림 9 캐쉬 탐색 성공률에 따른 쓰기 성능

읽기인지 구분할 수 없다. 즉 1 블록을 쓰기 위해서 1,2,3,4를 복호화하는 과정이 필요하다. 따라서 그림 9에서 읽기 사이즈가 4K바이트보다 작을 때 하위 시스템이 성능 저하를 보인다.

6. 결 론

본 논문에서는 기존의 암호화 파일 시스템을 상위 계층 암호화 파일 시스템, 하위 계층 암호화 파일 시스템의 두 종류로 분류하여 각각의 특성을 분석하고 불필요한 성능 저하 요인을 확인하였다. 그리고 이를 해결하기 위해서 ISEA(Indexed and Separated Encryption Approach)를 제안하였다.

ISEA는 (1)암복호화 계층 분리를 통한 페이지 캐쉬 이용 (2)페이지 인덱싱을 이용한 부분적인 페이지 캐쉬 암복호화를 통하여 암호화 파일 시스템의 성능을 개선하였다. ISEA는 암호화 시스템의 성능 개선을 위해 페이지 캐쉬 계층에서 암복호화를 지원하도록 설계되었다. ISEA는 페이지 캐쉬에 복호화된 데이타를 직접적으로 반영하고, 부분적으로 암복호화하기 때문에 상위 계층, 하위 계층 암호화 파일 시스템에 비해 대부분의 경우 좋은 성능을 보여주었다. 특히 페이지 캐쉬 탐색 성공률에 높아짐에 따라 상위 계층 암호화 파일 시스템에 비해 탁월한 읽기 성능을 보여주고, 읽기/쓰기 사이즈가 페이지 크기보다 적은 경우 하위 계층 암호화 파일 시스템에 비해 좋은 성능을 보여주었다.

참 고 문 헌

[1] S. Ravi, A. Raghunathan, P. Kocher and S. Hattangady, "Security in Embeddes systems: Design Challenges," Trans. on Embedded Computing Sys. Vol.3, No.3 pp. 461-491, 2004.

[2] M. Blaze, "A cryptographic file system for unix," Proceedings of the 1st ACM Conference on Computer and Communications Security. pp. 9-16 1993.

[3] G. Cattaneo, L. Catuogno, A. D Sorbo and P. Persiano, "The Design and Implementation of a Transparent Cryptographic File System for Unix," Proceedings of the FREENUX Track: 2001 USENIX Anuual Technical Conference, pp. 199-212, 2001.

[4] Microsoft Corporation, "Encryption File System for Windows 2000," Tech. Rep. 2000.

[5] E. Zadok, I. Badulescu and A. Shender, "Cryptfs: A Stackable vnode level encryption file system," Tech. Rep., 1998.

[6] C. Wright, M. Martino and E. Zadok, "Ncryptfs: A secure and convenient cryptographic file system," Proceedings of the Annual USENIX Technical Conference 2003, pp. 197-210.

[7] GNU License, "The GNU/Linux CryptoAPI," August 2003 http://www.kerneli.org

[8] Jetico Inc. "Bestcrypt corporate edition," 2001 http://www.jetico.com

[9] E. Riedel, M. Kallahalla and R. Swaminathan, "A framework for evaluating storage system security," Proceedings of the 1st USENIX Conference on File and Strorate Technologies. pp. 15-30, 2002.

[10] J. S. Heidemann and G. J. Popek, "Performance of cache coherence in stackable filing," Symposium on Operating System Principle, pp. 127-142, 1995.

[11] D. P. Bovet and M. Cesati, "Understanding the Linux Kernel," O'Reilly, 2006.

[12] Kingston Technology company, "DataTraveler Elite," 2006, http://www.kingston.com/flash/dt_elite.asp

[13] ABIT Computer corporation, "Secure IDE," 2003, http://abit-usa.com/products/multimedia/secureide

[14] R. Dowdeswell and J. Ioannidis, "The Cryptographic disk driver," Proceedings of the Annual USENIX Technical Conference, FREENIX Track, 2003.

[15] P. C. Gutmann, "Secure File System(SFS) for DOS/Windows," 1994. http://www.cs.auckland.ac.nz/~pgut001/sfs/

[16] ARC Advisory Group, "Microsoft Security Overview," 2003.

[17] GNU License, "Gnu Privacy Guard," 1999 http://www.gnupg.org

[18] J. K. Ousterhout and F. Douglis, "Beating the I/O Bottleneck: A Case for Log-Structured File System," Operating Systems Reviews, Vol.23, No.1, pp. 11-28, 1989.

[19] Hybus Corporation, "X-hyper270b," 2005 http://www.hybus.net

[20] Aleph. One, "Yaffs2," 2002 http://www.aleph1.co.uk

허 준 영

1998년 서울대학교 컴퓨터공학과 졸업(학사). 2002년~현재 서울대학교 컴퓨터공학부 박사과정(수료). 관심분야는 시스템 및 운영체제, 무선 센서 네트워크, 결함 허용 시스템, 임베디드 시스템 보안

박 재 민

2005년 서울대학교 컴퓨터공학과 졸업(학사). 2007년 서울대학교 컴퓨터공학과 졸업(석사). 현재 삼성전자 소프트웨어 연구소 재직. 관심분야는 시스템 및 운영체제, 임베디드 시스템 보안. OS 가상화

조 유 근

1971년 서울대학교 졸업(학사). 1978년 미국 미네소타 대학교 컴퓨터 과학 박사 졸업. 1985년 미국 미네소타 대학교 방문 교수. 2001년 한국 정보과학회 회장 1979년~현재 서울대학교 컴퓨터공학부 교수. 관심분야는 운영체제, 알고리즘, 시스템 보안, 결함 허용 컴퓨팅