NAND 플래시 파일 시스템을 위한 안전 삭제 기법

(A Secure Deletion Method for NAND Flash File System)

이 재 흥 † 오 진 하 ††
(Jaeheung Lee) (Jinha Oh)

김 석 현 † 이 상 호 †
(Seokhyun Kim) (Sangho Yi)

허 준 영 † 조 유 근 †††
(Junyoung Heo) (Yookun Cho)

홍 지 만 ††††
(Jiman Hong)

요 약 대부분의 파일 시스템에서는 파일이 삭제되더라도 메타데이타만 삭제되거나 변경될 뿐, 파일의 데이타는 물리 매체에 계속 저장된다. 그렇기 때문에 지워진 파일을 복구하는 것도 가능한데 이러한 복구를 불가능하도록 하는 것을 원하는 사용자들도 있다. 특히 이러한 요구는 플래시 메모리를 저장 장치로 사용하는 임베디드 시스템에서 더욱 더 많아지고 있다. 이에 본 논문에서는 NAND 플래시 파일 시스템을 위한 안전 삭제 기법을 제안하고 이를 가장 대표적인 NAND 플래시 파일 시스템인 YAFFS에 적용하였다. 제안 기법은 암호화를 기반으로 하고 있으며, 특정 파일을 암호화하는데 사용된 모든 키들이 같은 블록에 저장되도록 하여, 단 한번의 블록 삭제 연산으로 파일 복구가 불가능하도록 하였다. 시뮬레이션 결과는 제안 기법으로 인해 파일의 생성 및 변경 시 발생하는 블록 삭제를 감안하더라도 파일 삭제 시 발생하는 블록 삭제 횟수가 단순 암호화 기법의 경우보다 더 작다는 것을 보여준다. 본 논문에서는 제안 기법을 YAFFS에만 적용하였지만, 다른 NAND 플래시 전용 파일 시스템에도 쉽게 적용 가능하다.

키워드 : 안전 삭제, 파일 시스템, 플래시 메모리, 키 관리

Abstract In most file systems, if a file is deleted, only the metadata of the file is deleted or modified and the file's data is still stored on the physical media. Some users require that deleted files no longer be accessible. This requirement is more important in embedded systems that employ flash memory as a storage medium. In this paper, we propose a secure deletion method for NAND flash file system and apply the method to YAFFS. Our method uses encryption to delete files and forces all keys of a specific file to be stored in the same block. Therefore, only one erase operation is required to securely delete a file. Our simulation results show that the amortized number of block erases is smaller than the simple encryption method. Even though we apply our method only to the YAFFS, our method can be easily applied to other NAND flash file systems.

Key words : Secure deletion, File system, Flash memory, Key management

1. 서 론

일반적으로 사람들은 파일이 삭제되면, 삭제된 파일이 저장되었던 모든 블록들이 물리적으로 지워진다고 생각한다. 하지만 대부분의 파일 시스템에서는 파일이 삭제되더라도 메타데이타만 삭제되거나 변경될 뿐, 파일의 데이타는 물리 매체에 계속 저장되어 있는 경우가 많다[1]. 그러므로 악의적인 사용자가 삭제된 파일을 복구시키는 것은 그다지 어려운 일이 아니다.

많은 사람들이 가지고 있는 또 다른 오해는 파일의 데이타가 덮어 쓰여지게 되면 삭제된 파일을 복구 시킬 수 없다고 생각하는 것이다. 하지만 하드디스크 같은 자기 매체의 경우에는 데이타가 덮어 쓰여지게 되더라도 절대 복구 불가능한 것만은 아니다[2]. 그러므로 자기 매체를 위한 많은 안전 삭제 방법들은 정해진 패턴 또는 난수 데이타를 여러 번 덮어 씀으로써 안전 삭제를 구현한다[3,4].

점점 더 많은 임베디드 시스템들이 실생활에 이용됨에 따라, 임베디드 시스템에 저장되는 기밀 정보의 양도

· 본 연구는 2단계 BK21 사업 지원으로 이루어졌으며 서울대학교 컴퓨터연구소에서 연구장비를 지원하고 공간을 제공하였음
· 이 논문은 제34회 추계학술대회에서 'NAND 플래시 파일 시스템을 위한 안전 삭제 기법'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 서울대학교 컴퓨터공학부
jhlee@ssrnet.snu.ac.kr
shkim@ssrnet.snu.ac.kr
shyi@ssrnet.snu.ac.kr
jyheo@ssrnet.snu.ac.kr
†† 비 회 원 : 서울대학교 컴퓨터공학부
jhoh@ssrnet.snu.ac.kr
††† 종신회원 : 서울대학교 컴퓨터공학부 교수
ykcho@snu.ac.kr
†††† 정 회 원 : 숭실대학교 컴퓨터학부 교수
jiman@ssu.ac.kr

논문접수 : 2007년 12월 4일
심사완료 : 2008년 2월 20일

Copyright@2008 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.

정보과학회논문지: 컴퓨팅의 실제 및 레터 제14권 제3호(2008.5)

따라서 늘어나게 되었다. 이러한 임베디드 시스템들은 휴대성이 좋기 때문에 도난 당하기도 쉽다. 그리고 일단 도난 당하게 되면, 저장된 파일 뿐만 아니라 이미 삭제된 파일 역시 타인에게 노출될 수 있다.

플래시 메모리는 빠른 접근 속도, 낮은 전력 소비, 충격 및 온도에 대한 내성, 작은 크기와 무소음 등의 장점으로 인해 임베디드 시스템에서 많이 사용되고 있다[5]. 이러한 플래시 메모리에는 NAND 타입과 NOR 타입의 두 종류가 있는데, NAND 플래시 메모리가 용량 대비 가격이 저렴해서 임베디드 시스템을 위한 대용량 저장 장치로서 널리 사용되고 있다.

플래시 메모리는 물리적인 특성 상, 이미 쓰여진 곳에 다른 내용을 쓰기 위해서는 삭제 연산을 필요로 한다. 플래시 메모리에서 삭제 연산은 블록 단위로 수행되고, 쓰기 연산은 페이지 단위로 수행된다. 일반적으로 블록은 512 바이트 크기의 페이지 32 개나 2048 바이트 크기의 페이지 64 개로 구성되는데, 이러한 삭제 단위와 쓰기 단위의 불일치 때문에 많은 플래시 파일 시스템들은 로그 기반 구조를 채택하고 있다[6]. YAFFS[7] 역시 로그 기반 구조를 채택하고 있다.

안전 삭제 방법은 크게 두 가지로 나뉘어진다[8]. 첫 번째는 파일의 전체 데이타를 덮어 쓰는 것이다[3]. 그리고 두 번째는 데이타를 암호화하고 암호화에 사용된 키만을 앞의 방법으로 안전하게 삭제하는 것이다[9]. 플래시 메모리에서는 지우는데 드는 부하가 암/복호화의 경우보다 크기 때문에 두 번째 방법이 보다 바람직하다.

본 논문에서는 NAND 플래시 파일 시스템을 위한 안전 삭제 기법을 제안하고 이를 YAFFS에 적용하였다. 제안 기법은 암호화를 기반으로 안전 삭제를 수행한다. 다른 대부분의 플래시 전용 파일 시스템들과 마찬가지로 YAFFS도 로그 기반 구조 파일 시스템이기 때문에, 파일이 변경된 적이 있을 경우, 현재의 암호화 키뿐만 아니라 기존의 암호화 키들도 플래시 메모리에 같이 존재할 수 있다. 파일을 안전하게 삭제하기 위해서는 이러한 키들을 모두 다 지워주어야 하는데, 제안 기법은 특정 파일을 암호화하는데 사용된 모든 키들을 같은 블록에 저장되도록 하여, 단 한번의 블록 삭제 연산으로 이를 가능하게 한다. 또한 파일 암호화 시 사용된 키가 파일의 메타데이타를 저장하는 객체 헤더에 저장되기 때문에, 제안 기법은 암호화 키뿐만 아니라 파일의 메타데이타 전체를 안전하게 삭제한다. 본 논문에서는 제안 기법을 YAFFS에만 적용하였지만, 다른 NAND 플래시 전용 파일 시스템에도 쉽게 적용할 수 있다.

본 논문의 나머지 부분은 다음과 같이 구성되어 있다. 먼저 2장에서 안전 삭제에 관한 기존의 연구들과 YAFFS에 대해 살펴보고, 3장에서 본 논문에서 제시하는 안전 삭제 기법을 소개할 것이다. 4장에서는 3장에서 제시한 기법을 평가하고, 5장에서 결론을 맺을 것이다.

2. 관련 연구

2.1 안전 삭제

안전 삭제와 관련된 기존의 연구들은 덮어 쓰기와 암호화의 두 가지 종류로 나뉜다.

덮어 쓰기의 경우, 미리 정해진, 또는 무작위의 데이타 시퀀스를 삭제하려는 파일 위에 덮어 쓴다. 자기 매체에 기록된 데이타는 덮어 쓰기가 수행된 이후라도 자기 현미경(Magnetic Force Microscopes)을 통한 복구가 가능한 것으로 알려져 있다[10]. 그러므로 덮어 쓰기 기반의 대부분의 안전 삭제 기법들은 여러 번 데이타를 덮어 쓰도록 구현되었다. 미국의 국가표준기술연구소(The National Institute of Standards and Technology, NIST)는 자기 매체의 경우에 최소한 3번은 덮어 쓰기를 하도록 권고하고 있다[11]. 민감한 자료를 지우는 것에 대한 미국 국방성 표준은 DoD5220.22-M으로도 알려져 있는 국가 산업 보안 프로그램 작업 교범(the National Industrial Security Program Operating Manual, NISPOM)에 기술되어 있다[12]. Guttmann의 안전 삭제 방법은 지금까지 나온 방법들보다 더 엄격하다. Guttmann의 방법은 안전 삭제를 위해 전체 35번의 덮어쓰기를 수행한다. 또한 Bauer는 비동기적인 안전 삭제 기법을 제안하고 리눅스 커널에 구현하였다[4].

암호화 기반 안전 삭제는 암호화에 사용되었던 키를 앞의 방법으로 지움으로써 파일 복구를 불가능하게 한다[9]. 이러한 기법을 이용하기 위해서는 모든 파일들이 서로 다른 키로 암호화되어야만 한다.

2.2 YAFFS

YAFFS는 "Yet Another Flash File System"의 약자로서 NAND 플래시 메모리를 위해 설계된 파일 시스템이다[7]. 플래시 메모리는 특성 상, 이미 쓰여진 곳에 다른 내용을 쓰기 위해서 삭제 연산을 필요로 한다. 이러한 삭제 연산은 블록 단위로 이루어지기 때문에 부하가 꽤 크고, 그렇기 때문에 YAFFS는 같은 자리에 다시 쓰는 것을 가능한 한 피하도록 로그 기반 구조로 설계되었다.

YAFFS에서 각각의 페이지는 파일 ID와 Chunk 번호를 가진다. YAFFS에서 각 파일은 자신만의 고유한 파일 ID를 가진다. Chunk 번호는 각 페이지의 파일 내에서의 위치에 의해 결정된다. 각 파일의 메타데이타는 Chunk 번호가 0인 페이지에 저장되고, 파일 데이타는 Chunk 번호가 1, 2, 3, …인 페이지에 저장된다. YAFFS2에서는 각각의 페이지가 Chunk 번호뿐만 아니라 시퀀스 번호도 가진다. 시퀀스 번호는 최근 나온

NAND 플래시들의 "Write once" 요구를 만족시키기 위해 추가되었다. 시퀀스 번호는 단조적으로 증가하고 모든 새로 쓰여진 블록에 표시된다. 여러 페이지가 동일한 파일 ID와 Chunk 번호를 가질 때, YAFFS2는 가장 큰 시퀀스 번호를 가진 페이지를 선택한다. 각 페이지의 파일 ID, Chunk 번호와 시퀀스 번호는 플래시 메모리의 "Spare data" 영역에 저장된다.

3. 제안 기법

본 절에서는 NAND 플래시 파일 시스템을 위한 안전 삭제 기법을 제안하고 이를 YAFFS에 적용한다. 제안 기법은 파일을 안전하게 삭제하기 위해 암호화를 이용하는 데, 모든 파일은 파일마다 서로 다른 키로 암호화된다. 이 키들은 파일이 생성되거나 변경될 때 임의로 생성되고, 파일의 메타데이타를 저장하는 부분인 객체 헤더에 같이 저장된다. 그러므로 특정 파일의 키를 저장한 객체 헤더들이 모두 삭제된다면 파일을 복구하는 것은 불가능하다. 이전 절에서 언급한 바와 같이 YAFFS는 로그 기반 구조 파일 시스템이므로 하나의 파일 시스템 안에 동일한 파일 ID와 Chunk 번호를 가지는 페이지들이 여러 개 있을 수 있다. 그러므로 파일을 복구 불가능하게 만들기 위해서는 파일의 현재 객체 헤더뿐만 아니라 기존의 모든 객체 헤더들도 같이 삭제되어야만 한다. 제안 기법은 특정 파일의 키를 저장하는 객체 헤더들을 모두 하나의 블록에 저장되도록 한다. 그래서 단 한 번의 블록 삭제 연산만으로 안전하게 파일을 삭제할 수 있도록 한다.

3.1 자료 구조

플래시 메모리는 고정된 크기를 가진 블록으로 구성된다. 제안 기법은 전체 블록을 헤더 블록과 데이타 블록으로 나눈다. 페이지 크기인 객체 헤더들은 헤더 블록에만 들어가고 파일 데이타들은 데이타 블록에만 들어간다. 특정 블록이 헤더 블록인지 아니면 데이타 블록인지를 알기 위해서는 그 블록 안의 페이지들의 Chunk 번호가 0인지 여부를 살펴보면 된다. Chunk 번호가 0이면 그 블록은 헤더 블록이고 그렇지 않으면 그 블록은 데이타 블록이다.

본 논문에서 제안하는 기법은 이진 해시 트리 구조를 사용한다. 이 트리는 DRAM 메모리에 존재하고, 파일의 생성이나 변경으로 인해 새로 생성되는 객체 헤더가 지정된 헤더 블록에 저장되도록 하는데 사용된다. 이 트리 구조에서 각 노드는 유일한 Value 값을 가지며, 루트 노드의 경우 이 값은 NULL이다. 각 노드의 왼쪽 자식 노드는 부모 노드의 Value 값의 가장 왼쪽에 0을 덧붙인 값을 가지며, 오른쪽 자식 노드는 부모 노드의 Value 값의 가장 왼쪽에 1을 덧붙인 값을 가진다. 각각

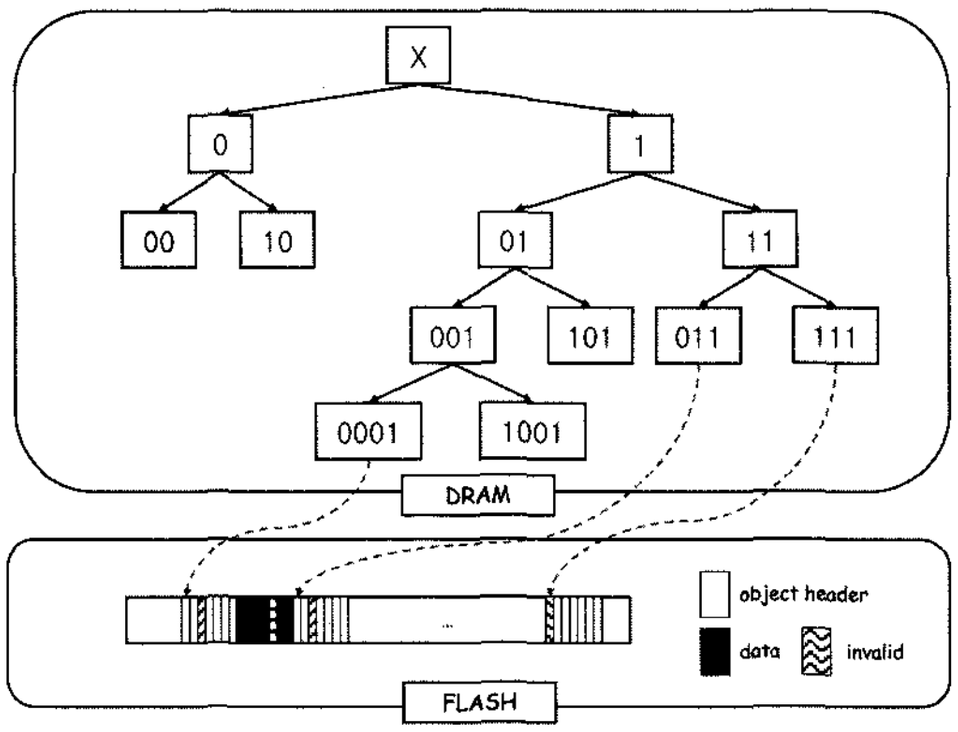

그림 1 이진 해시 트리와 이와 연결된 헤더 블록의 예

의 단말 노드는 플래시 메모리에 존재하는 서로 다른 헤더 블록에 대한 포인터를 가진다. 파일의 생성이나 변경으로 인해 객체 헤더가 새로 생성될 경우, 이 객체 헤더는 해당 파일 ID의 n개의 LSB(Least Significant Bits) 값이 깊이가 n인 노드의 Value 값과 같은 단말 노드가 가리키는 헤더 블록에 저장된다. 그러므로 같은 파일 ID를 가지는 모든 객체 헤더들은 같은 헤더 블록에 저장된다.

그림 1은 제안 기법에서 사용하는 이진 해시 트리와 이와 연결된 헤더 블록의 예를 보여준다. 이 그림에서 값이 '0001'인 노드는 파일 ID가 '0001'로 끝나는 객체 헤더를 저장하는 헤더 블록을 가리킨다. 그래서 파일 ID가 '0001'로 끝나는 모든 객체 헤더는 같은 헤더 블록에 저장된다.

YAFFS에서 객체 헤더는 하나의 페이지를 차지한다. 하나의 블록은 32 또는 64 개의 페이지로 구성되어 있으므로 각 헤더 블록은 최대 32 또는 64 개의 객체 헤더를 담을 수 있다. 그러므로 헤더 블록에 저장되는 객체 헤더의 수를 32 또는 64 개로 제한해야 한다. 다음 절에서 제안 기법이 어떻게 하나의 헤더 블록에 저장되는 객체 헤더의 수를 제한하는지 설명한다.

3.2 연산

3.2.1 OBJECT_HEADER_ADD 연산

OBJECT_HEADER_ADD 연산은 파일의 생성이나 변경으로 인해 새로 생성되는 객체 헤더를 위한 공간을 헤더 블록 안에 확보하기 위해 수행된다.

(1) 생성하거나 변경하려는 파일의 ID를 얻는다.

(2) 노드의 깊이가 n이고, 파일 ID의 n개의 LSB 값이 자신의 Value와 같은 값을 가지는 단말 노드를 탐색한다. 구체적으로는 루트 노드에서 시작해서 단말 노드가 발견될 때까지 파일 ID를 LSB부터 살펴보면서 0이면 왼쪽, 1이면 오른쪽 자식 노드로 탐색을

수행해 나가면 된다.

(3) (2)에서 발견된 단말 노드가 가리키는 헤더 블록에 새로운 객체 헤더가 들어갈 공간이 있는지 확인한다. 빈 공간이 있다면 객체 헤더를 그 헤더 블록에 저장하고 연산을 끝낸다.

(4) 만약 빈 공간이 없다면 그 헤더 블록을 DRAM에 복사하고, 그 블록에 유효한 객체 헤더가 몇 개 있는지 확인한다.

(5) (4)에서 구한 유효한 객체 헤더의 수가 한 블록의 페이지 수의 반 이상이면, 기존 헤더 블록의 Value 값의 왼쪽에 0과 1을 덧붙인 Value 값을 가지는 두 개의 헤더 블록을 각각 할당하고, 기존의 유효한 객체 헤더와 새롭게 생성된 객체 헤더를 그들의 파일 ID에 따라 새로 생성된 두 개의 헤더 블록에 알맞게 저장한다.

(6) 만약 (4)에서 구한 유효한 객체 헤더의 수가 한 블록의 페이지 수의 반을 넘지 않으면, 이전의 헤더 블록의 Value 값과 동일한 Value 값을 가진 하나의 헤더 블록을 할당하고, 이 블록에 기존의 유효한 객체 헤더와 새롭게 생성된 객체 헤더를 저장한다.

(7) (4)에서 복사된 블록을 지운다.

만약 (7)에서 블록을 지우는 것이 새로운 블록 할당과 저장 이전에 수행된다면 정전 등의 경우에 해당 헤더 블록의 모든 객체 헤더를 잃어버리는 경우가 생길 수도 있다. 그러므로 기존의 블록을 지우는 것은 새로운 블록 할당과 저장 이후에 수행되어야만 한다.

3.2.2 OBJECT_HEADER_DEL 연산

OBJECT_HEADER_DEL 연산은 파일을 안전하게 삭제하고자 할 때 수행된다.

(1) 안전하게 삭제하고자 하는 파일의 ID를 얻는다.

(2) 노드의 깊이가 n이고, 파일 ID의 n개의 LSB 값이 자신의 Value와 같은 값을 가지는 단말 노드를 탐색한다. 구체적으로는 루트 노드에서 시작해서 단말 노드가 발견될 때까지 파일 ID를 LSB부터 살펴보면서 0이면 왼쪽, 1이면 오른쪽 자식 노드로 탐색을 수행해 나가면 된다.

(3) (2)에서 발견된 단말 노드가 가리키는 헤더 블록을 DRAM으로 복사한다.

(4) (1)에서 얻은 파일 ID를 가진 객체 헤더들을 모두 찾아 유효하지 않은 것으로 설정한다.

(5) 기존의 헤더 블록의 값과 같은 값을 가지는 하나의 헤더 블록을 할당하고 모든 유효한 객체 헤더를 새로 할당된 헤더 블록에 저장한다.

(6) (3)에서 복사된 블록을 지운다.

4. 평 가

본 논문에서 제안한 안전 삭제 기법의 성능을 평가하기 위해 제안 기법에서 일정 개수의 파일이 생성되고, 또 그 중 일부가 변경되었을 때, 파일의 생성과 변경으로 인해 발생하는 블록 삭제 연산의 횟수를 계산하는 시뮬레이션 프로그램을 작성하여 실행하였다.

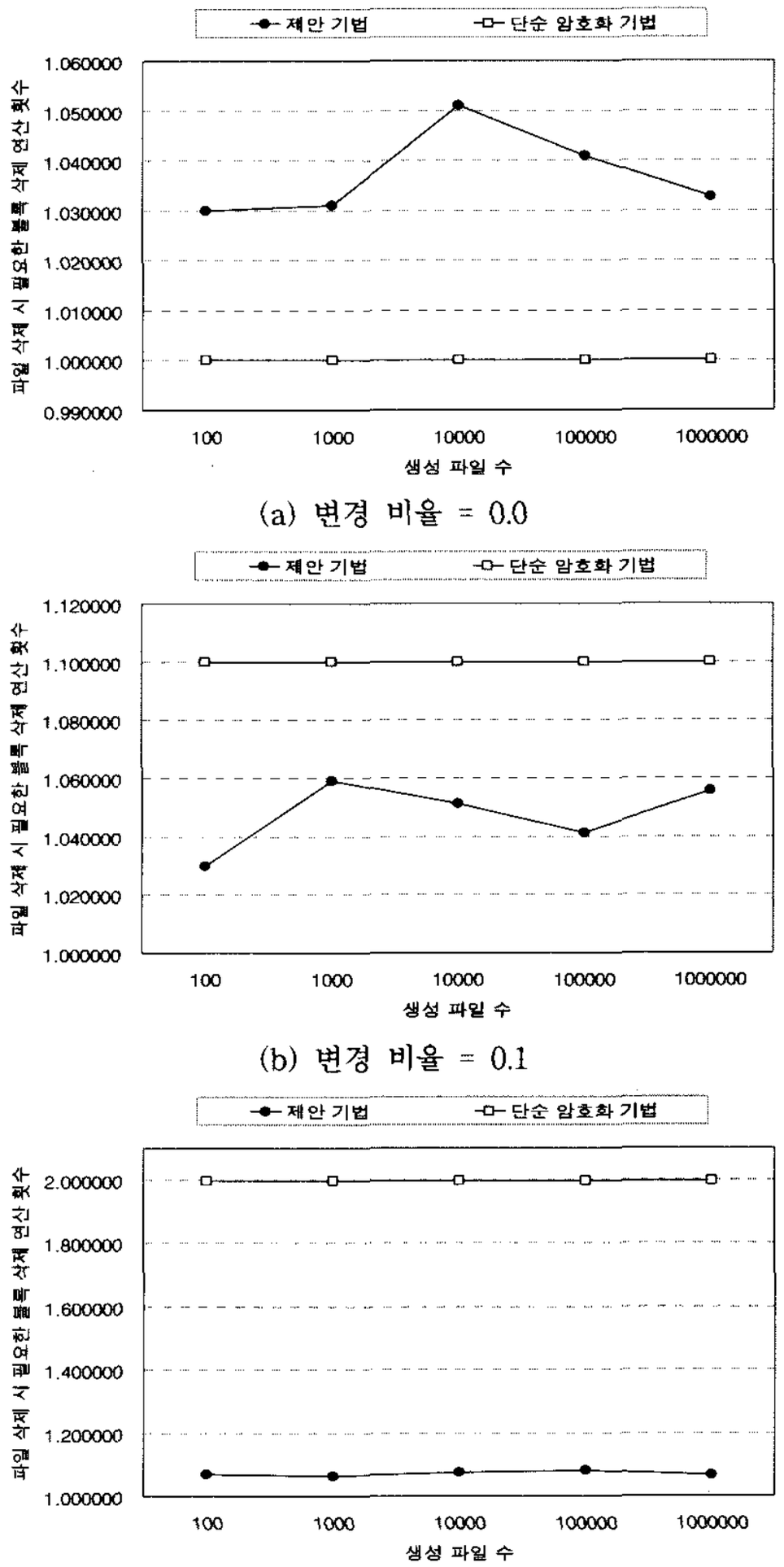

그림 2 생성 파일 수와 파일 변경 비율의 다양한 조합에 대해 계산된 파일 삭제 시 필요한 블록 삭제 횟수

그 후, 제안 기법을 단순 암호화 기법과 비교하였는데, 이 기법은 파일 데이타를 난수로 생성된 키로 암호화 하고 그 키가 포함된 객체 헤더를 비어 있는 아무 공간에나 저장한다. 이 기법으로 저장된 파일에 대해 안전 삭제를 수행하기 위해서는 해당 파일의 키를 포함한 모든 객체 헤더를 찾아 삭제 연산을 해 주어야만 한다. 각각의 파일에 대해 적용된 파일 변경 횟수의 평균을 변경 비율 m이라 하면, 변경 비율 m의 정의에 의해 하

나의 파일은 평균적으로 m + 1개의 객체 헤더를 파일 시스템 안에 가지게 된다. 전체 파일 시스템에 n개의 블록이 존재한다고 가정할 때, 단순 암호화 기법에서 안전 삭제를 위해 지워야 하는 블록의 수는 우연히 같은 블록에 객체 헤더들이 저장될 확률을 고려하여 통계학에서 잘 알려진 "bins & balls" 문제를 적용하면 다음과 같이 계산된다[13].

$$E = n(1-(1-\frac{1}{n})^{m+1})$$

그림 2는 생성 파일 수와 파일 변경 비율의 다양한 조합에 대해 파일의 생성과 변경으로 일어나는 블록 삭제를 감안해 계산한 파일 삭제 시 필요한 블록 삭제 연산 횟수를 보여준다. 단순 암호화 기법에서 n 값은 256으로 가정하였고, 제안 기법의 경우 실험을 10회 수행하여 나온 평균값을 적용하였다. 단순 암호화 기법의 경우, 위의 공식에 나타난 파일 삭제 시 필요한 삭제 연산 횟수 E를 나타낸 것이고, 제안 기법은 파일 삭제 시의 블록 삭제 연산 오버헤드의 일부를 파일 생성과 변경 시로 분배하는데, 이것까지 고려하여 계산된 것이다.

그림 2에서 보다시피 변경 비율이 0인 경우를 제외하고는 제안 기법이 단순 암호화 기법보다 더 좋은 성능을 보임을 알 수 있다. 이러한 성능 차이는 변경 비율이 커질수록 더 커진다. 그러므로 제안 기법은 파일이 생성된 후 바로 삭제되는 경우가 많은 실행 환경보다, 몇 번의 변경이 일어난 뒤 삭제되는 경우가 많은 실행 환경에서 더욱 효율적임을 알 수 있다.

5. 결 론

본 논문에서는 NAND 플래시 파일 시스템을 위한 안전 삭제 기법을 제안하고 이를 가장 대표적인 NAND 플래시 파일 시스템인 YAFFS에 적용하였다.

제안 기법은 파일의 데이타를 암호화된 상태로 저장하고, 파일 암호화 시 사용된 키를 모두 삭제함으로써 파일 복구를 불가능하게 한다. 플래시 메모리의 물리적 특성으로 인해 플래시 전용 파일 시스템은 대부분 로그 기반 구조 파일 시스템이다. 로그 기반 구조 파일 시스템에서는 하나의 특정 파일에 대해 많은 키가 파일 시스템 상에 존재할 수 있으므로 제안 기법은 특정 파일을 암호화하는데 사용된 모든 키들이 같은 블록에 저장되도록 하여, 단 한번의 블록 삭제 연산으로 파일 복구가 불가능하도록 하였다.

시뮬레이션 결과는 제안 기법으로 인해 파일의 생성 및 변경 시 발생하는 블록 삭제를 감안하더라도 파일 삭제 시 발생하는 블록 삭제 횟수가 단순 암호화 기법의 경우보다 더 작다는 것을 보여준다. 본 논문에서는 제안 기법을 YAFFS에만 적용하였지만, 다른 NAND 플래시 전용 파일 시스템에도 쉽게 적용 가능하다.

참 고 문 헌

[1] Nikolai Joukov and Erez Zadok. Adding Secure Deletion to Your Favorite File System. In Proceedings of the third international IEEE Security in Storage Workshop (SISW 2005), San Fransisco, CA, December 2005. IEEE Computer Society.

[2] I. Mayergoyz, C.Seprico, C.Krafft, and C. Tse. Magnetic Imaging on a Spin-Stand. Journal of Applied Physics, 87(9):6824-6826, May 2000.

[3] P. Gutmann. Secure Deletion of Data from Magnetic and Solid-State Memory. In Proceedings of the Sixth USENIX UNIX Security Symposium, pages 77-90, San Jose, CA, July 1996. USENIX Association.

[4] S. Bauer and N. B. Priyantha. Secure Data Deletion for Linux File Systems. In Proceedings of the 10th USENIX Security Symposium, pages 153-164, Washington, DC, August 2001. USENIX Association.

[5] F. Douglis, R. Caceres, F. Kaashoek, K. Li, B. Marsh, and J. A. Tauber. Storage alternatives for mobile computers. In Proceedings of the 1st Symposium on Operating Systems Design and Implementation (OSDI), pages 25-37. 1994.

[6] Mendel Rosenblum and John K. Ousterhout. The Design and Implementation of a Log-Structured File System. ACM Transactions on Computer Systems, 10(1), 1992.

[7] Aleph One Ltd, Embedded Debian. Yaffs: A NAND-Flash Filesystem. http://www.aleph1.co.uk/ yaffs/.

[8] Nikolai Joukov, Harry Papaxenopoulos, Erez Zadok. Secure Deletion Myths, Issues, and Solutions. In Proceedings of the 2nd ACM Workshop on Storage Security and Survivability, Alexandria, Virginia, October 2006.

[9] D. Boneh and R. Lipton. A Revocable Backup System. In Proceedings of the 6th USENIX UNIX Security Symposium, pages 91-96, San Jose, CA, July 1996, USENIX Association.

[10] R. Gomez, A. Adly, I. Mayergoyz, and E. Burke. Magnetic Force Scanning Tunnelling Microscope Imaging of Overwritten Data. IEEE Transactions on Magnetics, 28(5):3141.3143, September 1992.

[11] T. Grance, M. Stevens, and M. Myers. Guide to Selecting Information Technology Security Products, chapter 5.9: Media Sanitizing. National Institute of Standards and Technology (NIST), October 2003.

[12] Defense Security Service. National Industrial Security Program Operating Manual (NISPOM), chapter 8: Automated Information System Security. U.S. Government Printing Office, January 1995.

[13] S.M. Ross, Probability Models for Computer Science, Harcourt/Academic Press, 2002.