Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.24, NO.5, Oct. 2014 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2014.24.5.885

XFS 파일 시스템 내의 삭제된 파일 복구 기법 연구*

안 재 형,† 박 정 흠, 이 상 진‡
고려대학교 정보보호대학원

The Research on the Recovery Techniques of Deleted Files in the XFS Filesystem*

Jae-hyoung Ahn,† Jung-heum Park, Sang-jin Lee‡
Center for Information Security Technologies, Korea University

요 약

일반적으로 사용하는 컴퓨터의 저장소는 예상치 못한 오류나 사고로 인해 파일이 삭제되는 일이 발생할 수 있다. 또한 악의적인 목적을 가진 누군가가 안티 포렌식을 목적으로 직접 데이터를 삭제하기도 한다. 이렇게 삭제된 파일이 범죄와 연관된 증거이거나 업무상 중요한 문서인 경우 복구를 위한 대책이 있어야 한다. NTFS, FAT, EXT 와 같은 파일 시스템은 삭제된 파일을 복구하기 위한 기법이 활발히 연구되었고 많은 도구가 개발 되었다. 그러나 최근 NAS 및 CCTV 저장소에 적용된 XFS 파일 시스템을 대상으로는 삭제된 파일을 복구하기 위해 제안된 연구개발 결과가 거의 없으며 현재까지 개발된 도구들은 전통적인 시그니처 탐지 기반 카빙 기법에 의지하고 있어서 복구 성공률이 저조한 상황이기 때문에 이에 대한 연구가 필요하다.

따라서, 본 논문에서는 XFS 파일 시스템에서의 삭제된 파일을 복구하기 위해 메타데이터 기반 복구 방법과 시그니처 탐지 기반 복구 방법을 제안하며 실제 환경에서 실험하여 검증하였다.

ABSTRACT

The files in computer storages can be deleted due to unexpected failures or accidents. Some malicious users often delete data by himself for anti-forensics. If deleted files are associated with crimes or important documents in business, they should be recovered and the recovery tool is necessary. The recovery methods and tools for some filesystems such as NTFS, FAT, and EXT have been developed actively. However, there has not been any researches for recovering deleted files in XFS filesystem applied to NAS or CCTV. In addition, since the current related tools are based on the traditional signature detection methods, they have low recovery rates.

Therefore, this paper suggests the recovery methods for deleted files based on metadata and signature detection in XFS filesystem, and verifies the results by conducting experiment in real environment.

Keywords: XFS, file recover, data recover, XFS filesystem

I. 서 론

XFS(eXtended File System)는 1993년에 개발된 파일 시스템으로서 IRIX 5.3에 처음 적용되었고 2000년에 GNU GPL(General Public License) 하에 릴리즈 되어 리눅스 커널에 포함되었다[1].

XFS 파일 시스템은 데이터의 변경 이력을 저장하

접수일(2014년 8월 22일), 수정일(2014년 9월 12일), 게재확정일(2014년 9월 15일)
* 이 논문은 2014년도 정부(미래창조과학부)의 재원으로 한국연구재단-공공복지안전사업의 지원을 받아 수행된 연구임(2012M3A2A1051106)
† 주저자, ajh0418@korea.ac.kr
‡ 교신저자, sangjin@korea.ac.kr(Corresponding author)

는 저널링 기능을 포함하고, B+tree를 이용하여 데이터를 관리함으로써 파일 입·출력에 대한 속도를 개선했다. 특히, 다른 파일 시스템과 비교하여 가장 두드러지는 점은 높은 확장성이다. 최대 1 EB(Exabyte) 크기의 볼륨을 지원할 수 있는 EXT4 보다 8배 큰 볼륨을 다룰 수 있고, OS가 32비트일 경우 16 TB(Terabyte) 크기의 파일과 파티션을 다룰 수 있다[2].

대용량 데이터를 취급하기 위해 개발된 NAS(Network Attached Storage)를 제조하고 판매하는 'BUFFALO 사'는 자사의 모든 NAS 제품에 XFS 파일 시스템을 채택하였다. 또한 '삼성 테크윈 사'의 CCTV 제품에도 적용되어 판매하고 있다.

삭제된 데이터가 중요한 의미를 가진다면 반드시 복구해야 한다. 삭제된 데이터가 저장소 내에 남아 있음에도 불구하고 복구에 실패한다면 디지털 포렌식 수사 시 결정적인 증거를 놓칠 수 있고, 개인 및 집단에게 의미 있는 파일을 잃어버릴 수 있다. 이를 대비하여 많은 데이터 복구 도구의 개발과 연구가 이루어졌으나 대부분이 NTFS(New Technology File System), FAT(File Allocation Table), EXT(Extended), HFS(Hierarchical File System) 파일 시스템을 대상으로 하고 있다. 현재까지는 XFS 파일 시스템을 지원하는 파일 복구 도구가 거의 없으며, 관련 연구도 부족한 실정이다.

본 논문에서는 XFS 파일 시스템에서 삭제된 파일을 복구하기 위해 필요한 구조 분석 및 파일의 상태 변화에 따른 실험 결과를 정리하였다. 그리고 삭제된 파일의 복구 기법을 제시하고 이를 구현하였다.

2장에서는 XFS 파일 시스템의 파일 복구와 관련된 연구를 소개하고, 3장에서 XFS 파일 시스템의 구조를 상세하게 설명한다. 4장에서는 파일의 실제 데이터를 추적하기 위한 방법을 분석된 파일 시스템 기반으로 설명하고, 5장에서는 파일 삭제 메커니즘을 분석하기 위해 실험한 자료를 정리하였으며, 6장에서 삭제 메커니즘을 분석하여 삭제된 파일의 복구 방법을 제안한다. 7장에서는 제안한 복구 방법을 구현한 실험결과를 정리하였다. 그리고 7장에서 결론 및 향후 연구 방향을 제시하였다.

II. 관련 연구

디지털 장치는 사용자의 실수나 오작동, 고의적인 데이터 파괴로 인해 데이터를 잃는 경우가 발생할 수 있으며, 이를 대비하여 데이터 복구에 대한 연구와 기술이 꾸준히 개발되어 왔다[3].

전통적인 데이터 복구는 파일 시스템 구조에 기반하여 파일의 메타데이터를 통해 복구하는 방식이었다. 이에 따라 NTFS, FAT, EXT와 같은 파일 시스템에 대한 분석이 활발히 이루어졌고[4], 복구 알고리즘이 개발되었다. 이 방식은 삭제된 파일의 메타데이터를 참조하기 때문에 복구율이 높고 속도가 빠르며 삭제된 파일의 거의 모든 메타데이터를 복구할 수 있다는 장점이 있지만 메타데이터가 덮어 쓰여질 경우에는 복구가 불가능하다.

이를 보완하기 위해 시그니처 탐지를 기반으로 삭제된 데이터를 복구하는 카빙 기법이 개발되었다. 카빙 기법은 메타데이터가 없어도 복구가 가능하지만, 파일의 종류마다 구조가 다르고 기록되는 데이터의 형태가 다르기 때문에 복구를 위한 고유한 알고리즘이 필요하다. 또한, 파일의 데이터가 연속적이지 않은 블록에 할당되는 경우 복구가 어려우며 많은 시간이 소요된다는 단점이 있다.

최근에는 전통적인 카빙 방식의 단점을 보완하기 위해 연속적이지 않은 블록에 할당된 파일을 복구하기 위한 기법의 연구[5][6]가 있었지만, 단점을 완전히 해결하지 못하고 있다.

대부분의 상용 도구는 FAT, NTFS와 같은 파일 시스템을 대상으로 복구하는 경우 복구율이 높고 속도가 빠르다. 하지만 대부분의 도구들이 XFS 파일 시스템을 지원하지 않고 있으며, XFS 파일 시스템을 지원하는 복구 도구 중 하나인 UFS Exploerer[7]의 성능을 실험한 논문[8]에서는 해당 도구가 3 GB(Gigabyte) 이상 크기의 단일 파일에 대해서 복구에 성공했지만, 1 KB(Kilobyte) 이하의 단일 파일의 복구에 실패했으며 이에 따라 XFS 파일 시스템을 대상으로 삭제된 파일을 복구할 수 있는 새로운 도구의 개발이 필요함을 언급하고 있다. 하지만 해당 논문이 발표된 이후 XFS 파일 시스템을 대상으로 삭제된 파일을 복구하기 위한 기술의 연구가 여전히 미비한 상황이며 관련 연구가 필요한 실정이다.

III. XFS 파일 시스템 구조

XFS 파일 시스템의 전체적인 구조는 Fig.1.과 같으며, 하나의 파티션은 할당 그룹(Allocatino Group)이라는 단위로 나눠진다. 각각의 할당 그룹은

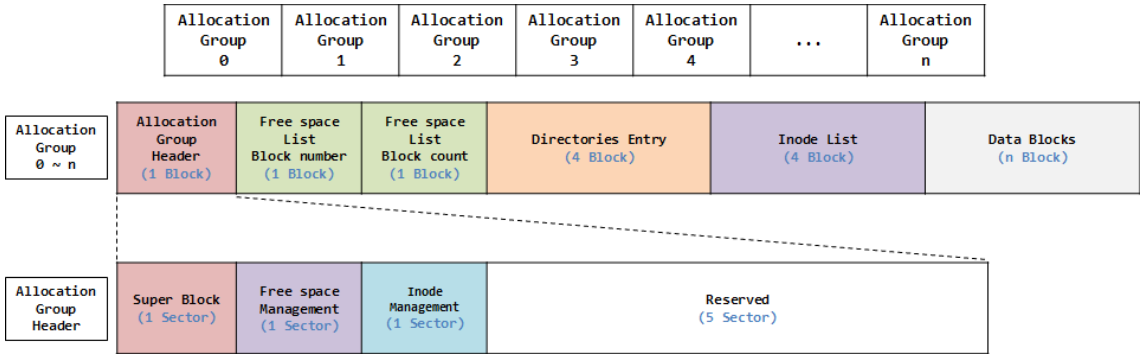

Fig. 1. XFS filesystem structure

독립적으로 존재하며 병렬적으로 처리된다. 파일 시스템을 생성할 때 할당 그룹의 크기와 수를 지정할 수 있는데 이를 지정하지 않은 경우에는 8개의 할당 그룹으로 생성된다[9].

각 할당 그룹은 Super Block(SB)으로 시작하며, SB에는 해당 할당 그룹의 블록 당 섹터의 개수, 할당 그룹의 크기, 루트 아이노드의 위치 등과 같은 기본적인 정보가 기록되어있다.

할당 그룹에는 미할당 영역 관리를 위한 Allocation Group Free space Management(AGF) 블록이 있다. Fig.2.는 XFS 파일 시스템을 디버깅하기 위해 개발된 도구인 "XFS_DB"[10]를 이용하여 AGF의 정보를 출력한 모습이다. AGF에는 미할당 영역에 대한 목록 정보인 Allocation Group Free space List(AGFL) 블록에 대한 정보가 기록되어 있다.

AGFL 블록은 다음과 같은 구조를 가지는 레코드가 연속해서 나열된다.

```
xfs_db> agf 0
xfs_db> p
magicnum = 0x58414746
versionnum = 1
seqno = 0
length = 4521884
bnoroot = 157
cntroot = 1050361
bnolevel = 2
cntlevel = 2
flfirst = 37
fllast = 42
flcount = 6
freeblks = 16119
longest = 64
btreeblks = 4
```

Fig. 2. Allocation Group Free space Management Block

```
typedef struct xfs_alloc_rec {
    BYTE4                ar_startblock;
    BYTE4                ar_blockcount;
} xfs_alloc_rec_t, xfs_alloc_key_t;
```

AGFL은 미할당 영역에 대한 정보가 담긴 구조체를 2가지 방식으로 관리한다. 첫 번째는 미할당 블록의 오프셋을 오름차순으로 정렬하는 방법이며, 정렬된 블록의 위치는 각 할당 그룹 AGF 블록의 bnoroot에 기록되어 있다.(Fig.2. 참조) bnoroot에 기록된 블록으로 이동하면 Fig.3.과 같이 미할당 블록에 대한 정보를 확인할 수 있다.

두 번째는 미할당 블록의 연속적인 개수를 기준으로 오름차순으로 정렬하는 방식이며, 해당 블록의 위치는 AGF의 cntroot에 기록되어 있다.(Fig.2. 참조) cntroot 블록으로 이동하면 Fig.4.와 같이 미할

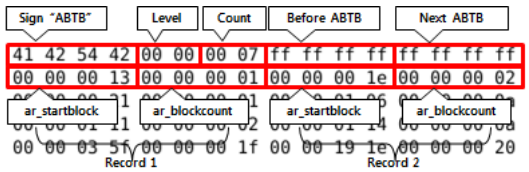

Fig. 3. AGFL - physical address in ascending order

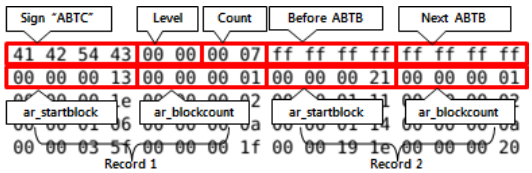

Fig. 4. AGFL - free block counter in ascending order

당 블록에 대한 정보를 확인할 수 있다

XFS 파일 시스템은 파일의 메타데이터를 아이노드에 저장한다. 아이노드는 기본 256 byte이며 96 byte의 "아이노드 코어"와 160 byte의 "데이터 포크" 영역으로 구성된다. MAC(Modify Access Create) Time, 크기, 포맷과 같은 메타데이터는 "아이노드 코어"에 기록되고, 하위 경로에 존재하는 파일의 목록, extent 또는 비트맵 블록 주소에 대한 메타데이터는 "데이터 포크" 영역에 기록한다. "XFS_DB"를 이용하여 초기 상태의 아이노드를 출력한 모습은 Fig.5.와 같다.

아이노드는 디렉터리와 파일에 대한 아이노드로 분류할 수 있으며, 파일에 대한 아이노드는 데이터가 조각난 개수에 따라 2가지 포맷으로 구분할 수 있다. 디렉터리에 대한 아이노드는 "local" 포맷으로 분류하며 아이노드의 "core.format"의 값이 1이다. "local" 포맷인 경우 데이터 포크 영역에 파일 목록이 기록되며 구조는 Table 1.과 같다.

파일에 대한 아이노드는 데이터가 조각난 개수에 따라 "extents"와 "btree" 포맷으로 나눈다. 아이노드의 포맷이 "extents"인 경우, "core.format"의 값은 2이며 데이터 포크 영역에 1~9개의 extent를 기록한다.

```
core.magic = 0
core.mode = 0
core.version = 0
core.format = 0 (dev)
core.uid = 0
core.gid = 0
core.flushiter = 0
core.atime.sec = Thu Jan  1 08:00:00 1970
core.atime.nsec = 000000000
core.mtime.sec = Thu Jan  1 08:00:00 1970
core.mtime.nsec = 000000000
core.ctime.sec = Thu Jan  1 08:00:00 1970
core.ctime.nsec = 000000000
core.size = 0
core.nblocks = 0
core.extsize = 0
core.nextents = 0
core.naextents = 0
core.forkoff = 0
```

Fig. 5. Inode in initial state

Table 1. File name structure

Data name	Details	Size (byte)
namelen	Length of filename	1
offset	Index Offset	2
name	filename	namelen
inumber	Inode number	4

XFS 파일 시스템에서는 파일의 실제 데이터 위치를 128비트 크기의 extent를 이용하여 관리한다. extent를 통해 데이터의 물리적 주소, 논리적 주소, 할당된 블록의 개수를 알 수 있으며, 구조는 Fig.6.과 같다.

파일의 실제 데이터가 10개 이상 조각나는 경우 extent의 목록으로 이루어진 비트맵 블록을 생성한다. 비트맵 블록을 사용하게 되는 경우 "core.format" 값은 3이 되고, "btree" 포맷이라고 부른다. 이 경우 데이터 포크 영역에 비트맵 블록의 물리적 주소를 저장하며, 이때 하나의 비트맵 블록은 최대 254개의 extent를 저장할 수 있다. 만약 파일이 255개 이상의 조각이 나서 다중 비트맵을 사용한다면 Fig.7.과 같이 비트맵 블록 간에 더블 링크드 리스트 구조로 연결된다. 참고로 Fig.8.은 비트맵 블록이 생성되는 구조를 추상적으로 표현한 그림이다.

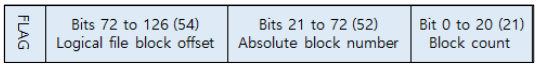

Fig. 6. Extent structure

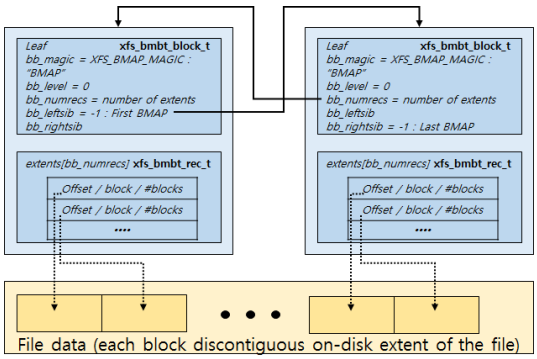

Fig. 7. Multiple bitmap block linked by double linked list

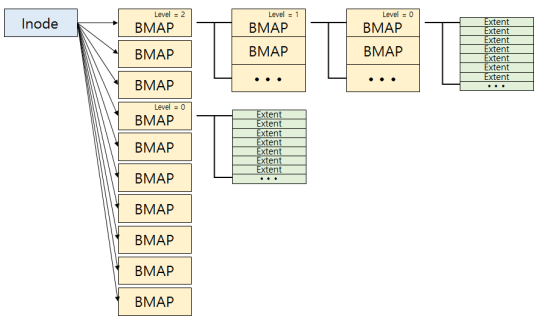

Fig. 8. Multiple Bitmap block structure

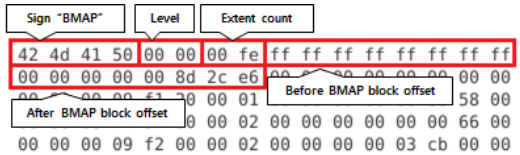

Fig. 9. Bitmap block

아이노드의 포맷이 "btree"인 경우 데이터 포크 영역에는 기본적으로 10개의 비트맵 블록의 주소를 기록할 수 있으며, 11개 이상의 비트맵이 필요할 경우 깊이(Level) 값을 증가시켜 첫 번째 비트맵 블록에 하위 비트맵 블록의 주소를 기록한다.(Fig.8. 참조)

비트맵 블록은 Fig.9.와 같이 "BMAP" 시그니처가 있으며 비트맵 블록의 깊이(2 byte) 그리고 비트맵 블록에 존재하는 extent의 개수(2 byte)로 구성된다. 또한 앞 비트맵 블록의 주소와 다음 비트맵 블록의 주소를 8 byte 크기로 기록하며, 이전 혹은 다음 비트맵 블록이 없을 경우 -1이 기록된다.

참고로 XFS 파일 시스템은 sparse 파일을 지원한다. 파일 내부의 특정 블록이 0x00으로 이루어져 있다면 실제 공간에 불필요하게 할당할 필요 없이 가상으로 공간을 할당하는 방법이다. 이와 같이 sparse 파일을 지원할 경우 디스크 공간을 효율적으로 관리할 수 있지만, extent의 개수가 증가한다.

IV. 데이터 추적 메커니즘

Fig.10.은 데이터의 실제 위치 정보를 찾는 방법을 표현한 다이어그램이다.

Fig. 11. Inode structure

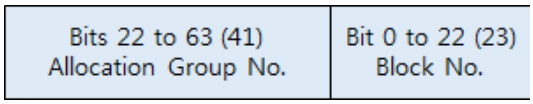

Fig. 12. Bitmap block pointer structure

먼저 할당 그룹의 Super block에 기록된 값을 통해 루트 아이노드(아이노드 목록)의 시작 위치를 알 수 있다. XFS 파일 시스템에서는 아이노드의 번호를 통해 해당 아이노드가 실제로 위치한 물리적 주소를 알아낼 수 있으며 구조는 Fig.11.과 같다.

4 byte 크기의 아이노드 번호 값 중 상위 5개 비트는 할당 그룹의 번호를 기록하며 나머지 27개 비트에 할당 그룹 내에서의 상대적인 블록 번호를 기록한다.

아이노드 번호를 통해 목표로 하는 파일의 아이노드를 찾은 다음, 해당 아이노드의 데이터 포크 영역에 기록된 extent를 분석하여 실제 데이터가 위치한 물리적 주소를 알 수 있다.

아이노드의 "core.format" 값이 "btree"인 경우 데이터 포크 영역에 비트맵 블록의 위치를 기록한다. 8 byte 크기의 비트맵 블록의 위치 값은 Fig.12와 같이 구성된다.

하위 23개 비트는 할당 그룹 내의 상대적인 블록 번호이며 나머지 비트는 할당 그룹의 번호를 낸다. 비

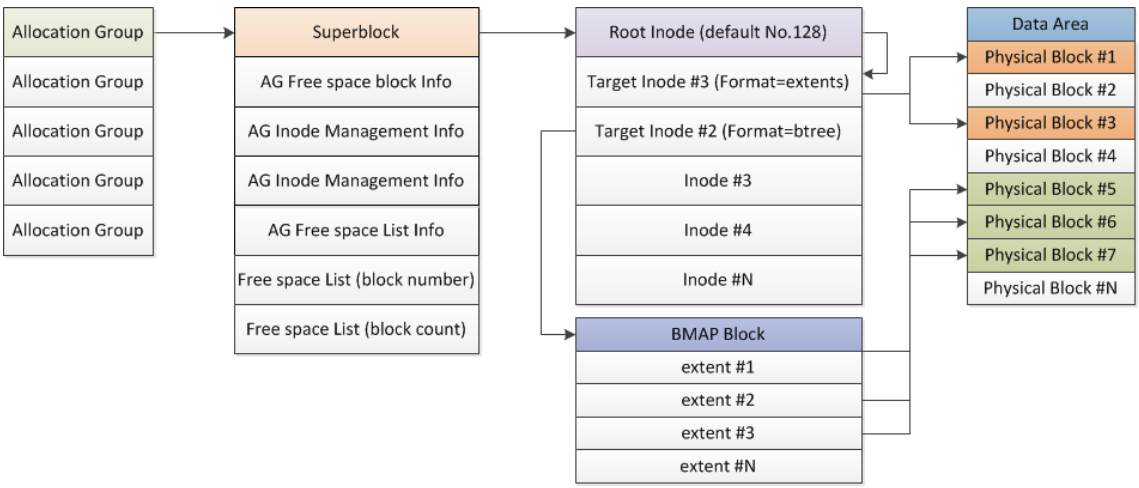

Fig. 10. The diagram for tracking data

트맵 블록으로 이동하면 extent 목록이 나열되어 있으며 데이터의 물리적 주소를 알 수 있다.

V. 파일 삭제 메커니즘

XFS 파일 시스템에서 파일을 삭제할 시에는 해당 파일의 메타데이터가 변경된다. 앞에서 설명한 것과 같이 아이노드는 3가지 포맷이 존재할 수 있기 때문에 이 장에서는 각 포맷 유형에 따라서 변경되는 부분을 상세하게 설명한다.

5.1 local 포맷 아이노드의 변화

XFS 파일 시스템은 Fig.13.과 같이 디렉터리에 포함된 디렉터리 및 파일의 엔트리를 해당 디렉터리 아이노드의 데이터 포크 영역에 생성 순서로 저장한다. 데이터 포크 영역의 첫 4 byte는 -1 (0xffffffff)로 기록되어 있다.

데이터 포크 영역의 첫 번째 byte에는 파일의 개수를 기록한다. Fig.13.의 디렉터리 엔트리에서 "DIR3"을 삭제할 경우 Fig.14.와 같이 파일의 개수를 기록하는 값이 1 감소한다.

Fig.13.에서 "DIR2"를 삭제한 결과는 Fig.15.와

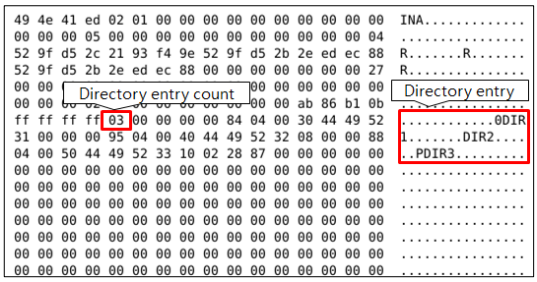

Fig. 13. Change of directory entry after deleting "DIR2"

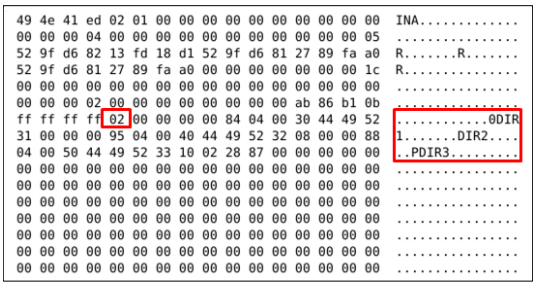

Fig. 14. Change of directory entry after deleting "DIR3"

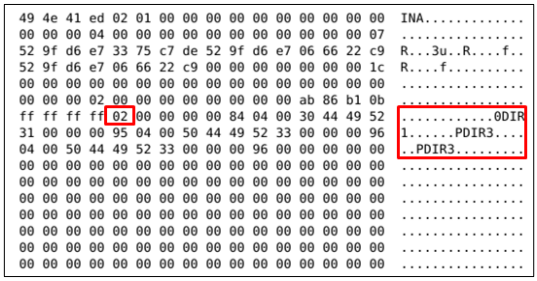

Fig. 15. Change of directory entry after deleting "DIR2"

같다.

"DIR2"를 삭제할 경우 파일의 개수를 기록하는 값이 1 감소하며, 디렉터리 엔트리가 Fig.15.와 같이 정렬된다.

5.2 extents 포맷 아이노드의 변화

XFS 파일 시스템에서는 파일이 삭제될 경우 해당 아이노드의 권한 값 및 크기에 관련된 메타데이터가 0으로 초기화 된다.

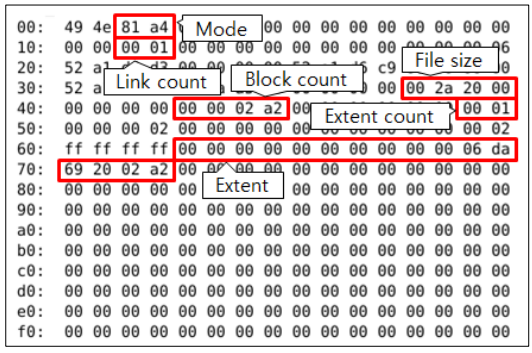

Fig. 16. Inode before deleting a file - extent

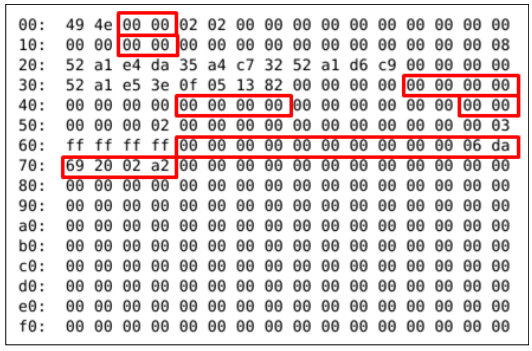

Fig. 17. Inode after deleting a file - extent

Fig.16.과 Fig.17.은 파일 삭제 전과 후의 extents 포맷 아이노드에 대한 변화이다. 데이터 포크 영역에 기록된 extent 정보는 변경되지 않는다.

5.3 btree 포맷 아이노드의 변화

Fig.18.과 Fig.19.는 btree 포맷 파일의 삭제 전과 후의 아이노드 변화이다.

데이터 포크 영역에 기록된 비트맵 블록의 위치 정보가 변경되지 않는다.

데이터 포크 영역에 기록된 비트맵 블록의 물리적 주소로 이동하면 Fig.20.과 같이 비트맵 블록이 존재하며, 해당 비트맵 블록에 유효한 extent 개수 및 extent 목록이 기록되어 있다.

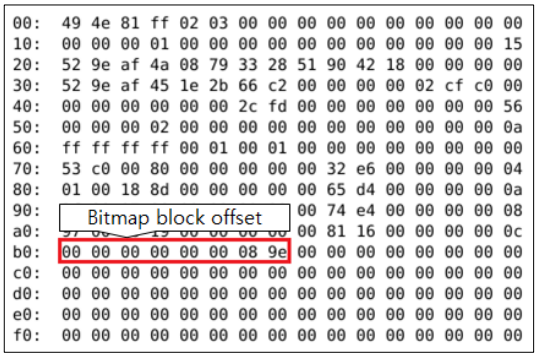

Fig. 18. Inode before deleting a file - btree

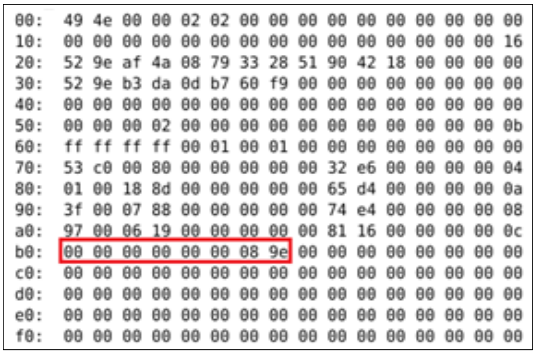

Fig. 19. Inode after deleting a file - btree

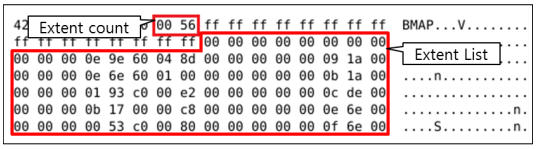

Fig. 20. Bitmap block before deleting a file

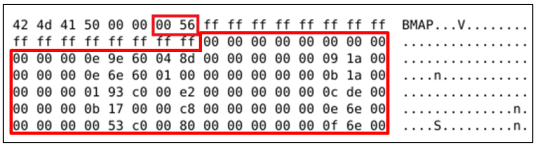

Fig. 21. Bitmap block after deleting a file

Fig. 21.은 파일 삭제 후 비트맵 블록의 상태이다. extent의 개수와 extent 목록이 변경되지 않는다.

VI. 복구 방법

이 장에서는 앞에서 설명한 XFS 파일 시스템에서의 파일 삭제 메커니즘에 기반한 복구 방법을 제안한다.

Fig.22.는 본 논문에서 제시하는 XFS 파일 시스템에서 삭제된 파일 복구의 순서도이다. 삭제된 파일을 복구하는 방법은 2가지로, 첫 번째는 삭제된 파일의 아이노드를 찾아 extent 또는 비트맵 블록의 주소를 확인하여 데이터를 복구하는 방법이다. 두 번째는 비트맵 블록의 시그니처를 탐색하여 복원하는 방법이다. 그 외에 삭제된 파일 복구에 대해서는 기존의 전통적인 시그니처 탐지 기반 복구 방법을[9] 사용하며 본 논문에서는 다루지 않는다.

6.1 메타데이터 기반 복구 방법

메타데이터 기반 복구 방법의 순서도는 Fig.23.과 같다.

먼저 각 할당 그룹에 있는 아이노드를 순차적으로 탐색한다. 아이노드의 목록은 각 할당 그룹의 시작 위치에서 1024 오프셋에 있다. 아이노드를 탐색하면서 버전, 포맷, 생성·수정·접근 시간에 0이 아닌 값이 있고, 그 외 변수의 값이 0으로 초기화 되어 있다면 삭제된 파일의 아이노드라고 판단하여 데이터 포크 영역에 있는 extent 및 비트맵 블록의 주소 정보를 확인한다. 삭제된 파일의 데이터 영역이 미할당 영역인지 확인 후 복구를 진행한다. 삭제된 파일의 데이터 블록이 미할당 영역으로 처리되어 있지 않다면 새로운 데이터로 덮어 쓰여졌다고 판단한다.

이 방법은 메타데이터가 기록되는 아이노드를 기반으로 복구하는 방식이기 때문에 아이노드의 메타데이터가 새로운 데이터로 덮어 쓰여졌다면 복구가 불가능하다. 또한 XFS 파일 시스템은 파일의 이름을 데이터 포크 영역에 디렉터리 엔트리 형태로 관리하

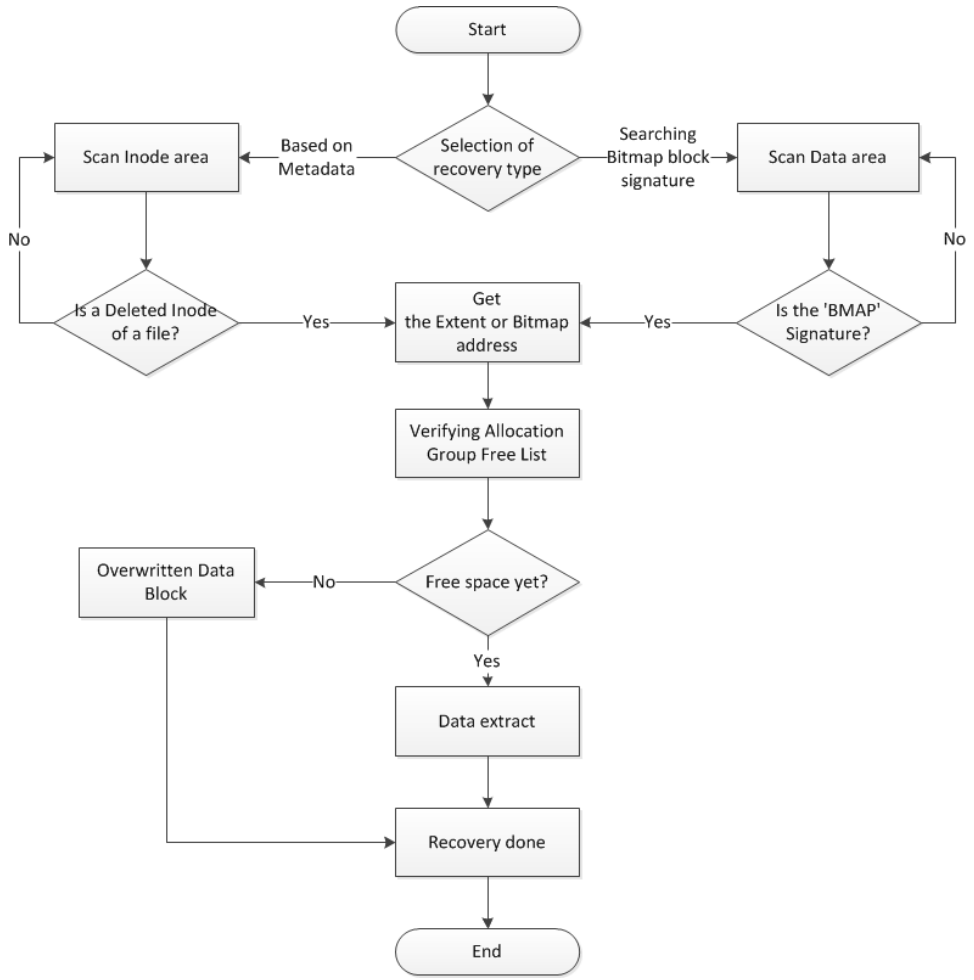

Fig. 22. Flow chart of deleted files recovery

며, 파일 삭제 시 디렉터리 엔트리를 정렬하기 때문에 경우에 따라 파일의 이름을 복구하지 못할 수도 있다.(Fig.13.과 Fig.15. 참조)

Fig.24.는 9개의 extent를 사용하는 파일이 삭제된 뒤, 1개의 extent를 사용하는 파일이 동일한 아이노드에 할당된 경우의 메타데이터 및 데이터 포크의 상태를 나타낸다.

Fig.25.는 Fig.24.의 파일이 삭제된 후 extent의 목록을 파싱한 결과이다. 파일이 삭제될 경우 extent의 개수가 0으로 초기화되기 때문에 삭제되기 전 파일의 extent가 몇 개 인지 알 수 없다. 하지만 첫 번째 extent를 분석해 보면 logical 블록 0번부터 122개 블록 크기의 데이터가 연속적으로 할당되어 있으므로 두 번째 extent의 logical 블록의 값은 최소 123이 되어야 한다. 이러한 특성에 따라 삭제된 파일의 extent를 구별할 수 있다. 하지만 XFS 파일 시스템은 sparse 파일을 지원하기 때문에 두 번째 extent의 값이 122 이상인 경우에는 추가적인 검증이 필요하며 이에 대한 연구가 더 필요하다.

또한 파일 삭제 시 아이노드의 포맷 값이 2로(extent 포맷) 초기화되기 때문에 삭제 전 아이노드의 속성을 알 수 없다. 따라서 데이터 포크 영역의 1번째 비트맵 블록의 주소를 참조하여 오프셋을 이동한 후 비트맵 블록이 있는지 확인하는 과정을 추가적으로 거쳐야 한다.

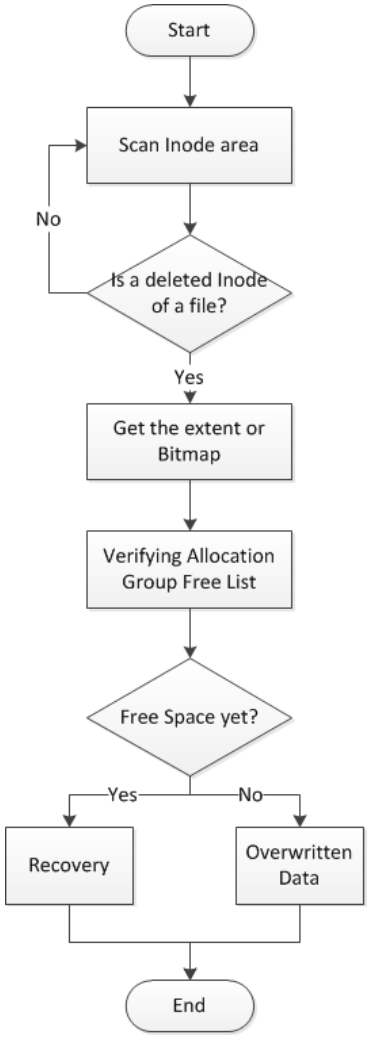

Fig. 23. Flow chart of recovery based on metadata

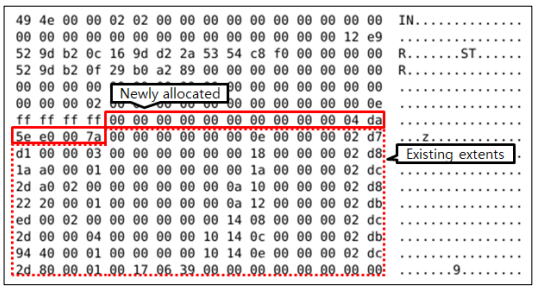

Fig. 24. The state change of inode after a new file is reallocated

```
AG No. = 0       Inode No. = 23
AG No. = 0       Inode No. = 24
AG No. = 0       Inode No. = 26
Found a deleted inode
This file used the extents format!!
26th Inode Recovery Start!!
No.1 Extent value : [0, 2544375, 122, 0]
No.2 Extent value : [7, 1490568, 3, 0]
No.3 Extent value : [12, 1491157, 1, 0]
No.4 Extent value : [13, 1499501, 512, 0]
No.5 Extent value : [1288, 1491217, 1, 0]
No.6 Extent value : [1289, 1498984, 512, 0]
No.7 Extent value : [2564, 1499496, 4, 0]
No.8 Extent value : [526854, 1498274, 1, 0]
No.9 Extent value : [526855, 1499500, 1, 0]
Recovery Done!!
```

Fig. 25. Extents in data fork overwritten by a new value

6.2 시그니처 탐지 기반 복구 방법

아이노드의 메타데이터가 새로운 데이터로 덮어 쓰여졌을 경우 시그니처 탐지 기반 복구 방법을 사용하여 데이터를 복구한다. 비트맵 블록은 "BMAP"라는 고유한 시그니처를 가지기 때문에 전통적인 시그니처 탐지 기반인 카빙 기법에 적용하여 파일을 복구할 수 있다. 비트맵 블록 시그니처 탐지 기반 복구 순서도는 Fig.26.과 같다.

먼저 데이터 영역의 블록을 스캔하면서 비트맵 블록의 "BMAP" 시그니처를 탐지한다. 비트맵 블록의 시그니처를 찾았다면, 이전 비트맵 블록의 주소 값이 -1 (0xffffffff)인지 확인한다. 데이터의 위치는 순차적으로 기록되기 때문에 첫 번째 비트맵 블록을 먼저 찾아야 한다. 이전 비트맵 블록의 주소 값이 -1이면 첫 번째 비트맵 블록이므로 extent를 확인하여 데이터 위치를 추적해 나간다. 해당 비트맵 블록의 extent를 모두 추적하였다면 다음 비트맵 블록의 주소 값을 확인한다. 다음 비트맵 블록의 주소 값이 -1이면, 마지막 비트맵 블록이므로 파일의 복구를 완료하고, 시그니처 탐지를 계속해서 수행한다.

extent를 확인하여 데이터를 추적하고 미할당 영역에 대한 정보를 비교하는 일은 메타데이터 기반 복구 방법과 동일하다.

이 방법은 삭제된 파일이 btree 포맷일 경우에만 복구가 가능하다. 만약 아이노드가 덮어 씌어졌고, btree 포맷이 아닌 경우 전통적인 카빙 방식에 의존해야 하며, 이미 알려져 있듯이 조각난 경우의 카빙 방법은 그 복구 성공률이 낮다.

XFS 파일 시스템은 파일의 목록을 데이터 포크

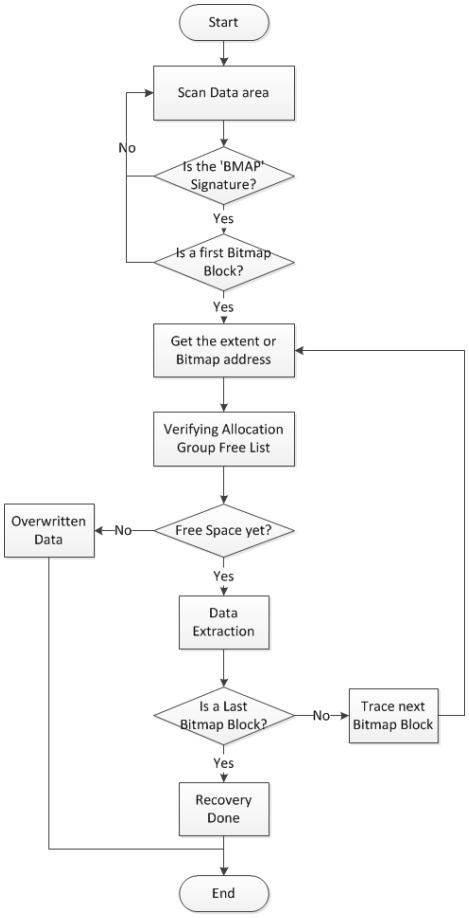

Fig. 26. Flow chart of recovery based on signature search

영역에 저장한다고 앞서 언급한 바 있다. 파일의 개수 및 파일 이름의 길이로 인해 160 byte 크기의 데이터 포크 영역에 파일 목록을 저장하지 못할 경우 비트맵 블록을 사용하여 저장한다. 이와 같은 비트맵 블록의 경우 btree 포맷에서 사용하는 비트맵 블록의 포맷과 동일하다. 따라서 복구 수행 시에 카빙 방식에 의해 탐지된 비트맵 블록이 디렉터리 엔트리를 저장하기 위한 것인지 데이터의 extent를 저장하기 위한 것인지 구별하여 처리해야 한다.

VII. 구현 및 실험 결과

본 논문에서 제시한 복구 방법을 이용하여 XFS 파일 시스템이 적용된 리눅스 환경에서 삭제된 데이터를 복구할 수 있는 프로토타입 도구를 구현했다.

Table 2. Test and development environment

구분	내용
Linux type	Slackware
Kernel version	3.1.0
Development Tool	MS Visual Studio 2010
Programming language	C / C++

상세한 실험 및 개발환경을 Table 2.에 나타내었다.

7.1 메타데이터 기반 복구 도구

Fig.27.은 본 논문에서 제안하는 메타데이터 기반 복구 방법을 구현한 도구의 수행 결과를 보여준다. Fig.27.에서 확인할 수 있는 것과 같이 아이노드가 새로운 데이터로 덮어 쓰여지지 않았다면 삭제된 파일을 복구할 수 있다.

```
AG No. = 0      Inode No. = 21
AG No. = 0      Inode No. = 23
AG No. = 0      Inode No. = 24
AG No. = 0      Inode No. = 26
Found a deleted inode
This file used the extents format!!
26th Inode Recovery Start!!
No.1 Extent value : [0, 2544375, 122, 0]
Recovery Done!!
```

Fig. 27. The recovery result of deleted files by the developed tool

7.2 시그니처 탐지 기반 복구 도구

Fig.28.은 본 논문에서 제안하는 시그니처 탐지 기반 복구 방법을 구현한 도구의 결과이다. 1개의 비트맵 블록에 11개의 extent로 이루어진 삭제된 파일을 복구하는데 성공하였다.

```
Found BMAP Block!!
Recovery Start!!
BMAP Block No.0 Extent No.1 Value = [0, 2020810, 24192, 0]
BMAP Block No.0 Extent No.2 Value = [24192, 2069170, 16384, 0]
BMAP Block No.0 Extent No.3 Value = [40576, 1988042, 32768, 0]
BMAP Block No.0 Extent No.4 Value = [73344, 1922506, 65536, 0]
BMAP Block No.0 Extent No.5 Value = [138880, 1791434, 131072, 0]
BMAP Block No.0 Extent No.6 Value = [269952, 1529290, 262144, 0]
BMAP Block No.0 Extent No.7 Value = [532096, 18055127, 524288, 0]
BMAP Block No.0 Extent No.8 Value = [1056384, 19627991, 1048576, 0]
BMAP Block No.0 Extent No.9 Value = [2104960, 25165836, 2097151, 0]
BMAP Block No.0 Extent No.10 Value = [4202111, 20676567, 622533, 0]
BMAP Block No.0 Extent No.11 Value = [4824644, 3118602, 418216, 0]
Recovery Done!!
```

Fig. 28. The recovery result of lost filles based on the signature search #1

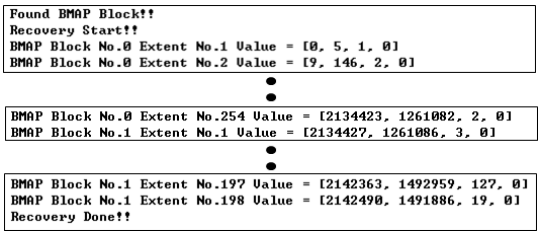

Fig. 29. The recovery result of lost files based on the signature search #2

Fig.29.는 2개의 비트맵 블록과 452개의 extent로 이루어진 삭제된 파일을 대상으로 복구 도구를 수행한 결과이다. 이와 같이 비트맵 블록을 2개 이상 사용할 경우에도 복구가 가능하다.

VIII. 결론 및 향후 연구 방향

본 논문에서는 XFS 파일 시스템에서 삭제된 파일을 복구하기 위해 메타데이터 기반 복구 방법과 비트맵 블록을 탐지하여 복구하는 알고리즘을 제시하였다. 그리고 삭제된 파일을 복구하기 위한 프로토타입의 도구를 구현하였고, 파일의 크기나 단편화의 정도와 상관없이 삭제된 파일의 아이노드가 덮어 쓰여지지 않았거나 비트맵 블록을 생성했던 파일이라면 복구 가능하다는 것을 증명하였다.

본 논문에서 제안한 복구 방법과 이를 기반으로 개발된 프로토타입 도구는 XFS 파일 시스템을 탑재한 NAS, CCTV에서 삭제된 파일을 복구하기 위해 활용될 수 있다.

References

[1] C. Hellwing, "XFS:the big storage file system for Linux," Communications of the ACM, 40, pp. 71-79, May 1997.

[2] A Sweeny, D. Doucette, W. Hu, C. Anderson, M. Nishimoto and G. Peck, "Scalability in the XFS File System," USENIX, 1996.

[3] Pal A and Memon N. "The Evolution of File Carving," iEEE SIGNAL PROCESSING MAGAZINE 2009.

[4] Brian Carrier, "File system forensic analysis," Addison-Wesley, 2005.

[5] Jungheum Park, "Advanced Digital Forensic System for Data Fragments," Korea University, 2013.

[6] M.Karresand and N.Shahmehri, "File type identification of data fragments by their binary structure," IEEE Information Assurance Workshop, June 2006.

[7] http://www.ufsexplorer.com

[8] Sekie Amanuel Majore, Changhoon Lee, Taeshik Shon, "XFS File System and File Recovery Tools," International Journal of Smart home, Vol7, No.1, Jan 2013.

[9] Silicon Graphics Inc, "XFS Filesystem Structure," 2006.

[10] http://linux.die.net/man/8/xfs_db

〈저자소개〉

안 재 형 (Jae-hyoung Ahn) 학생회원
2012년 2월: 인제대학교 컴퓨터공학부 졸업
2013년 3월~현재: 고려대학교 정보보호학과 석사과정
〈관심분야〉 디지털 포렌식, 정보보호

박 정 흠 (Jung-heum Park) 정회원
2007년 2월: 한양대학교 정보통신대학 컴퓨터전공 공학사
2009년 2월: 고려대학교 정보경영공학전문대학원 공학석사
2014년 2월: 고려대학교 정보경영공학전문대학원 박사
〈관심분야〉 디지털 포렌식, 안티-안티 포렌식

이 상 진 (Sang-jin Lee) 종신회원
1987년 2월: 고려대학교 수학과 학사
1989년 2월: 고려대학교 수학과 석사
1994년 8월: 고려대학교 수학과 박사
1989년 10월~1999년 2월: ETRI 선임 연구원
1999년 3월~2001년 8월: 고려대학교 자연과학대학 조교수
2001년 9월~현재: 고려대학교 정보보호대학원 교수
2008년 3월~현재: 고려대학교 디지털포렌식연구센터 센터장
〈관심분야〉 디지털 포렌식, 심층 암호, 해쉬 함수