논문 17-42-01-30 The Journal of Korean Institute of Communications and Information Sciences '17-01 Vol.42 No.01
https://doi.org/10.7840/kics.2017.42.1.241

차량용 블랙박스 영상파일의 무결성 검증에 해시함수 이용 방법

최 진 영◆, 장 남 수°

Integrity Verification in Vehicle Black Box Video Files with Hashing Method

Jin-young Choi◆, Nam Su Chang°

요 약

최근 차량용 블랙박스의 보급이 확산됨에 따라 이를 법적 증거로 사용하는 경우가 증가하고 있으며, 이에 따라 영상데이터의 무결성 검증에 대한 필요성이 대두되고 있다. 그러나 임베디드 시스템으로 분류되는 블랙박스는 적은 용량과 낮은 처리속도를 가지므로 영상파일 저장과 무결성 검증 처리의 한계점을 가진다. 본 논문에서는 제한된 자원을 가진 블랙박스 환경에서 고속경량 해시함수 LSH와 HMAC의 안전성을 이용하여 영상파일의 무결성을 보장하는 기법을 제안한다. 또한 이 기법을 구현하여 블랙박스 기기에서 무결성 검증 시의 CPU Idle Rate를 측정한 실험 결과를 제시하고, 제안한 기법의 효과성과 실용 가능성에 대해 검증한다.

Key Words : Blackbox, Integrity, Verification, Lightweight, Hash function

ABSTRACT

Recently, as a vehicle black box device has propagated, it has been increasingly used as a legal proof and there are the needs to verify an integrity of the video data. However, since the black box classified as the embedded system has a small capacity and low processing speed, there are limitations to the storage of video files and the integrity verification processing. In this paper, we propose a novel method for video files integrity in the black box environment with limited resources by using lightweight hash function LSH and the security of HMAC. We also present the test results of CPU idle rate at integrity verification in vehicle black box device by implementing this method, and verify the effectiveness and practicality of the proposed method.

Ⅰ. 서 론

최근 차량관련 스마트 카, 스마트 네트워크 등의 이슈로 임베디드 환경에서 빠른 영상처리를 위한 연구[1]와 차량에서 필요한 보안 요구사항에 대한 연구가 증가하고 있다[2,3,4]. 이중에서 차량용 블랙박스는 주행 시 상시 녹화는 물론, 주행/주차 상태에 상관없이 충격 영상을 별도로 저장하는 기능을 수행하고 있다. 특히 사고 발생 시의 영상에 대해서는 차후 보험사나 법원에서 증거로서 사용하기 위해서 무결성 검증 기능이 요구되고 있다. 영상데이터는 임베디드 기기로서는 처리하기에 만만치 않은 크기이므로 일반적인(상시) 영상파일 저장과 더불어 무결성 검증 기능을 수행하는 것은 큰 부담이 된다.

블랙박스의 상태는 크게 2가지로 나눌 수 있다. 자동차가 달리고 있을 때는 "주행" 상태로 인식하고 이

◆ First Author : Sejong Cyber University Graduate School of Information Security, seiyoung.choi@gmail.com, 정회원
° Corresponding Author : Sejong Cyber University Department of Information Security, nschang@sjcu.ac.kr, 정회원
논문번호 : KICS2016-11-354, Received November 17, 2016; Revised December 5, 2016; Accepted January 13, 2017

때는 카메라로 캡쳐 되는 모든 내용을 SD카드에 write하는 "상시녹화"를 진행한다. 차가 멈추었을 때는 "주차" 상태로 인식하며 이 때는 카메라 캡쳐만 하고 SD카드에 write 하지는 않는다. write하지 않으면서도 캡쳐를 계속하는 이유는 모션 또는 충격과 같은 이벤트 감지를 24시간 진행하기 때문이다. 언제든지 이벤트가 발생하면 발생시점 이전 10~15초와 발생시점 이후 20~30초 분량을 포함하는 사고영상파일을 상시와는 별도의 분리된 위치(다른 디렉토리)에 저장해야 한다. 발생시점 이전 10~15초 저장을 위해서는 메모리 상에 10~15초 영상데이터가 들어갈 만큼의 버퍼를 잡아 놓고 항상 최근 영상을 유지하고 있어야 한다.

임베디드 시스템으로 분류되는 블랙박스는 메모리와 CPU 면에서 일반 PC에 비하여 적은 용량과 낮은 처리속도를 가지고 있다. 그러므로 임베디드 시스템에서의 메모리 절감과 처리 효율성은 선택 사항을 넘어 필수사항이며 무결성 검증 기능의 구현 가능성을 결정짓는 중요한 요인이 된다. 그런 면에서 볼 때 공개키를 사용하기 위해서는 많은 양의 코드와 프로세스 시간을 필요로 하게 된다. 이와 대조적으로 해시 함수를 이용한 간단한 알고리즘만으로 무결성 검증을 구현한다면 메모리 절감효과는 물론 연산 효율 면에서도 이득을 볼 수 있다.

최근 제안된 실시간 무결성 보장기법[5]에서는 블랙박스의 보안 요구사항을 무결성, 부인방지, 오류복구, 빠른 계산 속도로 보고 이를 충족시키기 위해 hash 함수와 공개키 암호를 사용하였다. 첫 데이터 블록을 hash하기 전에 공개키로 서명함으로써 부인방지 하였고 각 데이터 블록을 2중으로 hash 함으로써 데이터 재생공격을 피하였으며 새로운 데이터 블록이 올 때마다 이전 데이터 블록의 IVD 값에 새로운 블록의 hash값을 함께 hash한 결과를 넣어 줌으로써 후위데이터 삭제 공격과 데이터 교체 공격을 막아냈다. 그러나 무결성 검증의 구체적인 방법을 제안한 연구로서 의미가 크나 대다수의 블랙박스가 영상파일을 짧은 시간(예를 들어 1분) 단위로 쪼개어 보관하는 상황에서 파일 단위 무결성 검증과 함께 파일시스템 전체에 대한 무결성 검증이 필요한데 이를 지원하지는 못하고 있다. 또한 계산량이 많은 공개키 암호를 사용하여 임베디드 환경에서의 부담을 가중시키고 있다. 블랙박스의 영상파일은 특정인에 대한 부인방지가 아니고 블랙박스 장치에 대한 의미만이 있으므로 부인방지하기보다는 어느 블랙박스에서 생성한 파일인지를 검증할 방법을 제안하는 것이 합리적일 것이며 이는 비밀 키를 블랙박스과 검증서버에 각각 보관함으로써 해결할 수 있다.

또한 영상데이터와 개인정보보호를 위해서 무결성과 기밀성을 제공하는 방안도 제안되었다[6]. 무결성 검증을 위해서 차량 내부메모리에는 파일리스트와 파일이력정보를 저장하고 외부 메모리에는 파일자체와 파일간 무결성 정보를 보유하게 하여 내부와 외부 메모리를 병행사용 하는 방법을 제안하고 있다. 그러나 이는 사고 발생 시 블랙박스 훼손으로 인하여 외부메모리인 SD카드만이 사용 가능할 경우에는 온전한 무결성 검증을 수행할 수 없다.

무결성 보장 기법과 관련하여 순환형 데이터 블록 체이닝을 이용한 차량용 블랙박스의 상시 데이터 무결성 보장 기법도 제안되었다[7]. 블록 체인을 순환형으로 맨 뒤와 맨 앞 블록을 연결함으로써 후위삭제 공격을 방지함과 동시에 맨 앞의 블록을 overwrite할 때에도 무결성 보장이 되도록 하였다. 또한 하나의 파일 안의 모든 블록이 연결되도록 하는 대신 인접한 2개 블록만 연결되도록 함으로써 인접과계를 증명하면서도 파일시스템 오류 시에 부분적으로 무결성 검증 가능하도록 하였다. 그러나 AVI 파일단위 무결성 검증 데이터 저장방법을 구현하기 위해서는 한 블록을 쓸 때마다 맨 첫 블록을 가진 파일의 첫 IVD값을 새로써 주어야하므로 항상 2개의 파일을 연 상태로 양쪽에 IVD 값을 써 주어야 한다. 블랙박스 업체별로 고안한 특수한 파일시스템을 사용하지 않는 한 일반 파일시스템에서는 전원 off시에 열고 있던 파일의 안전을 보장하지 못하므로 더 큰 위험을 감수하게 된다. 또한 실제 블랙박스 환경에서는 파일 저장시간이 성능에 큰 부담으로 주고 있음에도, 성능 검증을 위한 속도측정에서 메모리 상의 계산 속도만을 측정하고 파일로 저장하는 시간은 배제함으로써 실제 상황을 반영하지 못했다. 이에 대한 대응책으로써 "무결성 검증값을 모아서 한번에 파일쓰기"하는 방법을 제안하고 있으나 이 경우에는 무결성 데이터 부분에 손상을 입는 경우 부분적인 무결성 검증기능이 무력화된다.

본 논문에서는 경량고속 해시함수와 비밀 키를 이용하여 HMAC의 변형 방식으로써 영상파일 내 어느 곳에서 파일이 끊기더라도 끊긴 지점까지의 무결성 검증이 가능하게 하는 방법을 제안하고자 한다. 제안하는 방법은 format free 방식 하에서 안전하게 닫지 못한 파일도 데이터가 저장된 곳 까지의 무결성 검증이 가능하다. 또한, 고의 또는 과실에 의해 부분적으로 훼손된 파일이라도 훼손되지 않은 나머지 부분들에 대해서는 무결성 검증이 가능하다.

본 논문의 구성은 다음과 같다. 2장에서는 블랙박

스의 요구사항을 살펴보며, 3장에서는 암호 알고리즘의 블랙박스 상에서 동작 성능을 비교한다. 4장에서는 제안하는 무결성 방법을 기술하고, 5장에서 성능을 측정하고 결론을 맺는다.

Ⅱ. 블랙박스의 요구사항

블랙박스의 가장 중요한 기능은 사고영상 저장의 신뢰성이라 할 수 있다. 그러나 사고 발생 시에는 상시 저장 때와는 다른 악조건이 주어진다. 즉, 그림 1과 같이 DRAM에 저장된 최근 10~15초 분량의 영상데이터 수십 Megabytes를 한꺼번에 SD카드 등의 파일 형태로 저장해야 한다. 거기에 덧붙여 사고 발생 시에는 전원 탈거 또는 시동 꺼짐으로 인해 주전원이 차단되는 경우가 발생 가능하다. 이 때는 보조전원(배터리)이 소진되기 전에 DRAM에 쌓인 영상을 모두 저장하고 안전하게 파일을 닫아야 하므로 더욱 빠르고 효율적인 처리가 중요하게 된다.

블랙박스 프로그램을 수행하면서 top으로써 감시해 보면 표 1과 같이 상시의 cpu idle rate가 35~40%를 유지하다가도 충격 발생시에는 2~3초간 0%로 떨어지는 것을 확인할 수 있다.

현재 cpu load가 큰 작업을 처리하고 있는 블랙박스에 추가로 무결성 검증 기능이 요구되고 있다. 이유는 블랙박스 영상 파일을 보험사나 법원 등에 증거자료로서 사용할 수 있도록 하기 위함이다. 이에 따라 자동차용 사고영상기록장치에 대한 KS 인증 심사기

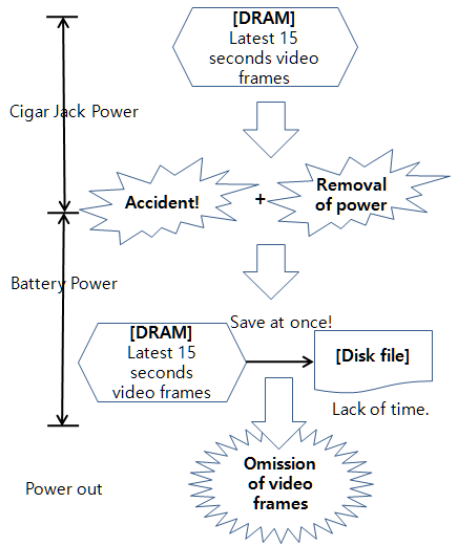

그림 1. 사고 시나리오
Fig. 1. Accident Scenario

표 1. 상시와 이벤트 발생 시의 cpu idle rate 비교
Table 1. Cpu Idle Rate Comparison at Normal and Event Recording

Cpu Idle Rate at Normal Recording	About 35 %
Cpu Idle Rate at Event Recording	About 0 % (for 2~3 seconds)

준 (2012년 제정되어 2015.8. 4차 개정됨. KS_C_5078)에 "사고기록정보의 무결성 기능 검증"이 포함되었고 한국산업기술시험원(KTI)에서는 이를 근거하여 아래와 같은 기준으로 검증시험을 수행하고 있다.

<사고기록 정보의 무결성 기능 검증 시험>
- 무결성 검증용 비밀키의 생성 확인.
- 무결성 기능을 위한 알고리즘 구현 확인.
- 무결성 검증값 생성 확인
- 사고기록 정보의 위/변조 또는 무결성 검증값 손상에 대한 탐지 및 검증여부 시험.

<영상사고기록장치의 무결성을 위해 사용되는 보안알고리즘>
- 해시 검증 방식 : MD5, SHA1, SHA2
- 대칭키 암호 방식 : SEED, AES
- 비대칭키 암호 방식 : RSA, ECDSA

블랙박스 영상 파일에 대해서는, 어떤 블랙박스가 작성한 파일인지 알리는 기능과, 최초에 작성된 상태에서 변형되지 않았다는 무결성 검증 기능이 함께 요구된다. 그러므로 통상 무결성 검증을 위해서 파일 내용 전체를 하나의 해시값으로 만들고 이에 대해 공개키로 서명을 한다. 그러나 Embedded linux 환경(ARMv7 Processor, Linux version 3.0.8, OpenSSL 사용) 에서 테스트한 결과 해시 결과값에 대해서 RSA로 암호화 하는 것은 큰 부담이 된다.

Ⅲ. 블랙박스에서의 암호 알고리즘 성능 비교

블랙박스 소비자 불만 사항에서 가장 큰 문제 중 하나가 장기 또는 단기 미녹화에 대한 것이다. 이러한 현상은 앞서 설명한 사고발생시 전원탈거와 같은 악조건 형성과 관련되어 발생 가능한 것이며 이에 대한 보완책으로서 format free 방식 채택 등으로써 파일을 닫지 못하더라도 차후 복구가 가능하게 하는 기술이 적용되고 있으나 이 때에도 법적 효력을 가지려면 무

결성 검증값을 같이 저장해 주어야 한다. 본 논문에서는 이러한 문제를 해결하기 위하여 HMAC을 활용하고자 한다. HMAC은 해시함수를 기반으로 구성되므로 효율적 구성을 위하여 본 절에서는 해시함수의 성능을 비교한다.

해시함수는 가장 널리 사용되는 SHA2와 최근 제안된 경량 해시함수 LSH를 대상으로 하였으며, 성능 측정을 위한 환경과 사용 소스의 출처는 다음과 같다.

[사용 환경]
- CPU : ARMv7 Processor revision 2 (ARMv7)
- 운영체제(OS): Linux version 3.0.8
- 구현언어 / 컴파일러: gcc version 4.5.1

[사용 소스의 출처]
- LSH : 한국암호포럼(http://kcryptoforum.or.kr/) 제공 소스(lsh_neon)]
- SHA : http://ftp.netbsd.org/ 제공 소스(sha2.c, sha2.h)

다양한 메시지 길이에 대해 SHA-256, SHA-512, LSH-256, LSH-512의 성능을 비교하면 그림 2와 같다. 256 및 512 Bytes의 모든 경우에서 LSH가 SHA보다 속도면에서 우수하였으며, 메시지의 길이가 길어질수록 그리고 256 bit보다는 512 bit 일 때 두 함수의 속도차가 더욱 커진다. SHA 수행시간에 대비하여, 더 좋은 성능을 보인 LSH의 수행시간은 표 2와 같으며, 4096 byte 이상의 길이를 512 bit 이상의 해시함수로 처리할 때 LSH가 효과적임을 알 수 있다.

영상처리에 활용하기 위하여 바이트 당 처리 시간을 측정하면 그림 3과 같다. 메시지가 길어질수록 바이트 당 처리 시간이 점점 줄어들다가 일정 길이 이상

```
LSH Benchmark Results:

SHA-256-256,   64-byte messages: (tMin=     13)    0.20 usec/byte
SHA-256-256, 4096-byte messages: (tMin=    347)    0.08 usec/byte
SHA-256-256,      long messages: (tMin=    170)    0.08 usec/byte
SHA-512-512,   64-byte messages: (tMin=     49)    0.77 usec/byte
SHA-512-512, 4096-byte messages: (tMin=   1549)    0.38 usec/byte
SHA-512-512,      long messages: (tMin=    750)    0.37 usec/byte
LSH-256-256,   64-byte messages: (tMin=      9)    0.14 usec/byte
LSH-256-256, 4096-byte messages: (tMin=    277)    0.07 usec/byte
LSH-256-256,      long messages: (tMin=    114)    0.06 usec/byte
LSH-512-512,   64-byte messages: (tMin=     40)    0.62 usec/byte
LSH-512-512, 4096-byte messages: (tMin=    771)    0.19 usec/byte
LSH-512-512,      long messages: (tMin=    362)    0.18 usec/byte
```

그림 2. 64-byte, 4096-byte, Long Messages 각각일 때의 SHA-256, SHA-512, LSH-256, LSH-512 성능테스트. Long message는 4096-byte - 2048-byte 시간.
Fig. 2. SHA-256, SHA-512, LSH-256, LSH-512 Performance Test for 64-byte, 4096-byte, Long Messages respectively. Long message is 4096-byte process time minus 2048-byte process time.

표 2. 다양한 메시지 길이에 대한 SHA, LSH 성능비교 (LSH 소요시간 / SHA 소요시간 * 100)
Table 2. SHA vs LSH Performance Time for Several Message Length (LSH time / SHA time * 100)

	64 byte message	4096 byte message	long message
SHA-256 vs LSH-256 Performance Time	69 %	79 %	67 %
SHA-512 vs LSH-512 Performance Time	81 %	49 %	48 %

```
LSH-256-256,     64-bytes messages: (tMin=     11)    0.17 usec/byte
LSH-256-256,    128-bytes messages: (tMin=     20)    0.16 usec/byte
LSH-256-256,    256-bytes messages: (tMin=     32)    0.12 usec/byte
LSH-256-256,    512-bytes messages: (tMin=     53)    0.10 usec/byte
LSH-256-256,   1024-bytes messages: (tMin=     95)    0.09 usec/byte
LSH-256-256,   2048-bytes messages: (tMin=    178)    0.09 usec/byte
LSH-256-256,   4096-bytes messages: (tMin=    345)    0.08 usec/byte
LSH-256-256,   8192-bytes messages: (tMin=    680)    0.08 usec/byte
LSH-256-256,  16384-bytes messages: (tMin=   1348)    0.08 usec/byte

LSH-512-512,     64-byte messages: (tMin=     44)    0.69 usec/byte
LSH-512-512,    128-byte messages: (tMin=     47)    0.37 usec/byte
LSH-512-512,    256-byte messages: (tMin=     92)    0.36 usec/byte
LSH-512-512,    512-byte messages: (tMin=    137)    0.27 usec/byte
LSH-512-512,   1024-byte messages: (tMin=    228)    0.22 usec/byte
LSH-512-512,   2048-byte messages: (tMin=    409)    0.20 usec/byte
LSH-512-512,   4096-byte messages: (tMin=    771)    0.19 usec/byte
LSH-512-512,   8192-byte messages: (tMin=   1496)    0.18 usec/byte
LSH-512-512,  16384-byte messages: (tMin=   2946)    0.18 usec/byte
```

그림 3. 64-bytes ~ 16384-bytes 메시지에 대한 LSH-256, LSH-512 의 수행시간
Fig. 3. LSH-256, LSH-512 Performance Time for 64-bytes ~ 16384-bytes Message

이 되면 바이트 당 처리속도가 더 이상 줄어들지 않는다. 따라서 최적의 성능을 위해서는 최소 입력값으로 아래 길이 이상의 메시지를 사용하는 것이 바람직하다.

표 3에서와 같이 LSH-256은 4096바이트 이상을 처리할 때 최적 성능을 보이며, LSH-512는 8192바이트 이상을 처리할 때 최적 성능을 보인다.

실제 블랙박스 영상 파일 처리에서 SHA와 LSH의 성능을 비교하면 다음과 같다. 샘플로 사용한 영상 파일 형식은 아래와 같으며 블랙박스에서 흔히 사용하는 전형적인 형태의 하나이다.

- 비디오포맷 : H.264, 영상파일포맷: AVI

표 3. 각 메시지 길이 별 최적 수행 시간 비교
((N byte 처리할 때의 usec/bytes) / (64 byte 처리할 때의 usec/bytes) * 100)
Table 3. Performance Time Percentage of Various Length Messages against for 64-bytes Performance Time
((usec/bytes for N bytes message) / (usec/bytes for 64-byte message) * 100)

bytes	64	128	256	512	1024	2048	4096	8192	16384
LSH-256	100%	94%	70%	58%	52%	52%	47%	47%	47%
LSH-512	100%	53%	52%	39%	31%	28%	27%	26%	26%

- 스트림 개수 : 비디오,오디오 스트림 각 1개씩, 총 길이 : 55,522,669 Bytes

영상 파일 무결성 검증에 해시함수를 사용할 때는 그림 4와 같이 일반적으로 frame 단위로 해시하는 방법을 취한다. 그러므로 실제 상황을 시뮬레이션 하기 위해 실제 영상 파일의 각 frame 길이를 추출하여 하나의 정수 배열로 만들었다. 그리고 이 배열의 각 원소값 byte수에 해당되는 메시지들을 차례로 모두 해시하여 전체 소요된 시간을 출력하였다. 그림 5와 같이 최소 frame 크기는 4096 byte, 최대 frame 크기는 74K-byte인 약 52.9M-byte 크기 영상파일을 해시할 때 소요된 시간을 확인할 수 있다. 결과에서 알 수 있듯이 전체적으로 LSH가 좋은 성능을 보이고 4096 byte 이상 길이의 메시지들을 처리할 때 byte당 처리 시간이 가장 짧다. LSH-256은 SHA-256에 비해 88.9%, LSH-512는 SHA-512에 비해 47.3%의 처리 시간이 각각 소요된다.

임베디드 시스템에서의 공개키 암호 함수와 해시함수의 성능을 비교하기 위해 같은 길이의 메시지를 RSA로 100회 처리하는 시간과 hash함수로 10000회 처리하는 시간을 system clock수로 비교하면 그림 6과 같다. RSA와 SHA256은 모두 OpenSSL 1.0.0d 라이브러리에 있는 소스를 사용하였다. system 1 clock을 약 0.001 msec (milli second)라고 간주할 때 RSA는 회당 75 msec가, hash 함수는 회당 0.004 msec가 소요되어 RSA는 해시함수를 사용할 때의 약 18000배의 시간이 소요된다. 2채널 전후방 각 30 frame이고 1분짜리인 블랙박스 영상 파일의 경우, 초당 frame 수를 (오디오와 subtitle frame까지 고려하여) 80개 전후로 볼 수 있다. 이 때 파일 전체에 4800개의 frame이 있으며 이를 무결성 보장하기 위해 수행하는 무결성 검증값 생성 횟수를 4800회라고 하면 RSA의 소요 시간은 해시함수의 약 4배이다. 이러한 효율성의 문제 때문에 임베디드 시스템에서의 공개키 사용은 큰 부담이 된다.

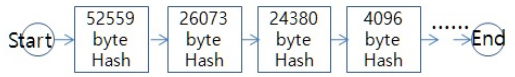

그림 4. 영상파일 해시 순서
Fig. 4. File Hash Sequence

```
H.264 frames : count=2456, minsize=4096 maxsize=74204 totalbytes=55522669

SHA-256-256,   real H.264 frames: (usec= 4790954)    0.09 usec/byte
SHA-512-512,   real H.264 frames: (usec=21161579)    0.38 usec/byte
LSH-256-256,   real H.264 frames: (usec= 4687722)    0.08 usec/byte
LSH-512-512,   real H.264 frames: (usec=10196619)    0.18 usec/byte
```

그림 5. 60초 길이의 블랙박스 영상파일 해시 시간
Fig. 5. Performance Time for Hash of All Packets in 60 Seconds Long Blackbox Video File

```
# ./openssltest
RSA start
public_encrypt : Encrypted length =256.
        100 try takes 250000 clocks.
Decrypted Text = Hello this is Choi
Decrypted Length = 18
private_encrypt : Encrypted length =256.
        100 try takes 7530000 clocks
Decrypted Text =Hello this is Choi
Decrypted Length =18
SHA256 : 10000 try takes 40000 clocks.
```

그림 6. 2048-bit key RSA와 SHA256 성능테스트
Fig. 6. RSA with 2048-bit key vs SHA256 Performance Test

Ⅳ. 제안하는 블랙박스 파일시스템 및 파일의 무결성 보장 기법

본 논문에서는 블랙박스에서의 무결성 검증 기능을 KS 기준에 부합하게 구현하는데 있어서 기존과 같이 무거운 공개키나 대칭키 암호 알고리즘을 사용하는 대신에 가벼운 해시함수만을 이용하고자 한다. 비밀키를 이용한 HMAC 의 변형 방식으로써 해시함수만을 이용하여 적은 메모리 자원을 가지고 높은 효율성을 발휘하며 데이터 생성의 주체를 확인하고 무결성 검증을 할 수 있다. 제안하는 방법 설명을 위하여 용어를 정의하면 다음과 같다.

- Normal 해시 : 메시지를 포함한 1st-pass 해시 과정과 이 때의 해시 값을 다시 처리하는 2nd-pass 해시 과정을 거친 해시 값.
- Double 해시 : Normal 해시의 결과값을 다시 키 값 포함하여 해시한 값
- i-key, o-key : 키 값 (64 byte 이내) 에 특정 상수를 XOR하여 만든 값
- frame IVD : frame마다 자신의 후미에 IVD (Integrity Verification Data) 값을 추가로 가지게 된다. 이는 대부분의 경우 normal 해시값을 사용하게 되며 특별한 경우 double 해시값을 사용하게 된다.
- file IVD : frame마다 자신의 후미에 IVD (Integrity Verification Data) 값을 추가로 가지게 된다. 이는 자신의 마지막 frame의 double 해시값 또는 다음 파일의 첫 frame의 double 해시값 중 하나이다.

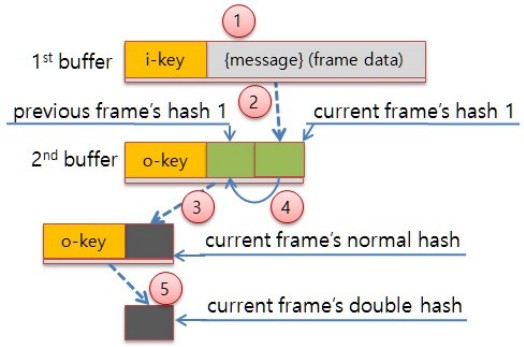

그림 7. normal hash와 double hash 값 생성
Fig. 7. Generating normal hash and double hash value

제안하는 방법을 기술하면 다음과 같다.

그림에서 짙은 회색으로 표시된 부분이 파일에 저장되는 hash 값이다.

① key에 0x3636…36 값을 XOR하여 생성한 i-key와 frame data를 인접시켜 버퍼에 준비한다. 이 과정을 위하여 frame data의 buffer를 잡을 때 미리 i-key 가 들어갈 여유 자리를 잡아 놓고 i-key 값을 넣어 둠으로써 hash하기 위해 별도의 메모리 복사가 필요하지 않도록 한다.

② 1st buffer 내용을 hash하여 결과값 hash 1을 2nd buffer 맨 뒤에 넣는다. hash 1은 HMAC에서의 first pass에 해당 된다. 2nd buffer 에는 key에 0x5c5c…5c 값을 XOR하여 생성한 o-key와 이전 frame의 hash 1 값과 현재 frame의 hash 1값이 인접하여 있다. 2nd buffer 은 메모리상에 준비해 두고 frame을 파일에 저장할 때마다 무결성 검증용 데이터 생성에 사용한다.

③ 2nd buffer 내용을 hash한 결과가 normal hash이며 이 값이 파일에 저장할 때 frame 뒤에 덧붙여지는 IVD 값이다. normal hash 는 HMAC의 second pass에 해당된다. 모든 frame들은 각각 자신의 normal hash 값(IVD)을 frame 내용 뒤에 덧붙이게 된다. 파일의 첫 frame과 마지막 frame에 대해서는 파일 간 무결성 검증을 위해 double hash값을 생성하는데 이 때 현재 frame의 normal hash값을 사용 한다. [그림 8]

④ 현재 frame의 hash 1 값을 이전 frame의 hash 1 값 자리에 복사하여 다음 frame 처리를 준비한다.

⑤ o-key 와 normal hash 값을 인접시켜 한 번 더 hash 하여 double hash 값을 계산한다. double hash는 파일의 첫 frame과 마지막 frame에 대해서만 생성

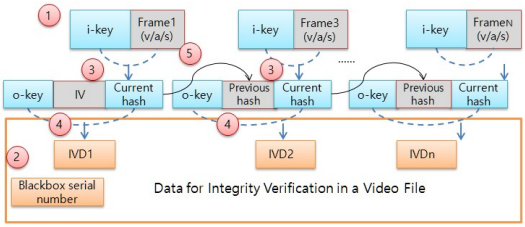

그림 8. 무결성 검증 데이터 생성 과정
Fig. 8. Process for Generating Integrity Data

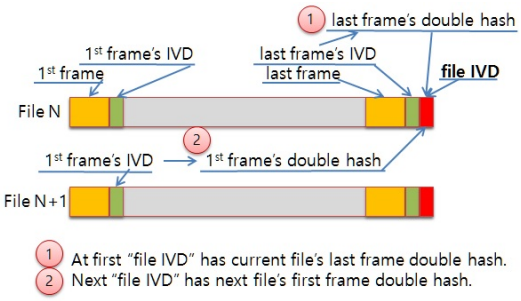

그림 9. IVD값 생성
Fig. 9. Generate file IVD

하며 파일 간 무결성 검증에 사용 된다. [그림 9]

① ~ ④ 과정은 frame당 1회씩 수행되고 ⑤ 과정은 파일당 2회씩(맨 처음 frame과 맨 마지막 frame일 때) 만 수행된다. Normal 해시는 일반적인 키를 이용한 HMAC 방식을 활용하였다. 이 결과값은 파일 내에 저장되어 각 frame의 무결성 검증에 사용되며 각 frame data뒤에 붙여서 저장한다. Double 해시는 Normal 해시에 의해 나온 해시 값 앞에 o-key를 붙여서 다시 한 번 해시한 값이다. 이 값은 파일간 연계성 검증에 사용한다. 그림 8과 같이 두 가지 해싱 방법을 이용하여 파일 내용에 대한 무결성 검증과 파일간 연계성 검증을 할 수 있다. 첫 번째부터 마지막 frame까지는 ①~⑤ 단계를 진행하여 "최종 해시값"을 파일에 write 한다. 이로써 파일의 어느 지점에서 끊기더라도 그 지점까지의 무결성 검증이 가능하며 파일 중간에 부분적으로 훼손된 부분이 있더라도 전후 관계가 맞기 시작하면 그 뒤 부터는 다시 무결성 검증이 가능하게 된다. 이렇게 되는 이유는 파일 내의 모든 frame내용을 연결하는 방식으로 해시 진행하지 않고 바로 이전 frame하고만 연결시켜 해시하기 때문이다. 중간에 해시 값이 맞지 않더라도 2개의 정상적인 frame이 들어오면 그 이후로는 다시 무결성 검증이 가능해진다.

① i-key, o-key 준비 : 키 값 (64 byte 이내) 에 특정 상수를 XOR하여 만든 각기 다른 두 값 i-key와

o-key를 준비한다.

② "검증"할 때 필요하므로 파일 첫 부분에 "제품고유번호"를 write한다.

③ 각 frame에 대하여 1st pass로서 frame 내용에 i-key를 앞에 붙여서 hash 한 후 결과 값을 Current hash 자리에 넣어 2nd pass를 준비 한다..

④ 2nd pass로서 o-key와 이전 frame hash값과 현재 frame hash값을 붙여서 한꺼번에 hash하여 결과값을 IVD (Integrity Varification Data) 로서 파일에 write 한다. 파일에 write된 값은 다음 hssh할 때 사용되지 않으므로 파일에 write하여 노출되어도 된다.

⑤ 다음 frame의 IVD값 생성에 사용되도록 하기 위해 현재 frame hash 값인 Current hash를 Previous hash 자리로 복사한다.

그림 8의 ③..⑤ 과정을 모든 frame에 대해 반복한다.

다음으로 파일 시스템에서의 무결성 검증방법을 기술한다. 파일 단위 삭제, 순서변경, 삽입 인지 방법으로서 파일의 file IVD(Integrity Verification Data) 값을 생성하여 파일 후미에 붙여 준다. 파일의 마지막 frame의 normal 해시값 o-key를 함께 hash하여 double 해시 값을 생성하여 이를 file IVD로 사용한다. 다음 파일을 열 때는 이전 파일을 열어서 file IVD 자리에 자신의 첫 frame의 double 해시 값을 써 준다. 이 때 이전 파일이 없으면 (자신이 첫 번째 파일) 첫 frame의 frame IVD 자리에 첫 frame의 double 해시 값을 써 준다. 그림 9와 같은 처리 순서를 따르면 다음과 같이 파일 간 중간, 후위 데이터 삭제 공격을 방지할 수 있다.

4.1 중간데이터 삭제 방지

연속된 파일 중 앞쪽 파일의 file IVD 자리에는 다음 파일의 첫 frame의 double해시 값이 있어야 한다. 다음 파일을 열어서 그의 첫 frame의 double 해시 값이 이전 파일의 file IVD 자리에 있으면 두 파일 사이의 인접 관계를 확인할 수 있다. 그 외의 다른 값이 있을 때는 중간데이터 삭제가 있었음을 알 수 있다.

4.2 후위데이터 삭제 방지

파일 IVD 자리에 자신의 최종 frame의 double 해시값이 들어 있으면 자신이 최종 파일인 것을 확인할 수 있다. 최종 파일인데 이의 file IVD 값이 자신의 마지막 frame 의 double 해시값이 아니라면 후위데이터 삭제가 있었음을 알 수 있다.

제안하는 무결성 검증방법은 블랙박스에서 다음을 지원한다.

- 데이터 재생 방지 : 각 IVD 값을 계산할 때마다 고유 key를 사용하므로 데이터 재생이 불가능
- 데이터 교체 방지 : 데이터가 바뀐다면 hash 함수의 결과값이 달라짐.
- 데이터 순서 바꾸기 방지 : 바로 전 frame의 normal hash값이 현재 frame의 IVD 값 생성에 사용되므로 방지됨
- 중간데이터 삭제 방지 : 바로 전 frame의 normal hash값이 현재 frame의 IVD 값 생성에 사용되므로 중간 frame 삭제를 방지함
- 후위데이터 삭제 방지 : 파일시스템 무결성 검증에서는 다음 파일이 생성될 때 그 파일의 첫 frame의 double hash값을 이전파일의 file IVD 값으로써 써 주므로, 어떤 파일의 fileIVD 값이 자신의 마지막 frame의 double hash값이 아닐 때 다음 파일이 없다면 후위데이터 삭제임을 알 수 있음
- 오류 복구 : SD카드 문제 등으로 인하여 파일의 일부가 손상되었더라도 체인 연결을 앞뒤로 인접한 2개 블록만을 연결하고 있으므로 정상적인 frame이 2개 이상 나온 이후로는 다시 무결성 검증이 가능

V. 성능 측정 및 결론

HMAC 변형 방식인 제안하는 방법의 성능을 블랙박스에서 측정하면 다음과 같다. 블랙박스 스펙은 HD 전방(1280x720)만 녹화하는 상태에서 테스트 하였으며 embedded linux에서 블랙박스 프로그램을 수행하며 동시에 top 프로그램을 수행하여 cpu idle rate를 측정하였다. 무결성 검증을 하지 않을 때는 그림 10과 같이 cpu idle rate가 29%로서 비교적 여유가 있다.

제안하는 방법에 SHA를 적용하여 무결성 검증을 수행하면 그림 11과 같다. 상시저장인데도 cpu idle rate가 0% 이므로 이벤트 발생 시에는 오동작할 가능성이 크다.

제안하는 방법에 LSH를 적용하여 무결성 검증을

```
Mem: 94864K used, 60392K free,  0K shrd,  7698K buff, 21004K cached
CPU:  15% usr  31% sys   0% nic  29% idle  21% io   0% irq   1% sirq
Load average: 1.67 0.47 0.16 1/66 1089
  PID  PPID USER     STAT   VSZ %MEM %CPU COMMAND
   62    57 root     S     355m 234%  39% /apps/cdrqp
   55     2 root     SW       0   0%   3% [mmcqd/0]
    4     2 root     IW       0   0%   2% [kworker/0:0]
   41     2 root     SW       0   0%   1% [kworker/0:2]
  100    62 root     S    67688  44%   0% /apps/cdrqp
  177     2 root     SW       0   0%   0% [flush-179:0]
  374    57 root     R     2484   2%   0% top
```

그림 10. 무결성 검증데이터 없을 때의 Cpu Idle Rate
Fig. 10. Cpu Idle Rate with No Integrity Data

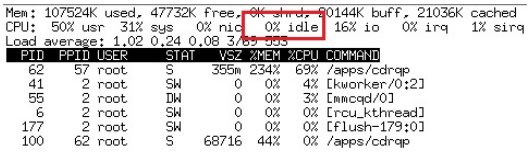

그림 11. SHA 로 무결성 검증데이터 생성시의 Cpu Idle Rate
Fig. 11. Cpu Idle Rate for Integrity Data with SHA

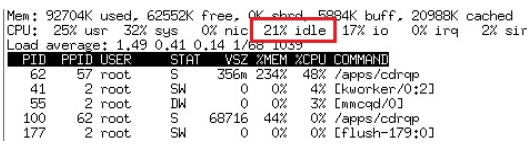

그림 12. LSH로 무결성 검증데이터 생성시의 Cpu Idle Rate
Fig. 12. Cpu Idle Rate for Integrity Data with LSH

수행하면 그림 12와 같다. cpu idle rate가 21%로서 무결성 검증을 하지 않을 때보다는 여유가 줄어들었지만 실행에 문제없는 상태임을 알 수 있다.

테스트 결과 LSH가 SHA 보다 나은 성능을 보였으며 이 결과가 영상파일 처리에서도 같게 나타남으로써 해시함수 성능에 따라 무결성 검증 기능의 효율성에 직접적인 영향을 주는 것으로 확인되었다. 무결성 검증을 하지 않을 때 29% idle과 LSH를 이용하여 무결성 검증할 때 21% idle은 정상 동작이 가능한 상태라고 할 수 있다. 그러나 SHA를 이용하여 무결성 검증할 때 0% idle 상태는 프로그램이 수행되기는 하였으나 이벤트 발생 시의 추가 처리를 고려하면 상용으로써 사용할 수 없는 과부하 상태이다. 또한 이는 전방카메라만을 처리할 때의 부하 정도를 테스트한 것이므로 후방카메라까지 처리하거나 HD보다 고해상도의 영상을 처리할 경우에는 보다 높은 성능이 필요할 것이다.

이미 국내 블랙박스의 현황은 대부분이 전/후방 2채널을 사용하며 전방 HD, 후방 VGA급 이상을 구현하고 있다. 현재도 블랙박스에 요구되는 화질은 HD에서 full HD로, 다시 UHD로 점점 높아져 가고 있다. 이렇게 될 경우 저장에 필요한 메모리 또는 디스크 용량은 full HD의 경우엔 HD의 약 2.2 배, UHD의 경우엔 HD의 약 9 배가 필요하다. 이런 대용량 데이터에 무결성 검증값을 실시간으로 생성하여 함께 저장하려면 암호 함수 중 가장 빠른 해시 함수를 활용하는 것이 가장 타당할 것이다. 그런 면에서 기존 사용하던 해시 함수의 성능을 개선한 더욱 고속이고 경량인 해시함수의 개발은 영상파일 무결성 검증 분야에 있어서 필수적이라 하겠다.

References

[1] S. Son, T. Kim, Y. Jeon, Y. Baek, "Smart camera technology to support high speed video processing in vehicular network," *J. KICS*, vol. 40, no. 1, pp. 152-164, Jan. 2015.

[2] B.-H. Lee, S.-B. Lee, J.-Y. Moon, and J.-H. Lee, "Lightweight DTLS message authentication based on a hash tree," *J. KICS*, vol. 40, no. 10, pp. 1969-1975, Oct. 2015.

[3] H. Seo and H. Kim, "User authentication method on VANET environment," *J. KICS*, vol. 37, no. 7, pp. 576-583, Jul. 2015.

[4] J. Shin, S. Oh, C. Jeong, K. Chung, and K. Ahn, "Improved an RFID mutual authentication protocol based on hash function," *J. KICS*, vol. 37, no. 3, pp. 241- 250, Mar. 2012.

[5] Y. Kim, B. H. Kim, and D. H. Lee, "Real-time integrity for vehicle black box system," *Korea Inst. Inf. Security & Cryptology*, vol. 19, no. 6, pp. 49-61, Dec. 2009.

[6] W. J. Kim and H. H. Kim, "A study on the application of the vehicle blackbox data security technology," *Korea Inst. Inf. Security & Cryptology*, vol. 24, no. 2, pp. 35-41, Apr. 2014.

[7] K. Yi, K.-M. Kim, and Y. J. Cho, "A car black box video data integrity assurance scheme using cyclic data block chaining," *J. KIISE*, vol. 41, no. 11, pp. 982-991, Nov. 2014.

최 진 영 (Jin-young Choi)

1989년 2월 : 서강대학교 전자계산학과 학사 졸업

2014년 3월~현재 : 세종사이버대학교 정보보호학과 석사과정

<관심분야> 정보보호, 소프트웨어 공학

장 남 수 (Nam Su Chang)

2002년 2월 : 서울시립대학교 수학과 학사 졸업

2004년 8월 : 고려대학교 정보보호대학원 석사 졸업

2010년 2월 : 고려대학교 정보경영공학전문대학원 박사 졸업

2010년 7월~현재 : 세종사이버대학교 정보보호학과 조교수

<관심분야> 공개키 암호 알고리즘, 공개키 암호 암호분석, 암호칩 설계 기술, 부채널 공격