FILESHADER: ENTRUSTED DATA INTEGRATION USING HASH SERVER

Juhyeon Oh and Chae Y. Lee

Department of Industrial & Systems Engineering, KAIST
juhyeonoh@kaist.ac.kr
cylee@kaist.ac.kr

ABSTRACT

The importance of security is increasing in a current network system. We have found a big security weakness at the file integration when the people download or upload a file and propose a novel solution how to ensure the security of a file. In particular, hash value can be applied to ensure a file due to a speed and architecture of file transfer. Hash server stores all the hash values which are updated by file provider and client can use these values to entrust file when it downloads. FileShader detects to file changes correctly, and we observed that it did not show big performance degradation. We expect FileShader can be applied current network systems practically, and it can increase a security level of all internet users.

KEYWORDS

SHA-1, File Integration, Web Cloud, File Transfer

1. INTRODUCTION

In the network system, file download and upload procedures are done by packet transfer. Usually file size is much bigger than normal packet size, and it causes packet fragmentation. When the file is delivered to other machines such as a server, a lot of fragmented packets will be transferred. However, some fragmented packet can be changed by eavesdrop and it could have a malicious attack pattern. In addition, entire file can be changed to malicious attack file during an uploading process. By this weakness of file transfer procedure, we need to detect file changes due to a possibility of file changes.

Hash server can be worked to solve this problem. Hashing has two advantages, speed and simple. Using hash value, we implemented FileShader which detects file changes before and after file transfer. It gets the correct hash value from the hash server who provided by original file provider, and compare with calculated hash value of downloaded or uploaded file.

While it checking file changes, a small overhead is possible. We evaluated this overhead and showed FileShader is practical and possible to be adopted to a current network system, so that user can prevent malicious attack pattern caused by file changes in the middle step of overall file transfer.

2. RELATED WORK

We proposed FileShader which is a hash-based solution for file security and network forensics issues. Below are previous related works.

David C. Wyld et al. (Eds) : CSITY, SIGPRO, AIFZ, NWCOM, DTMN, GRAPHHOC - 2016
pp. 15–25, 2016. © CS & IT-CSCP 2016 DOI : 10.5121/csit.2016.60402

2.1 Secured File System

The Secure File System (SFS) [1,2] provides strong authentication and a secure channel for communications. It includes an extensive authentication mechanism for individual users, and provides strong security for data in transit between clients and servers. It also allows servers to authenticate their users and clients to authenticate servers.

However, it still relies upon trusted file servers that data is stored by them. If a "trusted" server is physically compromised by the attacker, the data on it may be changeable to the attacker. In an environment where data storage is outsourced to companies, this security risk is unacceptable.

2.2 Network Forensics

Network forensics is the act of capturing, recording, and analyzing network audit trails in order to discover the source of security breaches or other information assurance problems. The term network forensics was introduced by the computer security expert Marcus Ranum in the early 90's [3], and is borrowed from the legal and criminology fields where "forensics" pertains to the investigation of crimes. According to Simson Garfinkel, network forensic systems can be implemented in two ways: "catch it as you can" and "stop look and listen" systems [4].

The main focus in this paper is to automate the process of detecting all the attacks and to prevent the damaged caused by further security issues. Our idea is to identify all possible security violations and prevention mechanisms to prevent further problems.

2.3 Hash Function

According to S. Bakhtiari[5], a one-way hash function, also known as a message digest, fingerprint or compression function, is a mathematical function which takes a variable-length input string and converts it into a fixed-length binary sequence. Furthermore, a one-way hash function is designed in such a way that it is hard to reverse the process, that is, to find a string that hashes to a given value (hence the name one-way.) A good hash function also makes it hard to find two strings that would produce the same hash value.

MD4 & MD5: Both MD4 and MD5 were invented by Ron Rivest. [6] MD stands for Message Digest. Both algorithms produce 128-bit hash values. MD5 is an improved version of MD4.

SHA: SHA[7] stands for Secure Hash Algorithm. It was designed by NIST and NSA. SHA produces 160-bit hash values, longer than MD4 and MD5. SHA is generally considered more secure that other algorithms and is the recommended hash algorithm.

3. DATA INTEGRATION

Data center is entrusted to people who have no role in creating and changing the data. Users trust these servers without any doubt. Security decisions are driven by preventing link level attack rather than internal data center security. However, most of recent security incidents are shown up by internal members. In particular, the data centers are operated by hired people who are physically insecure. They are able to change a data which is stored in a server. Thus, we can say data centers cannot be trusted anymore.

This paper proposes that how to transfer a file to user safely in unsecured and outsourced data center companies which is managed by people who have no permission to write data.

3.1 Security

The security is becoming a crucial part of the network. In the common sense, security has no deal with current network innovation. However, if the security is not managed, unimaginable disaster could be come up and it can cause huge damages to the network system.

Up to this point, most of the companies are increasing their investment to the security and trying to build security system by themselves, such as IDS (Intrusion Detection System) or firewalls. However, security is not an issue of companies anymore. All the network users are facing on the security issue. We suggest FileShader which can increase a security level of the entire internet user by using trusted hash server.

3.2 Data Integration

From the user's point of view, data integration checking by users themselves could have big overheads to their system itself. However, data integration should prevent various attacks and eliminate a potential security risk. Furthermore, the data integration process can be done without changing exist network device in the world.

By adding new secured hash server, data integration checking will be done automatically when the data transfer is finished. After that, the host can use the file from the unsecured server safely. If there are some suspicious bits or parts of the file, data integration process will let users know and let them to decide whether they discard file or not.

3.3 Case Study

The digital information such as image, video and data concerned with security issues require detection method for tampering data. Image can be modified easily for inappropriate propose. However, Image is very hard to authenticate its changes. Figure 1 shows how easy it is to modify and use. In forensic and criminal investigation, Digital data would be the most important evidence to determine the crime. In Figure 2, three suspects had disappeared in modified video without any unusual. Therefore, a valuable data such as contract, agreement and CCTV should be protected and verified by a trusted method.

Figure 1: Huh Kyung-young modifies his picture to increase his fame.

Figure 2: There are three men in first picture but no one in second picture

Some hackers insert malicious code into a normal program to make zombie computer. It can be used to perform malicious tasks of one sort or another under remote direction and launch Distributed Denial of Service (DDoS) attacks against targeted systems. Malicious software will typically install modules to zombie computer. Figure 3 shows that anti-virus programs warn you about malicious software. However, anti-virus programs can't find all malicious software because malicious software may delete itself, or may remain present to update and maintain the modules.

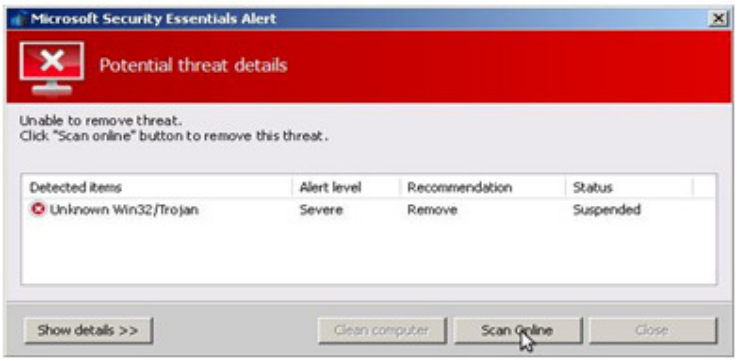

Figure 3: Trojan is flagged as malicious software and removed by anti-virus program.

4. PROBLEM DESCRIPTION

4.1 Hijacking Data

While the file transfer is processing, some malicious hackers can get the packet information doing Eves-Drop, and send the packet which has malicious code with same flow information, to the receiver who was getting the file. After the malicious packets are delivered, receiver side will reconstruct the file with malicious packets. When the receiver runs this reconstructed file, the host would be injected by virus or bots (Figure 4). Up to this point, we need file a integration process to ensure the file came from the source.

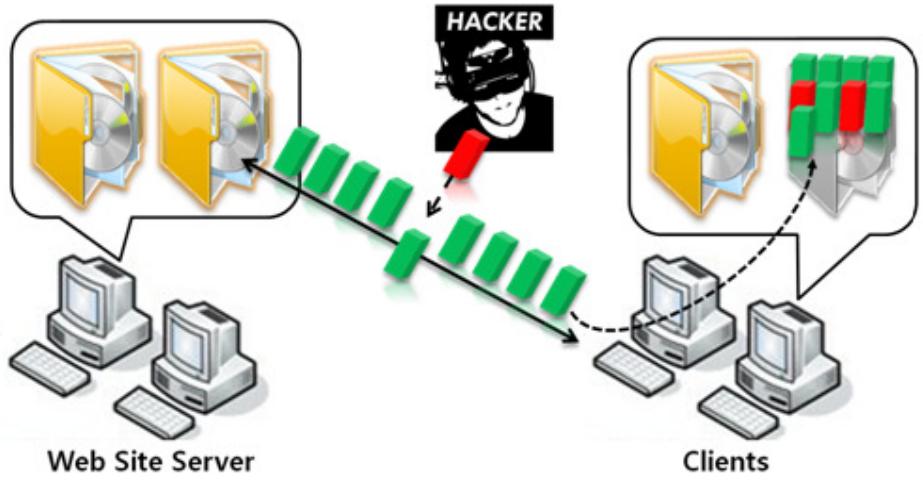

Figure 4: Receivers will construct the file with malicious packets.

4.2 Problem Definition

The problem of current network is file insurance. When the people download a file, no one knows this file is exactly same as the file which the file provider uploaded. File can be changed at a lot of spots and it can contain some malicious attack patterns. Up to this point, we need a file integration checking tool which can give us the fact that this file is certificated.

4.3 Assumption

Our problem starts with an assumption that transport layer has already supported sufficient security. Transport Layer Security (TLS) and Secure Sockets Layer (SSL) provide high communication security over the Internet. There is no longer the data modification in the current network.

Most of the recent security problems occur from the internal people or infected computer in a data server. Consequently, we presume the file has been changed in a server if there is the attacker who accesses to the data server. Furthermore, we suppose the hash server is trusted by users. The hash server operates in the high secured and validated authority. Hash server has the unique hash value updated by data owner.

4.4 Goal

The aim of our approach is to guarantee the file-integrity. We envisage warning clients when data has been changed and dealing with “forensics” issue. In the current network, it is hard to detect file changes due to the file integration process. Let’s say, the file is fragmented to some packets, and one of them are changed by eavesdrop or some other attack. The receiver cannot detect file changes after it integrated the packets to one file.

Up to this point, we propose the FileShader which can detect file changes not in the packet level but in the file level using hash values. Another possible issue is a malicious file provider. When the clients are attacked by malicious data, it is important to figure out who provides the malicious data on the web server. We expect our system is able to identify the exact malicious file provider.

5. SOLUTION

5.1 Problem Approach

Being able to access your files from anywhere and from any computer is one of the great conveniences of the always-on Internet. These are so cheap and easy to use that there is almost no reason not to back some of your files up into the cloud. Most online storage providers also give you the ability to then share these files with your friends and colleagues. Cloud services have a good track record of keeping your data while providing you easy access to your files from wherever you are. However, it's dangerous to put data in the cloud as it becomes controlled by the provider and the company can’t really know how secure the data and the infrastructure are in general.

Therefore, Client wants to independently prove the integrity of their data when they use public or sharing data in a web and cloud services environment. Cloud Services platforms can use data integrity protection of its storage but don’t provide the guarantee of modification caused by an inappropriate user. For customers to have confidence in cloud service implementations, File provider need to build security into the structure of their clouds and not totally rely on cloud

vendors to provide those capabilities. Security in the cloud is ultimately the responsibility of the file provider use cloud vender. File provider will provide security with their customer. Table 1 shows the comparison of each service.

Table 1. Comparative benefit analysis

	Self-managed Data	Cloud-managed Data	Self-managed Data+FileShader	Cloud-managed Data+FileShader
COST	×	○	×	○
RISK	△	×	○	○
TRANSPARENCY	△	△	○	○
LEGALLY DEFENSIBLE	×	×	○	○

5.2 FileShader Design

FileShader operates a data integrity protection service that enables users to protect the integrity and independently verify of their data, files or any digital content throughout hash algorithms. We design that FileShader is an Internet-based application that can work from any platform without a lot of changes. We propose the effective process to provide data insurance you can get from any server. Figure 5 shows our process of FileShader.

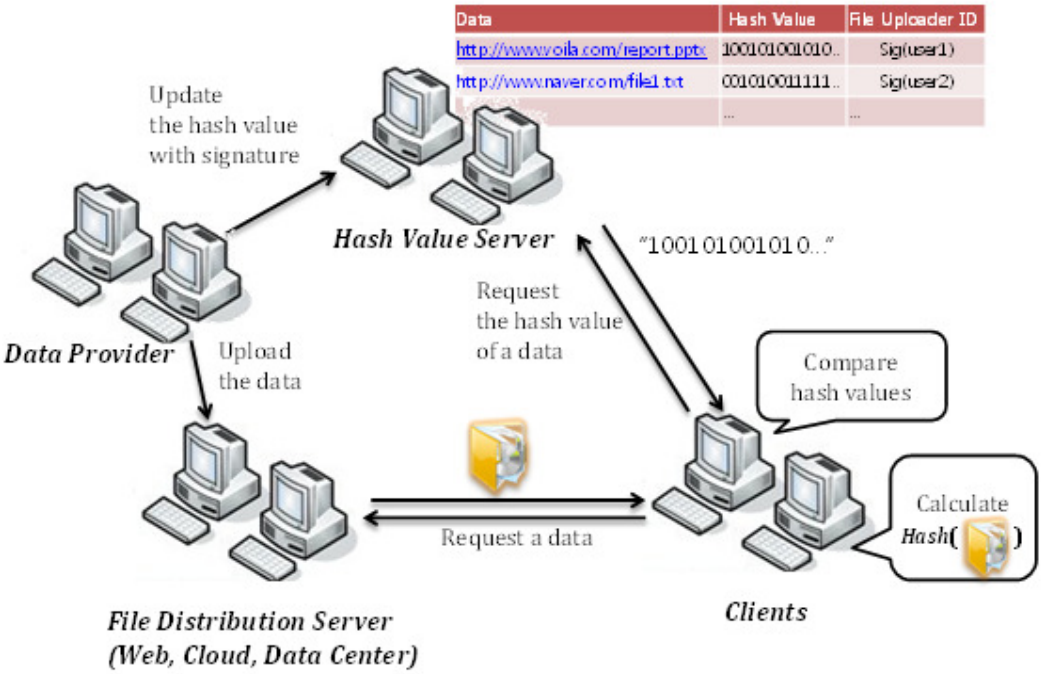

Figure 5: Integrity of a file is checked by comparing two hash values

FileShader Sealing and Validating Process

1. FileShader Sealing Process
   A. Data Provider creates a hash of a data or file from
   B. FileShader sends Hash to hash value server, via secure Internet connection such as SSL

C. Hash value server securely and verifiably bind the hash, and data provider to create the data integrity seal.
D. Hash value server sends the data integrity seal to the data provider
E. FileShader of data provider verifies and archives the data integrity seal.

2. FileShader Validating Process
   A. Client create a hash value of a file to compare it to the hash value in Hash value server
   B. Client sends a meta-data to Hash value server.
   C. Hash value server receives a meta-data from client and finds the data integrity seal.
   D. Hash value server sends the data integrity seal to the client. And then client compares it and the data integrity seal from validation process.

6. EVALUATION

6.1 Key Factor

Correctness: verifying file changes and show FileShader can detect file changes with exact hash values.

Performance: comparing between normal network downloading speed without FileShader and downloading speed with FileShader. It describes FileShader will not take that much overhead so that the user will not feel any downloading speed degradation and also FileShader is practical.

6.2 Method

We will evaluate the detection accuracy as well as the system overhead. To evaluate how accurately the system detects the data changed, the data center generates partially changed data. This data constructs a data pool with normal data. Then, the data center sends any data between this pool to the client. We check how many times the client detects the file changed accurately. This will show security robustness of our proposed approach.

As far as the system overhead is concerned, we will record the time taken when sending data with and without our proposed file integrity method. Accordingly, the system overhead is evaluated by the number computed by the time taken using our approach over that using existing approach. Since we add the process of calculating hash values, the number is over 1. Our objective is to reduce this number as possible as we can.

When using 128-bit hash values, we expect the system to detect the file changes100% since it is unlikely the hash collision is occurred. The size of hash values is relevant to the system overhead, however. To deal with the tradeoff between robustness and overhead, we will experiment by reducing the size of hash values. We will find the optimal point having small enough size where the detection rate is over the determined threshold.

6.3 Environment Setup

We prototyped client, web server, and hash value server. Different IP addresses and port numbers are assigned to web server and hash value server. Client tries connecting with these servers with two different sockets. Client has two functions: upload/download a file to a web server. Unlike

other client/server systems, client connects with hash value server. When uploading a file to a web server, client computes the hash value of the file and uploads the value to hash value server. Similarly, client receives the corresponding hash value from a hash value server as soon as downloading a file from a web server. Then, it compares the hash value to the newly computed hash value with the downloaded file. SHA-1 hash algorithm outputs completely different hash values if contents of file is changed. Consequently, client can easily validate files from a web server by comparing two hash values.

Our objective of the experiments is to reduce the overhead between the systems with/without the hash value server. To compute the overhead by calculating, uploading, and downloading the hash values; we measured the time taken with and without hash value server. Since the network is dynamic, execution time strongly depends on the network environments. File size may also affect the overhead. Accordingly, we tested a file for 10 times and calculated the average execution time.

7. RESULT

We implemented FileShader using C# to provide graphical user interface, so that user can easily use the FileShader and also FileShader can be more practical. We had a test with various file sizes; 64 byte, 512 byte, 1kB, 1MB, and 2MB.

7.1 Environment Setup

Basic user interface of FileShader is as shown below. When we put the file address at the file path section, FileShader will be automatically connected to web cloud severs and hash server. From the web cloud server, FileShader will download the file and calculate the hash value. On the other hand, from the hash server, FileShaer will send a request message to get the hash value of the file and wait. After FileShader gets the hash value from the hash server, it will compare both hash values, so that we can verify the downloaded file.

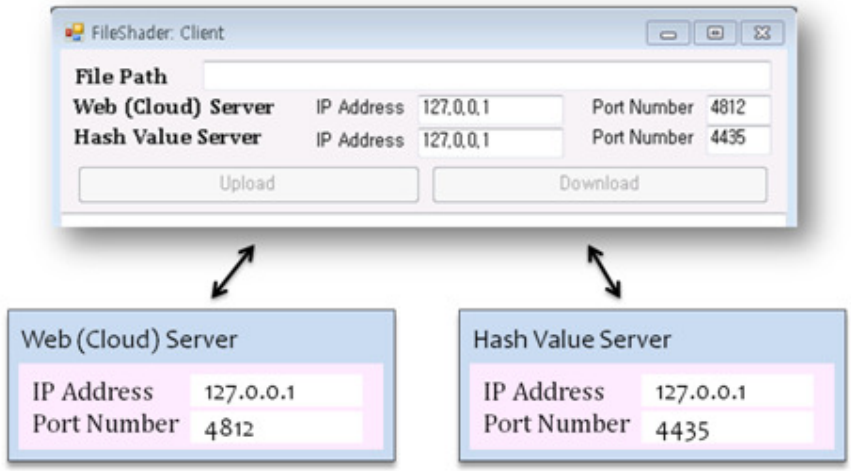

Figure 6: user interface

The next two pictures describe the correctness of the FileShader. We used file1.txt which has the data as shown left. The client connects to the hash server and gets the hash values to compare with file1.txt. The hash values were same, so the FileShader says “client_file1.txt is safe” which means this file is same as the file which uploaded by original file provider.

Figure 7: correct result of file1.txt

What if the file data is changed? We tried to change file1.txt manually with adding one question mark, '?', and kept experiment. FileShader calculates the hash value of changed file and compares with the hash value which comes from the hash server. In this time, two hash values are different, so it prints out file1.txt is not safe. This means this file is different from the file provider's uploaded file even the file name is same.

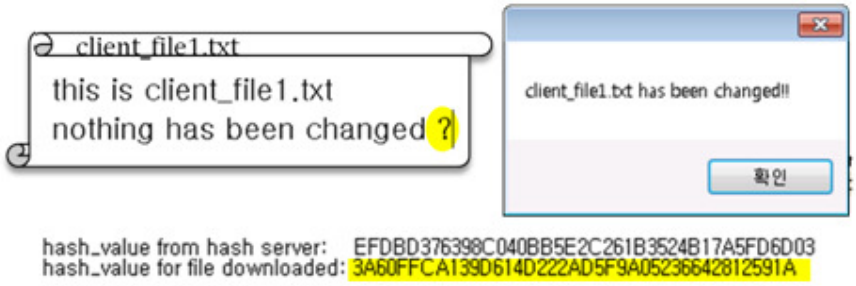

Figure 8: wrong result of file1.txt

FileShader uses hash value to verify the file. Thus, there is no weakness of file name and file size. Even the file name and file size is same, FileShader will detect the file changes.

7.2 System Overhead

We have done some experiments for the performance of Shader on a typical network system. We quantified the overheads of FileShader with various file sizes. The file size was changed from 64kB to 2MB. Figure 9 shows the system overhead with different file size in a file downloading process. The FileShader has a system overhead in a hash calculation and file verifying process. As we can see that normal download performance and FileShader performance has not big difference, less than 100ms, regardless of file sizes. This means that hash value calculation doesn't overheads a lot, and file verification is also light work due to the comparing hash values only.

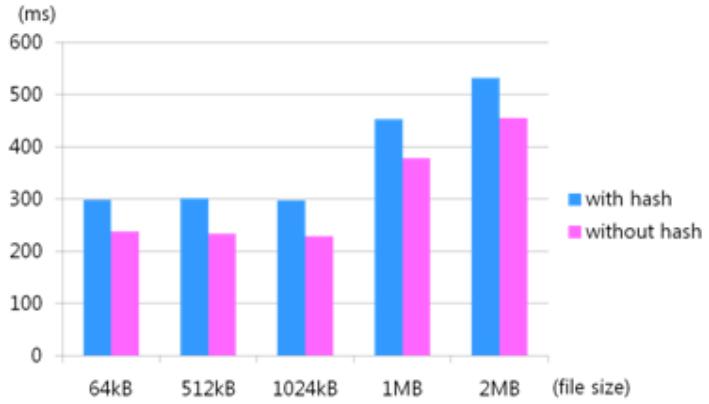

Figure 9: Overhead comparison of the download

Figure 10 shows the overhead of the uploading process. As we can see in the figure, overall system overheads are increasing when the file size is increased. The process of "without hash" is a normal uploading process. The difference between "with hash" and "without hash" is an overhead of FileShader. When a file is uploaded, FileShader should calculate the hash value of the entire file and store it in the hash server, so that it can use this value when the user downloads the file at the other machine through cloud interface. This process takes overheads, that is why, the system overheads is increased regards of file sizes.

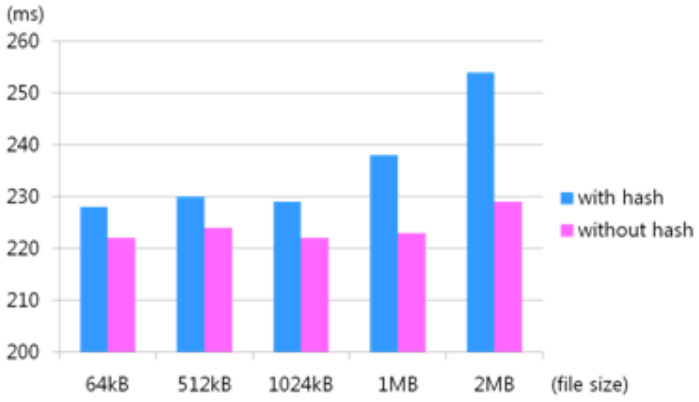

Figure 10: Overhead comparison of the upload

When downloading, client requests a file to the web server. The buffer size of client affects system overheads while in our experiment, two buffer sizes, of uploading and downloading, are different. Consequently, the downloading overheads are bigger than that of uploading.

This experiment has been done with simple file to test FileShader. It shows that the FileShader detects file changes correctly, and also it is totally acceptable to current network system.

8. CONCLUSION

We proposed FileShader which provides entrusted file integration and transfer using hash server. It detects file changes correctly, and performance degradation was negligible. We evaluated FileShader with correctness point of view, and performance point of view. With the evaluation

result, we showed FileShader is practical and can be applied to current network system. We expect FileShader can give a higher security level for the file transfer and efficiently prevent malicious attack from the implied attack pattern in the file which is injected in the file transfer process.

REFERENCES

[1] K. Fu, M. F. Kaashoek, and D. Mazieres, “Fast and secure distributed read-only file system,” 4th Sym-posium on Operating Systems Design and Implementation (San Diego, CA), October 2000, pages 181-196.

[2] D. Mazieres, M. Kaminsky, M. Kaashoek, and E.Witchel, “Separating key management from file system security,” Proceedings of the 17th ACM Symposium on Operating System Principles December 1999, pages 124-139.

[3] Marcus Ranum, Network Flight Recorder. http://www.ranum.com/

[4] Simson Garfinkel, Web Security, Privacy & Commerce, 2nd Edition.

[5] S Bakhtiari, "Cryptographic Hash Functions: A Survey", Department of Computer Science, University of Wollongong, 95-09, July 1995.

[6] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[7] D. Eastlake 3rd and P. Jones., “US Secure Hash Algorithm 1 (SHA1)”, RFC 3174, September 2001.

AUTHORS

Juhyeon Oh received the B.S. degree in Industrial Management Engineering from Korea University, Seoul, Korea in 2009 and M.S. degree in Industrial & Systems Engineering from Korea Advanced Institute of Science and Technology (KAIST) in 2011, Daejeon, Korea He is currently a Ph.D ca ndidate at KAIST. The focus of his current research is network optimization and video streaming over wireless networks.

Chae Y. Lee is a professor of Industrial Engineering at KAIST, Daejon, Korea. He received the B.S. degree in Industrial Engineering from Seoul National University,Seoul, Korea in 1979, and the M.S. and Ph.D. degrees in Industrial Engineering from Georgia Institute of Technology, Atlanta in 1981 and 1985, respectively. He is a member of INFORMS and IEEE Communication Society. His research area includes wireless and mobile communication networks, Internet data communications, heuristic search and optimization. He has published numerous papers in journals related to wireless communications, Operations Research and optimizations.