Implementation of Digital Signature for Maintaining File Integrity and Authentication

Feryandi Nurdiantoro - 13513042
Informatics / Computer Science
School of Electrical Engineering and Informatics
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
feryandi.n@gmail.com

***Abstract* — This paper aims to provide applicative usage of digital signature in daily usage, not only in network and internet based systems. Using digital signature, we can protect the integrity of the files from alteration and corruption of the data and could provide an early warning of possible virus and malware inside the file and could prevent it by delete the files or simply not open the files. Other usage for this type of digital signature is the file could provide the owner or creator of the file. The digital signature this paper provide is using ECDSA algorithm to provide secure signature and using SHA for digesting the file data.**

***Keywords* — ECDSA, Digital Signature, SHA, File Integrity, Security, Non-repudiation.**

I. Introduction

Transferring files between two or more hardware is essentials for our needs to exchange and spread the information through computers. Exchanging data could be dangerous in many ways such as alteration by viruses and malware, data alteration by someone in the middle of transmission, or the owner or creator of the files is unknown or cannot be identified.

In physical world, we could identify the sender of an item with writing names in the package and to keep the validity even more, the sender could give it his signature. Once the package is opened, the seal is broken and marked the package as have been altered by other than the owner. This could be a problem in digital world because everything is represented in bit, and anyone or anything could alter the bit so it could represent different data.

To achieve this in digital application, there is a method called digital signature. Digital signature is a chunk of data which contain the identity of the owner of the transmitted data and the data itself. Usually, digital signature is using asymmetric cryptography to prevent duplication or reconstruction of the signature by unintended party. The signature is generated using creators' or owner's private key, and validated using their public key and the file itself.

Digital signature like this usually used in internet protocol like HTTPS to ensure the integrity and identity of the website you'd open, this is one type of prevention method from scam websites and phishing. We encounter this mechanism everyday on line, but not off line when we using hardware such as hard drive, flash drive, CD or DVD as transfer media, we often don't really care about what's inside our drive even though these media is really vulnerable to viruses, malware or altered by third party intentionally or unintentionally. To prevent this unintended integrity loss and repudiation, this paper will try to prove an applicable solution to ensure the data's integrity is safe.

II. Literatures Study

A. Elliptic Curve Digital Signature Algorithm

Digital signature is one form of signature that in some countries legally accepted and as legal as physical signature. Digital signature, same as physical signature, is used to sign authenticity, integrity and non-repudiation of some data or items. In digital world, the signature could be attached to the file using steganography technique, or as simply as create a new signature file.

Elliptic Curve Digital Signature Algorithm (ECDSA) is an expansion of basic Digital Signature Algorithm using Elliptic Curve to enhance the security. ECDSA is consists of three phases, key generation, signature generation and signature verification. Before the phases begin, user must choose a set of elliptic curve which consist of,

a. Elliptic curve equation E, $y^2 = x^3 + ax + b$, where a and b are defined.
b. A prime number, p, which will be domain of the curve over field $F_{p.}$
c. Base point G on the selected curve
d. The prime order n of point G
e. Cofactor h of point G

Key Generation

In order to generate the public and private key, user must do the following steps:

1. Select a random integer d, where d is integer between 1 and n-1
2. Compute Q = d . G
3. User's private key is d and user's public key is Q

After the generation, user must keep his private key in safe in order to securely using the ECDSA. His public key may be distributed across unsecure channel. The public key is used to verify the signature and the private key is used to generate signature.

Signature Generation

To generate the signature for a message M, user must do the following steps:

1. Select another random integer k on between 1 and n - 1.
2. Compute (x, y) = k . G
3. Compute r = x mod n.
   If r is 0, then go back to step one.
4. Compute s = k^{-1} (hash(M) + d . r) mod n
   If s is 0, go back to step one.
5. Signature for the message M is pair of integers of (r, s)

This pair of integers (r, s) is what we called digital signature. It could be attached in some files or in the message itself to prove the authenticity.

Signature Verification

To verify a signature from message m, recipient must obtain the value of sender's public key, and the set of curve used to generate the signature then do this steps:

1. Verify that r and s are integer between 1 and n-1
2. Compute i = s^{-1} mod n
3. Compute j = hash(M) . i mod n
4. Compute k = r . i mod n
5. Compute (x, y) = j . G + k . Q
6. Compute v = x mod n
7. The signature is verified when the value of v = r, otherwise the signature is invalid.

B. File Checksums

File is consist of data and bits that could be read through. This is the same as string when we try to digest them with some hash function like SHA or MD5. The difference between digesting the file and a string is on the metadata of the file. A file contained with data is not only contain the data itself, but also the metadata. Metadata is information of the data the file contained. The example of metadata is file size and timestamps.

An image file when opened by some image viewer could looks the same as other image that have been on steganography. But deep inside, the data has been changed. With the hash function, this data could be digest and the function will return different value for each file. Even a slight change in metadata could make a big difference in the digest value. Hash collision are technically possible, but it is extremely unlikely and have little probability to appear as two files with different value even with same digest could not represent a same data to user so it's very unlikely become a problem.

The digests itself is commonly called checksums, to check whether is the file is same file or not. If the checksums is different, the file must be a different file. But if the checksums is same, it doesn't entirely mean the file is same.

III. Proposed Analysis And Solution Designs

Integrity of the file could be proven by the checksums of the file itself. Many websites using this method to get trusts from their users, using checksums and show it on the website so users could validate it later after downloaded.

This method has some design flaws, such as the file distributor could show wrong checksums, or change the original file with some malware and put the malware checksums so the file has right checksum but entirely different functionality. This is happened because the checksums is easily generated using well-known hash function and anyone could alter it.

To prevent this happening, a method called digital signature could help verify the integrity of the checksums and identify the signature's issuer.

A. ECDSA Using File Checksums

In normal ECDSA usage, users use message in a form of string as thing that ECDSA need to protect and certified. In this case, the program will use combined checksum of all files in the selected folder to generate the signature needed to protect it.

User will be able to select folder that need to be protected. But it's not restricted only to folder, user could select a flash drive to be protected too. After the user select the location of files, every files in that location will be digest with SHA and collecting their checksums. All of this checksums will be combined to one large string and then this string will be digest again with SHA to get the last digest that will be used for the ECDSA.

This last checksum will be saved as a new file in the root folder. Also, this file is indicated as the folder has been signed and protected so every tamper or change applied in the folder could make the entire folder lose the integrity. The signature file is consist of identifier that the signature is a signature file using the file extension and identifier byte. The file is started with 0x44 0x53 which in ASCII represented as "DS" and then followed by the value of r and s. Each segment of these parameter will be separated with 0x0A. This file format also could be expanded as needed and if a new features is implemented.

B. Identification and Validation Design

The identification process is begin with user request to check the folder. If a signature file is found within the folder, the application will check the value and validate it. If the software could find the signature file, folder could not be verified and only marked as vulnerable and cannot be verified, so the user could procced but without any guarantee that the folder has not been altered.

Validation process is begin shortly after the identification of signature file is success. The validation mechanism is using the validation provided by ECDSA. First, the selected location will be digest using the same method to generate digest from folder using the file checksums. After that, the folder digest hash value is used to verify the signature. The public key is also used to verify the signature. Public key is could be transferred using unreliable and unsecure connection, thus not handled by this application. This application is assumed that public key is could be transferred to the recipient and cannot altered nor be repudiated.

Another feature that could be implemented is alerting the user about invalid signature and integrity. This could prevent user accessing an external drive that has been altered by viruses, malware or manually by other people. So this also could prevent the computer from vulnerable files.

C. *Curve Standard*

Another problem faced when using ECDSA is the curve equation used in process of generating signature. With different curve standard definitely will change the signature significantly. To overcome this problem, program that will be implemented in this paper will be using Standard for Efficient Cryptography (SEC). The curve from SEC that will be used is secp256k1 with parameter explained below.

Table 1 Curve Parameter

Parameter	**Value**
Prime	FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F
A	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
B	00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000007
G	04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8
n	FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141

The curve secp256k1 will be used as default parameter curve in the application, but doesn't rule out the possibilities of using other standard curves, as in the proposed signature file there is space for the name of curve used for generation.

IV. Implementation And Experiments

A. *Implementation*

Language used to experiment and implement the proposed application in this paper is Java and using Java Swing as user interface. The program also using Path, File and BufferedReader library class for reading the file or do the recursive search on the target directory. There are two tabs where user can switch in between, the first one is for generating the digital signature for the targeted directory or file and the second one is for checking and verifying. User also could generate new key every time he wants to make a new signature. The private and public key will be saved as key file in the same directory with the digital signature file.

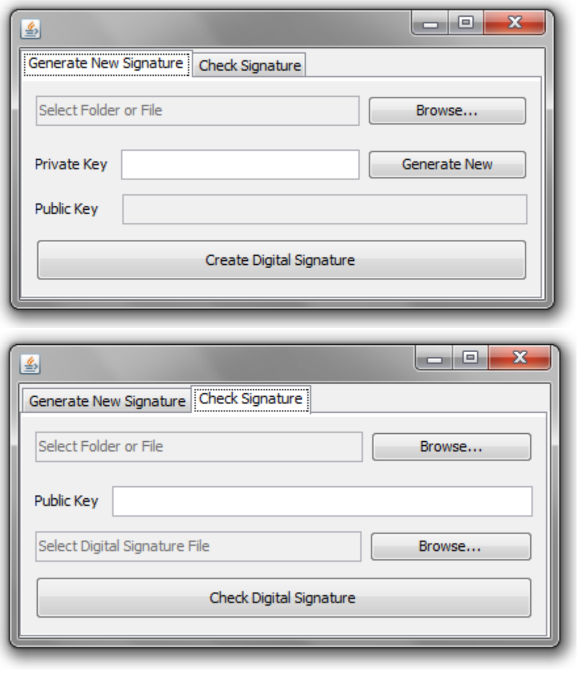


Figure 1 Program Interface

B. *Experiment*

As stated before, the goal of using this mechanism is to prevent security breach when transferring the files using unsecured channel or communication protocols. To achieve this goal, main security concepts must be covered. In general cases of transferring file, the main security concepts are integrity, non-repudiation, and authenticity. The confidentiality and availability aspect of security is not covered in this approach.

To verify this concepts, there are some experiments used to prove them and will be explained in this paper, there are:

1. Unaltered files and valid digital signature
2. Altered files and valid digital signature
3. Unaltered files and invalid digital signature
4. Altered files and invalid digital signature
5. Invalid public key for verification
6. Opening the files but not altering anything

Those experiments use more than file in more than one directory so will be useful when transferring files via hard drive or flash drive.

For his experiments, valid pair of public and private key is:

Table 2 Key Used

Private	d925ade5ffa5d233
Public (x, y)	caa46d10920bcf3eb38faae3838bd3348b6252409d1870c17e5dafe8d0c5b910
	7b4d0c20cc079479c94c6a75a1dbfbee80a9afe75733897deef2c5a4ff10e577

V. Experiment Result And Analysis

Files and directory used in this experiment are described in the following table.

Table 3 Folder Content

Location	File Name	Size
root	proposal makalah.docx	14 KB
root	undertale.png	10.6 MB
root/penting	Great Work.mp4	7.25 MB
root/penting	README.txt	5 KB
root/penting	Sosif.pptx	382 KB

A. *Unaltered files and valid digital signature*

In this case, the environment is in the best case scenario. The files is unaltered and embedded by valid digital signature.

Table 4 Experiment A

Total Files Digest Value
37a471f3b1c4fa7a5747c11c0292e7fc3d645928
Digital Signature File Content
DS04f7d911ed55852bc0c541b7b256a67223387bba858977a93608882c4e03a53a64faea1a9b8b60a8a2548eb66850f453c001681d1fef8b54b89ce36212cd9ddb01

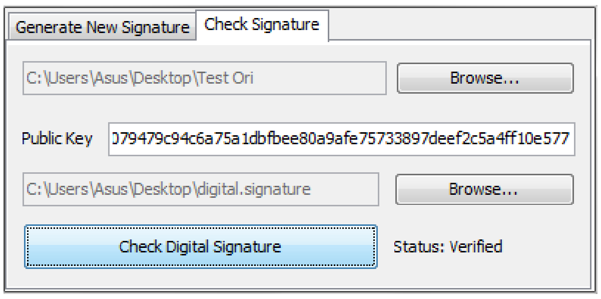

Generate New Signature | Check Signature
C:\Users\Asus\Desktop\Test Ori — Browse...
Public Key: 079479c94c6a75a1dbfbee80a9afe75733897deef2c5a4ff10e577
C:\Users\Asus\Desktop\digital.signature — Browse...
Check Digital Signature — Status: Verified

Figure 2 Result from Experiment A

The program said that the file and signature is match, so it's verified. This case is just showing that file verification with this method is feasible and could be used to protect your files.

B. *Altered files and valid digital signature*

This case is experimented by changing the content of the file insignificantly small. In this experiment, the proposal makalah.docx file is altered, and just one word altered.

Table 5 Experiment B

Total Files Digest Value
5a72253aed13de60012bf91862dbbaeb55310559
Digital Signature File Content
DS04f7d911ed55852bc0c541b7b256a67223387bba858977a93608882c4e03a53a64faea1a9b8b60a8a2548eb66850f453c001681d1fef8b54b89ce36212cd9ddb01

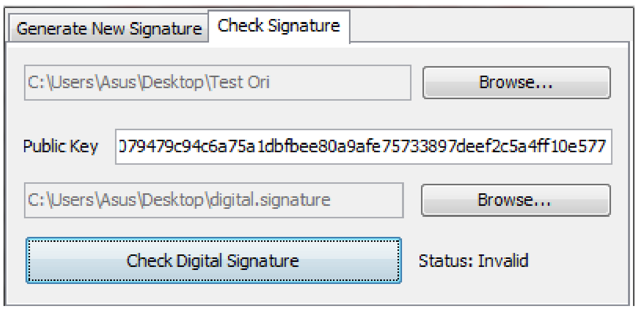

Generate New Signature | Check Signature
C:\Users\Asus\Desktop\Test Ori — Browse...
Public Key: 079479c94c6a75a1dbfbee80a9afe75733897deef2c5a4ff10e577
C:\Users\Asus\Desktop\digital.signature — Browse...
Check Digital Signature — Status: Invalid

Figure 3 Result from Experiment B

The impact on total files digest value is significant and really different. So the validation mechanism could detect it as alteration on the folder.

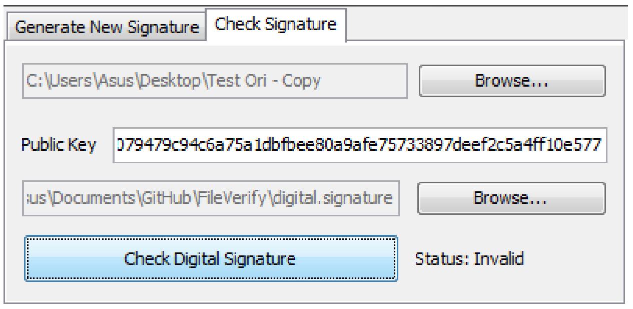
C. *Unaltered files and invalid digital signature*

In this case, valid digital signature is altered with other invalid signature but the files remain safe and be the same.

Table 6 Experiment C

Valid Total Files Digest Value
95ec6a0581eb72b7b74b149cfc368002489c6d08
Valid Digital Signature File Content
DS045e095e8ad830fe21fe1c4f45b3559772f820b7c3add551b1303992cf203001fe1fbb02bf3bce902a89ceb53eb9c0c6419914b4a885777db925edfa0041ab91be
Invalid Digital Signature File Content
DScc5ec95e8ac83cfe21fe1ccfc5b3559cc2f82cbcc3acc551c13c3992cf2c3cc1fe1fbbc2bf3bce9c2a89ceb53eb9ccc6c1991cbca8

85ccccb925ecfaccc1ab91be



C:\Users\Asus\Desktop\Test Ori - Copy

Public Key 079479c94c6a75a1dbfbee80a9afe75733897deef2c5a4ff10e577

:us\Documents\GitHub\FileVerify\digital.signature

Status: Invalid

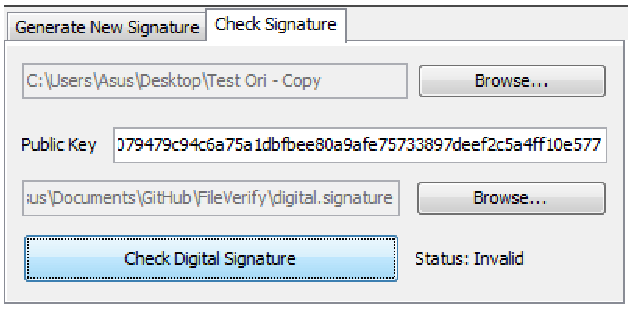
Figure 4 Result from Experiment C

D. Altered files and invalid digital signature

In this case, both data is altered or modified. This is represent a worst case scenario.

Table 7 Experiment D

Valid Total Files Digest Value
95ec6a0581eb72b7b74b149cfc368002489c6d08
Invalid Total Files Digest Value
611993704f527927b34b0f4ad8e959722f17d85b
Valid Digital Signature File Content
DS045e095e8ad830fe21fe1c4f45b3559772f820b7c3add551b1303992cf203001fe1fbb02bf3bce902a89ceb53eb9c0c6419914b4a885777db925edfa0041ab91be
Invalid Digital Signature File Content
DScc5ec95e8ac83cfe21fe1ccfc5b3559cc2f82cbcc3acc551c13c3992cf2c3cc1fe1fbbc2bf3bce9c2a89ceb53eb9ccc6c1991cbca885ccccb925ecfaccc1ab91be



C:\Users\Asus\Desktop\Test Ori - Copy

Public Key 079479c94c6a75a1dbfbee80a9afe75733897deef2c5a4ff10e577

:us\Documents\GitHub\FileVerify\digital.signature

Status: Invalid

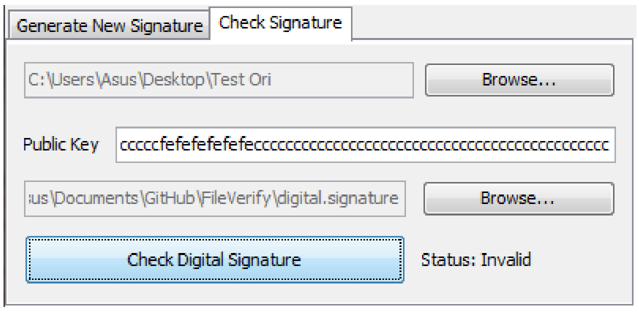
Figure 5 Result from Experiment D

E. Invalid public key for verification

Public key is an important part for this verification mechanism to work. Public key represent the user or party whom signed the file. So with invalid public key, the signature must not be valid. This also test for the non-repudiation mechanism.

Table 8 Experiment E

Invalid Public Key
fefefefefefeccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccfefefefefefeccccccccccccccccccccccccccccccccccccccccccccc



C:\Users\Asus\Desktop\Test Ori

Public Key ccccfefefefefefecccccccccccccccccccccccccccccccccccccccccccc

:us\Documents\GitHub\FileVerify\digital.signature

Status: Invalid

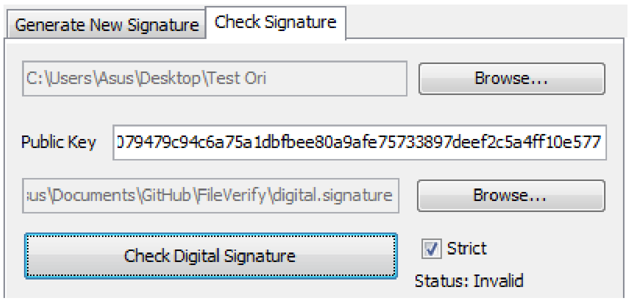
Figure 6 Result from Experiment E

F. Altering files but undoing the alter

This case is done by altering the README.txt file in Sublime Text or other editor, then undoing the alteration not using the CTRL+Z shortcut but retype the original text.

Table 9 Experiment F

Valid Total Files Digest Value
1d0f49e7b81de8b0be6d9d1ef0c7c5931f448c49
Invalid Total Files Digest Value
ee7150ef0a73cef4cdd05b85171e43a3dd526e83



C:\Users\Asus\Desktop\Test Ori

Public Key 079479c94c6a75a1dbfbee80a9afe75733897deef2c5a4ff10e577

:us\Documents\GitHub\FileVerify\digital.signature

[✓ Strict]

Status: Invalid

Figure 7 Result from Experiment F

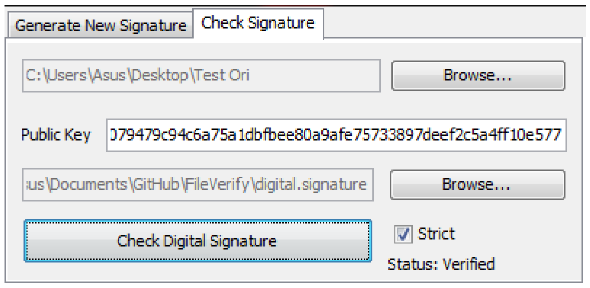
This feature could be enabled with strict mode in application. The strict mode is calculating the modified time and accessed time of the file served by the operating system's file system metadata. So any alteration even the alteration is not alter the data, will be counted as integrity breach.

G. Opening the files but not altering anything

This case is to prove that the file is haven't been read by someone else. Even this not the main concern here, but good for security and the concept of confidentiality.

Table 10 Experiment G

Total Files Digest Value Before Experiment
58905073b878503838c3 9e8cb1a33c2ebe7212a7
Total Files Digest Value After Experiment
58905073b878503838c3 9e8cb1a33c2ebe7212a7



C:\Users\Asus\Desktop\Test Ori

Public Key 079479c94c6a75a1dbfbee80a9afe75733897deef2c5a4ff10e577

us\Documents\GitHub\FileVerify\digital.signature

☑ Strict

Status: Verified

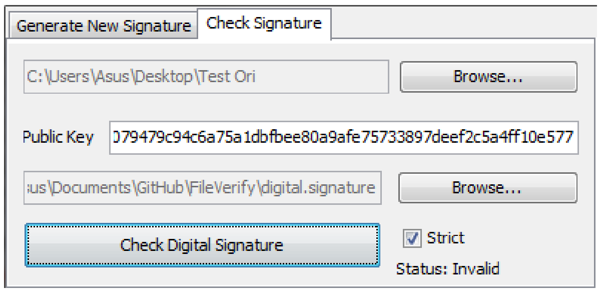
Figure 8 Result from Experiment G

The result is still valid even though the file has been accessed by opening it. This is because the every type of file system record the access time differently and not in the same manner. Windows, the operating system environment that used to run the experiment, is updating the access time every 1 hour in NTFS and 1 day in FAT. This is causing this invalid validity effect on the mechanism because the mechanism could not really rely on the file system metadata. But it will still works on case where the file is not opened in a day or so.

This case could be still tested, by copying the same file, delete the old one, and then rename it to the original one. This wouldn't change the modified time but will only change the accessed time. The result is explained below.

Table 11 Another Experiment G

Total Files Digest Value Before Experiment
58905073b878503838c3 9e8cb1a33c2ebe7212a7
Total Files Digest Value After Experiment
8aa4139a438a0fd949cc 9fbeb9d7f4663f8ddcb7



C:\Users\Asus\Desktop\Test Ori

Public Key 079479c94c6a75a1dbfbee80a9afe75733897deef2c5a4ff10e577

us\Documents\GitHub\FileVerify\digital.signature

☑ Strict

Status: Invalid

Figure 9 Result from Another Experiment G

VI. Conclusions

Digital signature could be used to maintaining file integrity and authenticity using the file digests with hash function. The metadata served by file system also help to prevent modification and access but every file system has their own manner to maintain this metadata so it should be relied too much.

Another application for this implementation is detecting viruses after lending or connecting flash drive or hard drive to other people or computer. So when the signature is not valid, user could prevent the virus to spread by not opening the drive or reformat it.

References

[1] http://searchsecurity.techtarget.com/definition/digital-signature accessed on May, 7th 2016 at 1:20 pm
[2] http://www.ijcaonline.org/volume2/number2/pxc387876.pdf accessed on May 7th 2016 at 2:45 pm
[3] http://www.secg.org/SEC2-Ver-1.0.pdf accessed on May 9th 2016 at 3:00 pm
[4] http://fotoforensics.com/tutorial-digest.php accessed on May 9th 2016 at 3:10 pm
[5] https://msdn.microsoft.com/en-us/library/windows/desktop/ms724290.aspx accessed on May 17th 2016 at 10:00 pm

Originality Statement

I hearby declare that this paper is my own writing, not an addaptation, nor translation of others' paper, and not plagiarism.

Bandung, 17th May 2016

[illegible]

Feryandi Nurdiantoro
13513042