An Implementation and Analysis on Elliptic Curve Digital Signature Algorithm and Its Variants

Pipin Kurniawati
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
13513089@std.stei.itb.ac.id

***Abstract*—This paper aims to provide implementation and analysis of digital signature in some schemes. These days, the usage of digital signature can protect the integrity of the files from alteration and corruption of the data, could provide an early warning of possible virus and malware inside the file and also could prevent it by deleting the files or simply not open the files. Other usage for this type of digital signature is to authenticate the actual owner or creator of the file. The digital signature this paper provided is using ECDSA and two of its variants to provide secure signature and using SHA1 for digesting the message.**

Keywords—digital signature, ECDSA, SHA1, authenticate.

I. INTRODUCTION

The concept of securing messages through cryptography has a long history. Indeed, Julius Caesar is credited with creating one of the earliest cryptographic systems to send military messages to his generals.

Throughout history, however, there has been one central problem limiting widespread use of cryptography. That problem is *key management*. In cryptographic systems, the term *key* refers to a numerical value used by an algorithm to alter information, making that information secure and visible only to individuals who have the corresponding key to recover the information. Consequently, the term key management refers to the secure administration of keys to provide them to users where and when they are required.

Historically, encryption systems used what is known as symmetric cryptography. Symmetric cryptography uses the same key for both encryption and decryption. Using symmetric cryptography, it is safe to send encrypted messages without fear of interception. However, there always remains the difficult problem of how to securely transfer the key to the recipients of a message so that they can decrypt the message. A major advance in cryptography occurred with the invention of public-key cryptography. The primary feature of public-key cryptography is that it removes the need to use the same key for encryption and decryption. With public-key cryptography, keys come in pairs of matched "public" and "private" keys. The public portion of the key pair can be distributed in a public manner without compromising the private portion, which must be kept secret by its owner. An operation (for example, encryption) done with the public key can only be undone with the corresponding private key.

The invention of public-key cryptography was of central importance to the field of cryptography. It provides secure encryption and 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 the corresponding public key.

However, like any other inventions in cryptography, digital signature has disadvantages too. Elliptic Curve Digital Signature Algorithm is one of the digital signature schemes and is the most secure digital signatures scheme according to many cryptologist. These days, many researches are developing different variants of ECDSA resulting many new variants of ECDSA.. This paper implements, analyzes, and describes three different variants of ECDSA.

II. FUNDAMENTAL THEORIES

A. Elliptic Curve Cryptography

Elliptic curve cryptography is based on the arithmetic of points on an elliptic curve[3]. Elliptic curves are represented by cubic equations similar to those used for calculating the circumference of an ellipse. An elliptic curve E over a field K is defined by a equation[1]:

$$E: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \dots\dots\dots\dots$$

Where α_1, α_2, α_3, α_4, $\alpha_6 \in K$ and $\Delta \neq 0$, where Δ is defined as follows

$$\Delta = -d_2^2 d_8 - 8d_4^3 - 27d_6^2 + 9d_2 d_4 d_6$$

where

$d_2 = a_1^2 + 4a_2$, $d_4 = 2a_4 + a_1 a_3$, and
$d_6 = a_3^2 + 4a_6$ and $d_8 = a_1^2 a_6 + 4a_2 a_6 a_1 a_3 a_4 + a_2 a_3^2 a_4^2$

The number of points on an elliptic curve, n, is the order of elliptic curve. Set of all points (x, y) which satisfies the above equation along with ∞, a point at infinity, are the points on the elliptic curve. The condition $\Delta \neq 0$ ensures that the elliptic curve is smooth, that is, there are no points at which the curve has two or more distinct tangent lines.

A.1. Point Addition

Addition of points on an elliptic curve is defined by Chord and Tangent rule. Let P = (*x1, y1*) and Q = (*x2, y2*) be two distinct points on an elliptic curve *E*. Then the sum *R*, of *P* and *Q*, is defined as follows: Draw a line connecting *P* and *Q* extend it to intersect the elliptic curve at a third point. Then the sum, *R* is the negative of the third point. Negative of a point is defined by reflection of the point about the x-axis.

The double *R*, of *P*, is defined as follows: Draw the tangent line to the elliptic curve at *P*. Let it intersects the elliptic curve at a second point. Then the double *R* is the reflection of this point about the x-axis.

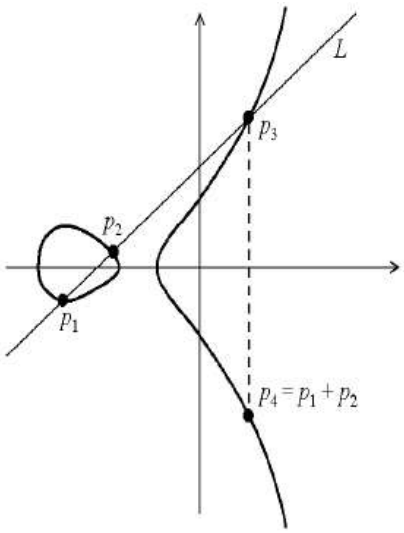

Figure 1. Point Addition
Source: https://i.imgur.com/EdPk12E.gif

A.2. Point Multiplication

Point Multiplication is the arithmetic operation which computes *kp* where *k* is an integer and *p* is a point on elliptic curve. It is done by repeated addition. For example *Q=kp* means *Q* is obtained by adding *p k* times to itself (*p + p + p....k* times). Cryptanalysis involves determining k given *P* and *Q*. This operation dominates the execution time of elliptic curve cryptographic schemes.

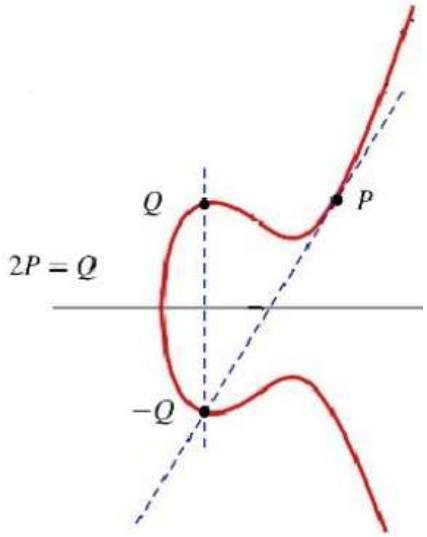

Figure 2. Point Multiplication
Source:
http://www.purplealienplanet.com/sites/default/files/image/point_doubling.png

B. Digital Signature Algorithm

A digital signature is an electronic version of a written signature. Digital Signature Algorithm (DSA) can be used by the recipient of a message to verify that the message has not been altered during transit as well as ascertain the originator's identity. Digital signature can be used in proving to the recipient or a third party that the message was, in fact, signed by the originator. Digital signatures may also be generated for stored data and programs so that the integrity of the data and programs may be verified at any later time[1].

The DSA is used by a signatory to generate a digital signature on data and by a verifier to verify the authenticity of the signature. Each signatory has a public and private key. The private key is used in the signature generation process and the public key is used in the signature verification process. For both signature generation and verification, the data or message is reduced by means of the Secure Hash Algorithm (SHA) specified in FIPS 180-1. An adversary, who does not know the private key of the signatory, cannot generate the correct signature of the signatory. In other words, signatures cannot be forged. However, by using the signatory's public key, anyone can verify a correctly signed message.

Simply put, the process of digitally signing starts by taking a mathematical summary (called a *hash code*) of the message or data. This hash code is a uniquely-identifying digital fingerprint of the message. If even a single bit of the message changes, the hash code will dramatically change. The next step in creating a digital signature is to sign the hash code with owner's private key. This signed hash code is then appended to the check.

The recipient of the message can verify the hash code sent by the sender, using his/her public key. At the same time, a

new hash code can be created from the received message and compared with the original signed hash code. If the hash codes match, then the recipient has verified that the check has not been altered. The recipient also knows that only the actual owner/sender could have sent the message because only he/she has the private key that signed the original hash code.

C. Elliptic Curve Digital Signature Algorithm (ECDSA)

Elliptic Curve Digital Signature Algorithm (ECDSA) was first proposed in 1992 by Scott Vanstone in response to NISTs proposal of DSS. It was later accepted in 1998 as an ISO standard, ANSI standard in 1999, IEEE standard, and NIST standard in 2000.

ECDSA is a lightweight signature algorithm because it operates in elliptic curve group. ECDSA uses multiplication in elliptic curve, which is same as repeated additions of two points, as a basic operation. In general, there are three main components in every digital signature algorithm, those are key generation, signature generation, and signature verification.

Key generation is the first thing to do when using digital signature. When a sender sends a message to a receiver, they must agree on a set of domain parameters of the elliptic curve that they will use later. The sender must have a private key *dA* (a random value less than *n*, *n* is the order of the curve). The sender must keep the private key to themselves. The sender also have a public key *QA* (the value of *QA* depends on the private key, $QA = dA * G$, *G* is a generator point). The sender can share her public key to the receiver for verification.

When the sender sends a message, he/she will sign it with a function called signature generation. Meanwhile when receiving a message, the receiver can prove that the message comes from the actual sender and it was not altered during the sending process with signature verification. The procedure of signature generation and verification will be explained later on.

III. Procedures of ECDSA

As mentioned previously, there are three main procedures of digital signature algorithm. Those are key generation, signature generation, and signature verification. ECDSA has also these 3 components which will be explained in the following items along with its security analysis.

A. Key Pair Generation

Let Alice be the signatory for a message *M*. Alice performs the following steps to generate a public and private key:

- Select a unique and unpredictable integer, *d*, in the interval [*1, n-1*], *n* is the order of the curve
- Compute $Q = dg$, *g* is a generator point
- Alice's private key is *d*
- Alice's public key is the combination of *E, g, n,* and *Q*

B. Signature Generation

When Alice sends a message, she will sign it with a function called signature generation. There are six steps in generating an ECDSA signature:

- Select a unique and unpredictable integer *k* in the interval [*1,n-1*]
- Compute $kg = (x_1,y_1)$, where x_1 is an integer
- Compute $r = x_1 \bmod n$; If $r = 0$, then go to step 1
- Compute $h = H(M)$, where *H* is one hash algorithm, for example SHA1 (as used in this implementation) $e = HASH(m)$
- Compute $s = k^{-1}(h + dr) \bmod n$; If $s = 0$, then go to step1
- The signature of Alice for message *M* is the integer pair (*r, s*)

C. Signature Verification

Let Bob be the receiver of message *M* signed by Alice. Bob can verify the authenticity of Alice's signature (*r, s*) for message M by performing the following steps:

- Verify that values *r* and *s* are in the interval [*1,n-1*]. If not, then the signature is not valid
- Compute $w = s^{-1} \bmod n$.
- Compute $e = HASH(m)$, where H is the same secure hash algorithm used by Alice in generating the signature
- Compute $u_1 = ew \bmod n$
- Compute $u_2 = rw \bmod n$
- Compute $(x_0,y_0) = u_1g + u_2Q$
- Compute $v = x_0 \bmod n$
- The signature for message *M* is verified only if $v = r$, otherwise the signature is not valid

IV. Procedures of Variant 1 (ECGDSA)

This variant of ECDSA is called Elliptic Curve German Digital Signature Algorithm. The ECGDSA signature scheme was developed in 1990. Two ideas were behind the ECGDSA development: Firstly, to transfer ElGamal's concept of a digital signature scheme based on the discrete logarithm problem in the multiplicative group of some finite field to elliptic curves. Secondly, to run the new scheme on RSA hardware — existing or just under development. As these hardware implementations did not support fast modular division, the scheme makes use of the modification suggested by Agnew, Mullin, and Vanstone by avoiding modular inversions for the calculation of ephemeral keys.

As mentioned above, one of the disadvantages of ECDSA scheme is the calculation of inverse in signing phase. Calculation of inverse is one of the expensive operations in Modular Arithmetic, so avoiding it will reduce the cost and time. In ECGDSA, inverse calculation is done in the key pair generation phase and not in Signing phase. A key will remain constant for a stable amount of time so signing is done more frequently than key generation. ECGDSA will save time and cost than ECDSA.

A. Key Pair Generation

Let Alice be the signatory for a message *M*. Alice needs to perform the following steps in order to generate a public and private key.

- Select a unique and unpredictable integer, *d*, in the interval [*1,n-1*], *n* is the order of the curve
- Compute $Q = (d^{-1} \bmod n)g$, *g* is a generator point
- Alice's private key is *d*
- Alice's public key is the combination (*E, g, n, Q*)

B. Signature Generation

When Alice sends a message, she will sign it with a function called signature generation. There are six steps in generating an ECDSA signature:

- Select a unique and unpredictable integer k in the interval [*1,n-1*]
- Calculate $kg = (x_1,y_1)$, where x_1 is an integer
- Calculate $r = x_1 \bmod n$; If $r = 0$, then go to step 1
- Calculate $h = H(M)$, where *H* is one hash algorithm, for example SHA1 (as used in this implementation) $e = HASH(m)$
- Calculate $s = (kr\text{-}h)\, d \bmod n$; If $s = 0$, then go to step1
- The signature of A for message M is the integer pair (*r,s*)

C. Signature Verification

Let Bob be the receiver of message *M* signed by Alice. Bob can verify the authenticity of Alice's signature (*r, s*) for message M by performing the following steps:

- Verify that values r and s are in the interval [*1,n-1*]. If not, then the signature is not valid
- Calculate $w = r^{-1} \bmod n$
- Calculate $e = HASH(m)$, where H is the same secure hash algorithm used by Alice in generating the signature
- Calculate $u_1 = ew \bmod n$
- Calculate $u_2 = sw \bmod n$
- Calculate $(x_0,y_0) = u_1g + u_2Q$
- Calculate $v = x_0 \bmod n$
- The signature for message *M* is verified only if $v = r$, otherwise the signature is not valid

V. Procedures of Variant 2

In ECGDSA (Variant 1), there is no need of finding inverse in signing phase but there is a need in key generation phase. In this scheme proposed by Zhang, Q, et al. there is no need in finding inverse in both key generation and signing phase. This scheme embeds the information of signature into a point on the ellipse.

A. Key Pair Generation

Let Alice be the signatory for a message *M*. Alice needs to perform the following steps in order to generate a public and private key.

- Select a unique and unpredictable integer, *d*, in the interval [*1,n-1*], *n* is the order of the curve
- Compute $Q = (dg \bmod n)$, *g* is a generator point
- Alice's private key is *d*
- Alice's public key is the combination (*E, g, n, Q*)

B. Signature Generation

When Alice sends a message, she will sign it with a function called signature generation. There are six steps in generating an ECDSA signature:

- Select a unique and unpredictable integer k in the interval [*1,n-1*]
- Calculate $kg = (x_1,y_1)$, where x_1 is an integer
- Calculate $r = x1 \bmod n$; If $r = 0$, then go to step 1
- Calculate $h = H(M)$, where *H* is one hash algorithm, for example SHA1 (as used in this implementation) $e = HASH(m)$
- Calculate $s = (kh + (r \text{ xor } h)d)g \bmod n$; If $s = 0$, then go to step1
- The signature of A for message M is the integer pair (*r,s*)

C. Signature Verification

Let Bob be the receiver of message *M* signed by Alice. Bob can verify the authenticity of Alice's signature (*r, s*) for message M by performing the following steps:

- Verify that values r and s are in the interval [*1,n-1*]. If not, then the signature is not valid
- Calculate $e = HASH(m)$, where H is the same secure hash algorithm used by Alice in generating the signature
- Calculate $w = e^{-1} \bmod n$
- Calculate $u = (r \text{ xor } e) \bmod n$
- Calculate $(x_0,y_0) = w(s - uQ)$
- Calculate $v = x_0 \bmod n$
- The signature for message *M* is verified only if $v = r$, otherwise the signature is not valid

VI. Implementation and Experiment Design

This paper implements the ECDSA along with its two variants to create digital signature embedded in a message string, and also the reverse process to verify the digital signature. The language used by this implementation is JAVA, with a NetBeans IDE. Figure 3. shows the main user interface. User can generate pair key by selecting the 'Generate Key' command. The program will call a function to generate public key based on user's private key. User can also embed the digital signature to a message after entering a private key and then verify the message after entering a public key.

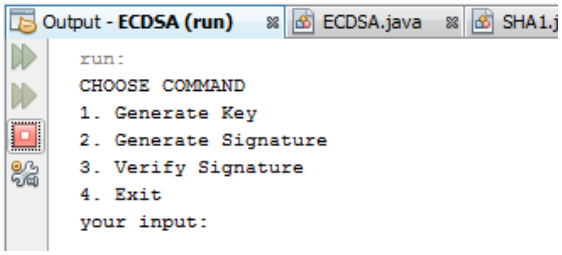

Figure 3. Main User Interface

By embedding digital signature, three concepts in security is covered. Those are authenticity, integrity, and non-repudiation. In this experiment, normal flow scenario is done to show the work of 3 ECDSA schemes. Each of the command for key generation, signature generation, and signature verification will run on the three schemes. The experiment uses curve with the parameters shown in Table 1.

Table 1. Elliptic Curve Parameters

Parameter	Value
prime	4451685225093714772084598273548427
a	4451685225093714772084598273548424
b	2061118396808653202902996166388514
koblitz	32
xG	188281465057972534892223778713752
yG	341987549103317082716786189608268 8
n	4451685225093714776491891542548933

In conducting the time complexity comparison of the three ECDSA schemes, the execution time of each command and each data/message size will be recorded. The result of execution time will be taken from an average of five attempts for each command.

VII. Experiment Results and Analysis

A. Time Complexity

Time taken for key generation, signature generation, and signature verification of all the schemes/variants are measured and detailed consecutively in Table 2, Table 3, and Table 4. Here the key size used in each experiment is 192 bits and the processor used for implementation is shown below.

- Processor: Intel(R) Core(TM) i3-2100 CPU @3.10GHz 3.40GHz
- RAM: 6.00 GB
- System Type: 64-bit Operating System
- Hard Disk: 1TB
- Operating System: Microsoft Windows 7 Ultimate

Tabel 2. Time taken for generating key using ECDSA variants

Algorithm	Execution Time (ms)
ECDSA	78
Variant 1	83
Variant 2	78

As shown in the table above, variant 1 of ECDSA needs the longest execution time in generating pair key. This could happen because variant 1 (ECGDSA) involves inverse calculation in the key pair generation phase. Calculation of inverse is one of the expensive operations in Modular Arithmetic, so using it will increase the cost and time complexity.

Tabel 3. Time taken for generating signature using ECDSA and its variants in millisecond

Message Size (kB)	Execution Time (ms)		
	ECDSA	Variant 1	Variant 2
20	49	44	73
40	93	78	141
80	180	151	279
160	349	296	562
320	681	577	1099

The correlation between the data size and execution time in generating digital signature can be shown in the chart below.

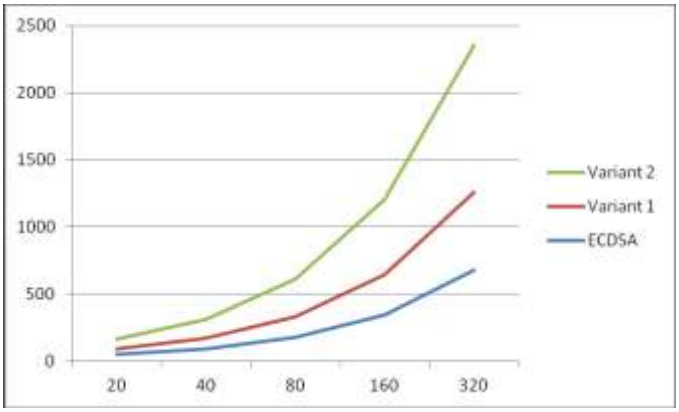

Figure 4. Time taken for generating signature in millisecond

Tabel 4. Time taken for verifying signature using ECDSA and its variants in millisecond

Message Size (kB)	Execution Time (ms)		
	ECDSA	Variant 1	Variant 2
20	87	85	109
40	125	125	218
80	190	193	437
160	259	274	869
320	347	393	1732

The correlation between the data size and execution time in verifying digital signature can be shown in the following figure 5.

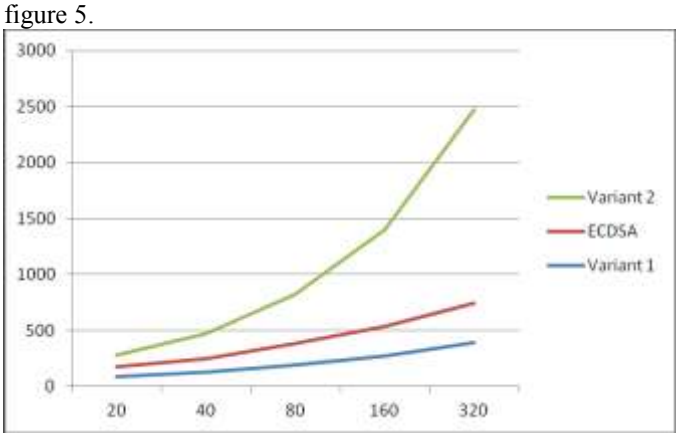

Figure 5. Time taken for verifying signature in millisecond

As shown in the two charts above, it can be seen that the execution time for generating and verifying digital signature for each scheme increase linearly with the increasing size of the data. This can occur because the algorithm needs time to create message digest. Moreover, with increasing size of the message given, the difference of execution time for the three schemes will be more visible.

In general, variant 2 of ECDSA needs the longest execution time to generate and verify the digital signature. This can happen as a result of the usage of elliptic curve. Variant 2 uses more elliptic curve operations and the time taken in each of the phases is large compared to the other schemes.

B. *Security Analysis of ECDSA*

Public key is generated by computing the point Q, where $Q=dg$. In order to crack the elliptic curve key, eavesdropper Eve would have to discover the secret key d when Q and g are provided. The order of the Elliptic curve, E is a prime number n, then computing d given dg and g would take roughly 2n=2 operations[2]. For example, if the key length n is 192 bits, then Eve will be required to compute about 296 operations. If Eve had a super computer and could perform one billion operations per second, it would take her around two and a half trillion years to find the secret key. This is the elliptic curve discrete logarithm problem behind ECDSA. The curve parameter should be chosen so carefully to secure Elliptic curve from well known attacks.

The secret k used for signing two or more messages should be generated independent of each other. In particular, a different secret k should be used for signing different messages otherwise the private key d can be recovered. However if a secure random or pseudorandom number generator is used, then the chance of generating a repeated k value is negligible. If same secret k is used to generate signature of two different messages *M1* and *M2* then it will result in two signatures (r,s_1) and (r, s_2).

$s_1 = k^{-1}(h_1 + dr)$,
$s_2 = k^{-1}(h_2 + dr)$
where $h_1 = SHA1(M1)$ and $h_2 = SHA1\ (M2)$
$ks_1 - ks_2 = h_1 + dr - h_2 - dr$
$k = (h_1 - h_2)/(s_1 - s_2)$
$d = (ks - h)/r$

VIII. Conclusion

These are the conclusions for this paper proposed algorithm and implementation:

- Variant 1 of ECDSA needs has the longest execution time in generating pair key in comparison with ECDSA and variant 2 scheme. This could happen because variant 1 (ECGDSA) involves inverse calculation in the key pair generation phase.
- Variant 2 needs the longest execution time to generate and verify the digital signature because it uses more elliptic curve operations.
- Execution time for generating and verifying digital signature for each scheme increase linearly with the increasing size of the data.
- This implementation can be developed by using the safer hash function to give a better security.

Acknowledgment

The author would like to thank her parents for their utmost support. The author would also give her gratitude to Mr. Rinaldi Munir for the knowledge they gave in class and all their support on the course IF4021 Cryptography for the last semester, and also for the chance to write this paper. Last but not least, the author would also like to thank other people who had given their help and support in any way that lead to the finishing of this paper.

References

[1] Aqeel Khalique ,Kuldip Singh Sandeep Soodv,"Implementation of Elliptic Curve Digital Signature Algorithm", International Journal of Computer Applications 2010.

[2] Hung-Zih Liao, Yuan-Yuan Shen, "On the Elliptic Curve Digital Signature Algorithm"Tunghai Science Vol. 8: 109126 July, 2006

[3] N. Koblitz," Elliptic curve cryptosystems", Mathematics of Computation 48, 1987, pp. 203-209

[4] Rinaldi, Bahan Kuliah IF3058 Kriptografi: Digital Signature

[5] https://www.vocal.com/cryptography/dsa-digital-signature-algorithm/ accessed on December, 18th 2016 at 2.23 pm

[6] https://developer.blackberry.com/native/reference/core/com.qnx.doc.crypto.lib_ref/topic/manual/ecc_ecgdsa.html accessed on December, 16th 2016 at 3.25 pm

Originality Statement

I hereby declare that this paper is my own writing, not an adaptation, nor translation of others' papers, and not a work of plagiarism.

Bandung, December 18th 2016

Pipin Kurniawati - 13513089