Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 09, September-2020

An Enhancement of Data Security in Cloud Computing with an Implementation of a Two-Level Cryptographic Technique, using AES and ECC Algorithm

Dickson Kodzo Mawuli Hodowu
Department of Computer Science,
KNUST, Kumasi, Ghana

Dennis Redeemer Korda
Department of Computer Science,
KNUST, Kumasi, Ghana

Dr. Edward Danso Ansong
Department of Computer Science
KNUST, Kumasi, Ghana

***Abstract*—Cloud computing has expanded significantly in recent years to become one of the major research areas. Aside its numerous benefits, it is also encompassed with challenges most importantly with the issue of data security. In essence, the paper presents an improved technique in ensuring data security in Cloud Computing. This study proposes an enhanced data security model using a two-level cryptographic technique - symmetric (AES) and asymmetric (ECC) cryptographic technique. This new technique seeks to pass data through the cloud space in a secure way, preventing third parties or intruders from accessing or making meaningful information out of the data being passed through the cloud space. The paper considered the time taken to perform cryptographic operations, enhance the security of data against intruders, denying them from having access to the real data enabling the confidentiality, integrity of the data, increasing level of speed with the use of smaller keys of ECC in the cryptographic process and increase in the trust level of usage of the Cloud Computing platform.**

Keywords—Cloud Computing, AES, ECC

I. INTRODUCTION

Cloud computing (CC) has expanded significantly in recent years to become one of out of the many research areas. CC is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction [1]. The term describes an improved technology that ignores the storage of data onto traditional data centers but leverages on external distributed data centers to store, process and access data. The cloud is divided into several models based on the type of services (SaaS, PaaS, and IaaS), offered to its clients. Cloud services may be used in a private, public, community or hybrid as deployment models.

Many individuals and companies now leverage on the use of cloud computing, becoming the necessary medium for storage, easy accessibility and preventing of data loss as compared to traditional computing. However, knowing that security cannot be 100% guaranteed, it is necessary to continuously find more secure ways in protecting the cloud environment from attacks and data breaches as it is a rising concern raised by many users. Among many individuals and organizations, data is a valued asset to them. A successful theft of 143,000,000 client accounts from Equifax, which is a customer credit reporting agency, a cybercrime with shocking outcomes due to the kind of personally recognizable information attained, was one of the many important data breaches that impacted end users in the year, 2017 [2].

Therefore, our main objective is to find a better means of securing data to minimize the loss and leakage of information, thereby improving the confidentiality, integrity of data passing through the cloud space, and increasing the level of trust of users. In that regard, we propose an enhanced two-level cryptographic conceptual framework that is implemented in the work. A good cause for this study rests on the skill or combination of algorithms of the system proposed.

II. LITERATURE SURVEY

A considerable amount of literature on cloud security and privacy has been published by several researchers. This phase of the paper presents some of these reviews on data security on CC.

In 2015 Abbas and Maryoosh proposed a work that provided a reliable, efficient, and scalable method for improving CC data storage security. Their approach encompassed three main aspects of using Private Key Generator (PKG), Trusted Cloud (TC), and the User [3].

In 2016, Abbas and Maryoosh presented a new architecture in improving the security of data within CC. This involved the usage of a Modified Identity-Based Cryptography (MIBC) as well as the Elliptic Curve Integrated Encryption Scheme (ECIES) algorithm which were the two encryption techniques used. The objective of their research was aimed at providing reliable, effective and scalable methods for improving data security in the cloud [3].

In 2016, Muthurajan and Narayanasamy examined how to improve the integrity and security of data transfer founded on the Schnorr scheme built on the Elliptic Curve. The paper suggested a VM-based cloud model utilizing HCSA to eliminate unrelated content and avoid system duplication. The model involves two stages, the system model and the threat model analysis [4].

In 2018, Smitha Nisha Mendonca aimed at developing a high-performance AES to further expand its prevalent application. The data sent from the user to the cloud is encrypted using AES algorithm. Therefore, request made to access the data must be decrypted at the user's end. The original text is not written on the cloud. A key managing server is installed to stores the keys. The research suggests that this type of encryption means protects both the data and encryption keys. It was found that AES encryption algorithm has a minimum storage capacity as well as good efficiency without any flaw or restriction [5].

In 2019, Kumari, et al., based on several research made, proposed an ECC built mutual authentication framework for secured communication. This model involved three phases. The setup phase, the extraction phase, and lastly the mutual authentication and session phase. It was observed that users authenticate each other and establishes a session key. Using the session key, participants can connect securely over the public communication network. Also, the protocol does not allow measurement of bilinear pairing, stressing that it makes the protocol in the communication domain very efficient [6].

III. TWO-LEVEL CRYPTOGRAPHIC TECHNIQUE

The scope pertaining to this research is tailored to developing an enhanced protection of data using encryption, leveraging a two-level cryptographic technique. In this state we incorporate both usage of Symmetric and Asymmetric Cryptography, AES and ECC respectively. The target is to develop a model to enhance the protection of data in cloud employing cryptographic technique. This will provide a reliable, efficient and scalable method for improving the security and safety of data passing through the cloud.

A. Advanced Encryption Standard (AES)

The symmetric key cryptography is established by Belgian cryptographers known as Joan Daemen and Vincent Rijmen [5]. Normally used for faster encrypting or decrypting of huge size of data. This is because there is no regeneration of another key but uses the same key for both its encrypting and decrypting process.

The AES uses 128-bit (with 10 encryption rounds), 192-bit (with 12 encryption rounds), or 256-bit (with 14 encryption rounds) keys to encrypt 128-bit blocks of data [7]. It is proven to have an increased level of security, that comes with large key size, and can encrypt messages faster than the DES, 3-DES [8]. The encryption with its decryption process involves the following systematic processes are: Byte substitution, Shift Rows, Mix Columns, add round key and then the inverse of these processes in order to decrypt it.

1) Encryption Process

In the course of the encryption process, each round is made up of the 4 sub-processes mentioned above.

a) Byte Substitution

The first modification of the AES encryption cipher is substituting data using a substitution table. The 16 input bytes are replaced by the fixed table which is the Substitution Box(S-box) and contains all likely combination of eight-bit order. The resultant new 16 bytes are arranged into a matrix of four columns and rows [5].

b) Shift Rows

The second transformation shifts rows of data. Each row in a matrix produced from the byte is moved/changed to the left. Any of the entries that shifts off is reinserted to the right side. The first row remains as it is, and the next row is shifted by a byte which is situated to the left. Then the 3rd row shifts 2 positions towards the left whereas the 4th row is shifted by 3 (byte) position towards the left. The resultant matrix then comprises of the same 16 bytes nonetheless at different positions [5].

c) Mix Column

At this point, the third transformation mixes the columns. Each of the columns of four-byte is then changed employing a special mathematical function of the Galois field (GF). The function takes the 4 bytes of one column by means of input and outputs four new bytes to replace the original column. The step is not done in the final round [5].

d) Add Round Key

The last transformation which is simple, that is carrying out exclusive OR operation on each column using a different section of an encryption key. The 16 bytes of the resulting matrix produced from the mix column stage are then considered to be 128 bits. At this stage, the 128-bit state is bitwise EX-ORed with a 128-bit round key. The output is the cipher text if it is the last line. Else, the resultant 128 bits will be read as 16 bytes and will start another likely process of substitution. It is a column-wise operation between four bytes of state column, one word of round key [5].

2) Decryption Process

This involves the reverse or opposite of the encryption process until plaintext or original data is gotten.

a) Add Round Key

This has its own inverse function, which the round keys are selected in the reverse order.

b) Inverse Shift Row

The inverse shift row works exactly the same way in the reverse order. The 1st row is kept so, then the others (2nd, 3rd and 4th row) is shifted by one-byte, 2-bytes and 3-bytes position to the right respectively.

c) Inverse Byte Substitution

This is done by using predefined substitution table identified as the inverse s-box.

d) Inverse Mix Column

The transition in the reverse mix column is achieved by using polynomials of lower than 4 degrees over the Galois field (GF) 28 of which the elements of the state column are the coefficients.

B. Elliptic Curve Cryptography (ECC)

The ECC is a mathematical technique used to create encryption keys, secure digital signatures and more. It is used in making lesser, quicker, and more efficient cryptographic keys. In 1985, the ECC's usage has been suggested by Neal Koblitz and Victor Miller [9].

1) The Arithmetic of Elliptic Curve

ECC's main attraction is its ability to provide an equivalent security level but this time with much lesser key size as likened to the RSA cryptography. The elliptic curve equation can be defined by equating all its coefficient and variables taking within the set of integers and ranging from 1 to n-1, performed by calculating the modulus. If using elliptic curve for cryptography the coefficients and variables in a finite Abelian group are restricted [10].

Below is the equation for the ECC:

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 09, September-2020

$$y^2 \bmod p = (x^3 + ax + b) \bmod p$$

2) *Elliptic Curve Discrete Logarithm Problem*

ECC is founded on the density of the Elliptic Curve Discrete Logarithm Problem (ECDLP). Suppose **E** is an elliptical curve over certain finite field Fq and then **P** representing a point order **n** on **E**. The ECDLP shall find the integer **d** on **E** ∈ [1, n–1], where there is such an integer,

$$Q = dP, \text{ where } dP = \underbrace{P + P + \ldots + P}_{\text{d times}}$$

DLP doesn't look like ECDLP, except the ECDLP is even tougher than the DLP [9].
The notion of finite field is the important characteristic of the elliptic curve, giving a way of limiting values on the elliptic curve.

3) *Security of Elliptic Curve Cryptography*

With the many asymmetric key cryptography techniques, the theory of elliptic curve proves that it can be used to generate easier, lesser, and more effective cryptographic keys. Using a 160-bits in ECC almost offer similar level of security against attacks as compared to 1024-bits in RSA [11]. Because of its key length, it is recommended for security applications whose electronic components (wireless devices, smart cards, pc cards) may have a reduced level of computational power [9]. The security of the algorithm depends on the difficulty in resolving. ECC generates smaller keys which eventually produces better performance as compared to others like the RSAs. ECC is more scalable compared to RSA which gets more cumbersome as key expands and less vulnerable even to Quantum computing and since its more of a new cryptographic technique as compared to the others [12].

Table I. Reasonable Bit Length [9]

ECC (bits)	RSA (bits)	Key Size Ratio
160	1024	1:6
256	2048	1:8
384	7680	1:20
512	15360	1:30

C) *The Proposed Model*

In this model, the first level is tagged as 1st level Security, where the data is at rest. Both the encryption and decryption of data is done via algorithm of AES. At second stage of security tagged as 2nd level Security, uses the ECC algorithm. At this stage, the encrypted data is in transit across the cloud space. It uses ECC in generating digital signatures and encryption keys to allow for safe transit of the data across the cloud space. The ECC, a public-key cryptosystem is also used for key management and authentication. The figure below clearly shows the architecture of the proposed model.

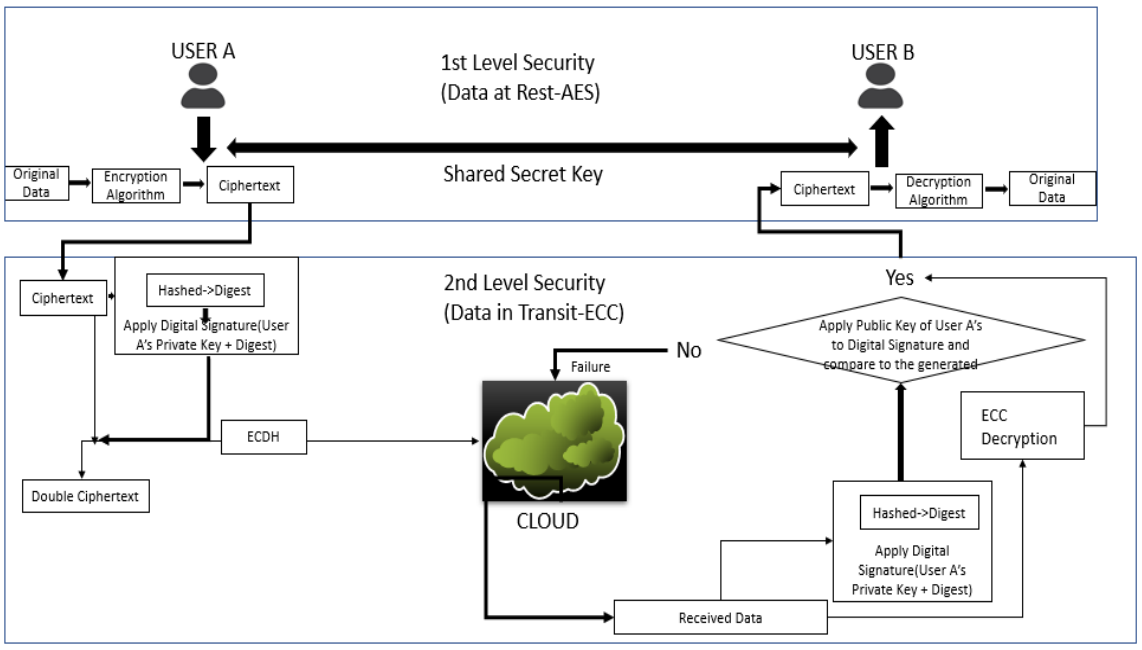


Figure 1. Proposed General Conceptual Two-Level Cryptographic Diagram

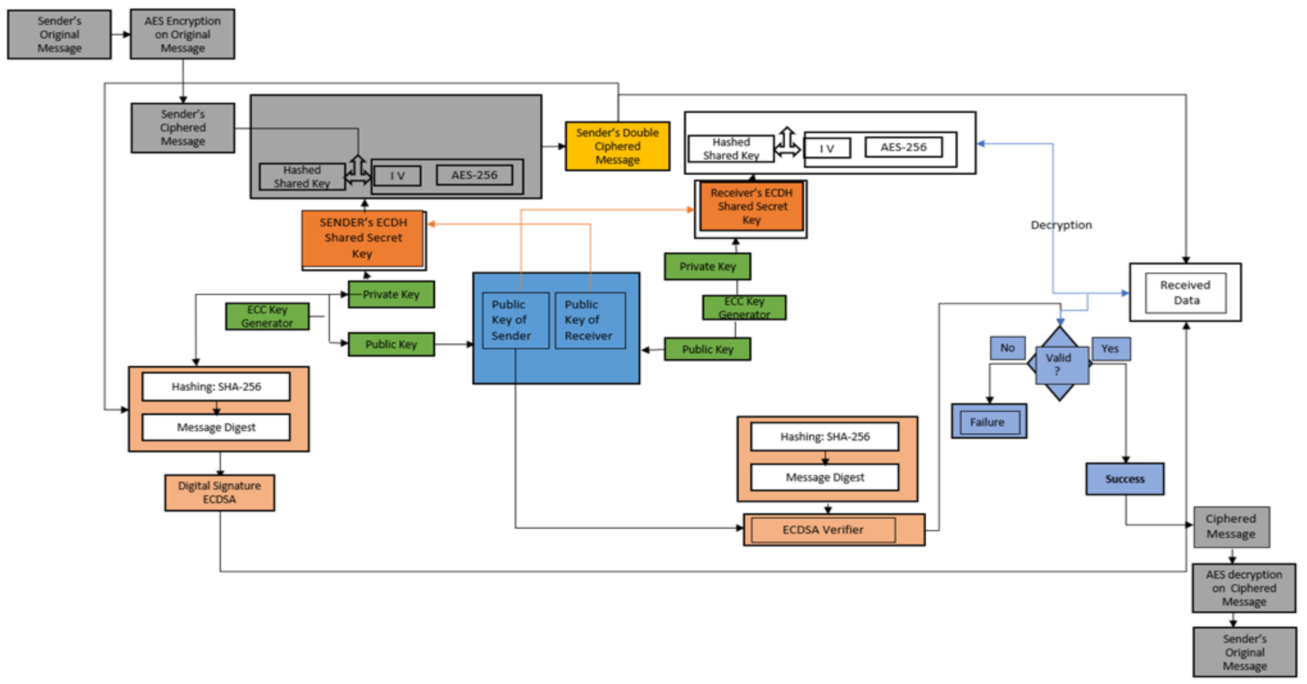


Figure 2. Proposed Detailed Model Showing Data Transiting in Cloud Space

1) AES and ECC Algorithm

The AES and the ECC algorithms have been separately defined below. The data at rest uses AES cryptography, which follows the algorithm below. After that the algorithm of ECC is looked at.

a) Secret Key Generation Using AES

In generating the key, the following steps are followed:
Step One: Using a key generator, AES key is generated
Step Two: Initialize the key size
Step Three: Generate the secret key

b) To encrypt/ decrypt data at rest

Since the AES makes usage of the same key for its encrypting and decrypting purpose, below shows its algorithm.
Step One: Create a cipher
Step Two: Initialize the cipher for encryption
Step Three: Encrypt or Decrypt the File

Let us consider the second level security using ECC in the Cloud space. The ECC like other asymmetric keys generates 2 keys be it as a public key and private key.
Considering the parameters that fully defines the Elliptic Curve cryptosystem, we have:
P: Specification of the finite field
a, b: Coefficient for defining Curve
G: Generator point on the curve where the operation starts
n: Order of G
h: Division of total points on the curve and the order of G

c) The ECC Key Generation Steps

In the generation of the keys, the below (mathematical/cryptographic) steps are follows:

Step 1: The sender selects a random number '$\boldsymbol{d_A}$' as the private key. The random key or number is represented by variable **d**, and selects in the range (1 to n-1).
Step 2: The public key is created by the sender using the formular

$$\mathbf{P_A = d_A * G}$$

Step 3: Likewise, receiver chooses a private key '$\mathbf{d_B}$' and generates corresponding public key

$$\mathbf{P_B = d_B * G}$$

Step 4: Both sender and receiver now generate the security keys to be

$$\mathbf{K = d_A * P_B} \quad \_\ \text{Sender}$$
$$\mathbf{K = d_B - P_A} \quad \_\ \text{Receiver}$$

d) *Generation of Signatures (Elliptic Curve Digital Signature Algorithm- ECDSA)*

The ECDSA uses keys derived from ECC, using a secured hash function. This secured has function has the property of irreversibility (theoretical impossible to determine the message from its digest, the property of collision resistance and thirdly, it has the property of high avalanche effect (change in message produces a modification in the message's digest).
In order to sign a message 'm' by the sender, the below steps are followed:
Step 1: A cryptographic hash function is calculated using

$$e = \text{HASH}(m)$$

Step 2: An integer k by the sender is randomly selected within the range [1, n-1]
Step 3: A pair of integers (r, s) is computed to define the signature for the message, where

$$r = x1(\text{mod } n) \text{ where } (x1, y1) = k * G \text{ and}$$
$$s = k\text{-}1(e + d_A * r)$$

Step 4: The signature is then sent to the receiver

2) Algorithm for Encryption

In the encryption algorithm, the message is converted to a cipher text. Assume the message that is to be sent to the Receiver from the Sender is indicated as **'m'.**

Step 1: Let **m** have a point, **M** on elliptic Curve

Step 2: Sender then selects a random number k from [1, n-1]

Step 3: The Cipher text produced will be a pair of point (C_1, C_2)

$$C_1 = k * G$$
$$C_2 = M + (k * G)$$

3) Algorithm for Decryption

Converting back the ciphered text to its plain text using the algorithm below

Step 1: Receiver calculates product of C_1 and its private key

Step 2: Receiver deducts the product from the second point C_2 as indicated below to get the original message M

$$M = C_2 - (d_B * C_1)$$

4) Algorithm for Verifying the Signature

Before the sender's signature is authenticated, the receiver is to be aware of the sender's public key P_A which is always available for use on the cloud space. Below are the steps taken in order to verify the authenticity of the message.

Step 1: Firstly, the receiver will verify the pairs (r, s) to check if it is in the array of [1, 1-n]

Step 2: The hash function 'e' is calculated by the receiver, same as during signature generation,

Step 3: Receiver then calculates, w = s-1 mod(n)

Step 4: Receiver calculates $u_1 = e * w \pmod n$ and then $u_2 = r * w \pmod n$

Step 5: Calculate $(x1, y1) = u_1 * G + u_2 * P_A$

Step 6: If x1 = r (mod n), then signature remains valid.

IV. IMPLEMENTATION AND RESULT

During the simulation, tools were used and procedures followed. A virtual machine with Kali as guest OS with its host as the Windows 10 OS is used. The 'AESCrypt' and Openssl as tools used in the cryptographic process installed in the Kali Linux, dual core processor of 1.80GHZ and 2.00GHZ, a 16GB RAM size and a Kali VM of 20GB Virtual Hard Disk space.

The dataset for implementation is a text-based file. The ECDH key exchange or Shared Secret Key uses '.bin', public and private key generated by ECC is saved as '. pem' extension. The extension for the ECDSA is a '. hash'.

In the simulation, the client and the cloud server use an agreed upon elliptic curve encryption shared keys and size, and hashing function's size engaged in the signing process. Three folders are created to simulate Client machines (A and B) and the Cloud Space(server) where Public keys generated by both clients are shared or stored.

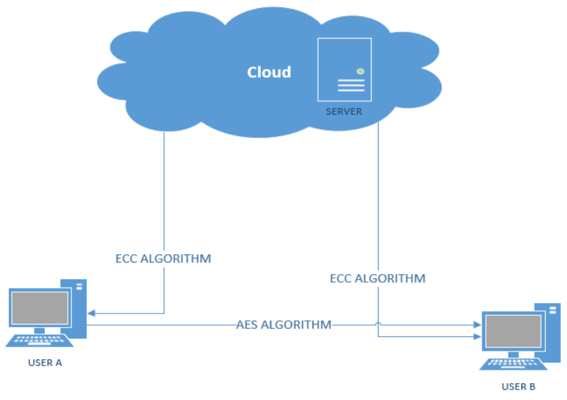
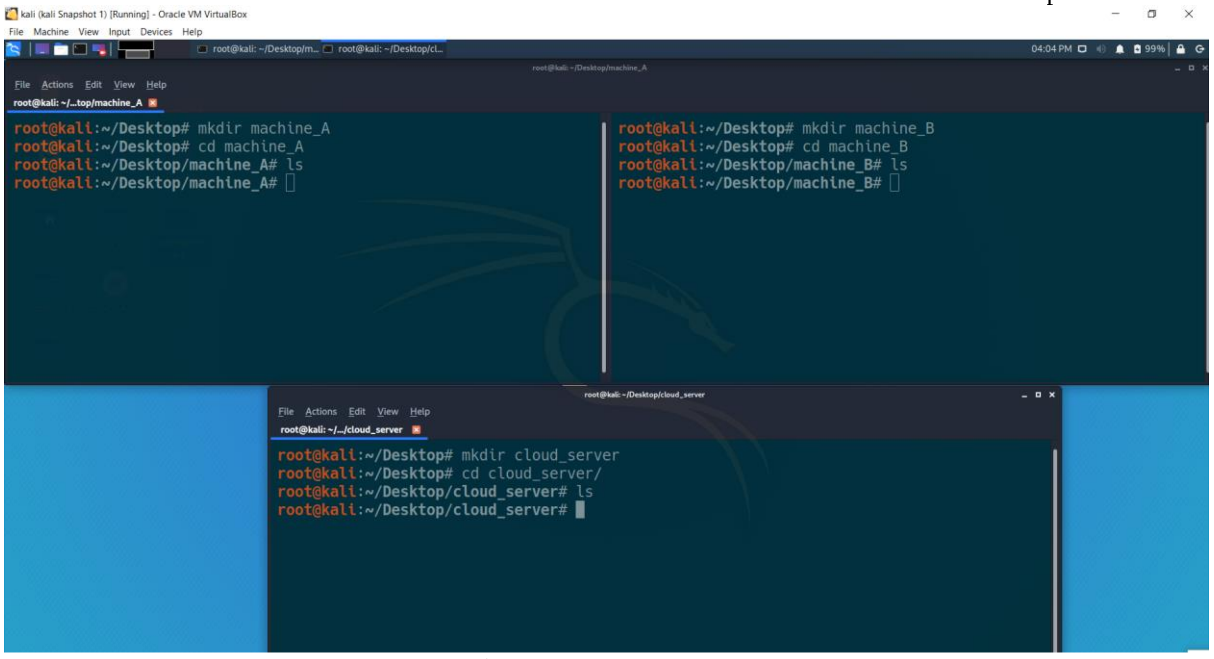


Figure 3. Cloud Server and Clients Connectivity

The assumed virtual machine and cloud space are created.


kali (kali Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
File Actions Edit View Help

```
root@kali:~/Desktop# mkdir machine_A
root@kali:~/Desktop# cd machine_A
root@kali:~/Desktop/machine_A# ls
root@kali:~/Desktop/machine_A#
```

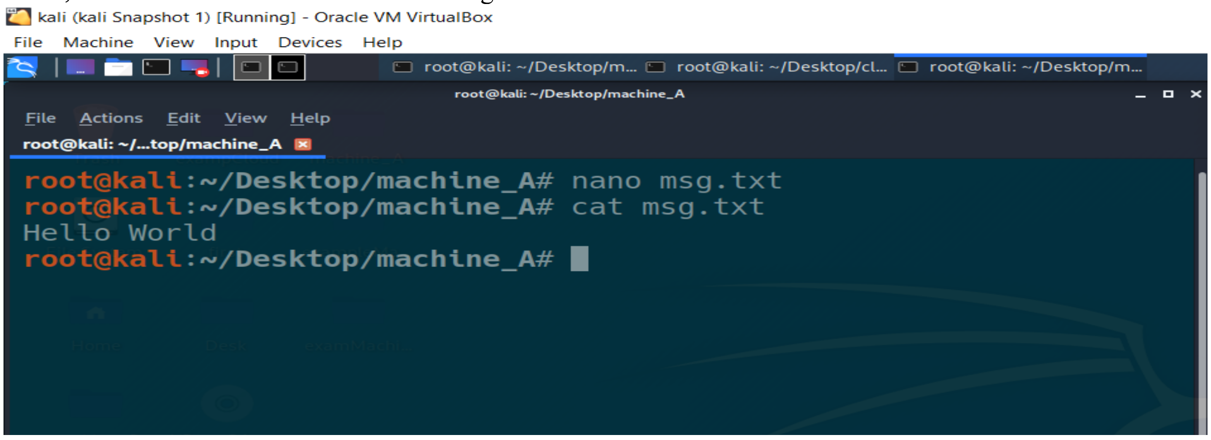
```
root@kali:~/Desktop# mkdir machine_B
root@kali:~/Desktop# cd machine_B
root@kali:~/Desktop/machine_B# ls
root@kali:~/Desktop/machine_B#
```

```
root@kali:~/Desktop# mkdir cloud_server
root@kali:~/Desktop# cd cloud_server/
root@kali:~/Desktop/cloud_server# ls
root@kali:~/Desktop/cloud_server#
```

Figure 4. Creation of the assumed Cloud Space and Clients Machines

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 09, September-2020

In machine A, the text file is created and named “msg.txt”.

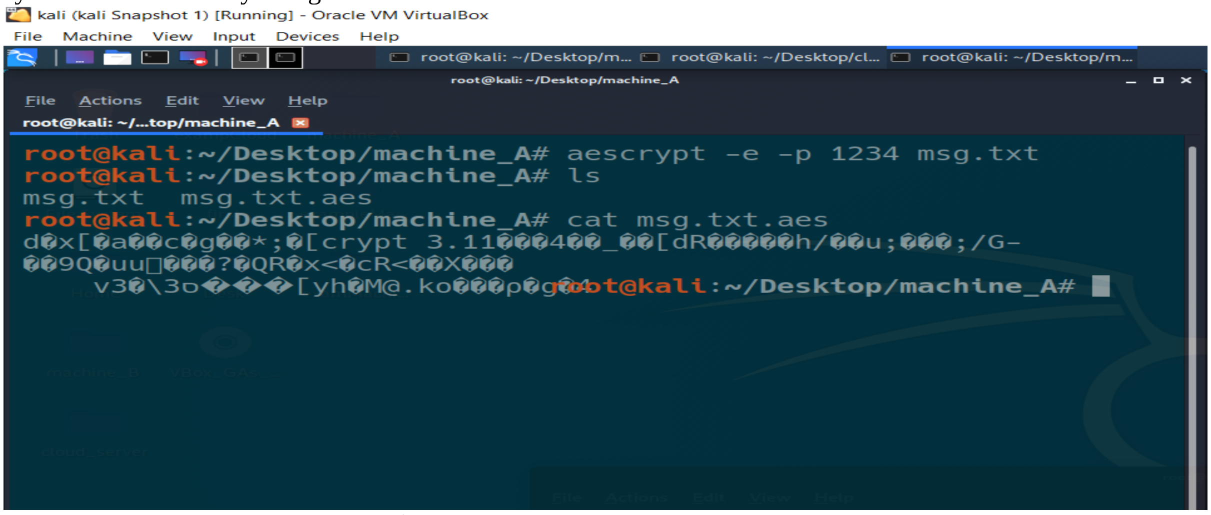

kali (kali Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@kali: ~/Desktop/m... root@kali: ~/Desktop/cl... root@kali: ~/Desktop/m...
root@kali: ~/Desktop/machine_A
File Actions Edit View Help
root@kali: ~/...top/machine_A

```
root@kali:~/Desktop/machine_A# nano msg.txt
root@kali:~/Desktop/machine_A# cat msg.txt
Hello World
root@kali:~/Desktop/machine_A# 
```

Figure 5. Plaintext File Creation

A. First Level Cryptography Using AES Algorithm

At the first level, AES is used to encode the message. A shares symmetric secret key is generated and known between the two parties or clients involved in the communication. The AEScrypt is the tool used at this stage. The figure below shows how the encryption is done.

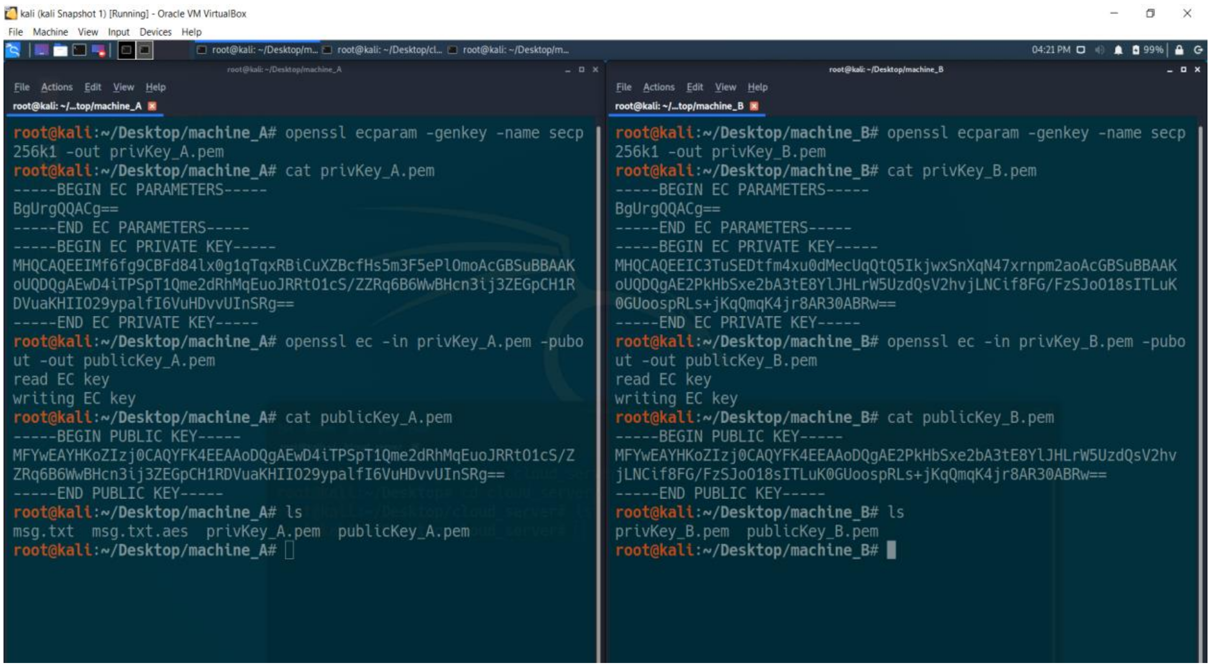

kali (kali Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@kali: ~/Desktop/m... root@kali: ~/Desktop/cl... root@kali: ~/Desktop/m...
root@kali: ~/Desktop/machine_A
File Actions Edit View Help
root@kali: ~/...top/machine_A

```
root@kali:~/Desktop/machine_A# aescrypt -e -p 1234 msg.txt
root@kali:~/Desktop/machine_A# ls
msg.txt  msg.txt.aes
root@kali:~/Desktop/machine_A# cat msg.txt.aes
d�x[�a��c�g��*;�[crypt 3.11���4��_��[dR������h/��u;���;/G-
��9Q�uu□���?�QR�x<�cR<��X���
    v3�\3o����[yh�M@.ko���p�g�root@kali:~/Desktop/machine_A# 
```

Figure 6. Initial Part of the First Level Cryptographic Process Showing Encrypting with AES

B. Key Generation Using the ECC Algorithm

At this stage, the Openssl tool is used in the cloud space. At the initial stage, the private keys are related and public keys from these private keys are generated.


kali (kali Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@kali: ~/Desktop/m...
root@kali: ~/Desktop/cl...
root@kali: ~/Desktop/m...
04:21 PM
99%
root@kali: ~/Desktop/machine_A
File Actions Edit View Help
root@kali: ~/...top/machine_A
root@kali: ~/Desktop/machine_B
root@kali: ~/...top/machine_B

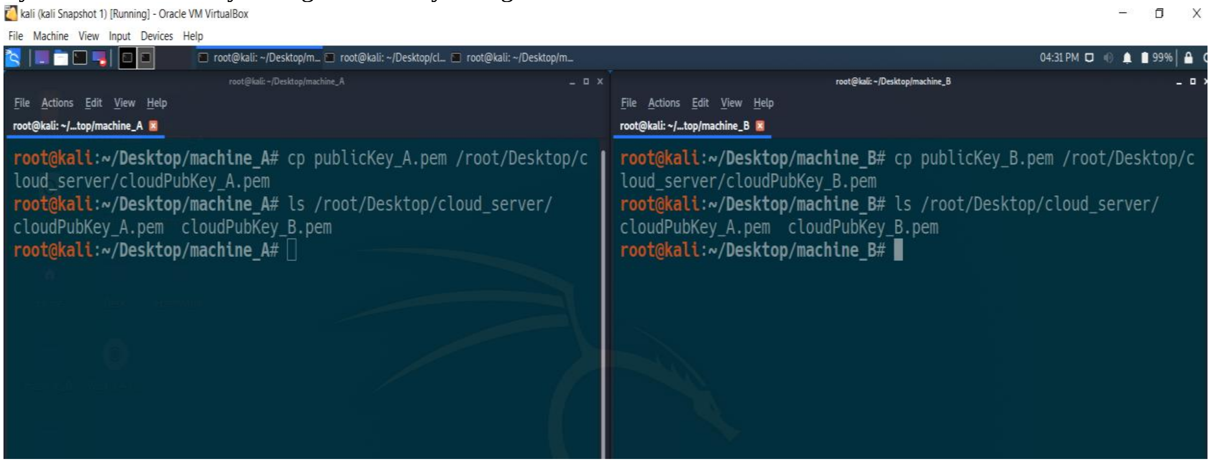
```
root@kali:~/Desktop/machine_A# openssl ecparam -genkey -name secp
256k1 -out privKey_A.pem
root@kali:~/Desktop/machine_A# cat privKey_A.pem
-----BEGIN EC PARAMETERS-----
BgUrgQQACg==
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MHQCAQEEIMf6fg9CBFd84lx0g1qTqxRBiCuXZBcfHs5m3F5ePlOmoAcGBSuBBAAK
oUQDQgAEwD4iTPSpT1Qme2dRhMqEuoJRRtO1cS/ZZRq6B6WwBHcn3ij3ZEGpCH1R
DVuaKHII029ypalfI6VuHDvvUInSRg==
-----END EC PRIVATE KEY-----
root@kali:~/Desktop/machine_A# openssl ec -in privKey_A.pem -pubo
ut -out publicKey_A.pem
read EC key
writing EC key
root@kali:~/Desktop/machine_A# cat publicKey_A.pem
-----BEGIN PUBLIC KEY-----
MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEwD4iTPSpT1Qme2dRhMqEuoJRRtO1cS/Z
ZRq6B6WwBHcn3ij3ZEGpCH1RDVuaKHII029ypalfI6VuHDvvUInSRg==
-----END PUBLIC KEY-----
root@kali:~/Desktop/machine_A# ls
msg.txt  msg.txt.aes  privKey_A.pem  publicKey_A.pem
root@kali:~/Desktop/machine_A# 
```

```
root@kali:~/Desktop/machine_B# openssl ecparam -genkey -name secp
256k1 -out privKey_B.pem
root@kali:~/Desktop/machine_B# cat privKey_B.pem
-----BEGIN EC PARAMETERS-----
BgUrgQQACg==
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MHQCAQEEIC3TuSEDtfm4xu0dMecUqQtQ5IkjwxSnXqN47xrnpm2aoAcGBSuBBAAK
oUQDQgAE2PkHbSxe2bA3tE8YlJHLrW5UzdQsV2hvjLNCif8FG/FzSJoO18sITLuK
0GUoospRLs+jKqQmqK4jr8AR30ABRw==
-----END EC PRIVATE KEY-----
root@kali:~/Desktop/machine_B# openssl ec -in privKey_B.pem -pubo
ut -out publicKey_B.pem
read EC key
writing EC key
root@kali:~/Desktop/machine_B# cat publicKey_B.pem
-----BEGIN PUBLIC KEY-----
MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAE2PkHbSxe2bA3tE8YlJHLrW5UzdQsV2hv
jLNCif8FG/FzSJoO18sITLuK0GUoospRLs+jKqQmqK4jr8AR30ABRw==
-----END PUBLIC KEY-----
root@kali:~/Desktop/machine_B# ls
privKey_B.pem  publicKey_B.pem
root@kali:~/Desktop/machine_B# 
```

Figure 7. ECC Key Generation Using Openssl

C. *Elliptic-curve Diffie–Hellman Key Exchange-Shared Secret Key Generation*

In generating the shared secret key, both public keys of the sender and receiver is first shared to the cloud. The shared secret key is a common key brought about by using the ECDH. Before performing the ECDH key exchange operation, the public keys of both users are shared on the cloud which any user may have access to. Figure 8 shows how it is done.

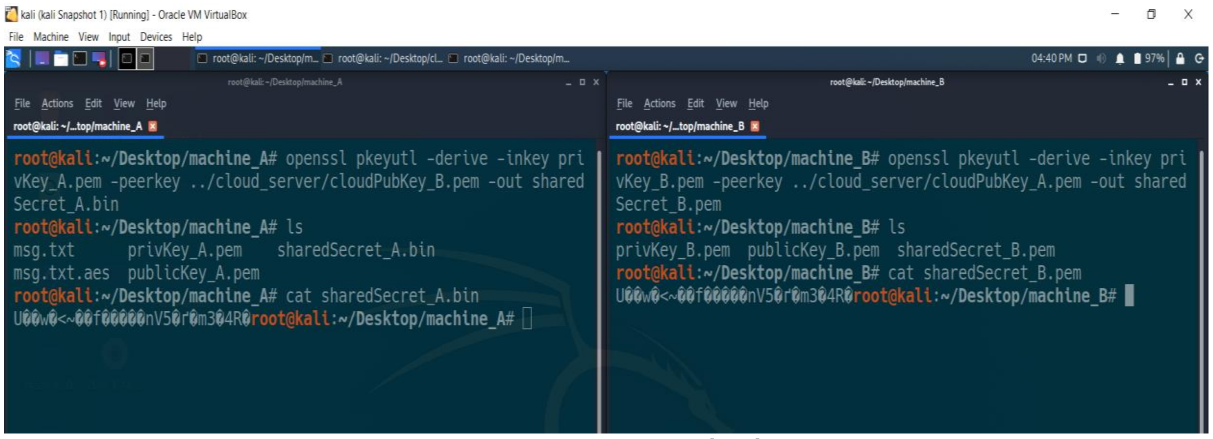

kali (kali Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@kali: ~/Desktop/m...
root@kali: ~/Desktop/cl...
root@kali: ~/Desktop/m...
04:31 PM
99%
root@kali: ~/Desktop/machine_A
File Actions Edit View Help
root@kali: ~/...top/machine_A
root@kali: ~/Desktop/machine_B
root@kali: ~/...top/machine_B

```
root@kali:~/Desktop/machine_A# cp publicKey_A.pem /root/Desktop/c
loud_server/cloudPubKey_A.pem
root@kali:~/Desktop/machine_A# ls /root/Desktop/cloud_server/
cloudPubKey_A.pem  cloudPubKey_B.pem
root@kali:~/Desktop/machine_A# 
```

```
root@kali:~/Desktop/machine_B# cp publicKey_B.pem /root/Desktop/c
loud_server/cloudPubKey_B.pem
root@kali:~/Desktop/machine_B# ls /root/Desktop/cloud_server/
cloudPubKey_A.pem  cloudPubKey_B.pem
root@kali:~/Desktop/machine_B# 
```

Figure 8. Public keys shared on the Cloud Space

In performing the ECDH key exchange operation, the private key of party A is paired with the public key of user B to produce shared key for Party A. Whereas the private key of Party B together with public key of Party A to produce the shared secret key for Party B. Figure 9 shows how the shared key is produced.

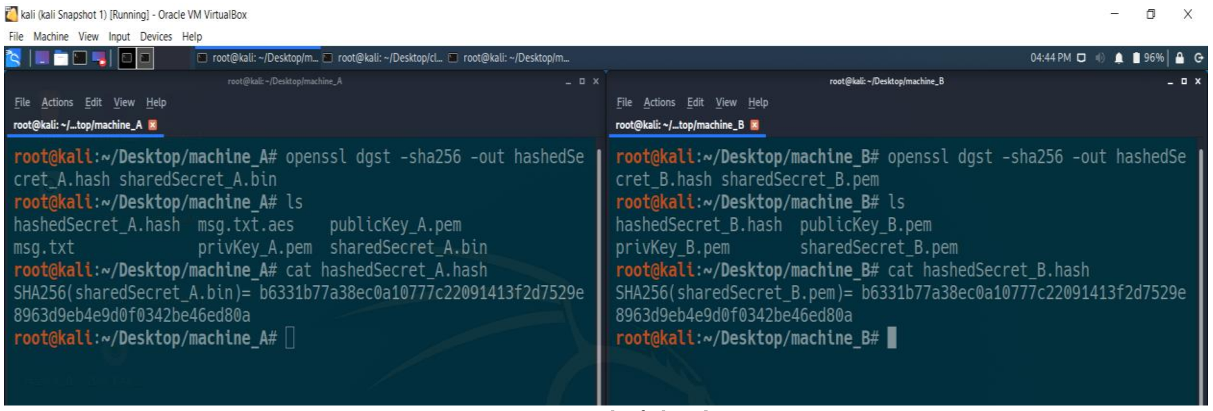

kali (kali Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
04:40 PM 97%
root@kali: ~/Desktop/machine_A
root@kali: ~/Desktop/machine_B
File Actions Edit View Help
root@kali: ~/...top/machine_A
root@kali: ~/...top/machine_B

```
root@kali:~/Desktop/machine_A# openssl pkeyutl -derive -inkey pri
vKey_A.pem -peerkey ../cloud_server/cloudPubKey_B.pem -out shared
Secret_A.bin
root@kali:~/Desktop/machine_A# ls
msg.txt      privKey_A.pem    sharedSecret_A.bin
msg.txt.aes  publicKey_A.pem
root@kali:~/Desktop/machine_A# cat sharedSecret_A.bin
U��w�<~��f�����nV5�r�m3�4R�root@kali:~/Desktop/machine_A#
```

```
root@kali:~/Desktop/machine_B# openssl pkeyutl -derive -inkey pri
vKey_B.pem -peerkey ../cloud_server/cloudPubKey_A.pem -out shared
Secret_B.pem
root@kali:~/Desktop/machine_B# ls
privKey_B.pem  publicKey_B.pem  sharedSecret_B.pem
root@kali:~/Desktop/machine_B# cat sharedSecret_B.pem
U��w�<~��f�����nV5�r�m3�4R�root@kali:~/Desktop/machine_B#
```

Figure 9. ECDH Key Agreement Protocol Implementation

We continue by hashing the shared secret key enabling it for authentication in which a hash function is performed on the key.

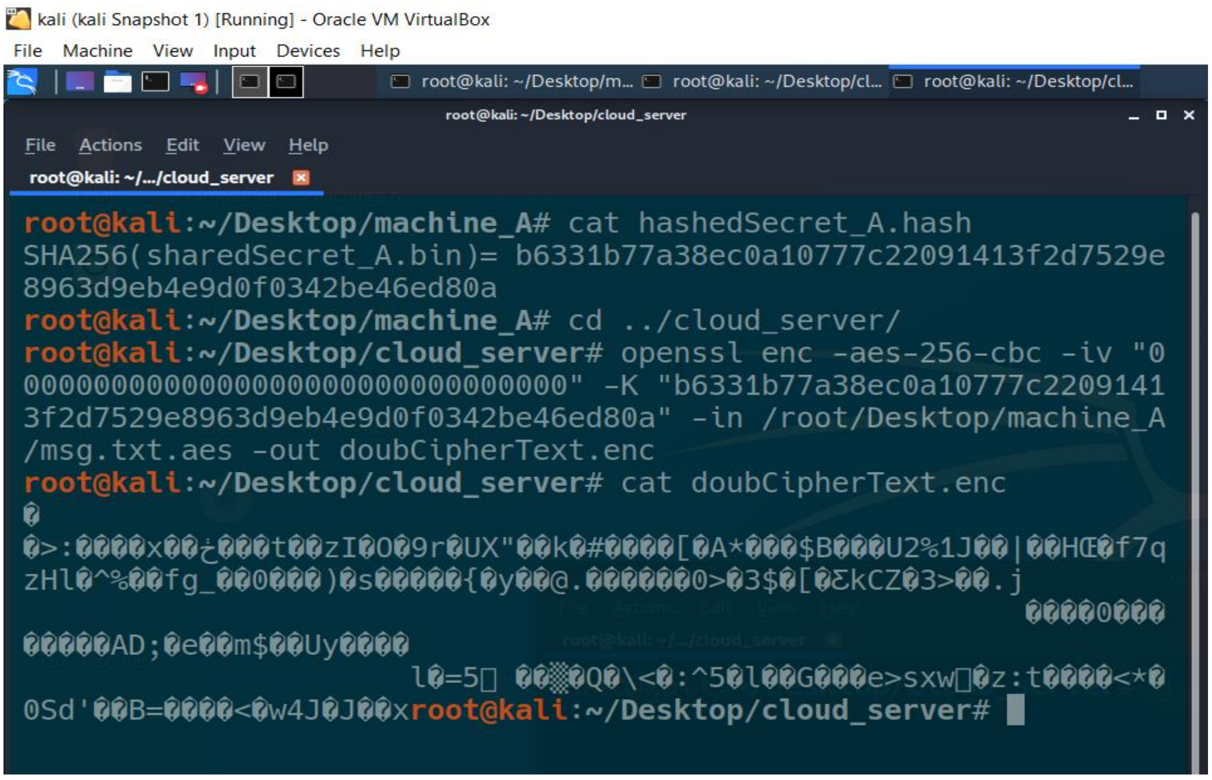

kali (kali Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
04:44 PM 96%
root@kali: ~/Desktop/machine_A
root@kali: ~/Desktop/machine_B
File Actions Edit View Help
root@kali: ~/...top/machine_A
root@kali: ~/...top/machine_B

```
root@kali:~/Desktop/machine_A# openssl dgst -sha256 -out hashedSe
cret_A.hash sharedSecret_A.bin
root@kali:~/Desktop/machine_A# ls
hashedSecret_A.hash  msg.txt.aes    publicKey_A.pem
msg.txt              privKey_A.pem  sharedSecret_A.bin
root@kali:~/Desktop/machine_A# cat hashedSecret_A.hash
SHA256(sharedSecret_A.bin)= b6331b77a38ec0a10777c22091413f2d7529e
8963d9eb4e9d0f0342be46ed80a
root@kali:~/Desktop/machine_A#
```

```
root@kali:~/Desktop/machine_B# openssl dgst -sha256 -out hashedSe
cret_B.hash sharedSecret_B.pem
root@kali:~/Desktop/machine_B# ls
hashedSecret_B.hash  publicKey_B.pem
privKey_B.pem        sharedSecret_B.pem
root@kali:~/Desktop/machine_B# cat hashedSecret_B.hash
SHA256(sharedSecret_B.pem)= b6331b77a38ec0a10777c22091413f2d7529e
8963d9eb4e9d0f0342be46ed80a
root@kali:~/Desktop/machine_B#
```

Figure 10. SHA256 Hash of Shared Secret Key

After hashing the shared secret key, it is used together with the AES256 to encrypt the ciphertext message earlier encrypted using the AES algorithm of the party's machine. The IV is added to the key to essentially scramble the key to encrypt the data passing through the cloud space. This process is done in a way that the client or user at the other end knows the operation used to reverse the process at the time of decryption.

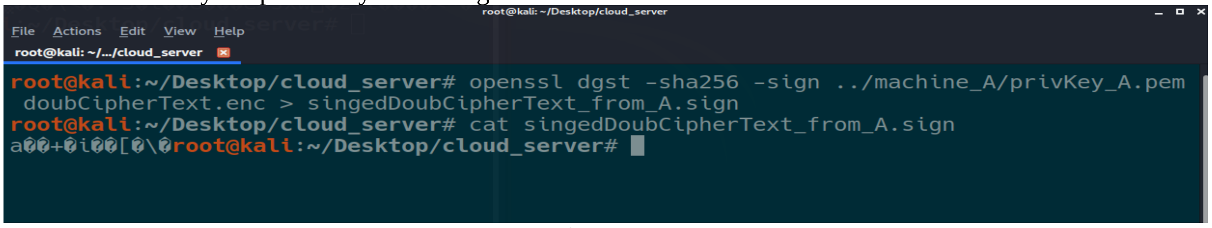

kali (kali Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@kali: ~/Desktop/m...
root@kali: ~/Desktop/cl...
root@kali: ~/Desktop/cl...
root@kali: ~/Desktop/cloud_server
File Actions Edit View Help
root@kali: ~/.../cloud_server

```
root@kali:~/Desktop/machine_A# cat hashedSecret_A.hash
SHA256(sharedSecret_A.bin)= b6331b77a38ec0a10777c22091413f2d7529e
8963d9eb4e9d0f0342be46ed80a
root@kali:~/Desktop/machine_A# cd ../cloud_server/
root@kali:~/Desktop/cloud_server# openssl enc -aes-256-cbc -iv "0
0000000000000000000000000000000" -K "b6331b77a38ec0a10777c2209141
3f2d7529e8963d9eb4e9d0f0342be46ed80a" -in /root/Desktop/machine_A
/msg.txt.aes -out doubCipherText.enc
root@kali:~/Desktop/cloud_server# cat doubCipherText.enc
�
�>:����x��¿���t��zI�O�9r�UX"��k�#����[�A*���$B���U2%1J��|��HŒ�f7q
zHl�^%��fg_��0���)�s�����{�y��@.������0>�3$�[�ƐkCZ�3>��.j
                                                    ����0���
�����AD;�e��m$��Uy����
                      l�=5▯ ��▒�Q�\<�:^5�l��G���e>sxw▯�z:t����<*�
0Sd'��B=����<�w4J�J��xroot@kali:~/Desktop/cloud_server# █
```

Figure 11. Encrypting Data in Cloud

D. Elliptic Curve Digital Signature Algorithm (ECDSA) Generation/Signing

The ECDSA is a digital signature algorithm uses derived keys from ECC. The ECC having generated a pair of keys (public and private) for the parties involved in its encryption process uses the private key to sign the message and authenticated or verified by the public key of the originator. To perform the message signing, the private key of sender together with a secured and common hashing algorithm known to the receiver is used. This has the property of irreversibility where it is almost impossible to determine the message from its digest. This has the property of irreversibility, collision resistance, and a high avalanche effect.

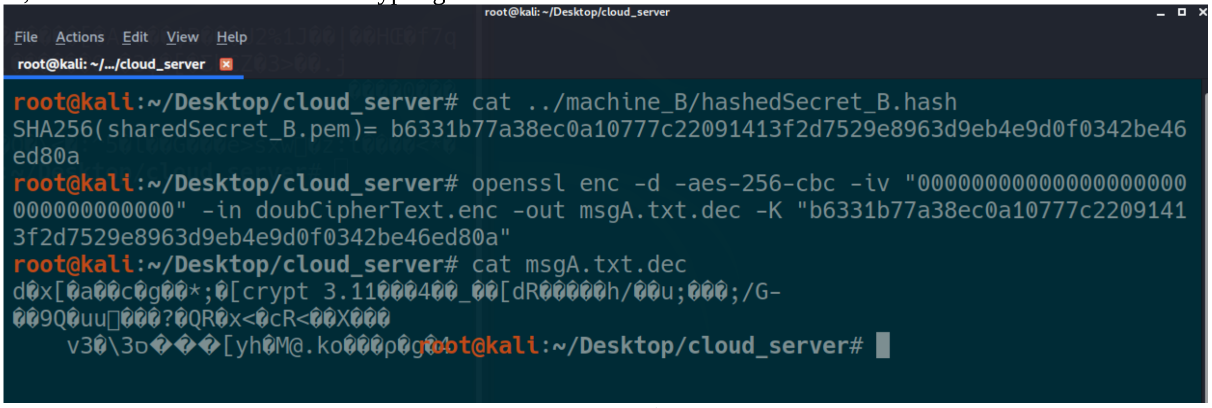

root@kali: ~/Desktop/cloud_server
File Actions Edit View Help
root@kali: ~/.../cloud_server

```
root@kali:~/Desktop/cloud_server# openssl dgst -sha256 -sign ../machine_A/privKey_A.pem
 doubCipherText.enc > singedDoubCipherText_from_A.sign
root@kali:~/Desktop/cloud_server# cat singedDoubCipherText_from_A.sign
a��+�i��[�\�root@kali:~/Desktop/cloud_server# █
```

Figure 12. Use of ECDSA to sign Message

E. ECC Decrypting in the Cloud Space

At this point, since the receiving client or user uses the same algorithm, it means that the IV used in encrypting the data is only known to the receiver and will be used in decrypting data.

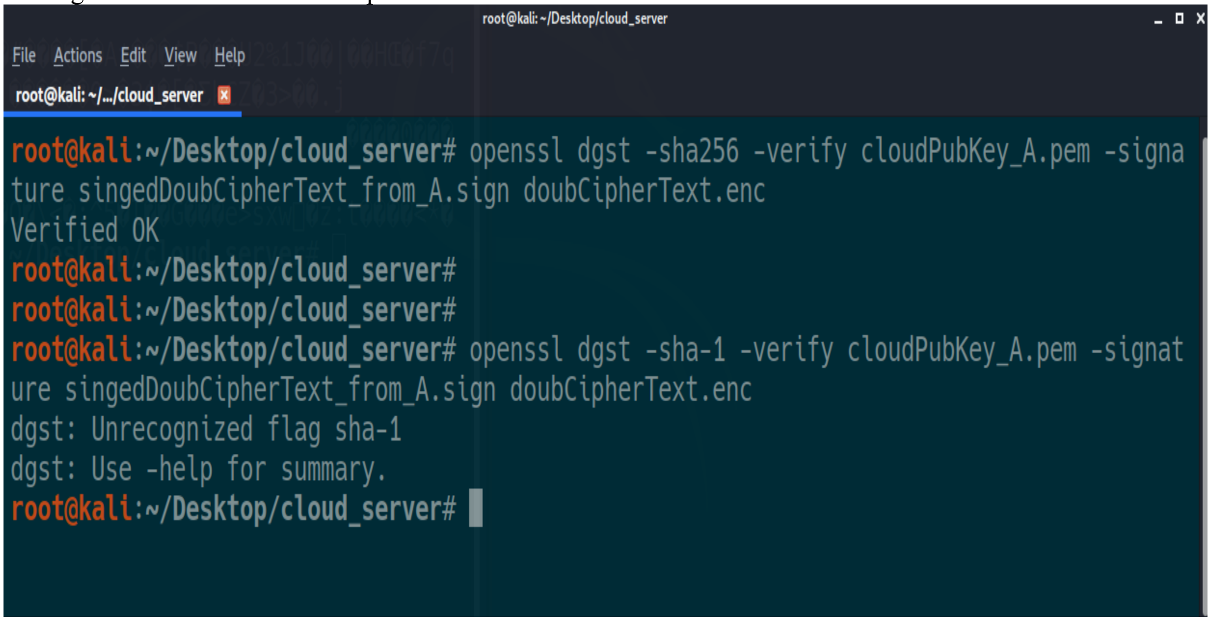

root@kali: ~/Desktop/cloud_server
File Actions Edit View Help
root@kali: ~/.../cloud_server

```
root@kali:~/Desktop/cloud_server# cat ../machine_B/hashedSecret_B.hash
SHA256(sharedSecret_B.pem)= b6331b77a38ec0a10777c22091413f2d7529e8963d9eb4e9d0f0342be46
ed80a
root@kali:~/Desktop/cloud_server# openssl enc -d -aes-256-cbc -iv "00000000000000000000
000000000000" -in doubCipherText.enc -out msgA.txt.dec -K "b6331b77a38ec0a10777c2209141
3f2d7529e8963d9eb4e9d0f0342be46ed80a"
root@kali:~/Desktop/cloud_server# cat msgA.txt.dec
d�x[�a��c�g��*;�[crypt 3.11���4��_��[dR�����h/��u;���;/G-
��9Q�uu▯���?�QR�x<�cR<��X���
    v3�\3ↄ���[yh�M@.ko���p�groot@kali:~/Desktop/cloud_server# █
```

Figure 13. ECC Decryption of Data in the Cloud

F. ECDSA Verification in Cloud

In the simulation, the verification of message is proven to be right indicating as "Verified OK". At the position when the right verification process is wrong, then the message doesn't get authenticated and therefore gives a flag, as seen in the below figure.

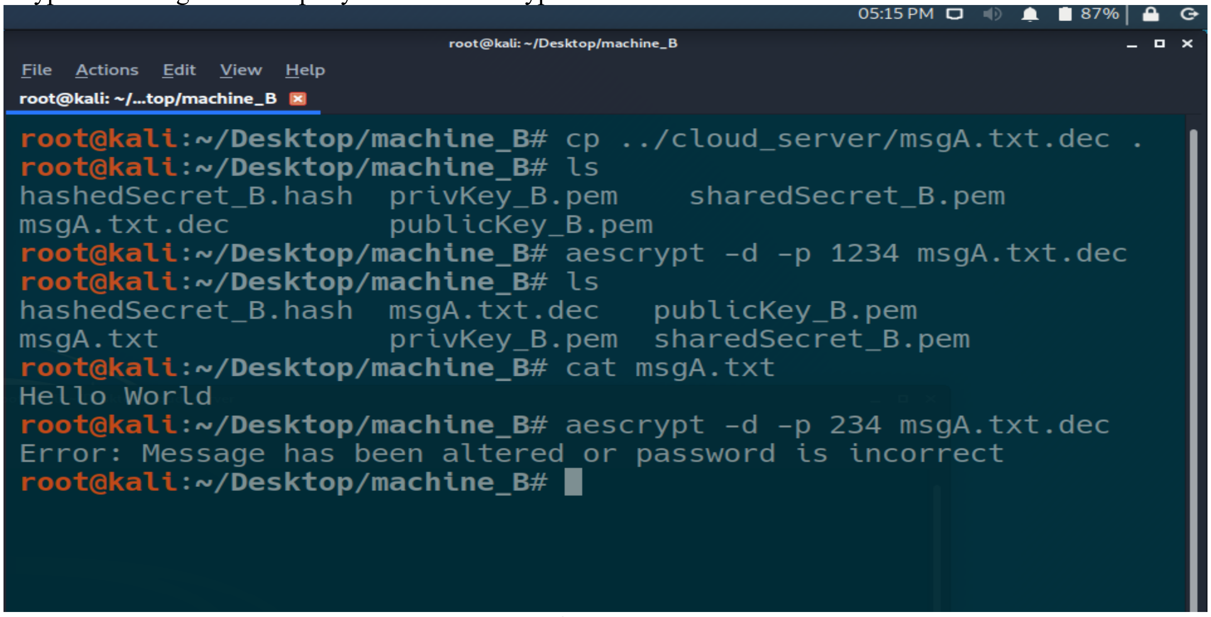

root@kali: ~/Desktop/cloud_server
File Actions Edit View Help
root@kali: ~/.../cloud_server

```
root@kali:~/Desktop/cloud_server# openssl dgst -sha256 -verify cloudPubKey_A.pem -signa
ture singedDoubCipherText_from_A.sign doubCipherText.enc
Verified OK
root@kali:~/Desktop/cloud_server#
root@kali:~/Desktop/cloud_server#
root@kali:~/Desktop/cloud_server# openssl dgst -sha-1 -verify cloudPubKey_A.pem -signat
ure singedDoubCipherText_from_A.sign doubCipherText.enc
dgst: Unrecognized flag sha-1
dgst: Use -help for summary.
root@kali:~/Desktop/cloud_server#
```

Figure 14. Verification of Signed Message

G. Final Stage of Decryption Using AES Algorithm

Since the common AES key is known by both parties, the tool "aescrypt" is used again on the party B's side to decrypt the ciphered data to its clear text. The figure below shows how the final stage of decryption is done and the message obtain and displayed for the party to read.

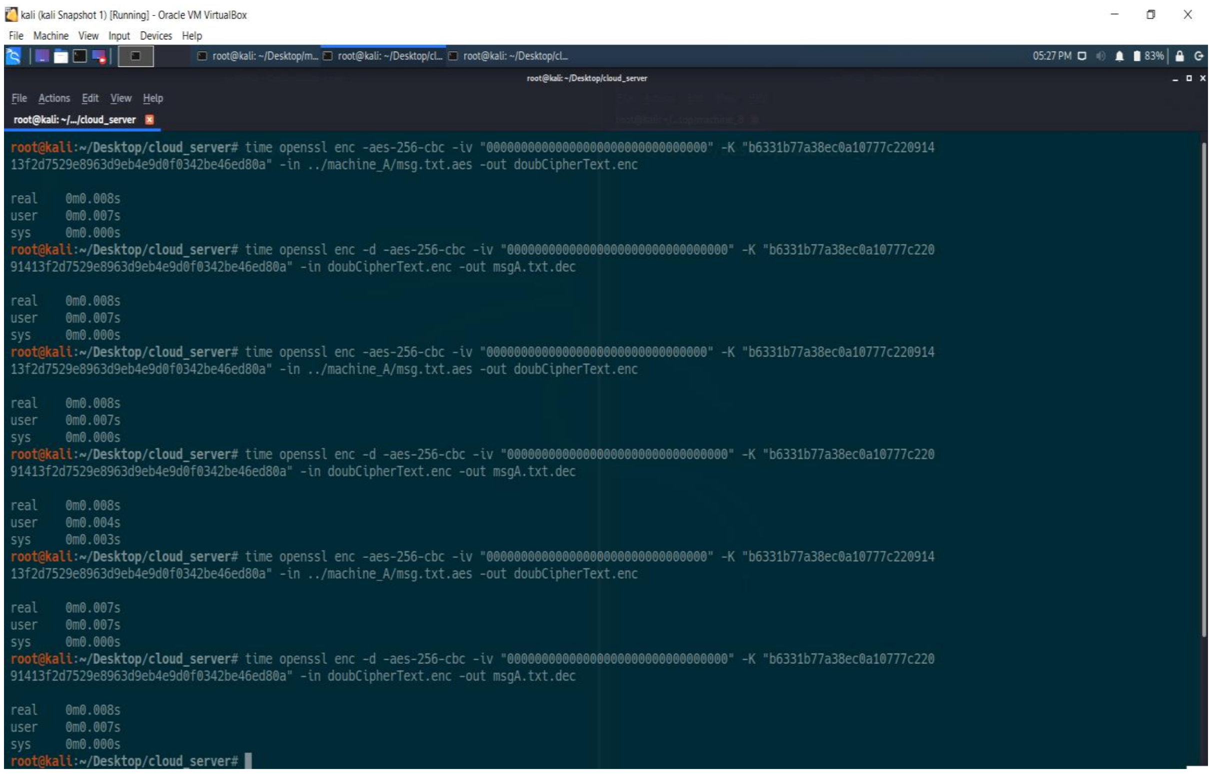

05:15 PM
87%
root@kali: ~/Desktop/machine_B
File Actions Edit View Help
root@kali: ~/...top/machine_B

```
root@kali:~/Desktop/machine_B# cp ../cloud_server/msgA.txt.dec .
root@kali:~/Desktop/machine_B# ls
hashedSecret_B.hash  privKey_B.pem    sharedSecret_B.pem
msgA.txt.dec         publicKey_B.pem
root@kali:~/Desktop/machine_B# aescrypt -d -p 1234 msgA.txt.dec
root@kali:~/Desktop/machine_B# ls
hashedSecret_B.hash  msgA.txt.dec   publicKey_B.pem
msgA.txt             privKey_B.pem  sharedSecret_B.pem
root@kali:~/Desktop/machine_B# cat msgA.txt
Hello World
root@kali:~/Desktop/machine_B# aescrypt -d -p 234 msgA.txt.dec
Error: Message has been altered or password is incorrect
root@kali:~/Desktop/machine_B#
```

Figure 15. Final Stage of Decryption Using AES Algorithm

H. Calculation of the Encryption and Decryption Time

The encryption and decryption time are the time it takes to encrypt and decrypt data in the cloud respectively.


kali (kali Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@kali: ~/Desktop/m...
root@kali: ~/Desktop/cl...
root@kali: ~/Desktop/cl...
05:27 PM
83%
root@kali: ~/Desktop/cloud_server
File Actions Edit View Help
root@kali: ~/.../cloud_server

```
root@kali:~/Desktop/cloud_server# time openssl enc -aes-256-cbc -iv "00000000000000000000000000000000" -K "b6331b77a38ec0a10777c220914
13f2d7529e8963d9eb4e9d0f0342be46ed80a" -in ../machine_A/msg.txt.aes -out doubCipherText.enc

real    0m0.008s
user    0m0.007s
sys     0m0.000s
root@kali:~/Desktop/cloud_server# time openssl enc -d -aes-256-cbc -iv "00000000000000000000000000000000" -K "b6331b77a38ec0a10777c220
91413f2d7529e8963d9eb4e9d0f0342be46ed80a" -in doubCipherText.enc -out msgA.txt.dec

real    0m0.008s
user    0m0.007s
sys     0m0.000s
root@kali:~/Desktop/cloud_server# time openssl enc -aes-256-cbc -iv "00000000000000000000000000000000" -K "b6331b77a38ec0a10777c220914
13f2d7529e8963d9eb4e9d0f0342be46ed80a" -in ../machine_A/msg.txt.aes -out doubCipherText.enc

real    0m0.008s
user    0m0.007s
sys     0m0.000s
root@kali:~/Desktop/cloud_server# time openssl enc -d -aes-256-cbc -iv "00000000000000000000000000000000" -K "b6331b77a38ec0a10777c220
91413f2d7529e8963d9eb4e9d0f0342be46ed80a" -in doubCipherText.enc -out msgA.txt.dec

real    0m0.008s
user    0m0.004s
sys     0m0.003s
root@kali:~/Desktop/cloud_server# time openssl enc -aes-256-cbc -iv "00000000000000000000000000000000" -K "b6331b77a38ec0a10777c220914
13f2d7529e8963d9eb4e9d0f0342be46ed80a" -in ../machine_A/msg.txt.aes -out doubCipherText.enc

real    0m0.007s
user    0m0.007s
sys     0m0.000s
root@kali:~/Desktop/cloud_server# time openssl enc -d -aes-256-cbc -iv "00000000000000000000000000000000" -K "b6331b77a38ec0a10777c220
91413f2d7529e8963d9eb4e9d0f0342be46ed80a" -in doubCipherText.enc -out msgA.txt.dec

real    0m0.008s
user    0m0.007s
sys     0m0.000s
root@kali:~/Desktop/cloud_server#
```

Figure 16. Encryption Time and Decryption Time in the Cloud Space

The simulation recorded different times during the encryption and decryption processes, and this was due to these factors: Load on Central Processing Unit (CPU), Memory and Hard Disk giving different output times. The outputs produced as shown in figure 16 are indicated as 'real', 'user' and 'sys'.

REAL Time: The real time between initiation and termination of the program including other processes that maybe running at the background.

USER Time: The CPU time used in the user-mode is known as the User. It uses the user space instead of the kernels.

SYS Time: The sys time refers to the time spent within the process in the kernel. This time is spent in the system calls within the kernel.

I. Discussions of Findings

The ECDH key exchange for producing Shared Key makes it totally impossible for third parties or attackers to produce same key in decrypting the data. The IV also adds up to the increasing level of security which makes it even harder for intruders to get to decrypt the data. Moreover, data is encrypted twice, that is first on the client's machine and later in the cloud. All this proves the enhancement of the security of data hence the confidentiality.

By implementing ECC, the smaller keys produce an equivalent level of encryption as compared to the other asymmetric encryption even at a faster rate showing the enhancement of the speed and security of data.

The integrity of the data is also maintained by the usage of the ECDSA which allows for the integrity check of the data. The trust level is enhanced from the end user's perspective as well. This is because, the user gets to encode the data even before passing through the cloud for further encryption. This will inform users of the privacy of their data and encourage more users to patronize the usage of cloud services.

Lastly, it is found that the load on both CPU, memory and disk determines the time it takes to encrypt and decrypt data.

V. CONCLUSION

This paper presented a two-level cryptographic technique and a model for the enhancement of data security in cloud computing. The model employs the use of both symmetric and asymmetric encryption algorithm (AES and ECC) to enhance the security of data against intruders or third parties, denying them from having access to the real data enabling confidentiality, integrity of the data, time taken to perform cryptographic operations and improving the trust level of users of CC and increasing level of speed with the use of smaller keys of ECC in the cryptographic process.

ACKNOWLEDGMENT

We would like to acknowledge the guidance, suggestions and tireless efforts offered to us by our Supervisor Dr. Edward Danso Ansong, in ensuring the accomplishment of this paper.

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," National Institute of Standards and Technology Special Publication, Gaithersburg, 2011.

[2] Cisco, "Cisco Global Cloud Index: Forecast and Methodology, 2016–2021 The White Paper," Cisco Global Cloud Index, US, 2018.

[3] A. P. D. S. A. Abbas and A. A. B. Maryoosh, "Improving Data Storage Security in Cloud Computing Using Elliptic Curve Cryptography," *IOSR Journal of Computer Engineering (IOSR-JCE)*, vol. 17, no. 4, pp. 48-53, 2015.

[4] V. Muthurajan and B. Narayanasamy, "An Elliptic Curve Based Schnorr Cloud Security Model in Distributed Environment," *The Scientific World Journal*, vol. 2016, pp. 1-8, 2016.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 9 Issue 09, September-2020

[5] S. N. Mendonca, "Data Security in Cloud using AES," *International Journal of Engineering Research & Technology (IJERT),* vol. 7, pp. 205-208, 2018.

[6] A. Kumari, M. Y. Abbasi, V. Kumar and A. A. Khan, "A Secure User Authentication Protocol Using Elliptic Curve Cryptography," *Journal of Discrete Mathematics and Cryptography*, vol. 22, pp. 521-530, 2019.

[7] E. Conrad, S. Misenar and J. Feldman, "Advanced Encryption Standard- An Overview," in *Eleventh Hour CISSP:: Study Guide*, Elsevier Inc, 2017.

[8] T. B. Azad, "Understanding XenApp Security," in *Securing Citrix Presentation Server in the Enterprise*, 2008.

[9] P. Salim Ali Abbas and A. A. B. Maryoosh, "Data Security for Cloud Computing based on Elliptic Curve Integrated Encryption Scheme (ECIES) and Modified Identity based Cryptography (MIBC)," *International Journal of Applied Information Systems (IJAIS),* vol. 10, pp. 7-13, 2016.

[10] W. Stallings, Cryptography and Network Security Principles and Practice, 5th Edition ed., Pearson Education, Inc, 2011.

[11] A. M. Sagheer, "Enhancement of Elliptic Curve Cryptography Methods," University of Technology, 2004.

[12] P. Nohe, "You should be using ECC for your SSL/TLS certificates," 2019. [Online]. Available: https://www.thesslstore.com/blog/you-should-be-using-ecc-for-your-ssl-tls-certificates/. [Accessed 15 01 2020].

[13] K. Rabah, "Data Security and Cryptographic Techniques-A Review," *Info Technology Journal*, pp. 106-132, 2004.

[14] A. P. Shaikh and V. Kaul, "Enhanced Security Algorithm using Hybrid Encryption and ECC," *IOSR Journal of Computer Eng. (IOSR-JCE),* vol. 16, no. 3, pp. 80-85, 2014.