Elliptic Curve Diffie-Hellman (ECDH) Analogy for Secured Wireless Sensor Networks

Stephen Aikins-Bekoe
Kwame Nkrumah University of Science and Technology
Department of Computer Science

James Ben Hayfron-Acquah
Kwame Nkrumah University of Science and Technology
Department of Computer Science

ABSTRACT

Wireless Sensor Networks (WSNs) with traditional cryptography are applied in many areas including healthcare, earth sensing and area monitoring. However, severe security constraints coupled with malicious attacks and threats revolve around the implementation of Wireless Sensor Networks which pose undesirable security performance as well as affect the maintenance of proper functionality of wireless sensor systems. Due to such circumstances, it is important to recognise the need for a holistic and robust security to ensure WSNs are well established and protected. In this study a more robust technique for a wireless sensor network system is employed. The algorithm for Elliptic Curve Diffie Hellman key exchange is studied and analyzed using PyCryptodome package and the Elliptic Curve Integrated Encryption Scheme. The study is carried out in comparison to Rivest-Shamir-Adleman (RSA) to assess the strengths of ECC in key generation and encryption/decryption process. The results obtained from the analysis reveals that ECC provides a higher level of security and also has very small key size in comparison to RSA, which makes possible implementations more compact for some level of security.

Keywords

Elliptic Curve, Diffie-Hellman, Wireless Sensor Network, Public key, Encryption, Decryption

1. INTRODUCTION

Recently, advances in integration between Miniature Embedded Processors, wireless interfaces and micro-sensors have influenced the forth coming of Wireless Sensor Network (WSN). The emerging technology of WSNs have gained worldwide attention due to their great importance in recent years. WSNs have been incorporated in several solicitation domains due to its rapid deployment, low cost, capability for self-organization, low energy and data processing cooperation which includes applications for habitat monitoring, military applications, intelligent agriculture and home automation.

Sensing element and node devices are limited resources deployed in hostile environments to properly sense data with efficiency. Although Wireless Sensor Networks with traditional cryptography are applied in many areas including healthcare, earth sensing and area monitoring, it also poses austere security challenges which includes sensor data forgery, denial of service attacks, eavesdropping and the compromise of sensor nodes physically (Ayaz & Mohiuddin, 2016). Hence, potential preparation of WSNs for any real-time applications must address several issues, together with system design, protocol functionalities and security. Provision of security to these resourced sensor networks may be a terribly difficult work in comparison to typical networks, like wide area network (WAN) and local area network (LAN) (Rehana, 2009). Henceforth, providing a more appropriate and secured network has collectively emerged as the essential issue in WSNs, thus the state-of-art ought to listen to the way to set-up secure, easy and reliable WSNs. Currently, traditional cryptography is not possible to protect Wireless Sensor Networks from threats or attacks because of the unpredictable wireless channel and the network security. This comes up as a result of the many limitations of resources such as computational power, limited energy and lower memory.

Wireless sensor networks (WSNs) collects data from its environment, store and process them, and finally sends the processed data to users, either upon event detection or on demand (Ali, 2013). They are identified as groups of widely distributed sensors used in monitoring and recording the physical conditions of its environment through organization or collection of data and reporting them to a central point (Boussag, 2017), through wireless links. This makes it crucial to encrypt sensitive data that are transported from a node to another node in wireless sensor networks so that it will not be modified by or disclosed to any unauthorized party.

Data encryption and decryption however, hinges on the cryptography scheme used and the generated key type. Cryptography involves the technique for securing communication in the presence of third parties, which is categorized into public key cryptography, secret key cryptography and hash function (one-way cryptography) based on the keys employed.

Public key cryptographic technique employs two keys - private and public keys - which are mathematically related. In the process of decryption and encryption, private and public keys are required for both process to work. Public key cryptography depends on mathematical functions that can be computed easily but relatively difficult to compute its inverse. Among the public key cryptographic schemes in modern days for key generation, Elliptic Curve Cryptography (ECC) is found the latest encryption method which offers high level of security.

The problem of authenticated querying is crucial in Wireless Sensor Networks. The unattended nature of some WSNs make it prone to node compromise attack. The resource and network constraints along with very different attacks impose many difficult necessities for the safety style in WSNs. This sophisticated security or authentication scheme requires that the design of wireless sensor networks must be robust against sensor compromise and attacks which introduces more security challenges. However, the robust design to achieve best security is often not achieved during wireless networking which mostly depends on the employed keys and the encryption/decryption process. In addition, longer cryptographic keys require more bandwidth, more space and an extra processor power. And also takes time for key generation, data encryption and decryption, which is mostly associated to most modern used cryptographic keys like the RSA.

In this study, the application of elliptic curves in cryptography for the construction of public and secret keys is carried out. To analyze the strength and feasibility of elliptic curves in cryptography, data encryption/decryption process is studied in line with elliptic curves. The Elliptic Curve - Diffie Hellman (EC-DH) cryptography technique identified as most robust technique for key generation is then employed to construct secured public and private keys. And the analysis carried out using the PyCryptodome package and the Elliptic Curve Integrated Encryption Scheme (ECIES).

The rest of the paper is organized as follows: the second section which discusses the concept of elliptic curve cryptography, Elliptic Curve Diffie-Hellman analog, discrete problem. The pycryptodome library and the elliptic curve integrated encryption scheme for analysis of the study is captured in section three. The fourth section considers implementation and analysis of results and finally the study is concluded in the fifth section.

2. PRELIMINARIES

2.1 Elliptic Curve Cryptography

The Elliptic Curve Cryptography (ECC) is one of the public key cryptographic schemes. In general, users or devices which form part of the communication processes in public key cryptography have a key pair - a private key and a public key - for the cryptographic scheme operations. With the private key, only one of the users know it, however the public key is made available to every user involved in the communication process. ECC is a modern encryption method which offers stronger level of security. In comparison to RSA algorithms, 256 bits of ECC equals 3072 bits of RSA keys (Haakegaard and Lang, 2015).

The essence of constructing short keys is to obtain less power of computations and secured and fast connections, which is ideal for Tablets and Smartphones. The best cryptography scheme for wireless applications is the ECC due to its limited battery life, compute memory and power (Blab and Zitterbart, 2005). The certificate from elliptic curve cryptography (ECC) allows for small key size while providing a higher security level. The smaller key sizes, the more compact its implementations for a certain security level, which is an implication of faster operations of cryptography. Method for the creation of ECC certificate key differs entirely from the other algorithms, which relies on using public keys for encryption processes and private keys for the process of decryption. ECC has longer potential lifespan which starts small and comes with slow potential for growth (Hankerson et al., 2000).

2.2 The Elliptic Curve

The mathematical operations of cryptographic schemes based on elliptic curves are defined on elliptic curves. In ECC, we want an elliptic curve E over a finite field F_p where p is a prime number more than 3. An elliptic curve is defined by equation 1:

$$y^2 = x^3 + px + q \tag{1}$$

where $p, q \in F_p$ and $4p^3 + 27q^2 \neq 0$ (Miller, 1985). The constant values in the equation offers different elliptic curves. Every point (x, y) that will satisfy the elliptic curve equation including a point at infinity must lie on the curve. In ECC, the private keys are random numbers and public keys are points in the curve obtained by multiplying the private keys with generator G in the curve. The parameters of the curve $'p'$ and $'q'$, the point generator G, together with few additional constants makes up the ECC domain parameter (Anoop, 2000).

More generally, the form of the elliptic curve is: (Vagle, 2000)

$$y^2 + py = x^3 + qx^2 + rxy + sx + t, \qquad p, q, r, s, t \in F_p \tag{2}$$

According to Miller (1985), a and b must be chosen for elliptic curves in cryptography such that

$$4p^3 + 27q^2 \neq 0.$$

2.3 Arithmetics of Elliptic Curves

2.3.1 Point Addition. This operation involves the addition of two points K and J in the elliptic curve to obtain L on the same elliptic curve. This is illustrated in figure 1.

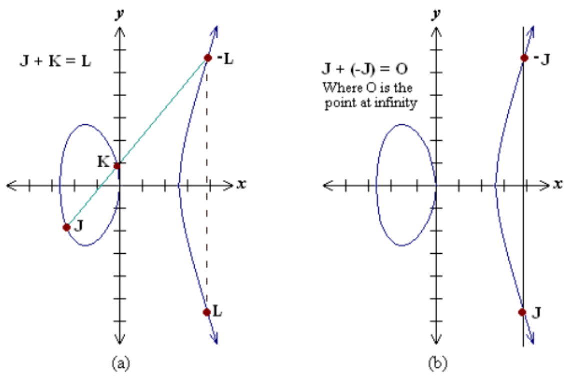


Fig. 1. Point Addition in Elliptic Curves

Considering the points K and J on the elliptic curve described in figure 1(a), provided $K \neq -J$, we can draw a line via points K and J to produce point $-L$ which also lies in the curve. When $-L$ is reflected along the $x-$axis, a point L is obtained which is the result of adding K and J. On the other hand, if the line passes through the point $K = -J$, then the line intersect at a point at infinity O, which we call the additive identity of the elliptic curve group as described in figure 1(b).

The reflection of points along the $x-$axis is a negative of the point. Analytically, considering the points $K = (x_K, y_K)$ and

$J = (x_K, y_K)$, the addition of these points in the curve produce one more point $L = (x_K, y_K)$, where

$$y_L = -y_J + s(x_J - X_L)(\text{mod } p) \text{ and } x_L = s^2 - x_J - x_K(\text{mod } p) \quad (3)$$

and

$$s = \frac{y_J - y_K}{x_J - x_K} \ (\text{mod } p)$$

is the gradient of the line through K and J.

We have that $J + K = O$, provided $K = -J$ thus, $K = (x_J, -y_J(\text{mod } p))$ where O is the point at infinity. (Silverman, 1986).

2.3.2 Point Doubling. Consider point $J = (x_J, y_J)$, where $y_J \neq 0$. If $L = 2J$ where $L = (x_L, y_L)$ then we obtain

$$y_L = -y_j + s(x_J - x_L) \ (\text{mod } p) \quad \text{and} \quad x_L = s^2 - 2x_J \ (\text{mod } p) \quad (4)$$

where

$$s = \frac{3x_J^2 + a}{2y_J} \ (\text{mod } p)$$

is the tangent at J and a is a parameter in the chosen elliptic curve. (Silverman, 1986)

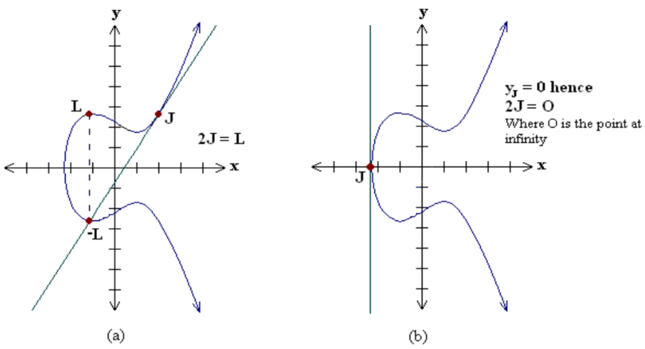


Fig. 2. Example of Point Doubling in Elliptic Curves

2.3.3 Point Subtraction. Given the two unique points $K = (x_K, y_K)$ and $J = (x_J, y_J)$, we have that $J - K = J + (-K)$ where $-K = (x_K, -y_K \ (\text{mod } p))$

2.3.4 Point Multiplication. Consider that the scalar k is multiplied with P to result in point $Q = kP$ on the elliptic curve. To multiply the point P by the integer k, point addition and point doubling are mainly. This method of integer multiplication is referred as 'double and add' method. For instance, given $k = 23$, we have $kP = 23P = 2(2(2(2P) + P) + P) + P$. (Silverman, 1986)

2.4 Discrete Logarithm Problem

The ECC security depends on how difficult is the Elliptic Curve Discrete Logarithm Problem. Given the points Q and P on an elliptic curve with scalar k so that $kP = Q$. With Q and P, it is still infeasible computationally, to find k, provided k is large enough. The scalar k is the discrete logarithm of Q to the base P (Vagle, 2000).

2.5 Elliptic Curve Diffie-Hellman (EC-DH) Analog

Elliptic Curve Diffie-Hellman protocol is a key agreement scheme allowing party A and party B to construct shared secret keys which are used for algorithms of private keys. The two parties do public information exchange to one another. Employing the public information and a private information, the two parties are able to generate a shared secret key. Third parties without an idea on the private information of both parties can not calculate the secret shared key from the public information available.

2.6 The Steps Involved in Elliptic Curve Cryptography

The processes of decryption and encryption in ECC can be categorized into three main steps namely:
The encryption and decryption process can be grouped into three main parts namely:

(1) Secret Key Generation
(2) Encryption
(3) Decryption

2.6.1 Secret Key Generation. The first step in elliptic curve cryptographic process is the generation of a secret key to encrypt messages before it is transfered to the intended recipient. The secret key construction is done using the Elliptic Curve Diffie-Hellman analog. ECDH is an improvement on the traditional Diffie-Hellman key agreement algorithm based on elliptic curves. Diffie-Hellman method generates secret shared keys between two parties in a communication so that a third party cannot see the secret just by observation of the communication. Hence the method of Diffie-Hellman does not need a prior contact between both parties. Each of the two parties generates dynamic private and public keys for use. The public keys generated are exchanged between them. Afterwards, each party uses its private key to combine with the public key of the other party to generate the shared secret. The steps involved in the generation of secret keys in elliptic curve cryptography is illustrated in the following subsection.

In generating shared secret keys between two parties using the Elliptic Curve Diffie-Hellman approach;

—Both parties must first agree on a publicly-known data items
  (1) The values of elliptic curve equation and a prime, p
  (2) The elliptic group obtained from the elliptic curve equation
  (3) A base point, B, obtained from the elliptic group

—Each of the two parties generates their key pair (private or public)
  (1) the private key is an integer, n, chosen from [1, p-1]
  (2) the public key, Q is the product of base point and private key (i.e., $Q = xB$)

—Each party then uses the public key, $Q = xB$ generated to generate a secret key by multiplying Q by the selected secret integer (i.e., xQ)

2.6.2 Encryption. The second step in the elliptic curve cryptographic process after the generation of a shared secret key is *encryption*. For party A to encrypt any message and send to party B, a secret shared key, P_S generated between the parties A and B is used To obtain the encrypted message M_E before sending to the other party, the secret shared key, P_S is added to the message such that $M_E = P_S + P_M$. Finally, a ciphered text $C_A = \{P_B, P_S\}$ of the encrypted message M_E is sent to the receiver for decryption.

2.6.3 Decryption. The third step involved in ECC is *decryption* where an encrypted message is decrypted. After the message is delivered to party B, the encrypted message is first decrypted to get the original message. To decrypt the encrypted message from A, party B has to subtract the shared secret key, P_S from the encrypted message M_E such that $P_M = M_E - P_S$, to obtain the original message. Hence, it is a challenge for an adversary or third parties to obtain the original message once he/she has only the ciphered text. For instance, for third parties to be able to decrypt the ciphered text, knowledge of the private key of the receiver is needed in order to obtain the secret shared key. Which implies, the third party is to compute the multiplier (i.e., solve the discrete logarithm problem) provided he is given the public key of the receiver and the point P on the elliptic curve.

3. SOFTWARE PACKAGES

3.1 The PyCryptodome Library

PyCryptodome toolkit is a self-contained Python package of low-level cryptographic primitive. The PyCryptodome toolkits support Python 2.6 or newer, and all versions of Python 3.
Unlike OpenSSL, PyCryptodome is not a wrapper to a separate C library. The algorithms here are implemented in pure Python to a largest extent possible. Just a part of the algorithm that are extremely critical to performance (e.g. block ciphers) are implemented as C extensions (Legrandin, 2018). It contains a built in module for elliptic curve cryptography for private and public key generation in cryptography. And uses the recommended NIST elliptic curves which is captured in section 3.1. All PyCryptodome is organized into sub-packages which are designed to solve specific class of problems.

The NIST Recommended Elliptic Curves

This is a group of recommended elliptic curves for use by the Federal government and contains the choice for underlying field and private key length. SHA-1 and the methods as described in IEEE and ANS X9.62 Standard 1363-2000 standards were used to generated the NIST curves. In 1999, a non-regulatory agency of the United States Department of Commerce, and a physical sciences laboratory, the National Institute of Standards and Technology (NIST), made the Elliptic Curve Digital Signature Algorithm a standard one in Federal Information Processing Standards (FIPS) 186-2, guidelines and specifications that apply to federal computer systems. The 15 elliptic curves of varying security levels, called *NIST curves* (Federal Informatin Processing Standards Publication, 2013) were recommended by NIST. Two kinds of these curves are:

—**Pseudo-random curves:** are elliptic curves with generated coefficients from the seeded cryptographic hash function output. To easily verify generated coefficients by this hash function, the domain parameter seed value mostly obtained alongside with these coefficients.

—**Special curves:** which have their underlying field and coefficients specifically selected in order to make optimal the efficiency of the operations of elliptic curves.

3.2 Elliptic Curve Integrated Encryption Scheme

The Elliptic Curve Integrated Encryption Scheme (ECIES) library is a hybrid encryption system proposed by Victor Shoup in 2001 which combines secp256k1 and AES-256-GCM (powered by coincurve and pycryptodome) to provide an API of encrypting with secp256k1 public key and decrypting with secp256k1's private key. The ECIES uses secp256k1 to generate an elliptic and encrypts/decrypts by AES-256-GCM with the keys generated from secp256k1. The secp256k1 is an Elliptic Curve Digital Signature Algorithm (ECDSA) which is based on elliptic curve cryptography. The secp256k1 is the curve

$$y^2 = x^3 + 7$$

over a finite field. It is defined in Standards for Efficient Cryptography (SEC). Most commonly-used curves have a random structure, but secp256k1 was constructed in a special non-random way which allows for especially efficient computation. As a result, it is often more than 30% faster than other curves if the implementation is sufficiently optimized. Also, unlike the popular NIST curves, secp256k1's constants were selected in a predictable way, which significantly reduces the possibility that the curve's creator inserted any sort of backdoor into the curve. The graph of the secp256k1 is shown in figure 3

Table 1. Some NIST Recommended Standardized Elliptic Curves

Curve	Possible identifiers
NIST P-256	'NIST P-256', 'p256', 'P-256', 'prime256v1', 'secp256r1'
NIST P-384	'NIST P-384', 'p384', 'P-384', 'prime384v1', 'secp384r1'
NIST P-521	'NIST P-521', 'p521', 'P-521', 'prime521v1', 'secp521r1'

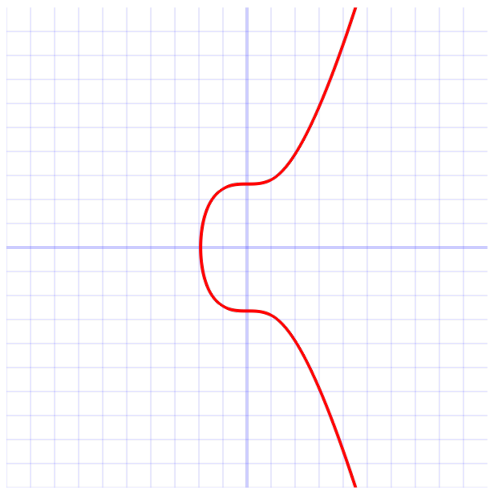


Fig. 3. The graph of secp256k1's elliptic curve $y^2 = x^3 + 7$ over the real numbers

AES-256-GCM encryption is an encryption option for updated installations and default encryption. The Galois/Counter mode (GCM) of operation (AES-128-GCM), however, operates quite differently. As the name suggests, GCM combines Galois field multiplication with the counter mode of operation for block ciphers. ECIES has two steps:

—Use ECDH to calculate an AES session key.

—Use this AES session key to encrypt/decrypt the data under AES-256-GCM.

4. IMPLEMENTATION AND ANALYSIS OF RESULTS

4.1 Key Generation with ECC in PyCryotodome

In this study, the PyCryptodome package among other packages is used for analyzing the operations of elliptic curve cryptography in generation of keys. The main package employed includes the Crypto.Publickey modules to generate, export and/or import public keys in ECC.

Figure 4 shows the python script for the generation of public and private with elliptic curve cryptography using the Crypto.PublicKey module in PyCryptodome.

```python
from Crypto.PublicKey import ECC
import sys

## This generates the Elliptic Curve and a point on the curve
#key = ECC.generate(curve='P-256')
key = ECC.generate(curve='secp384r1')

print 'generate key', key  ## print the key point

#################################################################
f = open('myprivatekey.pem','wt')

f.write(key.export_key(format='PEM'))

f.close()

### The following reloads the key into the application for ecnryption/decryption
f = open('myprivatekey.pem','rt')
key = ECC.import_key(f.read())
```

Fig. 4. The Python script for Key Generation

4.2 First Run of Script

The script is then run on "5.8GB, Intel Pentium(R) CPU B940 @ 2.00gHZ, Linux 64-bit OS, Intel Sandybridge Mobile Graphics" using the terminal with the command: "python ./ecckey.py". The output is shown in figure 5 which shows the used NIST curve, the private and the private keys generated.

```
richard@richard-VPCEH11FX:~/Desktop$ python ./ecckey.py
generate key EccKey(curve='NIST P-384', point_x=7249561678908176617233571318667945913383413043915624593250357132772954574439230596990323322160278685408006506249063, point_y=16964775432959197489176107715892994737709370438262100107320557953409892905566347955258631258917323386134854848582192, d=36706870466286409592276218340425926768269174168867729562816667613506005537925629872377791280975741993062788839277105)
```

Fig. 5. Output of the first run of the Script.

4.2.1 First Run. The 'secp384r1' in the code generates the recommended NIST elliptic curve, 'NIST P-384' in the output. However, different curves can be chosen from the NIST recommended curves by specifying the possible identifier in the code. The parameter d is a random integer (secret key) and (point_x,point_y) is the base point on the elliptic curve, which is used to generate the private key for encryption processes. The private key generated is written to a Privacy-Enhanced Mail (PEM) file format for storing and sending. The .pem file generated is shown in figure 6.

4.2.2 From the Second to the Fourth Run of the Script. The same python script with the same curve type was run three more times and the output are shown in figures 7, 8 and 9. In each of these cases, the .pem file generated for the encryption/decryption shows the same content as in figure 6 when it is opened.
From the output of the first run through to the fourth, it is observed that, the generated public key (i.e., d and (point_x, point_y)) varies.

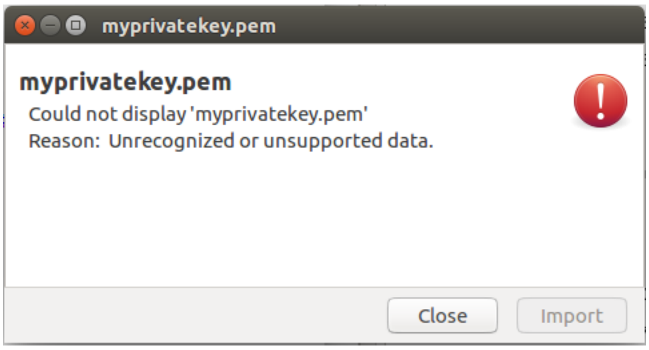


Fig. 6. The .pem private key file generated for encryption/decryption

```
richard@richard-VPCEH11FX:~/Desktop$ python ./ecckey.py
generate key EccKey(curve='NIST P-384', point_x=7249561678908176617233571318667945913383413043915624593250357132772954574439230596990323322160278685408006506249063, point_y=16964775432959197489176107715892994737709370438262100107320557953409892905566347955258631258917323386134854848582192, d=36706870466286409592276218340425926768269174168867729562816667613506005537925629872377791280975741993062788839277105)
```

Fig. 7. Output of the second run of the Script.

```
richard@richard-VPCEH11FX:~/Desktop$ python ./ecckey.py
generate key EccKey(curve='NIST P-384', point_x=7249561678908176617233571318667945913383413043915624593250357132772954574439230596990323322160278685408006506249063, point_y=16964775432959197489176107715892994737709370438262100107320557953409892905566347955258631258917323386134854848582192, d=36706870466286409592276218340425926768269174168867729562816667613506005537925629872377791280975741993062788839277105)
```

Fig. 8. Output of the third run of the Script.

```
richard@richard-VPCEH11FX:~/Desktop$ python ./ecckey.py
generate key EccKey(curve='NIST P-384', point_x=7249561678908176617233571318667945913383413043915624593250357132772954574439230596990323322160278685408006506249063, point_y=16964775432959197489176107715892994737709370438262100107320557953409892905566347955258631258917323386134854848582192, d=36706870466286409592276218340425926768269174168867729562816667613506005537925629872377791280975741993062788839277105)
```

Fig. 9. Output of the fourth run of the Script.

Thus with the elliptic curve cryptography, server public key is generated every time when the encryption/decryption process is initiated, which is as a result of the random secret integer each party generates.

4.3 Comparison to the Rivest-Shamir-Adleman (RSA) Key Type

From the pycryptodome library, the rsa key was used to compare its output to the ecc key. Rivest-Shamir-Adleman (RSA) is one of the first public-key cryptosystems which is used widely to secure transfer of data. In RSA the encryption key is public which is availbale to whosoever wants to send a message to a recipient while decryption is done with a secret key of the recipient. The script for the key generation of private and public key with RSA using the Crypto.PublicKey module is shown in figure 10.

For comparison purposes, the RSA script was run twice and the output is shown in figures 11 and 12, which shows the private and the private keys generated.

From the output of RSA script for key generation (figures 11 and 12), it is observed that the generated key with the RSA has many characters when compared to the ECC keys which accounts for the key size of RSA being larger than that of ECC.

```
generate key EccKey(curve='NIST P-384', point_x=72495616789081766172
3357131866794591338341304391562459325035713277295457443923059699032332
2160278685408006506249063, point_y=16964775432959197489176107715892994
7377093704382621001073205579534098929055663479552586312589173233861348
54848582192, d=3670687046628640959227621834042592676826917416886772956
2816667613506005537925629872377791280975741993062788839277105)
```

```
from Crypto.PublicKey import RSA
import sys

key = RSA.generate(2048)

print 'generate key', key

private_key = key.export_key()
file_out = open("private.pem", "wb")
file_out.write(private_key)

print 'generate key', private_key

public_key = key.publickey().export_key()
file_out = open("receiver.pem", "wb")
file_out.write(public_key)

print 'generate key', public_key
```

Fig. 10. RSA Key generation Script.

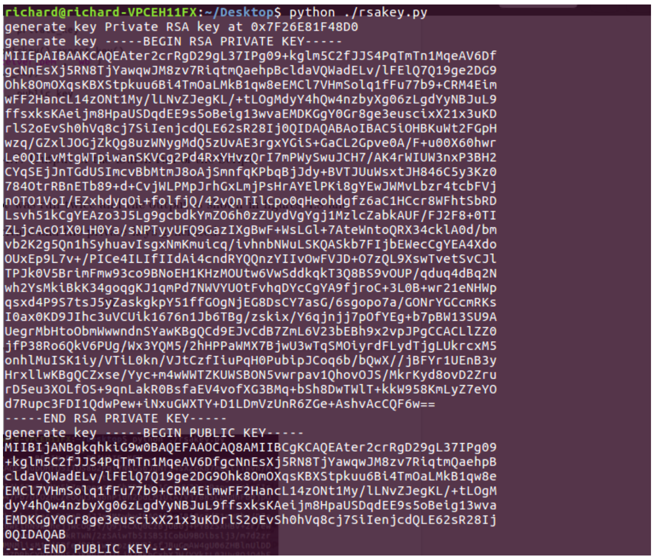


Fig. 11. Output of the first run of RSA Script.

4.4 Encryption/Decryption Process using ECIES

Figure 13 below shows the script for encryption/decryption using the ECIES library. The main module employed here is the Crypto.Ciper for encryption and decryption.

The corresponding output for the first and second run of the above script in figure 13 are shown in figures 14 and 15.

From the output of the first and second run, it is observed that the encryption keys with their corresponding encrypted messages are different with different characteristics in both cases.

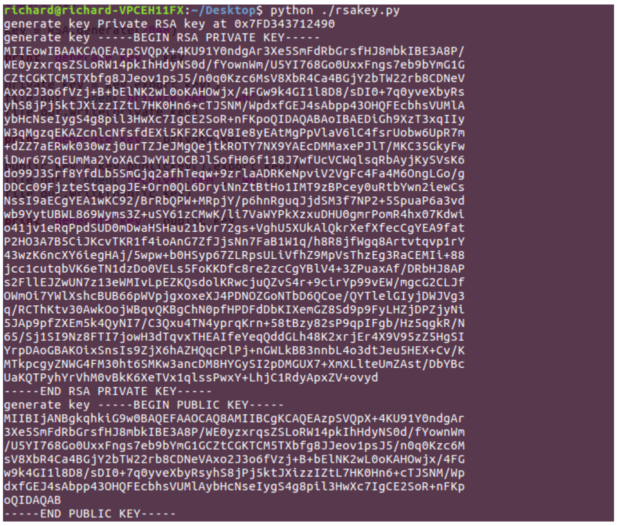
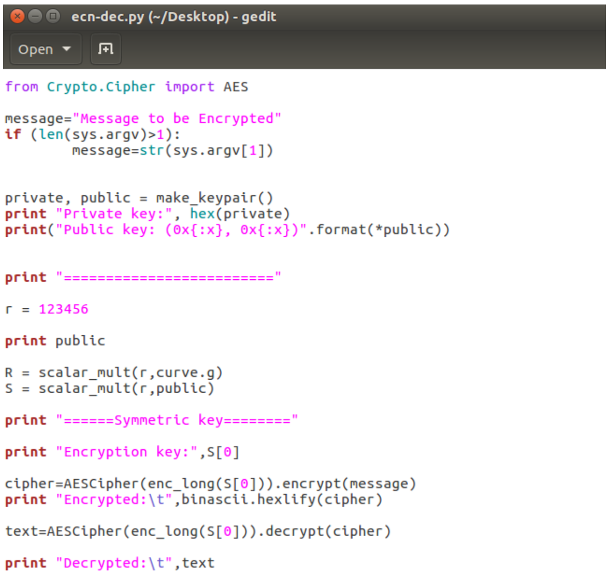


Fig. 12. Output of the second run of RSA Script.



```
from Crypto.Cipher import AES

message="Message to be Encrypted"
if (len(sys.argv)>1):
        message=str(sys.argv[1])


private, public = make_keypair()
print "Private key:", hex(private)
print("Public key: (0x{:x}, 0x{:x})".format(*public))


print "========================="

r = 123456

print public

R = scalar_mult(r,curve.g)
S = scalar_mult(r,public)

print "======Symmetric key========"

print "Encryption key:",S[0]

cipher=AESCipher(enc_long(S[0])).encrypt(message)
print "Encrypted:\t",binascii.hexlify(cipher)

text=AESCipher(enc_long(S[0])).decrypt(cipher)

print "Decrypted:\t",text
```

Fig. 13. Encryption/Decryption Script with ECIES.

4.5 Comparison to RSA Encryption

Figure 16 shows the script for encryption/decryption using the ECIES library. The main module employed here is the Crypto.Ciper from which PKCS1_OAEP is imported for the encryption.

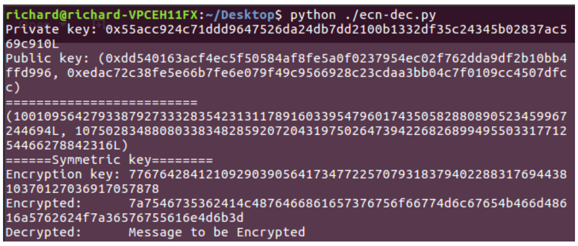


Fig. 14. First Output of Encryption/Decryption Script with ECIES.

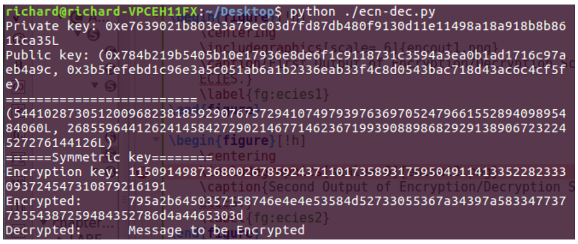


Fig. 15. Second Output of Encryption/Decryption Script with ECIES.

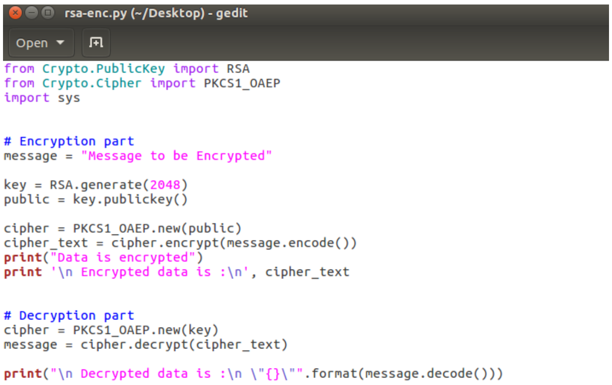


Fig. 16. Encryption/Decryption Script with RSA.

The output for the first and second run of the RSA encryption script (figure 16) are shown in figures 17 and 18

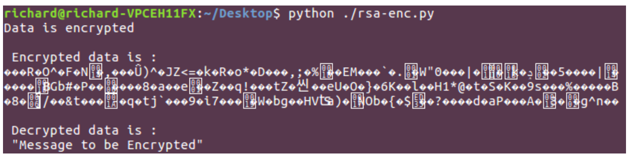


Fig. 17. First Output of Encryption/Decryption Script with RSA.

The output of the RSA encryption/decryption scheme shows a different characteristic of the encrypted data from that of ECC. Better still, it is observed that the number of characters for the encrypted data in RSA is more than that of ECC, which in effect makes RSA having larger key sizes than ECC.

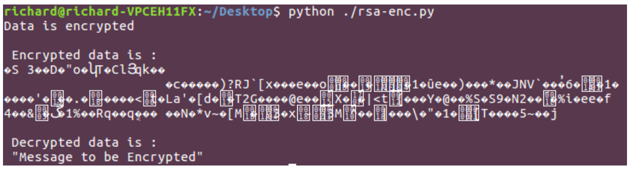


Fig. 18. Second Output of Encryption/Decryption Script with RSA.

5. CONCLUSION

The wireless sensor network system consists of spatially dispersed sensors usually dedicated to monitor and also record the physical conditions of the environment and also to organize the data collected at a central location. The security of the network system becomes a concern in the transmission of environmental data from one sensor node to another. Hence for safe transmission of data, good and highly secured keys must be generated to encrypt and decrypt information in order to protect the network system from third parties or attacks. In this study, the elliptic curve cryptography (ECC), a public key cryptography (PKC) based upon elliptic curves is studied and analyzed for its potential in encryption and decryption in wireless network systems using elliptic curve analogs.

From this study, it is observe from the output of the elliptic curve cryptographic key generation code that, for each attempt to send or transfer an information from a sender to a receiver, a new private key is generated. This is as a result of the random private key (integer) which is chosen to generate the shared private key for the two parties. The variant shared private key generated for each communication in the elliptic curve cryptography makes the network protocol less predictive by attackers during communication.

On the account of security, the encryption results in both Run 1 (19 and 20) and Run2 is considered. In both cases, it is observed that the encrypted message has different characteristics (see figures 19 and 20, *Encrypted*).

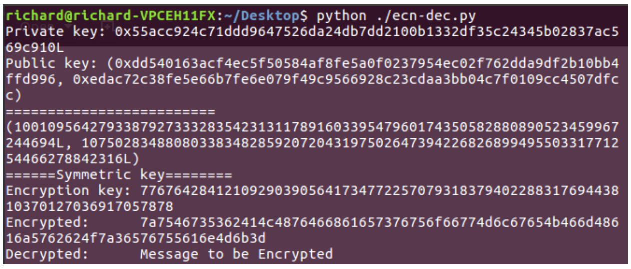


Fig. 19. First Output of Encryption/Decryption Script with ECIES.

In such conditions, finding a secret key by a third party to decrypt the encrypted message in the second run using the characteristics of the first encrypted message is almost highly impossible. Hence there is a high level of security for elliptic curve cryptography base network protocols, for a third party to intrude.

On the account of key size, the elliptic curve cryptography (ECC) certificate allows key size to remain small while providing a higher level of security. ECC key size in comparison to RSA key is shown in table 5.

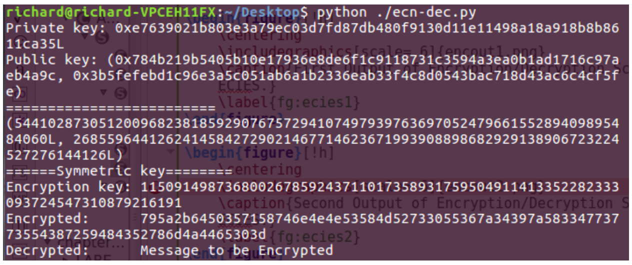


Fig. 20. Second Output of Encryption/Decryption Script with ECIES.

Table 2. Comparison of ECC and RSA (Source: www.ssl2buy.com)

Minimum size (bits) of Public Keys		**Key Size Ratio**	
RSA	ECC	ECC to RSA	Valid
1024	160-223	1:6	Until 2010
2048	224-255	1:9	Until 2030
3072	256-383	1:12	Beyond 2031
7680	384-511	1:20	
15360	512+	1:30	

This is evident from figure the output of the run of both the RSA and ECC key scripts as shown in figures 11 and 5. It was observed that the RSA key has much more characters in comparison to the ECC key which contributes to the ECC having much more smaller key size in comparison to the RSA key. The smaller ECC key size makes possible much more compact implementations for a given level of security. This in essence results in faster cryptographic operations, making it more feasible to run on smaller chips or for achieving more compact software. The short key size also allows for less computational power and fast and secure connection for wireless sensor networks which transmits data with very small sensor nodes.

6. RECOMMENDATION

For a secured wireless sensor network system, the elliptic curve cryptography is recommended.

7. REFERENCES

[1] Ali, M. Zulfiker, (2013). *A robust user authentication scheme for wireless networks*, Toronto, Ontario, Canada.

[2] Anoop M.S. (2000), Elliptic Curve Cryptography, An Implementation Guide.

[3] Ayaz H. M., Ummer I. & Mohiuddin B. (2016), *Secured Data Acquisition System for Smart Water Applications using WSN*, Indian Journal of Science and Technology, 9(10), DOI: 10.17485/ijst/2016/v9i10/86694

[4] Blab E. O. and Zitterbart M. (2005), *Efficient implementation of elliptic curve cryptography for wireless sensor networks.* Technical Report TM-2005-1, Institute of Telematics, University of Karlsruhe, Karlsruhe, German.

[5] Boussag L. (2017), *Implementation and Simulation of Security Protocols for Wireless Sensor Networks (WSNs).*

[6] Haakegaard R., and Lang J. (2015), *The Elliptic Curve Diffie-Hellman (ECDH)*

[7] Hankerson, D. et al. (2000) "Guide to Elliptic Curve Cryptography" Springer.

[8] Legrandin (2018), *PyCryptodome Documentation*, Release 3.6.1.

[9] Miller, V. (1985), "*Use of elliptic curves in cryptography*", Advances in Cryptology - CRYPTO '85 Proceedings, Springer Lecture Notes in Computer Science (LNCS), vol. 218

[10] Rehana, J., (2009). *Security of Wireless Sensor Network.* TKK T-110.5190 Seminar on Internetworking.

[11] U.S. Department of Commerce (2013), National Institute of Standards and Technology (NIST), "*Digital Signature Standard (DSS)*", FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION (FIPS), Gaithersburg, MD 20899-8900.

[12] Vagle L. J. (2000), *A Gentle Introduction to Elliptic Curve Cryptography*, BBN Technologies.

[13] Washingtion, Lawrence C. (2003), *Elliptic Curves: Number Theory and Cryptography.* Boca Raton, FL: CRC Press.