University of Baghdad
College of Engineering
JE
JOURNAL OF ENGINEERING

Journal of Engineering
journal homepage: www.joe.uobaghdad.edu.iq
Number 6 Volume 25 June 2019

Electrical, Electronics and communications, and Computer Engineering

WSN-WCCS: A Wireless Sensor Network Wavelet Curve Ciphering System

Assist. Prof. Dr. Nadia A. Shiltagh
University of Baghdad
College of Engineering
Baghdad – Iraq
dr.nadiaat123@gmail.com

Assist. Prof. Dr. Mahmood Z. Abdullah
Mustansiriyah University
College of Engineering
Baghdad – Iraq
drmzaali@uomustansiriyah.edu.iq

Ahmed R. Zarzoor*
Institute for Post-graduation Studies
Iraqi Commission for Computer &
informatics, Baghdad, Iraq
Ahmed.Arjabi@gmail.com

ABSTRACT

With wireless sensor network (WSN) wide applications in popularity, securing its data becomes a requirement. This can be accomplished by encrypting sensor node data. In this paper a new an efficient symmetric cryptographic algorithm is presented. This algorithm is called wireless sensor network wavelet curve ciphering system (WSN-WCCS). The algorithm idea based on discrete wavelet transformation to generate keys for each node in WSN. It implements on hierarchical clustering WSN using LEACH protocol. Python programming language version 2.7 was used to create the simulator of WSN framework and implement a WSN-WCCS algorithm. The simulation result of the proposed WSN-WCCS with other symmetric algorithms has shown that its execution time fastest among AES, 3DES and DES 15%, 55% and 17%.

Key words: Cryptography, Discrete Wavelet Transform, Wireless sensor network

نظام تشفير موجة الويفلت لشبكة الاستشعار اللاسلكية

الخلاصة

مع اتساع رواج تطبيقات شبكة الاستشعار اللاسلكية (WSN) امن بياناتها اصبح ضرورة وهذا يتم انجازه بواسطة تشفير بيانات نود الاستشعار. في هذا البحث نقدم خوارزمية جديدة و كفوءة للتشفير المتماثل. الخوارزمية تدعى نظام تشفير موجة الويفلت لشبكة الاستشعار اللاسلكية (WSN-WCCS). فكرة الخوارزمية تعتمد على التحويل المتقطع للويفلت لتوليد مفاتيح لكل النودات في شبكة الاستشعار اللاسلكية . الخوارزمية تم تطبيقها علي هيكل الكلاستر لشبكة الاستشعار اللاسلكية تستخدم بروتوكول LEACH. تم استخدام لغة البرمجة بايثون اصدار 2.7 لانشاء محاكاة هيكل شبكة الاستشعار اللاسلكية وتطبيق خوارزمية (WSN-WCCS) نتائج المحاكاة للخوارزمية المقترحة اظهرت انها اسرع من خوارزميات التشفير المتماثل الاخرى AES, 3DES, DES بحوالي 17%، 55%، 15%

الكلمات الرئيسية: التشفير، التحويل المتقطع للويفلت، شبكة الاستشعار اللاسلكية.

*Corresponding author
Peer review under the responsibility of University of Baghdad.
https://doi.org/10.31026/j.eng.2019.06.06

Article received: 2/5/2018
Article accepted: 25/6/2018

1. INTRODUCTION

In the last twenty-years wireless sensor network (WSN) development become a great interest area for industry such as transportation, civilian, healthcare, military and commercial applications. So, as this grown go day by day and beside its sensors small size, low cost and low power a security mechanisms become a big challenge due to its resources constrained. Furthermore, the nature of communication within WSN make eavesdropping and data modification more easy.

Cryptography is the core techniques to protect data in WSN. Due to the limitation of the WSN batteries power and computation speed make traditional security methods unstable to secure WSN data. However, there are two techniques in cryptography symmetric and asymmetric. In symmetric the same key is shared between two nodes and used for message encryption and decryption process. On other side, asymmetric two keys are used one to encrypt message and other one is used to decrypt message.

In fact, some researchers used asymmetric cryptography, **Kumaran, et al., 2016**, but most studies in WSN data security, **M. Panda,2015, Satyabrata, et al., 2016, and Li, Juan, 2017**, preferred to use symmetric cryptography (e.g., AES) to reduce energy consumption. This study is used a new symmetric cryptography algorithm is called WSN wavelet curve ciphering system (WSN-WCCS). In this algorithm a wavelet signal will be used to generate keys in such way that each node in the WSN has its own key. Our aim is developing data encryption algorithm using wavelet signal to achieve the following:

- Fast implementation for encryption and decryption process than other symmetric cryptography techniques such as AES, DES and 3DES.
- Flexibility in generating a number of keys for each node in WSN.
- Protection WSN data from eavesdropping and ciphertext attack.
- Reducing memory usage because it can perform computation in place without need to a temporary array

In this study a new algorithm is proposed to generate encryption keys based on discrete wavelet transformation. In which, secret key extracted from a multilevel up to 3 levels signal decomposition type **Daubechies** (one-dimension) as shown in section 3 in full details. The algorithm applied on hierarchical clustering WSN using LEACH protocol. Also, a comparison with other algorithm DES,3DES and AES have been made. It found a WSN-WCCS algorithm achieves significantly faster than other algorithms. The paper is presented as follows: section 2 contains a survey of related works on using discreet wavelet transformation DWT as base to generate keys in WSN, section 3 shows a study method which demonstrates key generation based on DWT multilevel decomposing using one dimension **Daubechies** and steps of building a WSN-WCCS algorithm, section 4 are discussed simulation result of the proposed algorithm with other symmetric algorithms (AES, DES and 3DES) and finally section 5 includes study conclusion.

3. RELATED WORK

Based to our knowledge there is a few studies that used discreet wavelet transformation DWT as base to generate keys in WSN. In **Keven and Fekri**, **2004** study, they used wavelet transformation through finite fields GF(256) to produce private key cryptosystem size 128 Byte. They conducted a charting on the field to their opposite in the field through nonlinear device. In a WSN-WCCS the key value is used in the process of encoding and decoding message using base64 method.

Another study **Koduganti and Avadhani**, **2011,** used DWT as base for authentication message in the network. They generated keys using one of generated algorithm then transformed generated keys using Haar technique. The transmitted encryption key is received then applied to encrypted message. So, the decryption key is retrieved from the reverse DWT. A **Daubechies** (one-dimension) wavelet is used in a WSN-WCCS because it has different types (from 1 to 20). In order to maximize complexity for opponent to know which type that has been used to generate secret keys in the algorithm.

In **DebayanGoswami et al.**, **2013,** study they generate keys consist of DWT decomposition code and bookkeeping to encrypt file format (.txt, .doc,.bmp). In the decryption process they used the inverse of DWT. A WSN-WCCS algorithm is different from this method in that its using coefficient details value as secret keys and that will be used for encryption/decryption message in wireless network. Beside the proposes algorithm applied on hierarchical clustering WSN, using node id to assign secret key to each node WSN.

3. STUDY METHOD

The basic idea of a WSN-WCCS is generating keys based on DWT multilevel decomposing using one dimension **Daubechies** wavelet. The result values of DWT process will be used as encryption keys assigned to each node in a WSN. So, to achieve that the following steps are:

3.1. Key Generation

Wavelet transformation is defined as an assortment of tools for transforming from time into convert domain to discover transfer coefficient. Subsequently, the coefficients also can be transformed again into the time domain for conversations many times. The fundamental issue in DWT are using low-pass (approximate coefficient) and high pass (details coefficient) filter to decompose signal simultaneously batch (h) into levels **Shukla and Tiwari, 2013,** as shown in **Fig.1.**

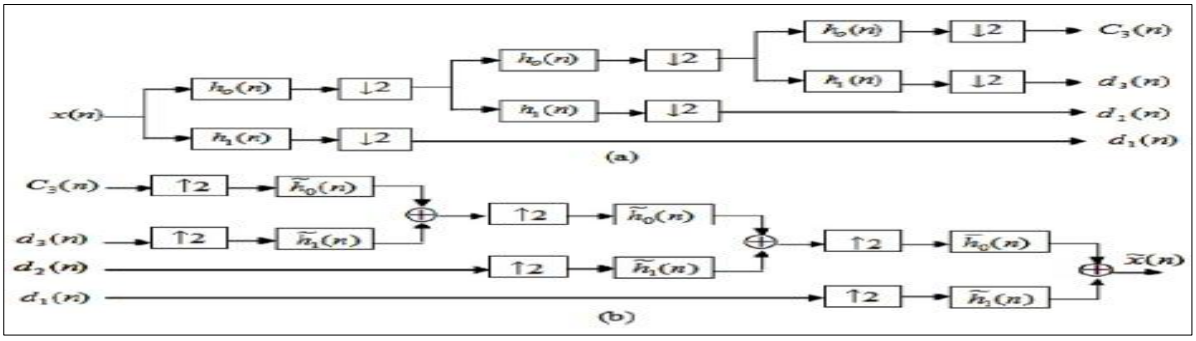

Figure 1 a. DWT b. inverse DWT for three levels coefficient **Haddadi, et al., 2014**

In **Fig 1**: n is the number of samples, x[n] is input samples to the DWT, d1(n), d2(n) and d3(n) are the coefficient details and $C_3(n)$ is coefficient approximate. Eq. (1) and (2) are used to calculate low-pass (coefficient approximate) and high-pass (coefficient details)

$$\hat{x}[n] = \sum_{k=-\infty}^{\infty} x[k]\,\widehat{h_0}\,(n)[2n-k] \qquad low-pass \qquad (1)$$

$$\hat{x}[n] = \sum_{k=-\infty}^{\infty} x[k]\,\widehat{h_1}\,(n)[2n-k] \qquad high-pass \qquad (2)$$

Eq. (3) is used to get the original input sample x[n]

$$x[n] = c3[n] \oplus d3[n] \oplus d2[n] \oplus d[n] \qquad (3)$$

In this algorithm **Daubechies** (one-dimension) wavelet is used because it has different types of **Daubechies** (from 1 to 20). In order to make it harder for opponent to know which type was used in the algorithm to generate keys. Posterior, coefficient details values are used as secret keys for encryption and decryption message in the network. Also, decomposition on input single made up to 3 levels. In order to increase the complexity for opponent in finding out which level the node request its secret key.

owever, a hierarchical clustering WSN is used in this study. The network organized into a number of clusters connected to the base station. Each cluster consists of two or more nodes connect to its cluster head. So, each node sensory data and send it to the cluster head, which in turn aggregate data and send it to the base station. The cluster head selected based on LEACH "**Less Energy Adaptive Clustering Hierarchy Protocol**", **Tandel, 2016**. In LEACH protocol all the nodes in the network can be selected as cluster head (CH). The node with highest energy will be elected as CH. In a WSN-WCCS algorithm the key values will be taken from d1, d2 and d3 lists and be used as secret key for node, CH and base station.

For example, if the network consisting of 50 nodes then 50 keys need to be generated as follow:

1. Generate 50 random numbers samples and store it in array XX [] to be used as input to the system
   XX=[54708062329, 95614112681, 38173952881, 26675615022, 43148396325, 93140605049, 982871032, 437773644, 2808779973, 8302567691, 30120833, 9633572973, 367073964, 5685987587, 7043562657, 924642211, 98155609, 407801576, 9182898601, 6653191725, 2182378616, 7340946955, 127564428L, 9102868022, 9655901195, 46263082828, 64938154071, 52819499975, 7220372196, 454533412, 8198891770, 9640024570, 2731045696, 5461756022, 7323979817, 5463058208, 32846240926, 56207566538, 30003067971, 34346330116, 63228815450, 9267229129, 5365922212, 8501765435, 101383779, 512533167, 5228889384, 8265300179, 5023473826, 4599860403]
2. Compute the DWT using Eq. (1) and (2) to compute low-pass (coefficient approximate) and high-pass (coefficient details) for **Daubechies** (one-dimension) wavelet type 2 (db2). The result will be store in three list d1, d2 and d3 as shown in the source code in the appendix A1.
3. The result **Daubechies** three levels details coefficient values for d1, d2 and d3 lists are shown in **Fig. 2.** Thus each value represents a secret key for each node. The secret key is assigned each node based on node id. For example, the key value is -1295000739.643582 is assign to the node id =5 in **Daubechies** level 3 details coefficient which represented in d3 list. So, in this step assigned each node in a WSN a key from d1 list using node id.

4. Encrypted message using key value with encode base64 method see **Table 1**. In base64 encode text to binary data in ASSCII see the source code of xor_crypt_string in the appendix A.1. For instance, encode the message "Hello" the steps are:

   1-EecodeBase64(Hello)→ SGVsbCA
   2-Encryption ("SGVsbCA",key="-1295000739.643582") → ciphertext= "ZVdYXVk="

   Here a concatenation operation in step 2 the results will be in the following order S-G2V9… in case the message is larger than the key, then recycle key value again i.e. starts again from -,1,2,9,5 and so on and apply result of the concatenation base64 encoder to get the ciphertext. "ZVdYXVk="

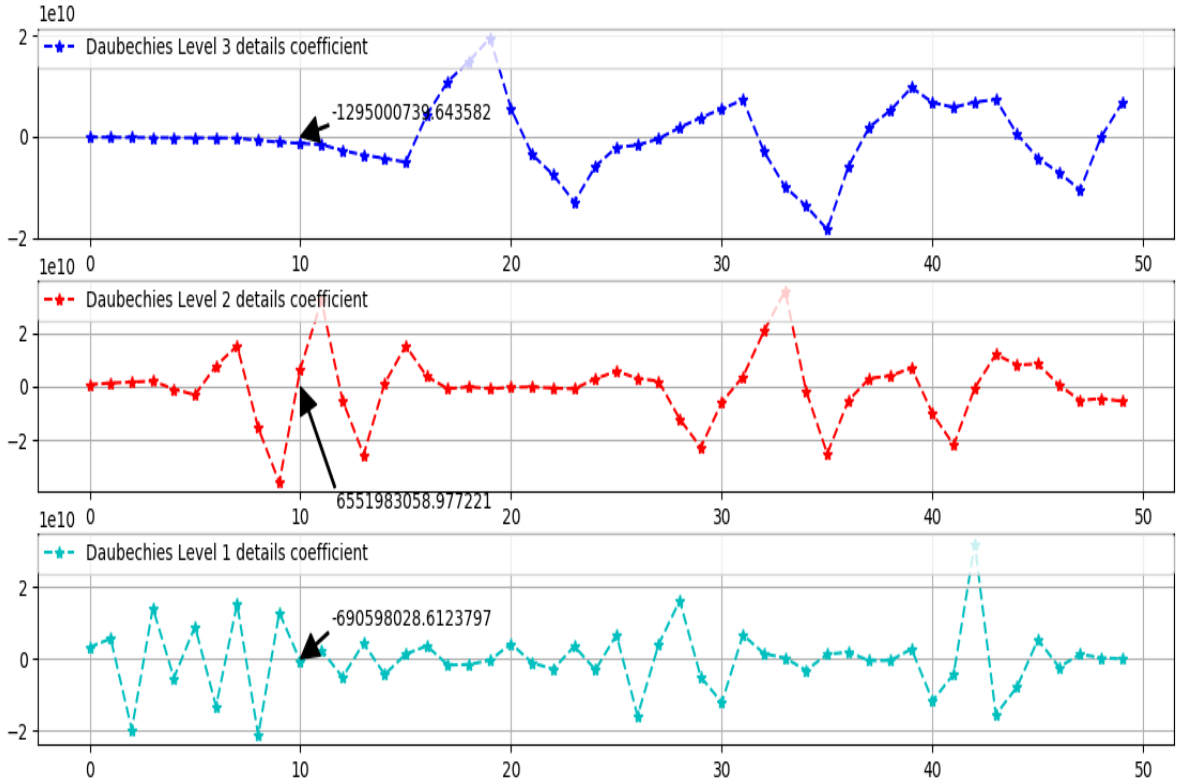

Figure 2. Daubechies level 1,2 and 3 details coefficient value graphic chart.

Table1.Base64 Encoding.

Text content	H								E								l								L								O							
ASCII	72								101								108								108								111							
Bit pattern	0	1	0	0	1	0	0	0	0	1	1	0	0	1	0	1	0	1	1	0	1	1	0	0	0	1	1	0	1	1	0	0	0	0	1	0	0	0	0	0
Index	18								6								21								44								27							
Base64-encoded	S								G								V								s								b							

Index	2								0																																
Base64-encoded	C								A																																

Note the **Base64-encoded** value is taken from mapping index value to Base64-encoded table. For example, 18 in the table equal to S.

5. The Decryption message process again the secret key will be in the decode base64. So, first step removes the key from the encoded message and apply it on the decode base64 as shown below:

   1-Encryption ("ZVdYXVk=", key= "-1295000739.643582") → M= "SGVsBCA"
   2-DecodeBase64(M)→ plaintext= "Hello"

3.2 LEACH Protocol

LEACH is one of the main routing protocol that used in forming cluster in WSN in order to prolong the network lifetime. In each cluster all the nodes connected to CH via single hop. So the nodes collection data send to their CH which in turn forward it the base station. The LEACH protocol constructed in two phases setup and steady.

In setup phase the network divided into groups of nodes or clusters. In the network all the nodes are equal in the probability to be elected as CH. The node with highest residual energy will be elected as CH. Thus, each CH will send advertised message to inform other nodes in the groups that it become as CH using Eq. (4) **Gupta, and Marriwala, 2017**. Other non CH nodes will send joining message to the elected CH in order to become Cluster Member (CM).

$$t(h) = \begin{cases} \dfrac{P_b}{1 - P_b \times \left[ro_n \bmod \left(\dfrac{1}{P_b}\right)\right]} & \forall h \in M \\ 0 & otherwise \end{cases} \qquad (4)$$

Where
h= a random number between (0 and 1),
P_b =Probability of CH
M= is the all the nodes that are not selected as CH
ro_n = round number.
t(h)= thereshold for node that became as CH in the current round
1/ P_b = first round that in which CH nodes had been elected.

The node elected as CH in the current round if its values is less than t(h) otherwise it remains normal node. Although, when a node elected as CH once it cannot be elected again as CH. So, the process continues till all the nodes in the network become CH once. In steady phase all CMs send their sensory data to their CH through single hop. The CH summed the collected data and send it the base station via other CHs or directly using static route. After specified period of time the network start again the setup phase.

3.3 WSN-WCCS Algorithm

1. Generate (n) random number of sample size 18 Byte and store it in xx[n] array
2. input x[n] samples into DWT function to
    a. Calculate low-pass and high-pass up to 3 levels from **Daubechies** wavelet (one-dimension) type 2
    b. Store the result of computation in d1, d2, d3 and C3 lists in a routing table.
3. Assign each cluster header in WSN a key from d2 list using cluster id.
4. Assign each base station in WSN a key from d3 list using BS id.
5. Assign each node in WSN a key from d1 list using node id.
6. Each a live node in the network sends its ID to the base station BS, to request a key
7. BS check the node id validity and send the key that assign to this node
8. Node use the key to encrypt message. Subsequently, send the encrypted message with its id to the destination node.
9. In BS again check node id that attach with the message in routing table. In order to find assigned key and used it to decrypt the message.

4. RESULTS

The algorithm applied on hierarchical clustering for a WSN that used LEACH protocol see appendix A2 LEACH protocol source code. The energy consumption is ignored in this study and consternated on the calculation speed of encryption algorithm and storage usage. Python language version 2.7 was used to create the simulator of WSN framework and implement a WSN-WCCS algorithm. The simulator implemented on laptop Lenovo, processor intel(R) Core(TM)-4200 CPU @ 1.60 GHz 2.30GHz and RAM 4.00GB. The network simulator parameters are assumed the same for the four algorithms (DES, AES, 3DES and WSN-WCCS) as shown in **Table2**. The **Fig. 3** shown WSN before using clustering and **Fig. 4** show WSN after clustering. In **Fig. 5** screenshot of running a WSN-WCCS in which final round 2545, total remaining energy is 0.0000 and node id: 36. Encrypted message “ektdWFoWQVxAVwE=”.

The WSN-WCCS algorithm is compare with other symmetric algorithms DES, 3DES and AES based on speed of execution time and storage usage. The comparison was done by running the fourth algorithms separately. The result shown in **Table 3.** A WSN-WCCS execution time is 15% faster than AES, 55% faster than 3DES and 17% faster than DES. In the storage usage it takes less storage than 3DES, slightly more storage than AES and DES.

Table 2. Simulation Parameters.

S.No	Parameter	Unit
1	number of nodes	50
2	Area	100 x 100 M
3	Message size	16 Byte
4	Header length	150 bits
5	INITIAL_ENERGY	2 Joules
6	Energy dissipated at the transceiver electronic (E _ELEC)	50e-9 Joules
7	Energy dissipated at the data aggregation (E_DA)	5e-9 Joules
8	Energy dissipated at the power amplifier (supposing a multi path Emp)	0.0013e-12 Joules
9	Energy dissipated at the power amplifier (supposing a line-of-sight free-space channel E_FS)	10e-12 Joules
10	Base station position	BS_POS_X = 10.0 BS_POS_Y = 10.0

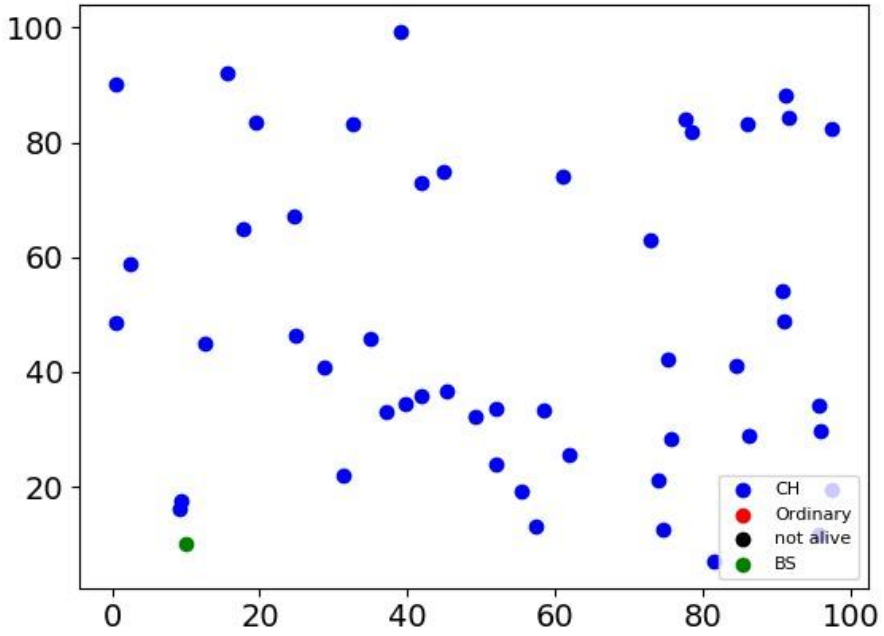

Figure 3. Initial phase WSN before clustering.

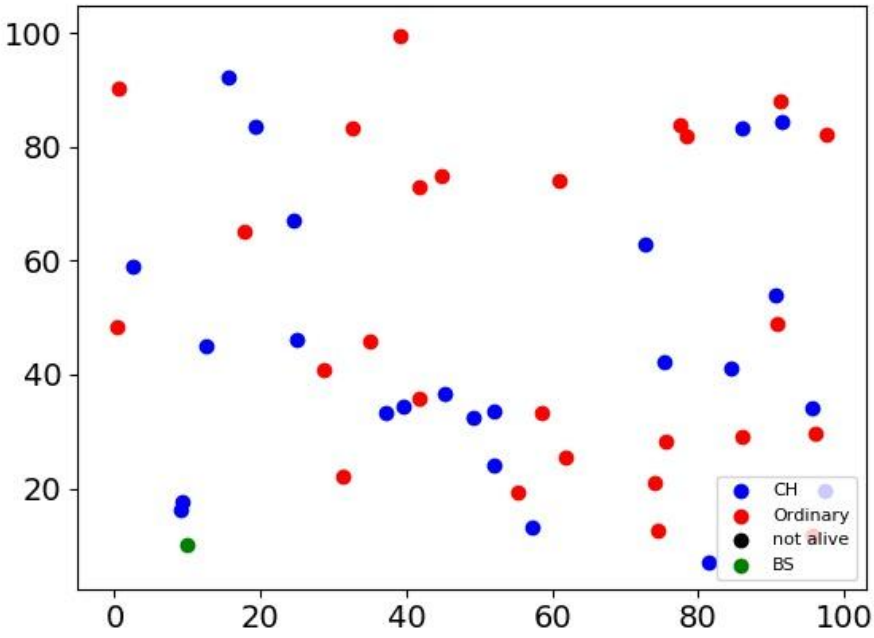

Figure 4. Initial phase WSN after clustering.

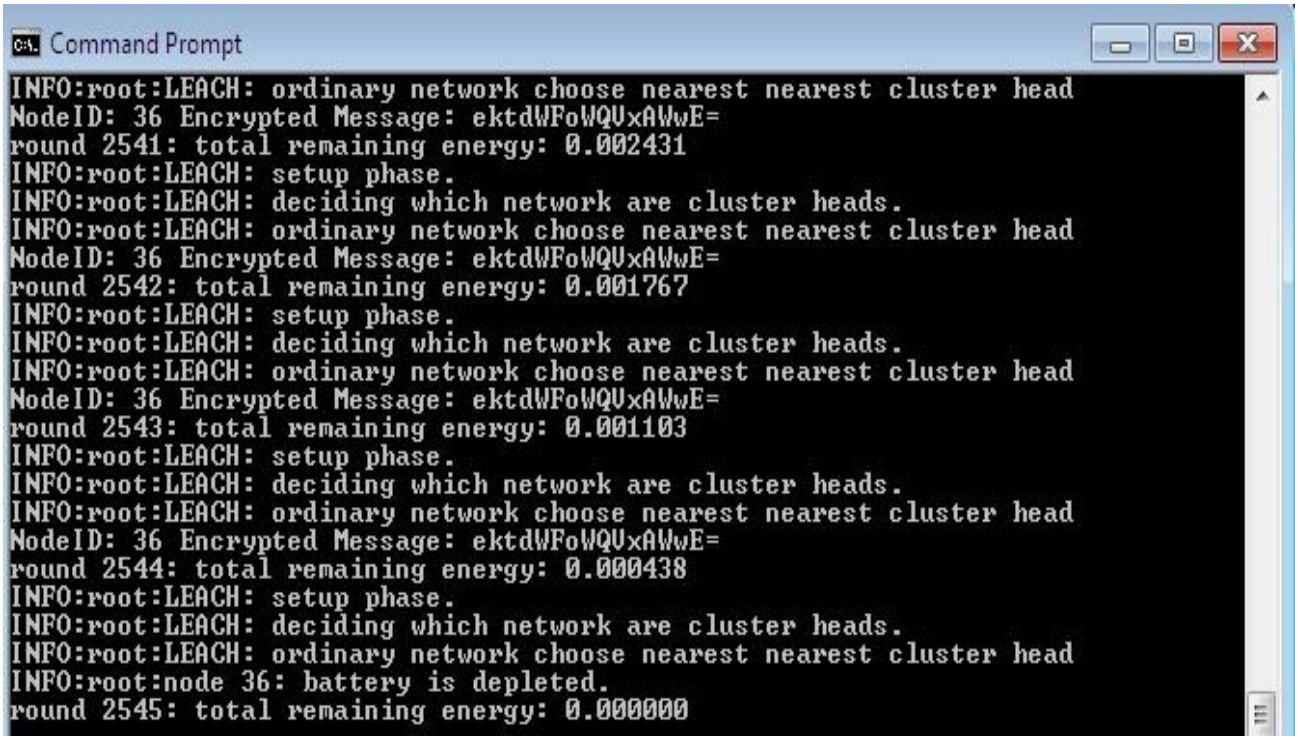

Figure 5. Screenshot of running WSN-WCCS.

Table 3. Performance Analysis.

Algorithm	Message Size	Key size	Execution time in seconds	Total storage usage in byte
AES	16 Byte	16 Byte	0.171	32 bytes
3DES	16 Byte	16 Byte	0.562	64 bytes.
DES	16 Byte	8 Byte	0.187	32 bytes
WSN-WCCS	16 Byte	18 Byte	0.016	34 bytes

In **Table 3**, the total storage usage for AES is 32bytes. But for 10 rounds the key expansion will be equal to 176 bytes 16 x (10+1). In a WSN-WCCS the total storage 34. While a DES the total storage is 32 bytes because DES used key sized 8 bytes for each block size is 8 bytes. Whereas, the message size is 16 bytes thus the total size is 32 (16x2). The 3DES algorithm increased the key size via the reused of DES therefore the total size storage is (32x2=64 bytes).

4.1 WSN-WCCS Cryptanalysis

The symmetric technique is based on using the same key between two nodes to encrypt and decrypt message. So, the secret key is considering vulnerable for attacker. Thus making its long and hard to predictable or cracked by the attackers is considered important issue. But due to a WSN resource constraints makes this issue a big challenge for researchers. This section is discussed the security issue of a WSN-WCCS algorithm in compare with other algorithms DES, 3DES and AES

The main disadvantage in DES algorithm is that a secret key size is small and can be creaked using brute force attack. In a WSN-WCCS the key sized is larger and randomly selected and assigned to the node in the network. The 3DES increased the key sized via the reused of DES algorithm using either two or three different keys (112 or 168 bit), but the main drawback that it taken more computation time, **Patil, et al., 2016**, which in turn increase the energy consumption, thus make it not appropriated for securing WSN framework. The AES algorithm is used in widely and efficiently in securing a WSN data, **Hung, et al., 2018**. The key length is 128 bit used in more than one rounds on 4x4 state matrix for full encryption/decryption using permutation and substitution operations. Whereas, a WSN-WCCS algorithm is used only substitution operation. Beside in this study the secret key length 18 Byte (144 bit) is randomly generated from **Daubechies** wavelet signals (from 1 to 20). Which maximums the complexity for adversary to crack the secret key that used with base64 encode/decode method. Although, a WSN-WCCS execution time in this study for encryption process is 15% faster than AES.

However, a WSN-WCCS based on the base64 decode and decode method. So, if the attacker discovered the way the secret is generated then attacker can easily reveal the original message using base64 decode table. In the proposed algorithm a random key is generated based on DWT multilevel decomposing using one dimension **Daubechies** wavelet. While the DES, 3DES and AES start with seed key is generated randomly using random generated then used shift rotate operation in DES and 3DES to generate secret key using permutation computation matrix in 16 rounds. Whereas AES four operations (Add Round key, Mix. Column, Shift Row and Subbyte) are used to generate secret key on 4x4 state matrix in 10 rounds for key sized 128 bits. Thus, there is a probability that the same seed key is used more than one times because it generates randomly. Thus, when adversary discovered the secret key values that used more than one times

then he or she can figure out how keys is generated randomly. While in a WSN-WCCS algorithm the level of security is increased because even the adversary can figure how the key is generated then he or she has to figure out which wavelet single is used (Haar, Daubechies. Coifle …etc.). Furthermore, even the adversary has ability to find out a wavelet signal in the algorithm is a **Daubechies**. Thus, the adversary must be able also to specify the signal type which is between (1 and 20). Even though, the attacker can be able to specify the signal type then he or she needs to compute DWT multilevel decomposing for many level. Which is very harder and take much time for adversary to check each point on each curve on each level.

5. CONCLUSION

Symmetric cryptograph was widely used to secure data in WSN network. In this study a new algorithm proposes, based on our knowledge e is the first algorithms that used DWT only to generate keys in WSN. A WSN-WCCS algorithm was designed based on DWT multilevel decomposing using one dimension **Daubechies** wavelet signal. It implements on hierarchical clustering WSN that use LEACH protocol.

In general, the study result has shown the proposed algorithm is faster than other symmetric techniques and efficient in storage usage. Even its used a larger key size than the others algorithms. Also, its take less storage usage about 30 bytes than 3DES algorithm and more storage usage about 2 bytes than AES and DES algorithms.

REFERENCES

- Kumaran, Senthil & Nallakaruppan, M.K. & Mohan, Senthilkumar., 2016, *Review of asymmetric key cryptography in wireless sensor networks*., International Journal of Engineering and Technology (IJET), Vol. 8 ,No. 2 , pp. 859-862.

- Panda, Madhumita, 2015, *Data security in wireless sensor networks via AES algorithm*. 2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO), Coimbatore, India, 9–10 January 2015, pp. 1–5.

- Satyabrata Roy, Jyotirmoy Karjee, U.S. Rawat, Dayama Pratik N., Nilanjan Dey, 2016, *Symmetric Key Encryption Technique: A Cellular Automata based Approach in Wireless Sensor Networks*, Procedia Computer Science, Vol. 78, pp. 408-414.

- Li, Juan. ,2017, *A Symmetric Cryptography Algorithm in Wireless Sensor Network Security*, International Journal of Online Engineering (iJOE), Vol. 13, No. 11, pp. 102-110

- Kevin Sean Chan and F. Fekri, 2004, *A block cipher cryptosystem using wavelet transforms over finite fields,* in *IEEE Transactions on Signal Processing*, Vol. 52, no. 10, pp. 2975-2991.

- Koduganti Venkata Rao & P. S. Avadhani ,2013, *Authentication based on wavelet transformations,* Journal of Discrete Mathematical Sciences and Cryptography, Vol. 9, No. 3, pp. 513–521

- DebayanGoswami, NaushadRahman, JayantaBiswas, AnshuKoul, Rigya Lama Tamang,Dr. A. K. Bhattacharjee, 2011, *A Discrete Wavelet Transform based Cryptographic algorithm*, IJCSNS International Journal of Computer Science and Network Security, Vol.11 No.4, pp. 178-182

- K. K. Shukla and A. K. Tiwari,2013, *Efficient Algorithms for Discrete Wavelet Transform*, SpringerBriefs in Computer Science, London, UK

- R. Haddadi, E. Abdelmounim, M. El Hanine, A. Belaguid, 2014, *Discrete wavelet transform based algorithm for recognition of QRS complexes*, World of Computer Science and Information Technology Journal (WCSIT), Vol. 4, No. 9, pp. 127-132.

- Reshma I. Tandel, 2016, *Leach Protocol in Wireless Sensor Network: A Survey*, (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 7, No. 4, pp. 1894-1896

- Gupta, S., & Marriwala, N. ,2017, *Improved distance energy based LEACH protocol for cluster head election in wireless sensor networks*, 2017 4th International Conference on Signal Processing, Computing and Control (ISPCC), pp. 91-96.

- Patil P, Narayankar P, Narayan DG, Meena SM, 2016, *A comprehensive evaluation of cryptographic algorithms: DES, 3DES, AES, RSA and Blowfish*. Elsevier B. V. Procedia Computer Science, Vol. 78, pp. 617-24.

- Hung, Chung-Wen; Hsu, Wen-Ting, 2018, *Power Consumption and Calculation Requirement Analysis of AES for WSN IoT,* Sensors, Vol. 18, No. 6, pp. 1-11.

Appendix A

A.1 WSN-WCCS source code

```
import matplotlib.pyplot as plt
import numpy as np
import random
import uuid
import hashlib
import pywt
import time
import numpy

# input samples that used to generate wavelet signal array XX[]
#=====================================================================#
XX=[54708062329, 95614112681, 38173952881, 26675615022, 43148396325, 93140605049,
   982871032, 437773644, 2808779973, 8302567691, 30120833, 9633572973, 367073964,
   5685987587, 7043562657, 924642211, 98155609, 407801576, 9182898601, 6653191725,
   2182378616, 7340946955, 127564428L, 9102868022, 9655901195, 46263082828, 64938154071,
```

52819499975, 7220372196, 454533412, 8198891770, 9640024570, 2731045696, 5461756022, 7323979817, 5463058208, 32846240926, 56207566538, 30003067971, 34346330116, 63228815450, 9267229129, 5365922212, 8501765435, 101383779, 512533167, 5228889384, 8265300179, 5023473826, 4599860403]

```
#=========================================================================#
# Encryption and Decryption Function  xor_crypt_string
#=========================================================================#
def xor_crypt_string(data, key, encode=False, decode=False):
   from itertools import izip, cycle
   import base64
   if decode:
      data = base64.decodestring(data)

   xored = ''.join(chr(ord(x) ^ ord(y)) for (x,y) in izip(data, cycle(key)))
   if encode:
      return base64.encodestring(xored).strip()
   return xored

#=========================================================================#
#Compute wavelet levels for high and low pass filters
#=========================================================================#
def wrcoef(X, coef_type, coeffs, wavename, level):
   N = np.array(X).size
   a, ds = coeffs[0], list(reversed(coeffs[1:]))

   if coef_type =='a':
      return pywt.upcoef('a', a, wavename, level=level)[:N]
   elif coef_type == 'd':
      return pywt.upcoef('d', ds[level-1], wavename, level=level)[:N]
   else:
      raise ValueError("Invalid coefficient type: {}".format(coef_type))

#=========================================================================#
#Compute CPU execution time for A WSN-WCCS
#=========================================================================#
def benchmark1():
   st = time.time()

   coeffs = pywt.wavedec(XX, 'db2', level=3)
   A3 = wrcoef(XX, 'a', coeffs, 'db2', level)
   D3 = wrcoef(XX, 'd', coeffs, 'db2', level)
   D2 = wrcoef(XX, 'd', coeffs, 'db2', 2)
   D1 = wrcoef(XX, 'd', coeffs, 'db2', 1)
   for i in xrange(50):
      print  (i,xor_crypt_string("The answer is no",str(D1[i]) ,encode=True))

   en = time.time()
   print ("WSN-WCCS Benchmark duration: %r seconds" % (en-st))
   print ("start time="+str(st), "End Time="+str(en))
```

```
#=====================================================================#
#Compute memory usage for A WSN-WCCS
#=====================================================================#
benchmark1()
from guppy import hpy; h=hpy()
h.heap()
print h.heap()
#=====================================================================#
```

A.2 Implement WSN-WCCS with LEACH protocol source code

The generated keys are stored in table in database name MData.db. The database created using SQLite 3.10.1 database browser. The table store node ID, cluster ID and base station ID and secret keys. Each node send request for secret key by sending its ID to the base station using **def sense** () functions. In which the node use **waveKey(node_id)** to get its secret key and used its in **xor_crypt_string()** . in order to encrypt the message "Hello world"

```
#=====================================================================#
def sense(self):
    node_id=(self.id)
    Nodekey=waveKey(node_id)
    Encrypt_ Message=xor_crypt_string("Hello world",str(Nodekey),encode=True)
    self.tx_queue_size = cf.MSG_LENGTH
    self.amount_sensed += cf.MSG_LENGTH
    print 'NodeID:',s,'Encrypted Message:', Encrypt_ Message
#=====================================================================#
def waveKey (Nid):
  con = lite.connect('MData.db')
  with con:
     cur = con.cursor()
     cur.execute("Select BSKeyA,BSKeyD,CusterID,CluserKey,NodeID,NodeKey from Table1 where
NodeID='" + str(Nid) +"'")
     record=cur.fetchall()
     if not cur.rowcount:
        print "No results found"
        secret_key=None

     else:
        for row in record:
           secret_ke = row[3]
           return secret_key
  cur.close()
  con.close()
#=====================================================================#
#Encryption / Decryption Function
#=====================================================================#
def xor_crypt_string(data, key, encode=False, decode=False):
  from itertools import izip, cycle
  import base64
  if decode:
```

```
        data = base64.decodestring(data)
    xored = ''.join(chr(ord(x) ^ ord(y)) for (x,y) in izip(data, cycle(key)))
    if encode:
        return base64.encodestring(xored).strip()
    return xored
#=============================================================================#
# the following functions are used to calculate the transmit energy, received energy
#=============================================================================#

  def transmit(self, msg_length=None, destination=None):
    logging.debug("node %d transmitting." % (self.id))
    if not msg_length:
      msg_length = self.tx_queue_size
    msg_length += const.HEADER_LENGTH

    if not destination:
      destination = self.network_handler[self.next_hop]
      distance = self.distance_to_endpoint
    else:
      distance = calculate_distance(self, destination)

    # transmitter energy model
    energy = const.E_ELEC
    if distance > const.THRESHOLD_DIST:
      energy += const.E_MP * (distance**4)
    else:
      energy += const.E_FS * (distance**2)
    energy *= msg_length

    # automatically call other endpoint receive
    destination.receive(msg_length)
    # after the message is sent, queue is emptied
    self.tx_queue_size = 0
    self.amount_transmitted += msg_length

    self.energy_source.consume(energy)
#=============================================================================#

  def receive(self, msg_length):
    logging.debug("node %d receiving." % (self.id))
    self._aggregate(msg_length - const.HEADER_LENGTH)

    self.amount_received += msg_length

    # energy model for receiver
    energy = cf.E_ELEC * msg_length
    self.energy_source.consume(energy)
#=============================================================================#
# LEACH Protocol
```

```
#=====================================================================#

class LEACH(RoutingProtocol):
  def setup_phase(self, network, round_nb=None):
     logging.info('LEACH: setup phase.')
    # decide which network are cluster heads
    prob_ch = float(const.NB_CLUSTERS)/float(const.NB_NODES)
    heads = []
    alive_nodes = network.get_alive_nodes()
    logging.info('LEACH: deciding which network are cluster heads.')
    idx = 0
    i=0

    while len(heads) != const.NB_CLUSTERS:
      node = alive_nodes[idx]
      u_random = np.random.uniform(0, 1)
      # node will be a cluster head
      if u_random < prob_ch:
        node.next_hop = const.BSID
        heads.append(node)

      idx = idx+1 if idx < len(alive_nodes)-1 else 0
  # ordinary network choose nearest cluster heads
    logging.info('LEACH: ordinary network choose nearest nearest cluster head')
    for node in alive_nodes:
      if node in heads: # node is cluster head
        continue
      nearest_head = heads[0]
      # find the nearest cluster head
      for head in heads[1:]:
        if calculate_distance(node, nearest_head) > calculate_distance(node, head):
          nearest_head = head

     node.next_hop = nearest_head.id
    network.broadcast_next_hop()
```