

# A Light Weight Cryptographic Hash Algorithm for Wireless Sensor Network

Manoj Kumar

Maharaja Surajmal Institute, New Delhi

Email: [manoj.rke77@gmail.com](mailto:manoj.rke77@gmail.com)

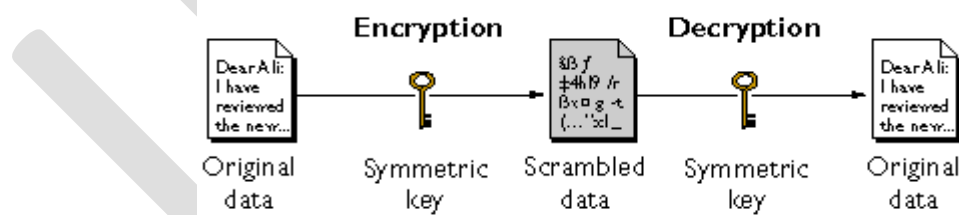
**Abstract-** Authentication of a message is a great research challenge in today's advanced wire and wireless communication. Cryptographic hash functions are used to protect the authenticity of information. Some of the most popular and commonly used cryptographic hash algorithms are MD5 and SHA1. These hash algorithms are used in a wide variety of security applications e.g. securing node/message in traditional networks.

However, the commonly used hash algorithms require huge computational overhead which is not affordable by applications in energy-starved network e.g. wireless sensor network (WSN). In these applications the major constraints are communication, computation and storage overheads; out of which communication and computation overheads consume high energy. Keeping this fact in mind, in this work, a light-weight, one-way, cryptographic hash algorithm is suggested with a target to produce a hash-digest with fixed and relatively small length for such an energy-starved wireless network. The primary focus is making the algorithm light-weight so that upon using it in application of network like WSN, the nodes can successfully run the algorithm with low energy. It is suggested that such algorithm must fulfill all the basic properties such as preimage resistance, collision resistance of a one-way hash function. The proposed algorithm is developed using NS2 simulation tool and results were compared with MD5 and SHA1.

**Keywords:** symmetric key, asymmetric key, message authentication, md5, sha1, cryptographic hash algorithm, ns2 simulation, wireless sensor network

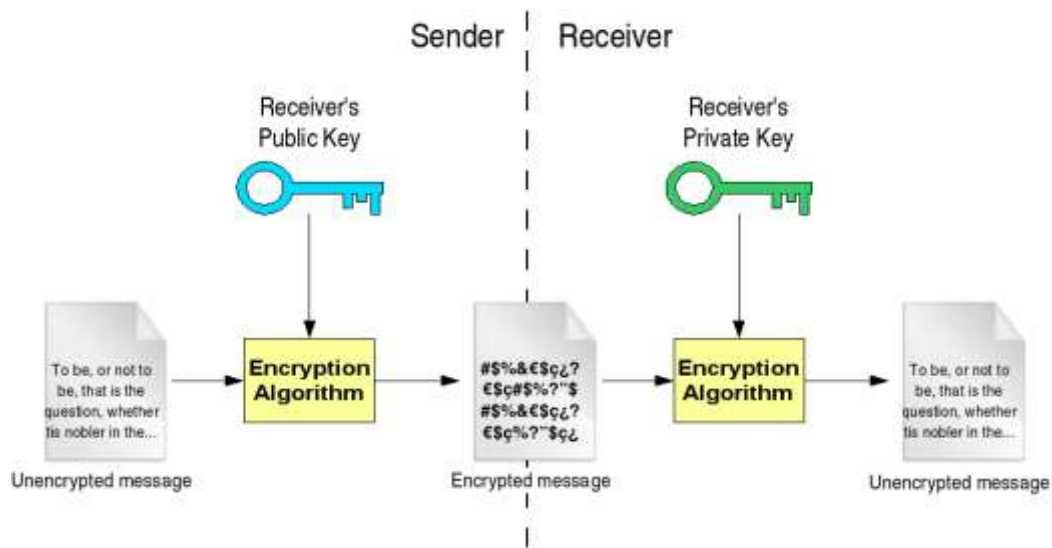
## 1. INTRODUCTION

Symmetric key & Asymmetric key cryptography are two different techniques available to use keys or secrets for encryption. Both types of algorithms are used for data encryption and decryption in cryptography and network security. Symmetric key cryptography is a conventional encryption technique which is used to encrypt and decrypt the data. The process used to generate cipher text in Symmetric key algorithms is less complicated due to which these algorithms execute much faster than Asymmetric key algorithms. The number of bits (i.e. length) used to define the key determines the strength of the security. A key can be 160-512 bits long. NIST has provided recommendations regarding the key length. The major challenge in implementing Symmetric key cryptography is that 2 parties must share the secret key in a secure way.



**Fig 1.1** Symmetric Key Encryption

Asymmetric key cryptography uses a pair of mathematically related keys. The key pair contains a public key and a private key. Public key is known to everyone and the private key is always in possession of its owner only. The working mechanism of Asymmetric key cryptography takes away the security risk involved in key sharing between 2 parties. The private key is never revealed in this process. The message is encrypted by applying the public key before sending. The encrypted message can only be decrypted by using the corresponding private key. In another use of Asymmetric key cryptography, a message encrypted with the private key is decrypted by the corresponding public key. It is virtually impractical to compute the private key even if the corresponding public key is known.



**Fig 1.2** Asymmetric Key Encryption

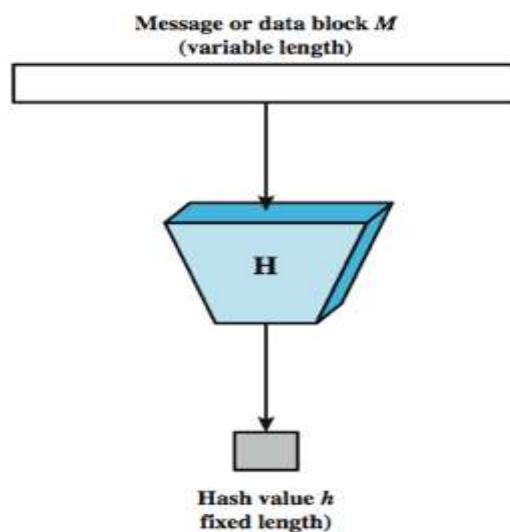
A different set of problem is still there-

When a message is received by Bob sent from Alice, 2 things are to be verified

- Is the message authentic (Integrity of the message has not been compromised)
- Has the message originated from Alice herself?

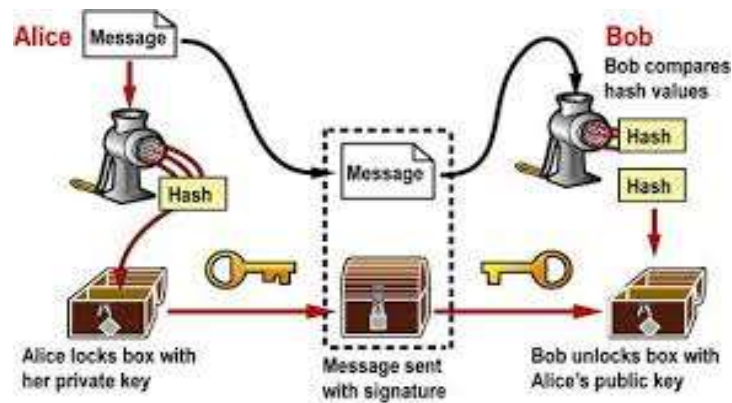
The answer of these questions is the Cryptographic Hash Functions.

A hash (also known as message digest or signature) is a one way function that by some means computes a fingerprint of the message. It is more widely known as the hash value of the message.



**Fig 1.3** Signature of Long Message with a Hash Function

So if Alice wants to ensure the integrity of the contents of her document, she can attach the fingerprint of the message at the bottom of the document. Bob knows the function scheme used by Alice to generate the hash at his end. If it tallies with the hash value from Alice, Bob confirms the message is authentic.



**Fig 1.4** Checking Integrity at Bob's End

Some of the popular hash functions are MD4, MD5, SHA-1, and SHA-512.

Certainly there may be many messages that can produce the same hash digest, because a message can be arbitrarily long and the hash digest will be of some predetermined length, for instance 128 bits in MD5. For instance, for 1000 bit messages and a 128 bit hash digest, there are on the average  $2^{872}$  messages that map to any of  $2^{128}$  message digest. So undoubtedly, by trying lots of messages, one would eventually find two or more messages that can generate the same hash digest. The problem is that "lots" is so many that it is essentially impossible. Assuming a good 128 bit hash digest function, it would take trying around  $2^{128}$  possible input messages before one would find a hash collision, or approximately  $2^{64}$  messages before finding two that had the same hash digest.

A reasonable way of constructing a hash function is to combine lots of vicious operations into a potential digest function, and then play with it. If any particular patterns are detected repeatedly in the output the function perhaps requires a modification or it is summarily rejected.

Ideally, a good hash function should be very easy to compute. However, there is no dimension of minimal function which is fully secure. It is safer for a hash function to be overload and do a lot of shuffling beyond what is needed. The function must use all the input data. The hash function must uniformly distribute the hash values across the entire set of available values. The hash digest tend to be calculated in several rounds. The designers find the least number of rounds necessary to generate a hash output which qualifies various randomness tests, and then do a few more just to make it more robust and safe.

### 1.1 Communication Architecture of WSN

A Wireless Sensor Network is generally composed of few hundred to several thousands of tiny sensor nodes. Such networks are used to monitor environmental conditions. These sensor nodes are densely deployed to create a communication network in a sensor field. A sensor node consists of 4 basic parts: a sensing unit, a processing unit, a power unit and a transceiver unit [1]. Sometimes it may have a location tracking system which helps to keep track of the respective location. It may also have power generator which provide longer power backup. In addition to these, a node may also have mobilizer (Fig. 1.5). Sensors and analog-to-digital converters (ADCs) are the two subunits of sensing units. A processing unit is generally consists of microcontroller or microprocessor and a small storage unit. Its main job is to process the gathered data and execute the communication protocols. The power unit of a sensor is generally limited (e.g., a single battery). It is sometimes supported by power scavenging devices (e.g., solar cells). For large networks, battery replacement is very difficult or even impossible. A transceiver unit provides a connection of the node to the network. Most of the sensing tasks and sensor network routing techniques require knowledge of location, which is provided by a location tracking system. Depending upon the application, a mobilizer is sometimes used to provide movement to the sensor node.

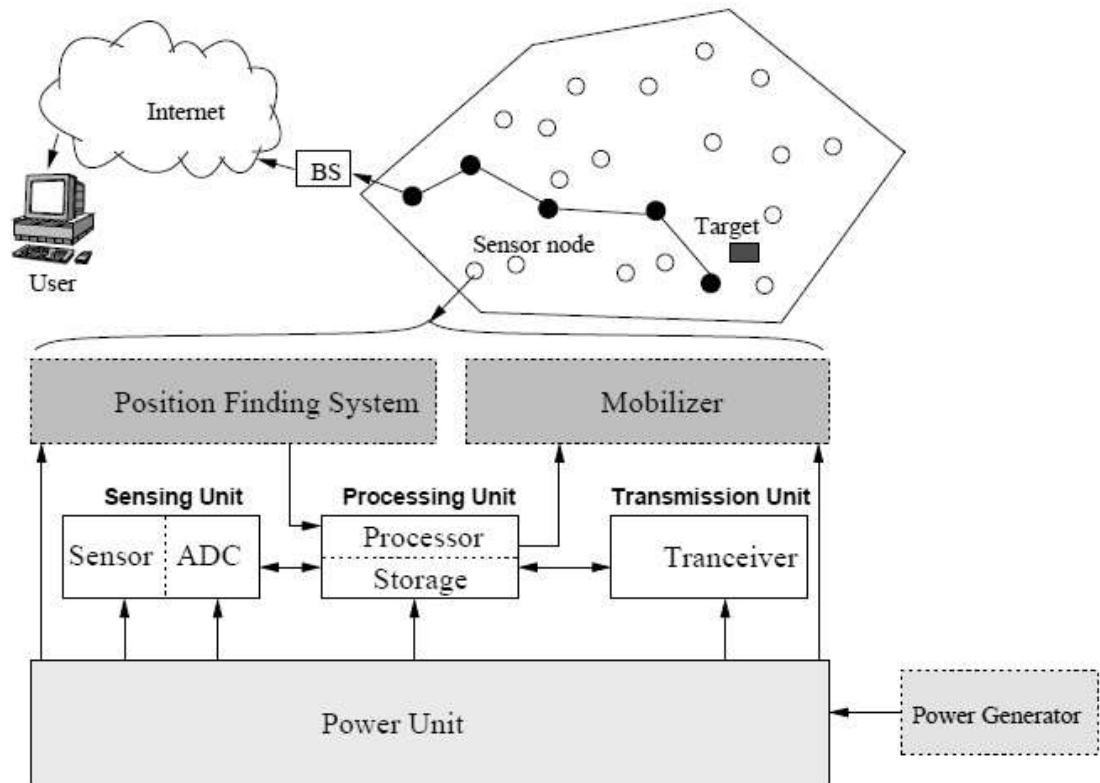


Fig 1.5 Wireless Sensor

The protocol stack used in sensor nodes contains the following layers [1]

- **Physical layer:** responsible for transmission and reception of data, generation and selection of carrier frequency, signal deflection, modulation, and data encryption & decryption.
- **Data link layer:** responsible for error detection and correction, the multiplexing of data streams, detection of data frames, medium access, reliable point-to-point and point-to-multipoint connections.
- **Network layer:** responsible for assignment of addresses and specify the process of packet forwarding to other nodes.
- **Transport layer:** responsible for the reliable transport of packets from one node to other node.
- **Application layer:** responsible for the interactions with the end users. It specifies how the data provided to individual sensor nodes upon request.

## 1.2 Constraints in WSN

Nodes of a Wireless Sensor Network are resource constrained as they have restricted computing capability, communication bandwidth and storage capacity. The small size of sensor node and limited computing power are 2 greatest constraints. So the security services in a sensor node must be implemented keeping in view of following hardware constraints of the sensor node:

### • Energy:

- Energy is required for the sensor transducer which converts one form of energy into another
- Energy is required to establish communication among sensor nodes
- Energy is also required for microcontroller & microprocessor computation

### • Computation:

Conventional complex cryptographic algorithms would require a lot of computing power for which the processors of sensor nodes is not feasible. Lightweight cryptographic algorithms are required to reduce the computing burden on sensor node.

• **Memory:**

Flash memory and RAM are usually included for the storage purpose in a sensor node. Flash memory is used to keep downloaded application code and RAM is used to keep sensor data, storing application programs, and intermediate computations. Generally, after loading OS and application code there is not sufficient space left to run complex algorithms. Due to the memory constraint it is not viable to use the majority of traditional cryptographic algorithms.

• **Transmission range:**

Limited operating power imposes restrictions on the communication range of sensor nodes. The actual transmission range of a signal depends on various environmental factors such as weather and terrain.

## 2. LITERATURE SURVEY

The most widely used hash functions are one-way functions for which finding an input which hashes to a pre-specified hash-value is very difficult. Two commonly used hash functions are MD5 and SHA-1. Both MD5 and SHA-1 are derived from MD4 in which weaknesses have been identified [3]. MD5 uses a hash algorithm with 128-bit long output hash was designed in 1991 and in 2005 it was shown [4] how rapidly random collisions for MD5 can be computed. MD5 is also not suitable for applications like SSL certificates or digital signatures. In [5] authors have revealed that how a couple of X.509 certificates can be produced that result in the same MD5 hash digest. This revelation led the cryptographers recommending the use of other algorithms like SHA-1 and other hash algorithms of SHA family. SHA-1 has been found to be weak [6] as well and most U.S. government applications now use the SHA-2 and SHA-3 family of hash functions [7, 8]. But most of the discussed hash functions are used in large traditional networks. Contrary to traditional networks, in a short-lived and energy-constrained network like Wireless Sensor Networks, a number of sensor nodes are deployed [9] in outdoor environment without human intervention. Reliability and data authenticity becomes the main worry to deal with such kind of networks. WSN suffers from number of constraints like low computing power, low battery life, and small memory. Due to these constraints, it is not able to deal with conventional cryptographic algorithms. So, it becomes compulsory to design a lightweight security hash algorithm for WSNs. Many similar works are reported towards hash-based security solutions and some of them [10, 11, 12] are mentioned here. Here [10, 11] prescribes solutions for WSNs whereas [12] is not usually meant for WSNs.

In [10], authors have presented a hash-based signature method which can be used to verify the messages for unicast and broadcast communication. It has claimed that both the signature generation and verification are quicker than the other existing schemes e.g. ECDSA (elliptic curve digital signature algorithm). In security analysis of the algorithm, authors have claimed that the signature scheme is preimage resistant and second preimage resistant. However, the authors have not claimed that the scheme is collision resistance, which is also another important property.

In [11] authors have designed a strong and efficient scheme against node capture attack using hash chain in WSN. The primary idea of the scheme is to use preloaded keys to calculate the hash value. The hashed key is used as a communication key. This scheme also shows an improvement in comparison of other existing schemes.

In another work [12] authors have proposed a cryptographic hash function Whirlwind which can be easily implemented in software. This method can be considered as an expansion in design compared to SHA-3 hash functions. This scheme uses large S-boxes which allow proficient implementation on a wide variety of platforms. The hash function produces 512-bit long hash digest by incorporating a compress function, computing initialization vector etc. The hash digest is represented by an 8x4 array of 16-bit elements each. Although the cryptanalysis of this scheme shows a progress but the performance of software implementation is not very fast compared to the other competing schemes.

## 3. PROPOSED LIGHT WEIGHT CRYPTOGRAPHIC HASH ALGORITHM FOR WSN

Any hash function which has to be evolved is required to satisfy a number of desirable characteristics.

### 3.1 Requirements of an Ideal Hash Function

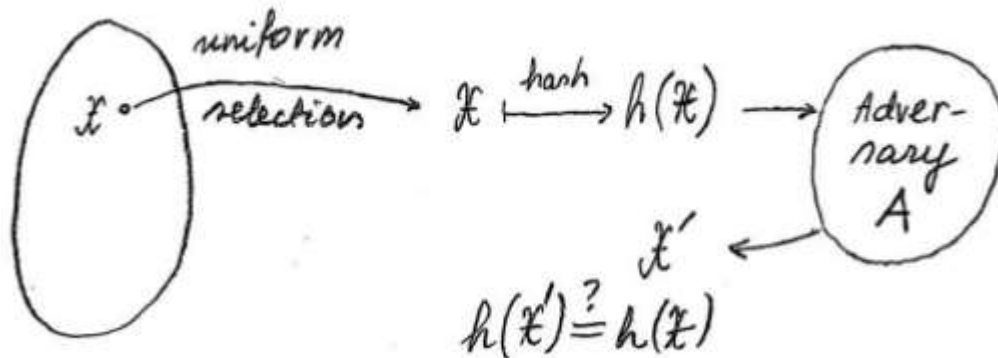
Following are the essential characteristics which a hash calculating functions need to follow-

- i. It should accept the arbitrary length inputs.
- ii. Its output length must be fixed size.

iii. It should be efficient and its computation must be fast.

iv. **Pre-image resistant:** It is “one-wayness” property of the hash function (i.e. it should not be possible to calculate the input from the hash output). A hash function for which preimage/input cannot be efficiently solved is said to be preimage resistant. So, a preimage resistant function must ensure that given  $h(X)$ , it should not be possible to calculate  $X$ .

v. **Second Preimage Resistant (Weak Collision Resistant):** Attacker should not be able to compute  $X'$  which has the same hash as  $X$  has. If it is computationally feasible  $h(X)$  cannot be considered as a fingerprint unique to  $X$ .



**Fig 3.1** Example

of Second Preimage Resistance

Fig. 3.1 explains that given a hash  $h(X)$  of a randomly chosen input  $X$ , it is hard to find an input  $X'$  with the same output  $h(X') = h(X)$ .

vi. **Strong Collision Resistant:** The difference between weak and strong collision resistance is very subtle. This can be clarified using Birthday Paradox.

- Given a person (& his birthday), identifying the second person with the same birthday corresponds to weak collision problem.  
 (There is a fix  $X_1$ , finding an another element  $X_2$  where  $h(X_1) = h(X_2)$ , this instance corresponds to weak collision)
- Identification of any 2 people in the group having the same birthday corresponds to the strong collision problem.  
 (Finding any 2 elements in a group of  $n$  elements s.t.  $h(X_1) = h(X_2)$ , this instance corresponds to strong collision)

### 3.2 Proposed Hash Function

The proposed algorithm takes a message of arbitrary length as input. The output is a 12 byte ( $12 \times 8 = 96$  bits) hash digest. The steps of the algorithm are as follows-

1. Declare the substitution table  $S_{Table\_1}$  containing prime numbers chosen randomly. This table is used in first transformation.

$S_{Table\_1}[] = \{521, 997, 983, 733, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 809, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 863, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509\};$

2. Declare the substitution table  $S_{Table\_2}$  which consists of 67 prime numbers chosen randomly to reduce overhead & ensure uniformity in transformation. This table is used in second transformation.

```
STable_2[ ] = {2911, 2899, 2893, 2887, 2871, 2857, 2851, 2837, 2833, 2827, 2809, 2797, 2791, 2779, 2773, 2767, 2749, 2743, 2737, 2731, 2723, 2719, 2713, 2711, 2701, 2687, 2683, 2677, 2659, 2653, 2647, 2641, 2629, 2623, 2617, 2611, 2587, 2563, 2539, 2503, 2497, 2447, 2431, 2419, 2413, 2407, 2401, 2389, 2383, 2377, 2359, 2347, 3193, 3173, 3157, 3139, 3121, 2257, 2251, 2239, 2227, 2221, 2197, 2191, 2179, 2173, 2167}
```

3. Initialize the following variables-

```
first_conv = 1;
second_conv_p3 = 7;
state[0] = 0x01234567;
state[1] = 0x89ABCDEF;
state[2] = 0xFEDCBA10;
```

where state[0], state[1] and state[2] are 3 variables which help in calculating the final hash value.

4. Preprocess the arbitrary long input message by converting each of the input character into 8 bit binary

5. Apply the padding in least significant position to make it divisible by 512

6. Split the message 3 times in a nested manner as follows

7. At first level, split the input message (of any length) in a block size of 512 bit each  
for ( i = 0; i < t; i++), t number of blocks of 512 bits each

8. At second level, split one block into 8 blocks of 64 bits each

```
for ( j = 0; j < 8; j++), 8 blocks of 64 bits each
```

9. At third level, split each of 64 bits of one block into 8 subblocks of 8 bits each.

```
for ( k = 0; k < 8; k++), 8 numbers of 8 bit subblock each
```

10. Obtain subblock[i][j][k] // result of 3-level split.

11. First substitute the inner most block (8 bit) using substitution table Stable\_1 as per follows and check subblock[i][j][k] contains for at least one 1, update subblock[i][j][k] accordingly-

```
if (subblock[i][j][k]!=0) {
    p[k]=abs(subblock[i][j][k]-31) //p[k] value doesn't exceed 97
    subblock[i][j][k]=STable_1[p[k]];
}
else {
    p[k]=1;
    subblock[i][j][k]=STable_1[p[k]];
}
```

12. Calculate the value of first\_conv variable

```
first_conv = STable_1[p[k]]* first_conv;
if ( first_conv > 65535)
    first_conv = first_conv % 65536;
```

13. Once the first conversion is over, second conversion takes place

```
second_conv_p1= first_conv % 67;
if (subblock[i][j][7] == 0)
    second_conv_p2 = second_conv_p1;
else
    second_conv_p2 = 67-second_conv_p1;
second_conv_p3 = second_conv_p3 + (second_conv_p1 + first_conv) % 256;
```

14. Third conversion takes place and swapping of values are done-  
after\_second\_conv = ( first\_conv % second\_conv\_p3 ) + first\_conv +  
STable\_2[second\_conv\_p2];  
after\_third\_conv = (after\_second\_conv % 256) + p[2] + p[0] % 127;  
after\_third\_conv = after\_third\_conv^state[0];  
Apply intra-hexnumber hexdigit swapping on each hexnumber.  
state[0] = state[1];  
state[1] = state[2];  
state[2] = after\_third\_conv;

15. Compute the final hash digest  
for (int i=0; i < 3; ++i) {  
hash[i] = (state[0]>> (i\*8)) & 0x0000ff;  
hash[i+4] = (state[1]>> (i\*8)) & 0x0000ff;  
hash[i+8] = (state[2]>> (i\*8)) & 0x0000ff;

16. Apply inter-hex number swapping on hash output.

The final hash digest is 96 bits long whatever may be the length of input message.

### 3.3 Performance Analysis

As discussed in section 5.1, every hash algorithm needs to follow six essential properties. The strength of the proposed algorithm can be evaluated by discussing the extent of maintaining these basic properties of a cryptographic hash function.

- i. The proposed algorithm follows the first property of a cryptographic hash function as the input message can be arbitrary long.
- ii. It satisfies the second property as it produces fixed length output.

The efficiency of the proposed algorithm can be observed from the implementation part as follows.

- iii. **Preimage Resistance** - For a given hash digest H with respect to an unknown input, it is not feasible to find an input message m such that h (m) =H where h (m) is the message digest of m. It symbolizes the one-way property of a hash function. The final hash digest is the sum of t times of 12 byte hexadecimal numbers where, t is number of 512 bit blocks. Let H be the final hash digest.

$H = \sum H_i$ , i varies from 0 to t-1 and  $H_i$  is hash digest of each of t number of 512 bit blocks. There will be  $^{H+t-1}C_{t-1}$  ways to calculate  $H_i$ . If t=1, the number of solutions would become one.

In a brute force attack, upon capturing the digest H, it is attempted to find a message m such that h(m) = H(given). The attacker has to perform of the order of  $2^{96}$  operations which will take high amount of time to perform brute force attack. Hence the algorithm is preimage resistant.

- iv. **Collision Resistance** – To find out h(m1) = h(m2) the followings need to be true

- i)  $hexnumber_{ij}(m1) = hexnumber_{ij}(m2)$  for all i, j where  $i \in \{0,1,\dots,t-1\}$  and  $j \in \{0,1,\dots,7\}$  and
- ii)  $subblock_{ij}(m1) \neq subblock_{ij}(m2)$  for all i, j where  $i \in \{0,1,\dots,t-1\}$  and  $j \in \{0,1,\dots,7\}$

For these conditions to be correct at least one of the following conditions need to be satisfied:

- a)  $first\_conv(m1) = first\_conv(m2)$  and  $sub\_block_{ij}(m1) \neq sub\_block_{ij}(m2)$
- b)  $after\_second\_conv(m1) = after\_second\_conv(m2)$  and  $sub\_block_{ij}(m1) \neq sub\_block_{ij}(m2)$
- c)  $after\_third\_conv(m1) = after\_third\_conv(m2)$  and  $sub\_block_{ij}(m1) \neq sub\_block_{ij}(m2)$

So the only feasible method to satisfy at least one of the above conditions is brute force attack. Hence our algorithm is collision resistant.  $2^{48}$  operations are to be performed by the brute force method to compute two messages having the same message digest which would be time consuming for a sensor node.



v. **Second Preimage Resistance** – For a given input message  $m_1$ , if it is impossible to find another input message  $m_2$  where the hash output of first message is equal to the hash output of second input message  $m_2$  i.e.,  $h(m_1) = h(m_2)$ .

Second preimage resistance or weak collision resistance is an easier or weaker version of strong collision resistance. So, a strong collision resistant function would also follow the property of second preimage resistant.

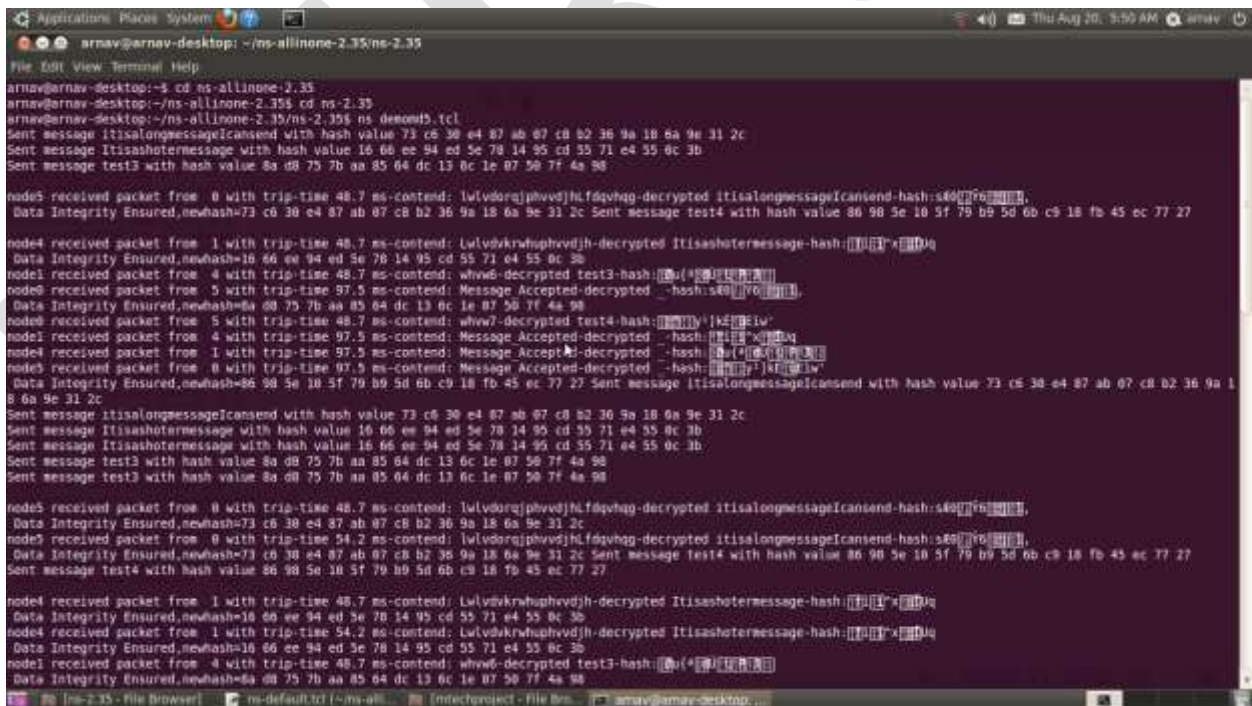
#### 4. IMPLEMENTATION

Network Simulator (Ver. 2), popularly known as NS2, is a discrete event driven simulation tool. There is a great need to simulate the protocols and algorithms before their actual implementation. NS2 has been proved useful in simulating the dynamic nature of communication networks. It is used to simulate the network functions and protocols of wired as well as wireless network. It provides the researchers a way to implement network protocols and help them to understand their corresponding behaviours using simulation.

In this work, cryptographic algorithm along with hash functions are being used to send data securely between two nodes. MD5, SHA-1 and one more protocol is being simulated while communicating between two nodes. In this work we have used CESAR cipher, for encryption/decryption of messages and MD5, SHA-1 and a proposed light weight cryptographic algorithm to calculate the hash digest. Results are compared. The proposed scheme is tested against MD5 and SHA-1. From the simulation of the experimental results, we can conclude that proposed algorithm may be used in Wireless Sensor Network for hashing the data exchanged between nodes.

##### 4.1 Demonstration and Results

For the demonstration to be carried out, six nodes have been created. Node 0, 1 will send message to node 4 and 5 respectively and from node 4 and 5 back to node 0 and 1. An acknowledgement packet is expected by each sender node from the receiving node. A script is created using the TCL language to simulate this scenario. The following figures are the proofs to the demonstration. Figure 4.1 shows the communication of node 0 and 1 with the node 5 and 4 respectively using the hash digest calculated through MD5 algorithm. Corresponding acknowledgements are also being issued as can be seen in this figure. Figure 4.2 shows the communication among the nodes using SHA-1 hash algorithm whereas Figure 4.3 replicate the scenario using the proposed hash algorithm.



```
arnav@arnav-desktop: ~/ns-allinone-2.35/ns-2.35
arnav@arnav-desktop:~$ cd ns-allinone-2.35
arnav@arnav-desktop:~/ns-allinone-2.35$ cd ns-2.35
arnav@arnav-desktop:~/ns-allinone-2.35/ns-2.35$ ns demoMd5.tcl
Sent message ItisalongmessageIcansend with hash value 73 c6 30 e4 87 ab 07 c8 b2 36 9a 18 6a 9e 31 2c
Sent message Itisashortmessage with hash value 16 66 ee 94 ed 5e 78 14 95 cd 35 71 e4 55 8c 3b
Sent message test3 with hash value 8a d8 75 7b aa 85 64 dc 13 6c 1e 07 50 7f 4a 98

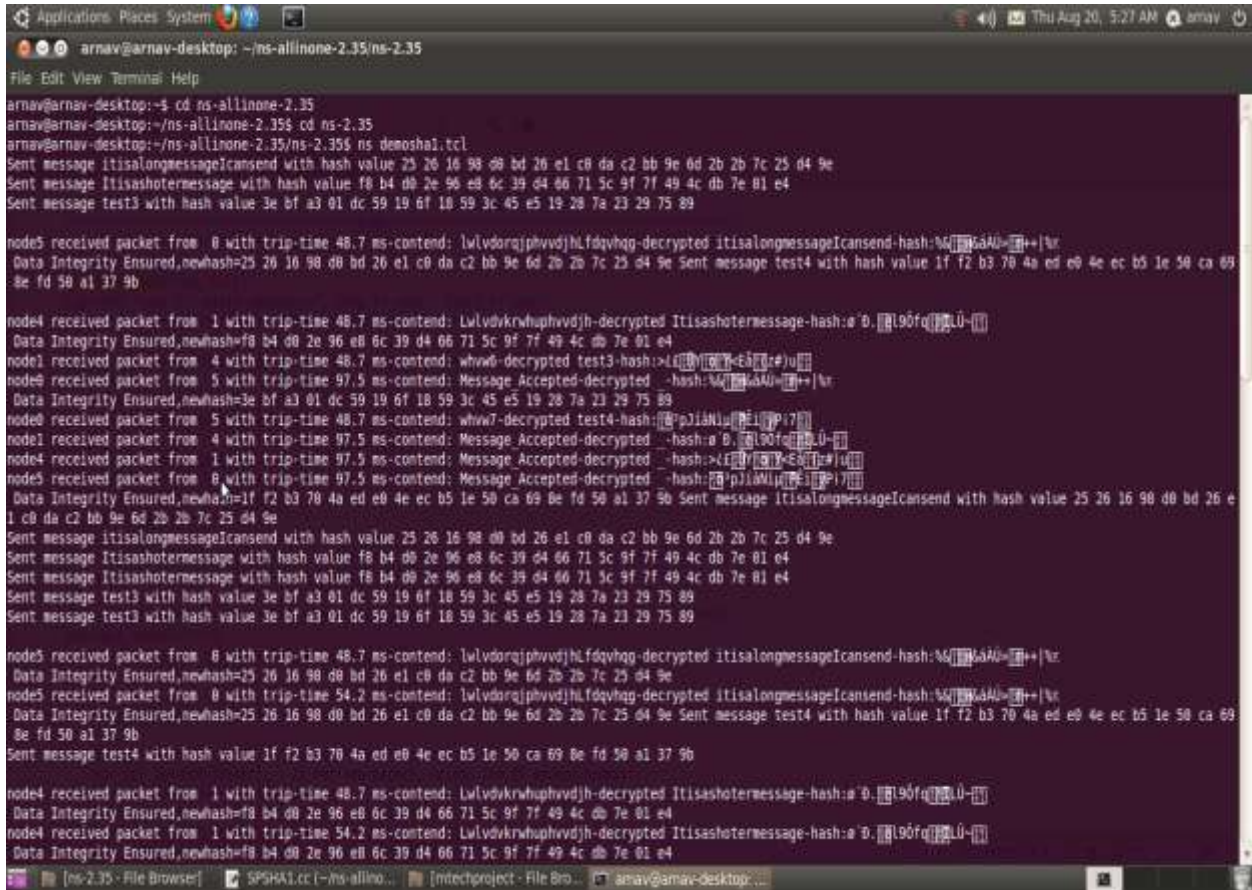
node5: received packet from 0 with trip-time 48.7 ms-content: lUlv0rgjphvvdjh.f0qvqg-decryptd ItisalongmessageIcansend-hash:s40[0]r6[0]
Data Integrity Ensured,newhash=73 c6 30 e4 87 ab 07 c8 b2 36 9a 18 6a 9e 31 2c Sent message test4 with hash value 86 98 5e 10 5f 79 b9 50 6b c9 18 7b 45 ec 77 27

node4: received packet from 1 with trip-time 48.7 ms-content: Lvlv0kvrhuphvvdjh-decryptd Itisashortmessage-hash:[0][0]x[0][0]0q
Data Integrity Ensured,newhash=16 66 ee 94 ed 5e 78 14 95 cd 35 71 e4 55 8c 3b
node1: received packet from 4 with trip-time 48.7 ms-content: whw6-decryptd test3-hash:[0][0]x[0][0]0q
node0: received packet from 5 with trip-time 97.5 ms-content: Message Accepted-decryptd --hash:s40[0]r6[0]
Data Integrity Ensured,newhash=8a d8 75 7b aa 85 64 dc 13 6c 1e 07 50 7f 4a 98
node0: received packet from 5 with trip-time 48.7 ms-content: whw7-decryptd test4-hash:[0][0]x[0][0]0q
node1: received packet from 4 with trip-time 97.5 ms-content: Message Accepted-decryptd --hash:[0][0]x[0][0]0q
node4: received packet from 1 with trip-time 97.5 ms-content: Message Accepted-decryptd --hash:[0][0]x[0][0]0q
node3: received packet from 0 with trip-time 97.5 ms-content: Message Accepted-decryptd --hash:[0][0]x[0][0]0q
Data Integrity Ensured,newhash=86 98 5e 10 5f 79 b9 50 6b c9 18 7b 45 ec 77 27 Sent message ItisalongmessageIcansend with hash value 73 c6 30 e4 87 ab 07 c8 b2 36 9a 18 6a 9e 31 2c
Sent message ItisalongmessageIcansend with hash value 73 c6 30 e4 87 ab 07 c8 b2 36 9a 18 6a 9e 31 2c
Sent message Itisashortmessage with hash value 16 66 ee 94 ed 5e 78 14 95 cd 35 71 e4 55 8c 3b
Sent message Itisashortmessage with hash value 16 66 ee 94 ed 5e 78 14 95 cd 35 71 e4 55 8c 3b
Sent message test3 with hash value 8a d8 75 7b aa 85 64 dc 13 6c 1e 07 50 7f 4a 98
Sent message test3 with hash value 8a d8 75 7b aa 85 64 dc 13 6c 1e 07 50 7f 4a 98

node5: received packet from 0 with tria-time 48.7 ms-content: lUlv0rgjphvvdjh.f0qvqg-decryptd ItisalongmessageIcansend-hash:s40[0]r6[0]
Data Integrity Ensured,newhash=73 c6 30 e4 87 ab 07 c8 b2 36 9a 18 6a 9e 31 2c
node5: received packet from 0 with trip-time 54.2 ms-content: lUlv0rgjphvvdjh.f0qvqg-decryptd ItisalongmessageIcansend-hash:s40[0]r6[0]
Data Integrity Ensured,newhash=73 c6 30 e4 87 ab 07 c8 b2 36 9a 18 6a 9e 31 2c Sent message test4 with hash value 86 98 5e 10 5f 79 b9 50 6b c9 18 7b 45 ec 77 27
Sent message test4 with hash value 86 98 5e 10 5f 79 b9 50 6b c9 18 7b 45 ec 77 27

node4: received packet from 1 with trip-time 48.7 ms-content: Lvlv0kvrhuphvvdjh-decryptd Itisashortmessage-hash:[0][0]x[0][0]0q
Data Integrity Ensured,newhash=16 66 ee 94 ed 5e 78 14 95 cd 35 71 e4 55 8c 3b
node4: received packet from 1 with trip-time 54.2 ms-content: Lvlv0kvrhuphvvdjh-decryptd Itisashortmessage-hash:[0][0]x[0][0]0q
Data Integrity Ensured,newhash=16 66 ee 94 ed 5e 78 14 95 cd 35 71 e4 55 8c 3b
node1: received packet from 4 with trip-time 48.7 ms-content: whw6-decryptd test3-hash:[0][0]x[0][0]0q
Data Integrity Ensured,newhash=8a d8 75 7b aa 85 64 dc 13 6c 1e 07 50 7f 4a 98
```

Fig 4.1 md5shot1



```
arnav@arnav-desktop: ~/$ cd ns-allinone-2.35
arnav@arnav-desktop:~/ns-allinone-2.35$ cd ns-2.35
arnav@arnav-desktop:~/ns-allinone-2.35/ns-2.35$ ns demosh1.tcl
Sent message itisalongmessageIcansend with hash value 25 26 16 98 d8 bd 26 e1 c8 da c2 bb 9e 6d 2b 2b 7c 25 d4 9e
Sent message Itisashotermessag with hash value f8 b4 d8 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 81 e4
Sent message test3 with hash value 3e bf a3 01 dc 59 19 6f 18 59 3c 45 e5 19 28 7a 23 29 75 89

node5 received packet from 8 with trip-time 48.7 ms-content: lUvdorajphvvd]hLfdqvhqg-decrypted itisalongmessageIcansend-hash:Na[90f]L-U-[]\r
Data Integrity Ensured,newhash=25 26 16 98 d8 bd 26 e1 c8 da c2 bb 9e 6d 2b 2b 7c 25 d4 9e Sent message test4 with hash value 1f f2 b3 70 4a ed e0 4e ec b5 1e 50 ca 69
8e fd 50 a1 37 9b

node4 received packet from 1 with trip-time 48.7 ms-content: Lvlvdvkrwhuphvvd]h-decrypted Itisashotermessag-hash:a 0. [90f]L-U-[]\r
Data Integrity Ensured,newhash=f8 b4 d8 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 81 e4
node1 received packet from 4 with trip-time 48.7 ms-content: whw6-decrypted test3-hash:>L[90f]L-U-[]\r
node9 received packet from 5 with trip-time 97.5 ms-content: Message Accepted-decrypted -hash:Na[90f]L-U-[]\r
Data Integrity Ensured,newhash=3e bf a3 01 dc 59 19 6f 18 59 3c 45 e5 19 28 7a 23 29 75 89
node0 received packet from 5 with trip-time 48.7 ms-content: whw7-decrypted test4-hash:[90f]L-U-[]\r
node1 received packet from 4 with trip-time 97.5 ms-content: Message Accepted-decrypted -hash:a 0. [90f]L-U-[]\r
node4 received packet from 1 with trip-time 97.5 ms-content: Message Accepted-decrypted -hash:>L[90f]L-U-[]\r
node5 received packet from 8 with trip-time 97.5 ms-content: Message Accepted-decrypted -hash:[90f]L-U-[]\r
Data Integrity Ensured,newhash=1f f2 b3 70 4a ed e0 4e ec b5 1e 50 ca 69 8e fd 50 a1 37 9b Sent message itisalongmessageIcansend with hash value 25 26 16 98 d8 bd 26 e
1 c8 da c2 bb 9e 6d 2b 2b 7c 25 d4 9e
Sent message itisalongmessageIcansend with hash value 25 26 16 98 d8 bd 26 e1 c8 da c2 bb 9e 6d 2b 2b 7c 25 d4 9e
Sent message Itisashotermessag with hash value f8 b4 d8 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 81 e4
Sent message Itisashotermessag with hash value f8 b4 d8 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 81 e4
Sent message test3 with hash value 3e bf a3 01 dc 59 19 6f 18 59 3c 45 e5 19 28 7a 23 29 75 89
Sent message test3 with hash value 3e bf a3 01 dc 59 19 6f 18 59 3c 45 e5 19 28 7a 23 29 75 89

node5 received packet from 8 with trip-time 48.7 ms-content: lUvdorajphvvd]hLfdqvhqg-decrypted itisalongmessageIcansend-hash:Na[90f]L-U-[]\r
Data Integrity Ensured,newhash=25 26 16 98 d8 bd 26 e1 c8 da c2 bb 9e 6d 2b 2b 7c 25 d4 9e
node5 received packet from 8 with trip-time 54.2 ms-content: lUvdorajphvvd]hLfdqvhqg-decrypted itisalongmessageIcansend-hash:Na[90f]L-U-[]\r
Data Integrity Ensured,newhash=25 26 16 98 d8 bd 26 e1 c8 da c2 bb 9e 6d 2b 2b 7c 25 d4 9e Sent message test4 with hash value 1f f2 b3 70 4a ed e0 4e ec b5 1e 50 ca 69
8e fd 50 a1 37 9b
Sent message test4 with hash value 1f f2 b3 70 4a ed e0 4e ec b5 1e 50 ca 69 8e fd 50 a1 37 9b

node4 received packet from 1 with trip-time 48.7 ms-content: Lvlvdvkrwhuphvvd]h-decrypted Itisashotermessag-hash:a 0. [90f]L-U-[]\r
Data Integrity Ensured,newhash=f8 b4 d8 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 81 e4
node4 received packet from 1 with trip-time 54.2 ms-content: Lvlvdvkrwhuphvvd]h-decrypted Itisashotermessag-hash:a 0. [90f]L-U-[]\r
Data Integrity Ensured,newhash=f8 b4 d8 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 81 e4
```

Fig 4.2 shashot1

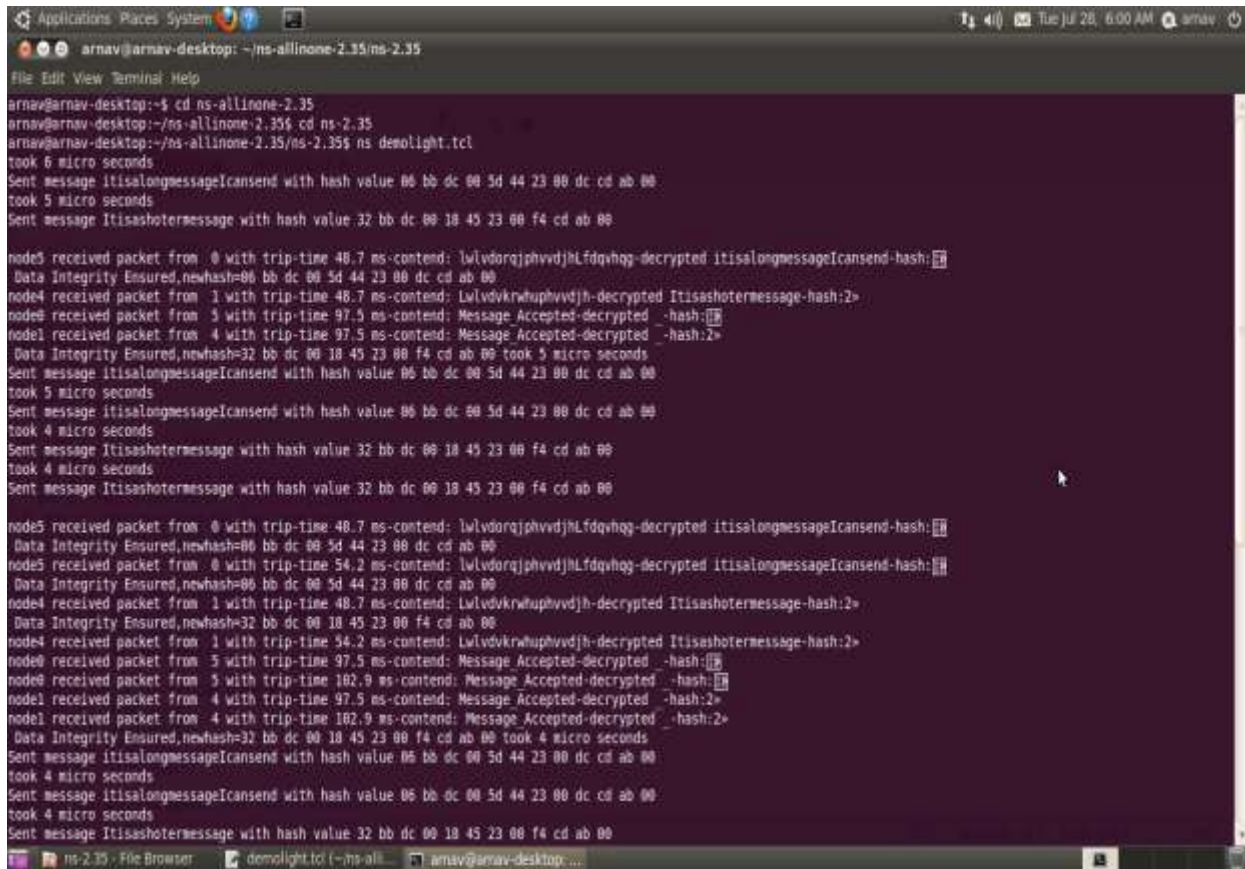


Fig 4.3 lightshot1

## 5. CONCLUSION AND FUTURE SCOPE

In this work the basic requirements to design a lightweight, one-way hash algorithm which produces a fixed & relatively small length hash digest which is applicable for securing energy-starved wireless network e.g. WSN are discussed. Operations like MOD and SWAP are used extensively to make the proposed hash algorithm lightweight. All the basic properties such as preimage resistance, strong collision resistance and weak or second preimage resistance are fulfilled in the proposed algorithm. The proposed algorithm is implemented using NS-2 simulator. It is compared with MD5 and SHA-1 has shown significant improvement in terms of communication overhead and computation speed.

As per Mica2 specification, energy consumption [15] [16] for transmitting one byte of data for ATmega 128 processor is 16.25µJ and energy required per clock cycle is 3.2nJ or 0.0032µJ. For the given specification, the energy requirements for all the competing schemes are computed below:

### Communication overhead:

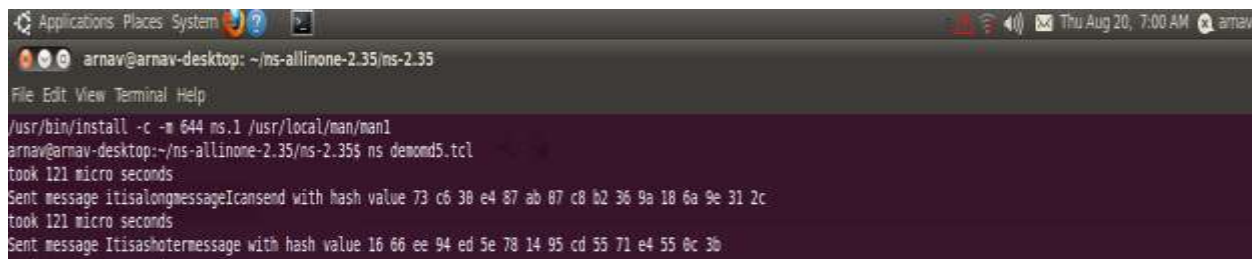
MD5 -- 16.25x128 (bit) = 16.25x16 (byte) = 260 µJ

SHA1-- 16.25x160 (bit) = 16.25x20 (byte) = 325 µJ

Proposed Algorithm—16.25x96 (bit) = 16.25x12 (byte) = 195 µJ

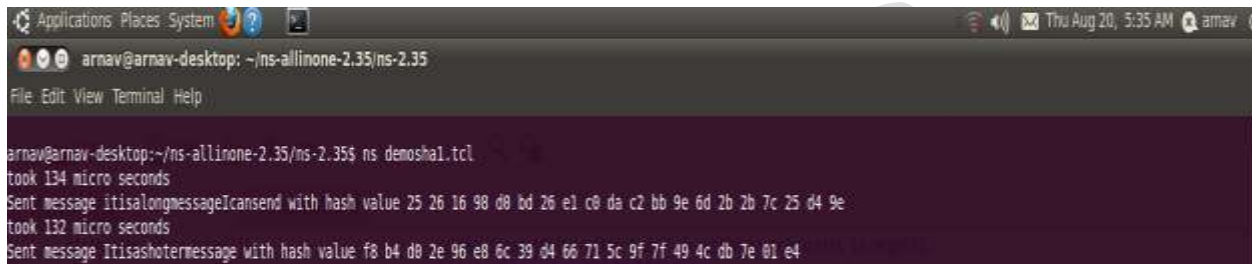
### Comparison of Computation Speed:

Following snapshots clearly indicates that the time taken to calculate the hash of a sample string “ItisalongmessageIcansend” takes approximately 121, 134 and 115 micro seconds using MD5, SHA1 and the proposed light weight algorithm respectively.



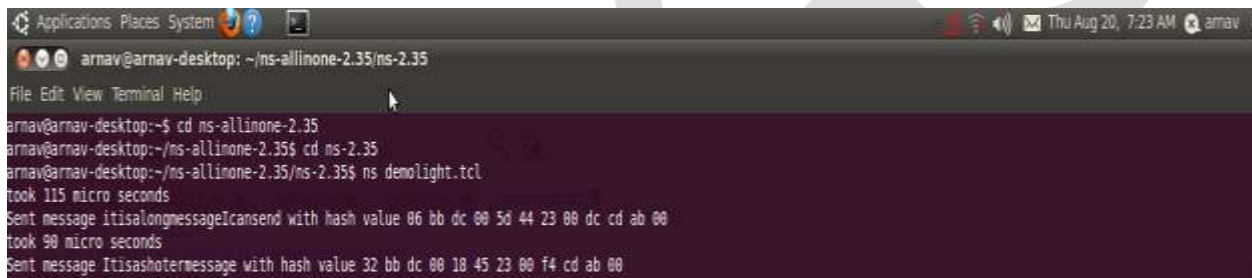
```
Applications Places System
arnav@arnav-desktop: ~/ns-allinone-2.35/ns-2.35
File Edit View Terminal Help
/usr/bin/install -c -m 644 ns.1 /usr/local/man/man1
arnav@arnav-desktop:~/ns-allinone-2.35/ns-2.35$ ns demond5.tcl
took 121 micro seconds
Sent message itisalongmessageIcansend with hash value 73 c6 38 e4 87 ab 87 c8 b2 36 9a 18 6a 9e 31 2c
took 121 micro seconds
Sent message Itisashotermesssage with hash value 16 66 ee 94 ed 5e 78 14 95 cd 55 71 e4 55 0c 3b
```

Fig 5.1 md5\_time\_shot



```
Applications Places System
arnav@arnav-desktop: ~/ns-allinone-2.35/ns-2.35
File Edit View Terminal Help
arnav@arnav-desktop:~/ns-allinone-2.35/ns-2.35$ ns demosh1.tcl
took 134 micro seconds
Sent message itisalongmessageIcansend with hash value 25 26 16 98 d8 bd 26 e1 c8 da c2 bb 9e 6d 2b 2b 7c 25 d4 9e
took 132 micro seconds
Sent message Itisashotermesssage with hash value f8 b4 d8 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 01 e4
```

Fig 5.2 sha1\_time\_shot



```
Applications Places System
arnav@arnav-desktop: ~/ns-allinone-2.35/ns-2.35
File Edit View Terminal Help
arnav@arnav-desktop:~$ cd ns-allinone-2.35
arnav@arnav-desktop:~/ns-allinone-2.35$ cd ns-2.35
arnav@arnav-desktop:~/ns-allinone-2.35/ns-2.35$ ns demolight.tcl
took 115 micro seconds
Sent message itisalongmessageIcansend with hash value 06 bb dc 00 5d 44 23 08 dc cd ab 00
took 90 micro seconds
Sent message Itisashotermesssage with hash value 32 bb dc 00 18 45 23 00 f4 cd ab 00
```

Fig 5.3 light\_time\_shot

The significant improvement in the communication and computation overhead has been observed while implementing the proposed light weight cryptographic hash algorithm in NS2. As a future extension, more efforts could be made to crosscheck the claim made in performance analysis using the Simulator.

## REFERENCES:

- [1] I. F. Akyildiz *et al.*, “A Survey on Sensor Networks,” *IEEE Commun. Mag.*, vol. 40, no. 8, Aug. 2002, pp. 102–114.
- [2]. Handbook of Cryptography by A. Menezes, P. van Oorschot, and S. Vanstone, CRC Press, 1996.
- [3]. H. Dobbertin: Cryptanalysis of MD4, Journal of Cryptology, vol. 11, No. 4, pp. 253-271, 1998.
- [4]. X. Wang and H. Yu: How to Break MD5 and Other Hash Functions, EuroCrypt 2005, Springer LNCS 3494, pp. 19–35, 2005.
- [5]. M. Stevens, A. Lenstra and B. Weiger.: Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities, EUROCRYPT 2007: 1-22.
- [6]. Rijmen, V.Oswald: Update on SHA-1, RSA 2005, LNCS 3376, pp. 58-71.
- [7]. R. P. McEvoy, F. M. Crowe, C. C. Murphy and W. P. Marnane: Optimisation of the SHA-2 Family of Hash Functions on FPGAs, IEEE Computer Society Annual Symposium on VLSI: Emerging VLSI Technologies and Architectures (ISVLSI 06), IEEE Computer Society, Washington DC, pp. 317-322, 2006.
- [8]. Y. Jararweh, L. Tawalbeh, H. Tawalbeh and A. Moh’d: Hardware Performance Evaluation of SHA-3 Candidate Algorithms, Journal of Information Security, vol. 3 No. 2, 2012, pp. 69-76. doi: 10.4236/jis.2012.3, 2008.

- [9]. I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci: Wireless sensor network: a survey, *Computer Networks*, vol. 38, pp. 393-422, 2002.
- [10]. Erik Dahmen and Christoph Kraus: Short Hash-based Signatures for Wireless Sensor Networks, 8th International Conference on Cryptology & Network Security (CANS), Kanazawa, Ishikawa, Japan, December, 2009.
- [11]. Tao Qen, Hanli Chen: An Enhanced Scheme against Node Capture Attack using Hash Chain for Wireless Sensor Network, *Information Technology Journal* 11 (1), pp. 102-109, 2012.
- [12]. Paulo Barreto, Ventsislav Nikov, Svetla Nikova, Vincent Rijmen, Elmar Tischhauser: Whirlwind: a new cryptographic hash function, Springer, vol. 56, pp.141-162, 2010.
- [13]. Amrita Ghosal, Subir Halder & Sipra DasBit, : A Dynamic TDMA based scheme for securing query processing in WSN, *Journal of Mobile Communication, Computation and Information*, vol.18, number 2, pp. 165-186, 2012.
- [14]. [https://en.wikipedia.org/wiki/Cryptographic\\_hash\\_function](https://en.wikipedia.org/wiki/Cryptographic_hash_function) accessed on 18-07-2015
- [15]. Atmel AVR8-bit Microcontroller ATmega 128 processor datasheet (<http://tools.ietf.org/html/rfc4270>).
- [16] Amrita Ghosal, Subir Halder & Sipra DasBit, : A Dynamic TDMA based scheme for securing query processing in WSN, *Journal of Mobile Communication, Computation and Information*, vol. 18, number 2, pp. 165-186, 2012