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.

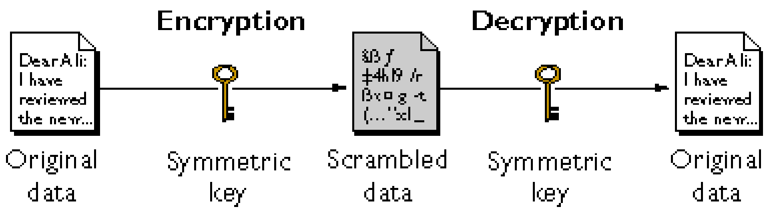


Fig 1.1 Symmetric Key Encryption

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.

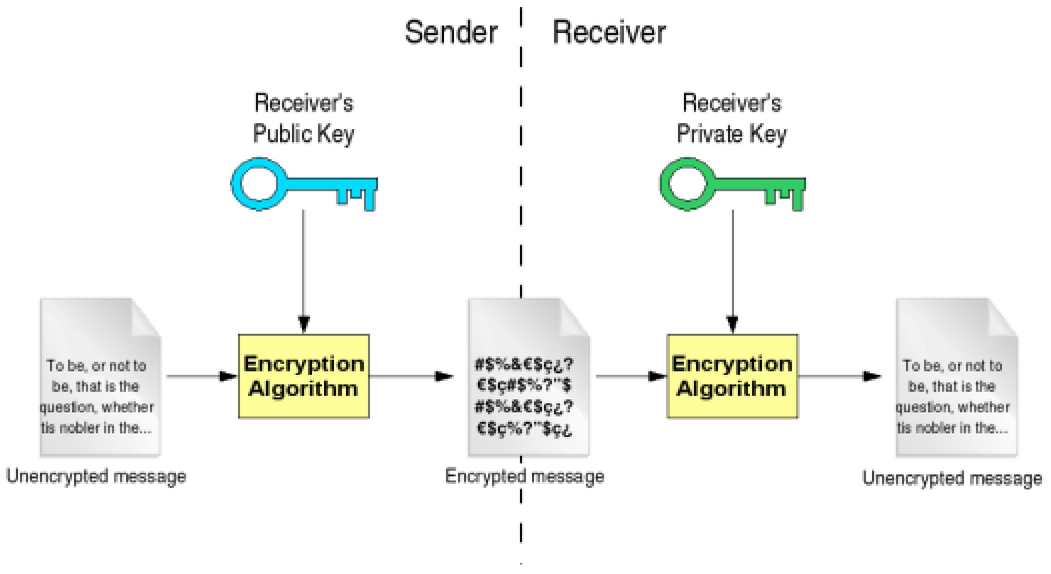

Sender
Receiver
Receiver's Public Key
Receiver's Private Key
Encryption Algorithm
Encryption Algorithm
To be, or not to be, that is the question, whether tis nobler in the...
#$%&€$ç¿? €$ç#$%?"$ #$%&€$ç¿? €$ç%?"$ç¿
To be, or not to be, that is the question, whether tis nobler in the...
Unencrypted message
Encrypted message
Unencrypted message

Fig 1.2 Asymmetric Key Encryption

A different set of problem is still there-

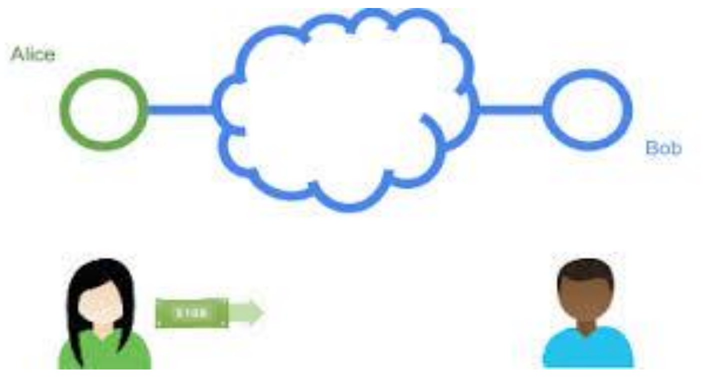


Fig 1.3 Alice Sending Message to Bob

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.

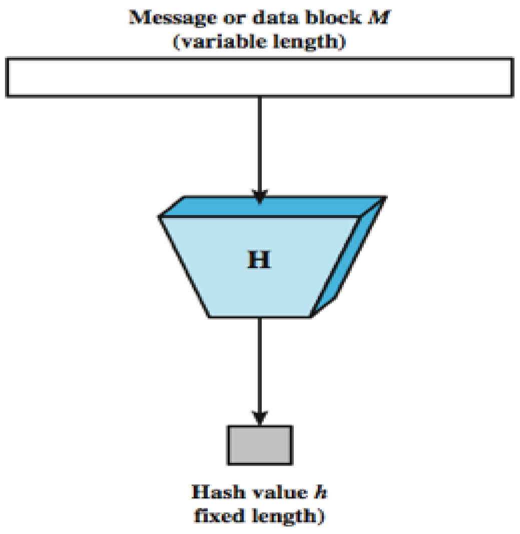


Fig 1.4 Signature of Long Message with a Hash Function

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.

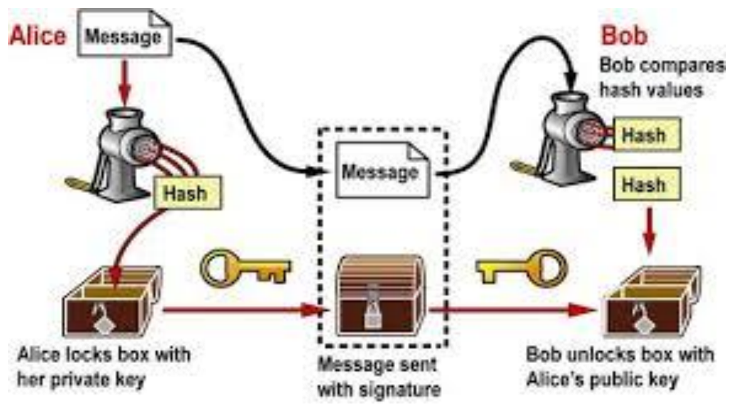

Alice
Message
Hash
Alice locks box with her private key
Message
Message sent with signature
Bob
Bob compares hash values
Hash
Hash
Bob unlocks box with Alice's public key

Fig 1.5 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 (see Appendix 1).

The message digest of a document is attached as a **fingerprint** or **signature** of the document to detect modification during transport. If you know the digest of the program, then nobody will be able to amend the program without being detected, because it is not easy to find another program with the same message digest.

In most applications, to challenge a system an attacker has to find an ambiguous message whose hash digest matches with a pre-existing message. To find 2 different messages having the same signature is a risk to the security of the system.

Suppose Alice wants to fire Fred, and writes her administrative officer Bob, to implement her decision. Alice computes a message digest, and cryptographically sign the message digest using her private key. Fred may play a trick by replacing the letter saying that Fred is brilliant and his salary ought to be doubled. However, Fred cannot generate a hash digest signed with Alice’s private key.

Suppose the message digest function produces only 64 bits long output, and is a good message digest function. The only way to find two messages with same hash digest would be by trying an adequate amount of messages that would have the same digest (as in birthday paradox). Total number of possible input messages is 2^{64}. Then using birthday paradox we would have to try about 2^{32} messages before we found two that matched.

There are sufficient computer generated variants of two letters that Fred can calculate message digests on various variants until he finds an exact match. It is computationally feasible to compute and test an order of 2^{32} messages, whereas due to computing limitations, it would not be feasible to deal with 2^{64} or more messages.

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.

Surprisingly, the drive for hash digest functions started with public key cryptography. RSA was invented, which made it possible to calculate digital signatures of messages. However, computing a signature on a long message with RSA was significantly slow. Instead of computing a signature over a long message, the message could be condensed into a smaller size by first computing a hash digest and then computing the RSA of the digest. So MD and MD2 were designed. MD was proprietary and never published. MD2 was documented in RFC 1319.

Then Ralph Merkle of Xerox designed a hash digest algorithm called SNEFRU that was quite a lot of times faster than MD2. This persuaded Ron Rivest into developing the MD4 algorithm documented in RFC 1320. Later weaknesses were found in SNEFRU and MD4. Ron Rivest decided to strengthen MD4 and created MD5 (RFC 1321), which is a little slower than MD4. NIST subsequently developed SHA, which is very much analogous to MD5, but more secure and also a little slower. NIST revised the algorithm at the last hour in an attempt to make it more secure and called the revised version SHA-1. The latest versions of SHA are SHA3-224, SHA3-256, SHA3-384 and SHA3-512.

The major difference between a secret key algorithm and a message digest algorithm is that in secret key algorithm, same key is used to encrypt and decrypt the data whereas a message digest algorithm is always impossible to reverse.

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. 1.6). 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.

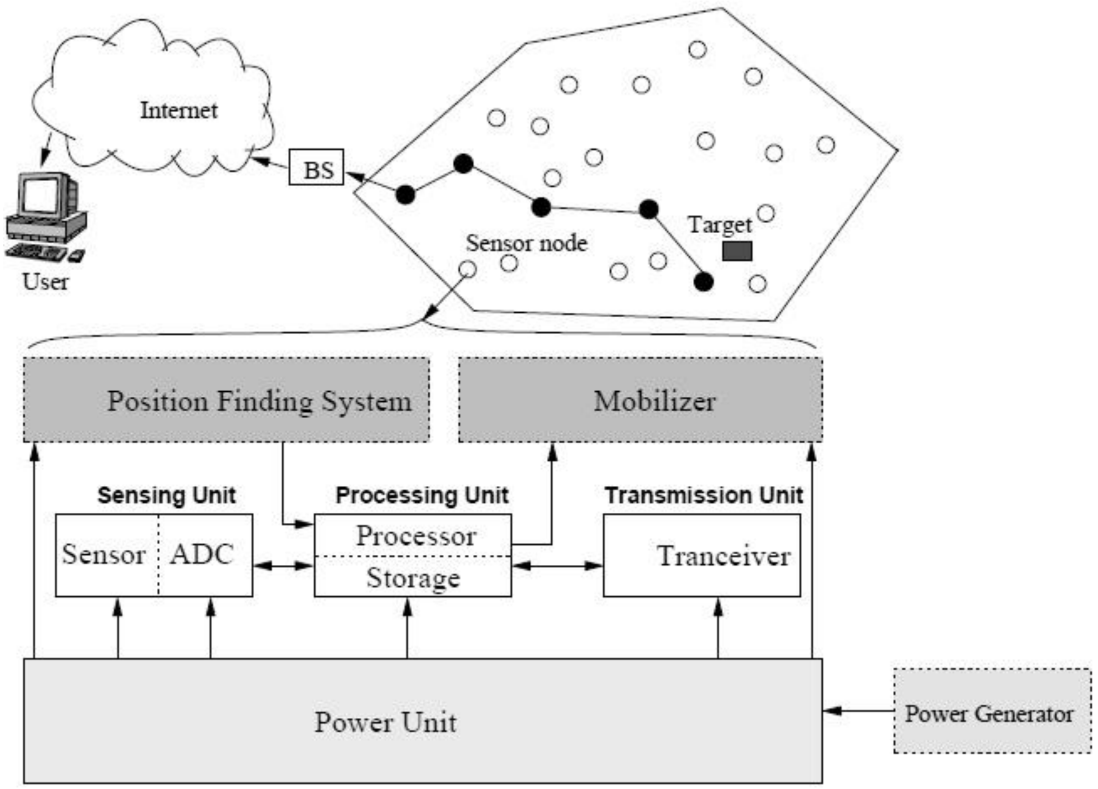


Fig 1.6 Wireless Sensor

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.

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.

Applications of Cryptographic Hash Functions

The hash functions are popularly used in following areas-

3.1 Verifying the Integrity of Messages or Files

Hashes are used to verify the authentication of messages or files integrity. Hash functions are used to calculate hash digest which can help to identify the changes made to a message (or a file) during transmission. It is achieved by computing the hash digests before sending the data to the recipient and attach this hash value with the data. At the other end the recipient will re-compute the hash of the data received and compare it with the received hash to verify the integrity of file or data. This is how the authenticity of the received data is verified using hash digest.

MD5, SHA1, or SHA2 hashes are sometimes posted on websites or forums along with the files to ensure the integrity of the file. This practice establishes a chain of trust so long as the hashes are posted on a site authenticated by HTTPS [14].

3.2 Password Verification

Another use of hash digest is verification of passwords used in any application. It was first invented by Roger Needham. User passwords are not stored as clear text in databases because clear text storage can lead to a massive security breach if the password file is compromised somehow. To reduce the risk of breach the hash digest of each password would be stored. During authentication of a user, when the password is provided by the user, hash digest of the password is computed and compared with the stored value in database. However it will prevent to retrieve the original passwords if forgotten. It can only be replaced with new ones. In latest trend, to calculate the hash value of the password, it is usually concatenated with a random, non-secret salt value before applying the hash functions. As different users may be using different salt values, it is not possible to store tables of pre-computed hash values for the common passwords. Key stretch functions are used for this purpose. It will increase the time needed to execute brute force attacks on stored password digests.

3.3 Digital Signatures

A digital signature is basically an encrypted hash of the message and used to verify a message. The recipient can ensure if the message is tampered with. It is done by computing the hash of the received message and comparing this value with the decrypted signature. Digital signatures assist the safe exchange of electronic data by specifying a way to check both the authenticity and the integrity of digital information exchanged. There is an explosion of organizations occupied in electronic data exchange as well as the quantity of data changing hands; the secure exchange of data has become a focal point for business users across almost all industries. A digital signature is computed by applying a series of mathematical process that convert data into a unique message digest. The sender encrypts the hash digest, attaches it to or embeds it in a file, and sends the parcel to the intended recipient. Once the parcel is received and the hash digest is decrypted, the authenticity and integrity of the received data can be ensured. If the digital signature verifies the identity of the sender one can be assured that it was truly sent by the individual related to the digital signature.

3.4 Pseudorandom Generation and Key Derivation

Pseudorandom bits can be generated using hash functions. It is also used to derive new keys or passwords. Usually, a pseudorandom function $\{PRF(p, n) \mid p \in S\}$ consists of polynomial time functions with a seed p and input variable n. When a cryptographic key K_c is regarded as the seed, that is, $p = K_c$, the output of the pseudorandom function can be used as keying material.

Traditional Hash Algorithms

There are several hash algorithms available these days which are widely in use to verify the integrity of the message. Two of the most popular algorithms are MD5 and SHA1. This chapter discuss about the mechanism of these two algorithms.

4.1 MD5 Algorithm

MD5 is a hash digest algorithm developed by Ron Rivest. It was designed to be somewhat more conservative than MD4 in terms of being less concerned with speed and more concerned with security. It is similar to MD4. The length of the hash digest it produces is 128 bit long. Several potential weaknesses have been diagnosed in MD5 by the researchers.

The input text in MD5 is processed in blocks of 512 bits each. This block is further divided into 16 sub blocks of 32 bits each. The algorithm come up with an output of 128 bits long message digest which is a set of 4, 32 bits sub-blocks.

Step 1. Append Padding Bits

To make the length of input message congruent to 448, modulo 512 (in bits), padding bits are added. Padding is always performed even if the length of the input message is already congruent to 448, modulo 512. To add padding a single "1" bit is followed by as many "0" bits are added so that the length of the processed text is short of 64 bit of 512 bits.

Step 2. Append Length

The length of the processed message (after step 1) excluding the padding bits is calculated and added to the last 64-bit (of 512 bit block) kept for this purpose. Sometimes if the length of the processed message (after step 1) is greater than 2^64, then only the low-order 64 bits of the calculated length is considered. The input message along with the padding bits and length of message become an exact multiple of 512 bits.

Step 3. Initialize MD Buffer

MD5 uses a buffer that consists of four-words (A, B, C, and D). Each of the buffers is 32-bit long word. Buffers are initialized to the following hex values:

A= 01 23 45 67
B= 89 ab cd ef

C= fe dc ba 98
D= 76 54 32 10

Step 4. Process Message in 16-Word Blocks

4.1: Copy the four chaining variables (A, B, C, D) into 4 corresponding temporary variables a,b,c,d. All the 4 variables (a-d) taken together will be a 128 bit long (32*4). This combined register is used for holding intermediate as well as final hash result.

4.2: Divide the current 512 bit long block into 16 sub-blocks of 32 bit each M [0] to M [15].

4.3: Now, MD5 consists of 4 rounds. The inputs to each round are

a. M [0]…..M [15] sub-blocks of a particular block
b. All the 4 variable (a-d)
c. Some predefined constants (K) are used

All the 4 rounds of MD5 vary in one major way. Step 1 of the each of four rounds is processed in different way. The other steps will remain the same.

- In each round, we have 16 input sub-blocks of 32 bits each, named M [0],M [1],....M [15]
- K is an array of constants. The number of elements in K is 64. The width of each element of K is 32 bit. 16 out of the 64 constants of K are used in each round

Let's summarize the iterations of all the 4 rounds. There are 16 iterations in each round.

One MD5 round operation can be expressed as-

$$a = b + ((a + \text{Process F}(b,c,d) + M[i] + K[k]) <<< s)$$

where

Process F = A non linear operation

M[i] = ith sub block of 32 bit of a 512 bit block

K[k] = A constant

<<<s = circular left shift by s bits

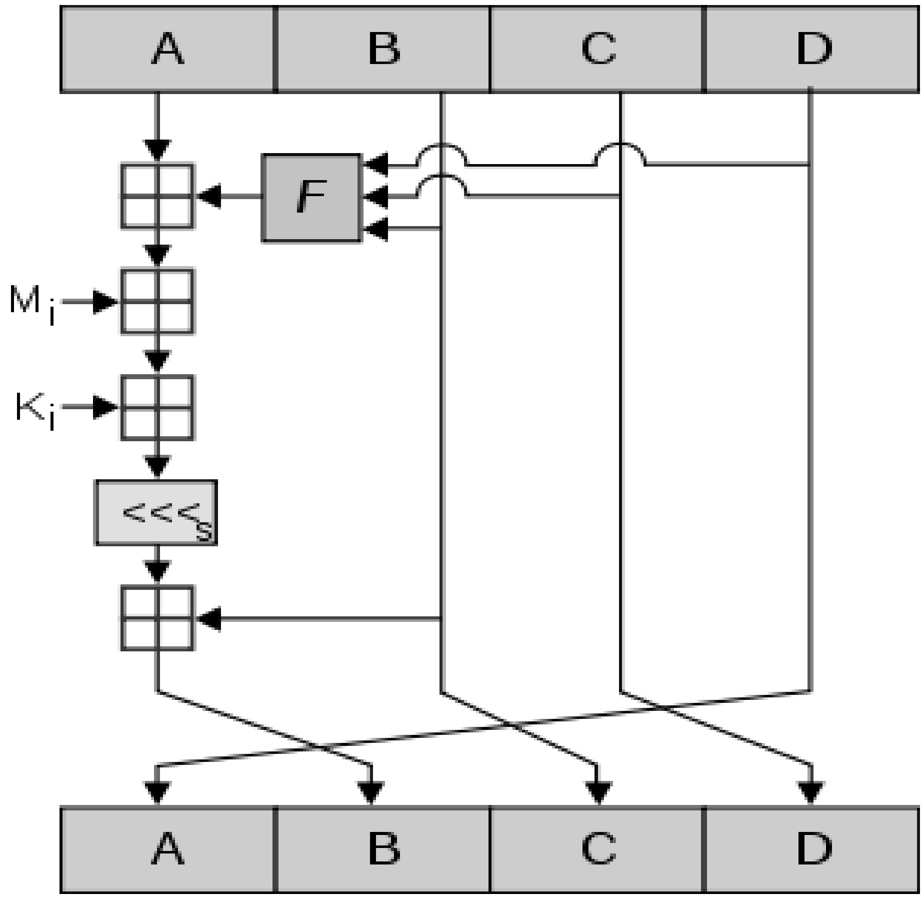


Fig 4.1 One Round of MD5

Understanding the process of F

Process F is different in the 4 rounds. This process is a collection of some basic Boolean operations.

Round	**Process F**
1	(b AND c) OR ((NOT b) AND (d))
2	(b AND d) OR (c AND (NOT d))
3	b XOR c XOR d
4	c XOR (b OR (NOT d))

Step 5. Output

The combination of A,B,C,D taken together will be the hash digest. We start with the low-order byte of A, and end with the high-order byte of D. This completes the description of MD5.

4.2 SHA1 Algorithm

SHA1 stands for “Secure Hashing Algorithm”. SHA1 is similar to MD5 which was designed by NSA and published by NIST. It was first published in 1995 as an improvement over the earlier version of hash algorithm SHA0. SHA1 is presently the most widely used SHA hash function. It is presently used in a wide variety of applications.

The algorithm come up with an output of 160 bits long message digest which is a set of 5, 32 bits sub-blocks.

Step 1. Padding: It is exactly same as in MD5

Step 2. Append Length: It is exactly same as in MD5

Step 3. Initialize MD Buffer: In SHA-1, 5 chaining variables A, B, C, D, E are initialized. In MD5 hash algorithm, 4 chaining variables A-D (32 *4=128 bits) were used in combination to keep intermediate as well as final results. As the hash output in SHA-1 is 160 (32*5=160) bits long, five chaining variables are considered (A-E). These registers are initialized to the following hex values, (low-order bytes first):

A = 01 23 45 67
B = 89 ab cd ef
C = fe dc ba 98
D =76 54 32 10
E = c3 d2 e1 f0

Step 4. Process Message in 16-Word Blocks

Now the actual algorithm begins. Steps are quite similar to those in MD5.

4.1: Copy the values of five chaining variables (A-E) into 5 corresponding temporary variables a-e. The 5 variables a-e taken together constitute 160 bit (32*5). This combination of 5 variables is used for holding intermediate as well as final results.

4.2: Divide the current 512 bit block into 16 sub-blocks of 32 bit each.

4.3: Now, we have 4 rounds. Each round consists of 20 steps. The inputs to each round are

a. Current 512 bit block
b. Register abcde
c. Some constants, designated as K[t].

It then updates the content of abcde using the SHA algorithm steps.

We had 64 constants defined as t in MD5. Here we have only 4 constants defined for K[t], one used in each of the 4 rounds.

Round	Values of t between	K[t] in hexadecimal
1	1 & 19	5A 92 79 99
2	20 & 39	6E D9 EB A1
3	40 & 59	9F 1B BC DC
4	60 & 79	CA 62 C1 D6

Table 4.1 Values of K[t]

4.4: SHA consists of 4 rounds, each round containing 20 iterations. [A total of 80 iterations] Mathematically, iteration consists of the following operations-

abcde = (e + Process F + s^5 (a) + W[t] + K[t]), a, s^{30} (b), c, d

where

abcde = register made up of 5 variables

Process F = logical operation (defined below)

s = Circular shift left of 32 bit sub-block by t bits

W[t] = A 32 bit constant derived from the current 32 bit block

K[t] = One of the 4 additive constant

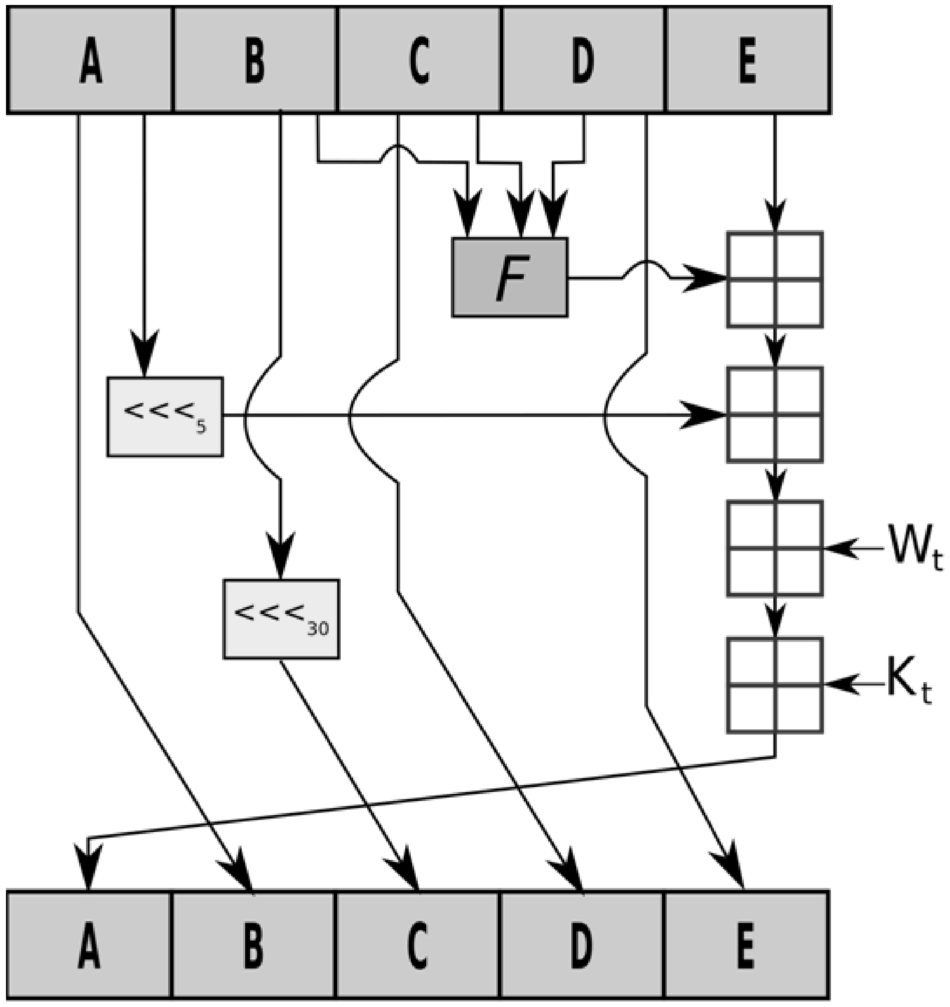


Fig 4.2 One Round of SHA-1

Process F in each SHA-1 round

Round	**Process F**
1	(b AND c) OR ((NOT b) AND (d))
2	b XOR c XOR d
3	(b AND c) OR (b AND d) OR (c AND d)
4	b XOR c XOR d

Step 5. Output

The combination of A,B,C,D & E taken together will be the hash digest. That is, we start with the low-order byte of A, and end with the high-order byte of E. This completes the description of SHA1.

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.

5.1 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.

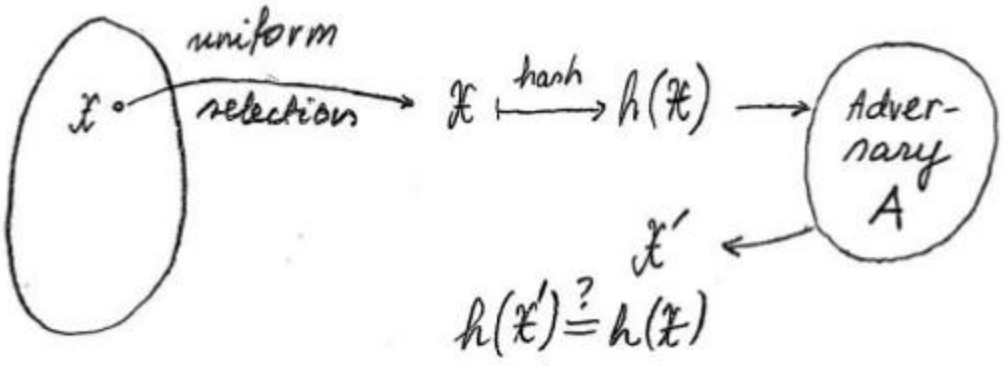


Fig 5.1 Example of Second Preimage Resistance

Fig. 5.1 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 (see Appendix 1).

- Given a person (& his birthday), identifying the second person with the same birthday corresponds to weak collision problem.
  (There is a fix X1, finding an another element X2 where h(X1) =h(X2), 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(X1) =h(X2), this instance corresponds to strong collision)

5.2 Proposed Hash Function

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

1. Declare the substitution table STable_1 containing prime numbers chosen randomly. This table is used in first transformation.

STable_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 STable_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[ ] = {2895, 2887, 2879, 2871, 2863, 2855, 2846, 2837, 2828, 2818, 2809, 2799, 2788, 2778, 2766, 2756, 2746, 2734, 2723, 2711, 2699, 2687, 2674, 2662, 2649, 2635, 2622, 2608, 2595, 2581, 2567, 2552, 2538, 2523, 2509, 2494, 2478, 2463, 2447, 2431, 2416, 2401, 2386, 2370, 2354, 2339, 3189, 3173, 3158, 3142, 3127, 2323, 2255, 2240, 2225, 2209, 2195, 2180, 2164, 2149, 2134, 2120, 2105, 2090, 2075, 2059, 2045};

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.

5.3 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) *hexnumberij* (*m*1) = *hexnumberij* (*m*2) for all i, j where i ϵ {0,1,…,t-1} and j ϵ {0,1,…7} and

ii) subblock_{ij} (*m*1) ≠ subblock_{ij} (*m*2) for all i, j where i ϵ {0,1,…,t-1} and j ϵ {0,1,…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) ≠ sub_block_{ij} (m2)

b) after_second_conv (m1) = after_second_conv (m2) and sub_ block_{ij} (m1) ≠ sub_ block_{ij} (m2)

c) after_third_conv (m1) = after_third_conv (m2) and sub_ block_{ij} (m1) ≠ 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 m1, if it is impossible to find another input message m2 where the hash output of first message is equal to the hash output of second input message m2 i.e., h (m1) = h (m2).

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.

Implementation

Network Simulator (Ver. 2), popularly known as NS2, is an 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. The basic architecture of NS2 is shown in Figure 6.1.

NS2 provides executable command ns (object) which take a Tcl simulation scripting file as an input argument.

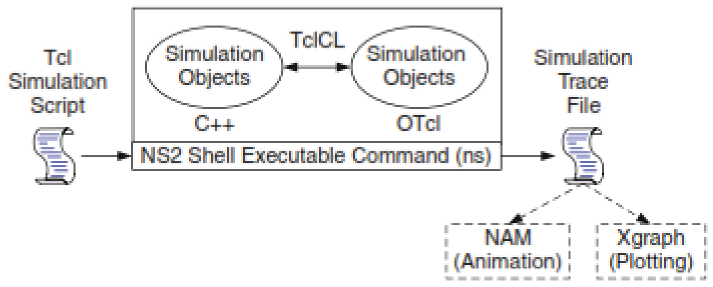


Fig 6.1 Basic Architecture of NS-2

6.1 Installation of NS2

Unix/Linux based systems are the primary operating systems in which NS2 can be installed easily. NS2 is also available on Windows platform which requires Cygwin software installed as prerequisites on the machine. If anyone has both the options to work on a Linux based or a Windows based system, it is recommended to go for Linux based system. A set of protocols comes with NS2 and are used in different kind of networks. The current version used is 2.3.5 which comes with a total of 72 various protocols.

6.2 Features of NS2

NS2 is an object oriented discrete event simulator, written in C++ language. The frontend of this tool is an OTcl interpreter. NS2 follows a class hierarchy in C++ language which is called the compiled hierarchy. A corresponding class hierarchy within OTcl interpreter is also followed which is called as interpreted hierarchy. There is a one-to-one correspondence between both the hierarchies from user's perspective. New simulator objects are constructed by the users through the NS2 interpreter.

- NS2 is an object oriented discrete event simulator
  – Being a discrete event Simulator it executes one event after another.
  – No locking or race around condition occurs as there is a single thread of control to execute the events.

- Protocols are mostly implemented using C++ language at back end.

- Most of NS2 source code is in C++ language.

- TCL language is used to write the scripts.

- Protocols implemented in NS2
  – Implementation of Drop tail queue and Interface queue.
  – Implementation of TCP, UDP, Transport layer and traffic agents.

- Scalability
  –To simulate a large network the processing at packet level must be fast enough to produce faster result.

- New objects can be added as NS2 is easily extensible.

6.3 Design and Implementation

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.

6.3.1 ***Approach***

Within the derived class encryption and decryption function for data field in the data packet are implemented. Message digests are calculated using MD5, SHA1 and the proposed cryptographic algorithm. These hash digests are used to ensure the integrity of data packet during transmission. Following the concepts of Object Oriented programming, much of the implementation is done using the C++ programming language.

Some of the implemented procedures are as follows:

- A Packet structure is created to carry the payload.
- A new class is inherited from the Agent class of NS2 to process the new packet format.
- Processing includes:

  - Message digests generation function

  - Encrypting function.

  - Decrypting function.

NS-2 packet types are considered as a simple byte arrays and handled accordingly. The instances of Packet class are created to carry the information from one node to other. A sample code is follows to add a new packet type in NS2-

```
struct hdr_Secure
{
char ret_val;                          //return value
double s_time;                         //send time
double r_time;                         //receive time
int sequence;                          //sequence number
char data[128];                        //data of 128 character long string
unsigned char hashvalue[16];           // hashvalue for 16 byte hex for md5
```

```
 //Access methods of Packet Header
 static int offset_;
 inline static int& offeset()
              {
                                          return offset_;
              }

 inline static hdr_Secure* access(const Packet* pac)
              {
                return (hdr_Secure*)pac->access(offset_);
              }
```

Following is a piece of code from .cc file to show the static binding of the packet.

```
static class SecureHeaderClass: public PacketHeaderClass
{
public:
SecureHeaderClass():PacketHeaderClass("PacketHeader/Secure", sizeof(hdr_Secure))
{
bind_offset(&hdr_Secure::offset_);      //linkage of variable
}
}class_Securehdr;    //End of the static class SecureHeader
```

Agents are defined to provide an interface with the nodes where network-layer packets are constructed or consumed. Agents are used in the implementation of protocols at various layers. The class Agent has an implementation partly in OTcl and partly in C++.

To create a new agent, one has to do define the following:

1. inheritance structure of the code and create the appropriate classes,

2. recv( ) and command( ) function,

3. OTcl linkage functions,

4. OTcl code to access the agent.

```
class SecureAgent: public Agent
{  ***********Declaration for MD5 starts******************/
   unsigned char data[64];         // md5 data
   unsigned long datalen;          // md5 data length
   unsigned long bitlen[2];        // md5 bit length
   unsigned long state[4];         // md5 4 initial variables
   unsigned char hash[16];         // md5 16 bytes for hex
   /*********Declaration for MD5 ends*********************/
public:
SecureAgent();
int seq;
int oneway;
/******** To give the control of our C++ object to OTcl***/
virtual int command(int argc,const char* const* argv);
/******** recv function to reveive matching data*********/
virtual void recv(Packet*, Handler*);
/******** encrypt function to encrypt sending data*******/
void encrypt(char* );
/******** decrypt function to decrypt receiving data******/
void decrypt(char* );
/********Functions for md5*************************/
void md5_init();
void md5_transform();
void md5_update(const char* st, size_t len);
void md5_final();
void print_hash();
};
```

The command( ) function must perform the desired action. Two command line arguments are passed in this function. The argument argc keeps the total no. of arguments passed by the user.

Packet* allocpkt( ) allocate new packet. It is used to create packets when a node has something to send to other node. The different fields of a packet are also populated with values before it is released to other node.

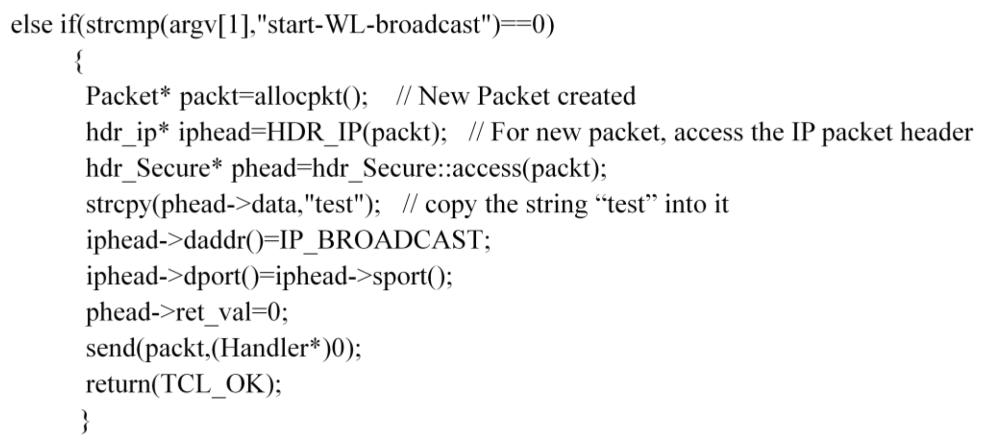
The argv arguments are used to collect the real arguments passed by the user: there are two types of command line arguments-

— The name of the command provided by the user in tcl script is collected by the parameter argv[0].

— The second command line argument argv contains an array of arguments specified by the user. dispatch_cmd() invokes the command method of Agent Class.

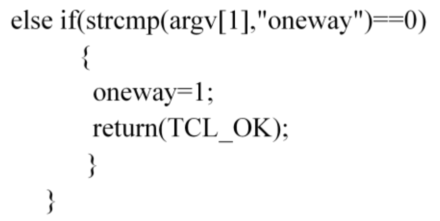
```
int SecureAgent::command(int argc, const char*const* argv)
{                                                   // chk for the input argument count
if(argc = =3)
  {
   if(strcmp(argv[1],"send") = =0)                  // send is command string used in tcl script
     {
      Packet* packt = allocpkt();                   // New Packet created
      hdr_Secure* header = hdr_Secure::access(packt);   // For new packet, access the Secure
                                                         // packet header
     // return field is initialized to 0, so the receiving node would generate Ack packet
      header -> ret_val = 0;
      header -> sequence = sequence++;                   // Sequence is incremented by 1
      header -> s_time = Scheduler::instance().clock();   // Note the Current time
     // the input string after "send" function in tcl script is argv[2], it is copied to header->data
     strcpy(header -> data, argv[2]);

      /**********************md5 work begins****************************/
      md5_init();                                            // md5 function
      md5_update(header ->data, strlen(header ->data));      // md5 function
      md5_final();                                           // md5 function
      /************************Ends***********************************/
      int indx;
      for (indx = 0; indx < 16; indx++)
          header -> hashvalue[indx] = hash[indx];
```

```
        printf("Sent message %s with hash value ",header->data);
        for (indx = 0; indx < 16; indx ++)
             printf("%02x ",hash[indx]);
             printf("\n");

        /********************** Encryption performed on sending data************/
        encrypt(header->data);

       /**********************Send the packet to other node******************/
        send(packt,(Handler*)0);
      //return TCL_OK, to inform the calling function that the command has been processed
        return(TCL_OK);
        }
```



```
  else if(strcmp(argv[1],"start-WL-broadcast")==0)
          {
            Packet* packt=allocpkt();    // New Packet created
            hdr_ip* iphead=HDR_IP(packt);   // For new packet, access the IP packet header
            hdr_Secure* phead=hdr_Secure::access(packt);
            strcpy(phead->data,"test");   // copy the string "test" into it
            iphead->daddr()=IP_BROADCAST;
            iphead->dport()=iphead->sport();
            phead->ret_val=0;
            send(packt,(Handler*)0);
            return(TCL_OK);
          }
```

```
// if the first argument is "oneway"
```



```
  else if(strcmp(argv[1],"oneway")==0)
          {
            oneway=1;
            return(TCL_OK);
          }
      }
```

```
 // command function of the base class is called, in case command is not processed by
    SecureAgent,
    return(Agent::command(argc,argv));
}
```

The recv( ) method is the primary entry point for an Agent which receives packets, and is invoked by upstream nodes while sending a packet. The direction of the packet received by the recv() method is identified. A callback handler is used for this purpose. If the packet is an information packet an acknowledgment is issued. If the received packet is an acknowledgement one, nothing is required to be done.

```
/********************Recv function***************************/
void SecureAgent::recv(Packet*packt, Handler*h)
{//received packet’s IP header and Secure packet header
hdr_ip* headip = hdr_ip::access(packt);
hdr_Secure* header = hdr_Secure::access(packt);

// If packet is received with return value as 0, a reply would be initiated
if(header -> ret_val = = 0)                          {
   double send_time = header -> s_time;      // copied old packet's send time
   char orig_data[128];
   char encrypt_data[128];
// copy the orig data of received packet into "encrypt data"
   strcpy(encrypt_data,header -> data);
   strcpy(orig_data, header -> data);
   int reciv_sequence = header -> sequence;
  // Display the received packet at this node
   unsigned char new_hash[16];
   char output[200];
   char authentic_reslt[50];
   decrypt(orig_data);
  /******************md5 work*****************************/
   md5_init();                                    // md5 function
   md5_update(orig_data,strlen(orig_data));  // md5 function
   md5_final();                                   // md5 function
   int indx;
   for (indx = 0; indx < 16; indx++)
         new_hash[indx] = hash[indx];
   indx = 0;
   while(indx < 16)
       {
        if(new_hash[indx] = = header -> hashvalue[indx])
          indx++;
        else
          break;
       }
   if(indx = = 16)
     {
       printf("\n Data Integrity Ensured,new_hash=");
// print the calculated hash
         for (int c = 0; c < 16; c++)
         printf("%02x ",new_hash[i]);

   strcpy(authentic_reslt,"Message_Accepted");
      }
   else {
   printf("\ndata modified, new_hash=");

       for (int c = 0; c < 16; c++)
         printf("%02x ",new_hash[i]);

   strcpy(authentic_reslt," Integrity Violation Error");
   }
  sprintf(output,"%s recv %d %3.1f %s %s %s",name(),headip->src_.addr_ >>
  Address::instance().NodeShift_[1], (Scheduler::instance().clock()-header->send_time)*1000,
  encrypt_data, orig_data, header->hashvalue);

  Tcl& tcl=Tcl::instance();
  tcl.eval(out);

  //Discard the packet
  Packet::free(packt);

/*********************Prepration for reply***********************/
Packet* packt_ret = allocpkt();  //Create a new Packet
//access the header for the new packet
hdr_Secure* header_ret = hdr_Secure::access(packt_ret);
header_ret -> ret_val = 1;
header_ret -> s_time = send_time;
header_ret -> r_time = Scheduler::instance().clock();
header_ret -> sequence = reciv_sequence;
strcpy(header_ret -> data,authentic_reslt);
      for (indx = 0; indx < 16; indx++)
         header_ret -> hashvalue[indx] = new_hash[indx];
send(packt_ret,0);  //send the packet
}
/******************Processing of Acknowledgement*************/
else
{
 char output[200];
```

```
sprintf(output,"%s recv %d %3.1f %s _ %s",name(),headip->src_.addr_>> Address
::instance().NodeShift_[1], (Scheduler::instance().clock()-header->send_time)*1000,       header-
>data, header->hashvalue);

Tcl&tcl=Tcl::instance();
tcl.eval(out);

//Discard the packet
Packet::free(pkt);
}
}
```

6.3.2. Adding a new Protocol in NS2

A set of steps are required for any procedure to be implemented. Similarly, for the protocol to be added in the simulation, a systematic set of steps will be required. All the following mentioned steps must be performed in the same order and not doing so will bring errors in the procedure.

a) Create a new header file Secure.h for the protocol.

b) Create Secure.cc file which contains the required code to execute the protocol.

c) Add the protocol ID to the common/packet.h file of NS2.

d) Edit class p_info() and static void initName() in packet.h.

e) Add the default value of the protocol to ns-default.tcl file.

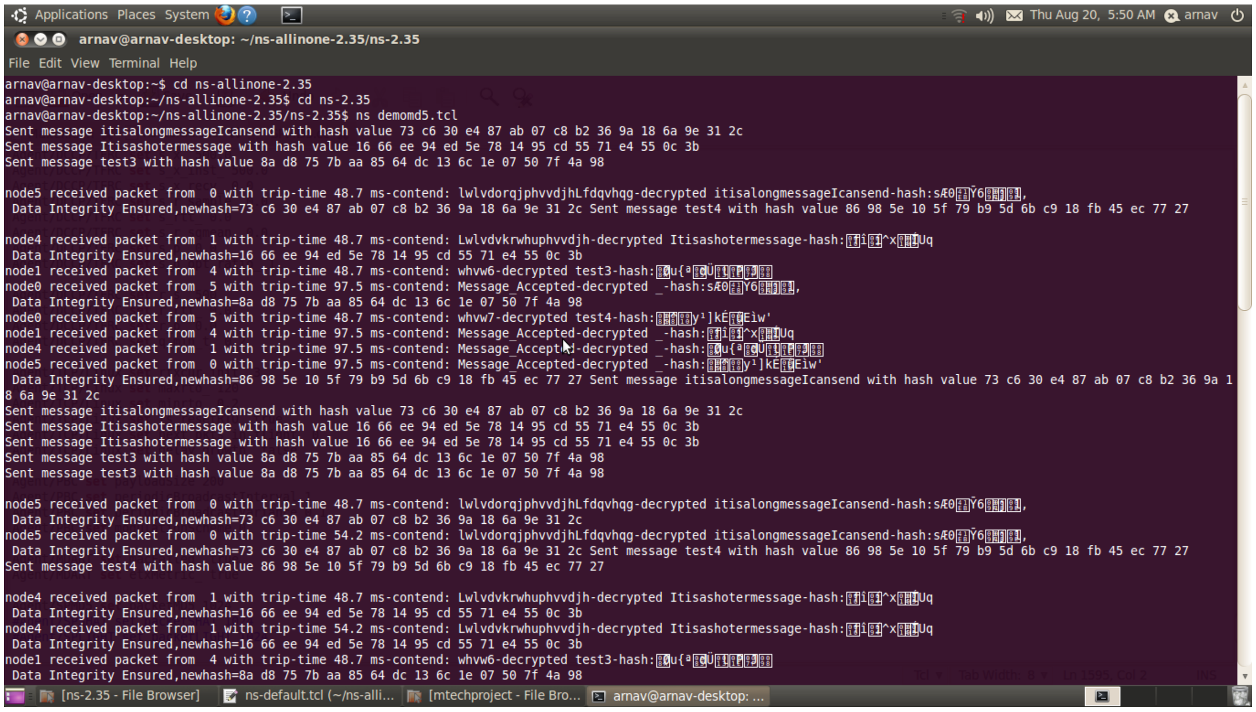
f) Add an entry for the new protocol packets under the section set protolist in the file ns-packet.tcl.

g) Add the object file Secure.o under the section OBJ_CC with other object files makefile.in file.

h) Recompile NS2 software

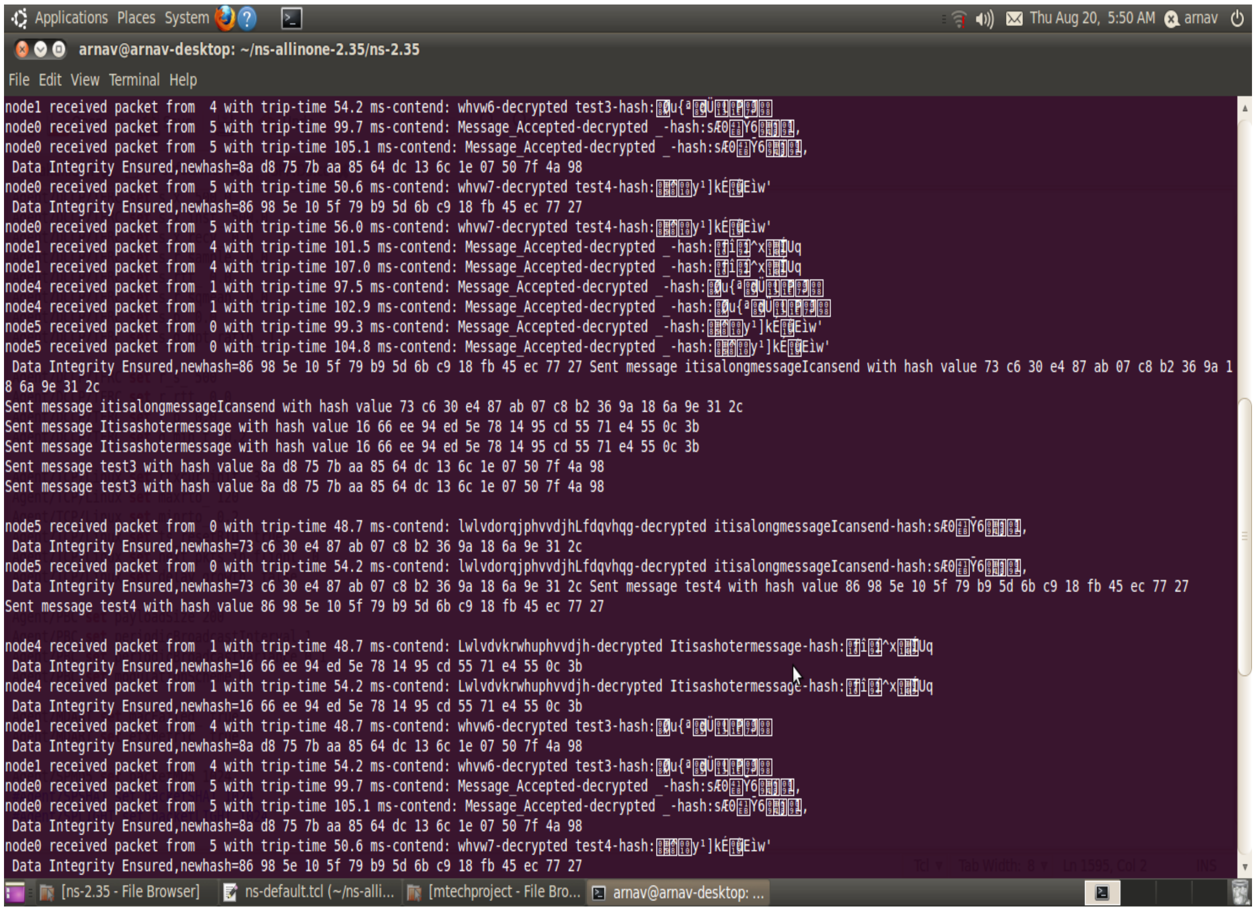
Demonstration and Results

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 7.1 to 7.3 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 these figures. The second set of figures Fig 7.4 to Fig 7.6 shows the communication among the nodes using SHA-1 hash algorithm whereas the set of figures Fig 7.7 and Fig 7.8 replicate the scenario using the proposed hash algorithm.


Applications Places System
Thu Aug 20, 5:50 AM arnav
arnav@arnav-desktop: ~/ns-allinone-2.35/ns-2.35
File Edit View Terminal Help
[ns-2.35 - File Browser]
ns-default.tcl (~/ns-alli...
[mtechproject - File Bro...
arnav@arnav-desktop: ...

```
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 Itisashotermessage with hash value 16 66 ee 94 ed 5e 78 14 95 cd 55 71 e4 55 0c 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-contend: lwlvdorqjphvvdjhLfdqvhqg-decrypted itisalongmessageIcansend-hash:sÆ0▯Ÿ6▯▯▯,
 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 5d 6b c9 18 fb 45 ec 77 27

node4 received packet from  1 with trip-time 48.7 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:▯î▯^x▯Uq
 Data Integrity Ensured,newhash=16 66 ee 94 ed 5e 78 14 95 cd 55 71 e4 55 0c 3b
node1 received packet from  4 with trip-time 48.7 ms-contend: whvw6-decrypted test3-hash:▯u{ª▯Ü▯▯▯
node0 received packet from  5 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:sÆ0▯Ÿ6▯▯▯,
 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-contend: whvw7-decrypted test4-hash:▯▯y¹]kÉ▯Eìw'
node1 received packet from  4 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:▯î▯^x▯Uq
node4 received packet from  1 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:▯u{ª▯Ü▯▯▯
node5 received packet from  0 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:▯▯y¹]kÉ▯Eìw'
 Data Integrity Ensured,newhash=86 98 5e 10 5f 79 b9 5d 6b c9 18 fb 45 ec 77 27 Sent message itisalongmessageIcansend with hash value 73 c6 30 e4 87 ab 07 c8 b2 36 9a 1
8 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 Itisashotermessage with hash value 16 66 ee 94 ed 5e 78 14 95 cd 55 71 e4 55 0c 3b
Sent message Itisashotermessage with hash value 16 66 ee 94 ed 5e 78 14 95 cd 55 71 e4 55 0c 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 trip-time 48.7 ms-contend: lwlvdorqjphvvdjhLfdqvhqg-decrypted itisalongmessageIcansend-hash:sÆ0▯Ÿ6▯▯▯,
 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-contend: lwlvdorqjphvvdjhLfdqvhqg-decrypted itisalongmessageIcansend-hash:sÆ0▯Ÿ6▯▯▯,
 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 5d 6b c9 18 fb 45 ec 77 27
Sent message test4 with hash value 86 98 5e 10 5f 79 b9 5d 6b c9 18 fb 45 ec 77 27

node4 received packet from  1 with trip-time 48.7 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:▯î▯^x▯Uq
 Data Integrity Ensured,newhash=16 66 ee 94 ed 5e 78 14 95 cd 55 71 e4 55 0c 3b
node4 received packet from  1 with trip-time 54.2 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:▯î▯^x▯Uq
 Data Integrity Ensured,newhash=16 66 ee 94 ed 5e 78 14 95 cd 55 71 e4 55 0c 3b
node1 received packet from  4 with trip-time 48.7 ms-contend: whvw6-decrypted test3-hash:▯u{ª▯Ü▯▯▯
 Data Integrity Ensured,newhash=8a d8 75 7b aa 85 64 dc 13 6c 1e 07 50 7f 4a 98
```

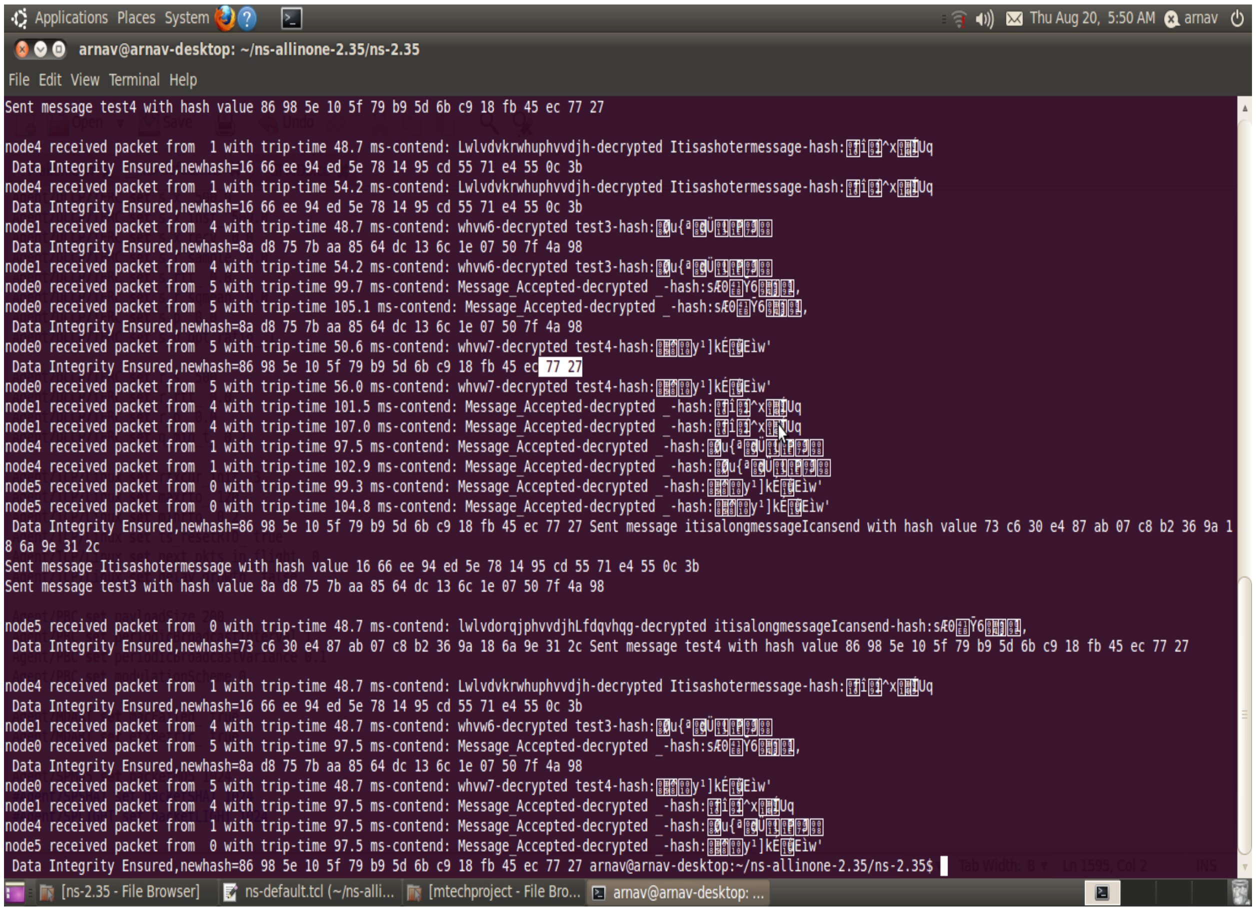
Fig 7.1 md5shot1



```
node1 received packet from  4 with trip-time 54.2 ms-contend: whvw6-decrypted test3-hash:▯u{ª▯Ü▯▯▯▯
node0 received packet from  5 with trip-time 99.7 ms-contend: Message_Accepted-decrypted _-hash:sÆ0▯Y6▯▯▯,
node0 received packet from  5 with trip-time 105.1 ms-contend: Message_Accepted-decrypted _-hash:sÆ0▯Y6▯▯▯,
 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 50.6 ms-contend: whvw7-decrypted test4-hash:▯▯y¹]kÉ▯Eìw'
 Data Integrity Ensured,newhash=86 98 5e 10 5f 79 b9 5d 6b c9 18 fb 45 ec 77 27
node0 received packet from  5 with trip-time 56.0 ms-contend: whvw7-decrypted test4-hash:▯▯y¹]kÉ▯Eìw'
node1 received packet from  4 with trip-time 101.5 ms-contend: Message_Accepted-decrypted _-hash:▯î▯^x▯Uq
node1 received packet from  4 with trip-time 107.0 ms-contend: Message_Accepted-decrypted _-hash:▯î▯^x▯Uq
node4 received packet from  1 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:▯u{ª▯Ü▯▯▯▯
node4 received packet from  1 with trip-time 102.9 ms-contend: Message_Accepted-decrypted _-hash:▯u{ª▯Ü▯▯▯▯
node5 received packet from  0 with trip-time 99.3 ms-contend: Message_Accepted-decrypted _-hash:▯▯y¹]kÉ▯Eìw'
node5 received packet from  0 with trip-time 104.8 ms-contend: Message_Accepted-decrypted _-hash:▯▯y¹]kÉ▯Eìw'
 Data Integrity Ensured,newhash=86 98 5e 10 5f 79 b9 5d 6b c9 18 fb 45 ec 77 27 Sent message itisalongmessageIcansend with hash value 73 c6 30 e4 87 ab 07 c8 b2 36 9a 1
8 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 Itisashotermessage with hash value 16 66 ee 94 ed 5e 78 14 95 cd 55 71 e4 55 0c 3b
Sent message Itisashotermessage with hash value 16 66 ee 94 ed 5e 78 14 95 cd 55 71 e4 55 0c 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 trip-time 48.7 ms-contend: lwlvdorqjphvvdjhLfdqvhqg-decrypted itisalongmessageIcansend-hash:sÆ0▯Y6▯▯▯,
 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-contend: lwlvdorqjphvvdjhLfdqvhqg-decrypted itisalongmessageIcansend-hash:sÆ0▯Y6▯▯▯,
 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 5d 6b c9 18 fb 45 ec 77 27
Sent message test4 with hash value 86 98 5e 10 5f 79 b9 5d 6b c9 18 fb 45 ec 77 27

node4 received packet from  1 with trip-time 48.7 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:▯î▯^x▯Uq
 Data Integrity Ensured,newhash=16 66 ee 94 ed 5e 78 14 95 cd 55 71 e4 55 0c 3b
node4 received packet from  1 with trip-time 54.2 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:▯î▯^x▯Uq
 Data Integrity Ensured,newhash=16 66 ee 94 ed 5e 78 14 95 cd 55 71 e4 55 0c 3b
node1 received packet from  4 with trip-time 48.7 ms-contend: whvw6-decrypted test3-hash:▯u{ª▯Ü▯▯▯▯
 Data Integrity Ensured,newhash=8a d8 75 7b aa 85 64 dc 13 6c 1e 07 50 7f 4a 98
node1 received packet from  4 with trip-time 54.2 ms-contend: whvw6-decrypted test3-hash:▯u{ª▯Ü▯▯▯▯
node0 received packet from  5 with trip-time 99.7 ms-contend: Message_Accepted-decrypted _-hash:sÆ0▯Y6▯▯▯,
node0 received packet from  5 with trip-time 105.1 ms-contend: Message_Accepted-decrypted _-hash:sÆ0▯Y6▯▯▯,
 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 50.6 ms-contend: whvw7-decrypted test4-hash:▯▯y¹]kÉ▯Eìw'
 Data Integrity Ensured,newhash=86 98 5e 10 5f 79 b9 5d 6b c9 18 fb 45 ec 77 27
```

Applications Places System
Thu Aug 20, 5:50 AM arnav
arnav@arnav-desktop: ~/ns-allinone-2.35/ns-2.35
File Edit View Terminal Help
[ns-2.35 - File Browser]
ns-default.tcl (~/ns-alli...
[mtechproject - File Bro...
arnav@arnav-desktop: ...

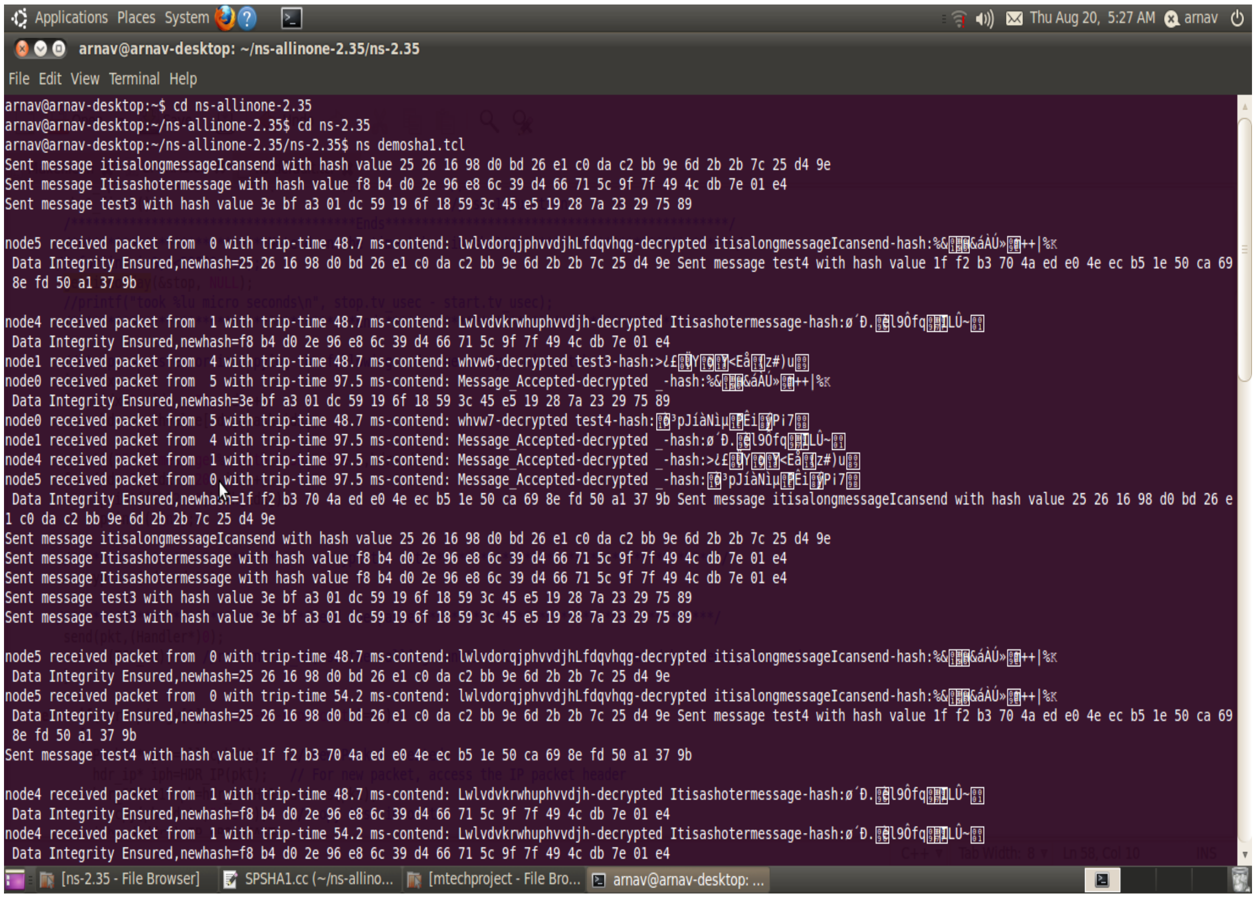
Fig 7.2 md5shot2



```
Sent message test4 with hash value 86 98 5e 10 5f 79 b9 5d 6b c9 18 fb 45 ec 77 27

node4 received packet from  1 with trip-time 48.7 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:□î□^x□Uq
 Data Integrity Ensured,newhash=16 66 ee 94 ed 5e 78 14 95 cd 55 71 e4 55 0c 3b
node4 received packet from  1 with trip-time 54.2 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:□î□^x□Uq
 Data Integrity Ensured,newhash=16 66 ee 94 ed 5e 78 14 95 cd 55 71 e4 55 0c 3b
node1 received packet from  4 with trip-time 48.7 ms-contend: whvw6-decrypted test3-hash:□u{ª□Ü□□□
 Data Integrity Ensured,newhash=8a d8 75 7b aa 85 64 dc 13 6c 1e 07 50 7f 4a 98
node1 received packet from  4 with trip-time 54.2 ms-contend: whvw6-decrypted test3-hash:□u{ª□Ü□□□
node0 received packet from  5 with trip-time 99.7 ms-contend: Message_Accepted-decrypted _-hash:sÆ0□Y6□□,
node0 received packet from  5 with trip-time 105.1 ms-contend: Message_Accepted-decrypted _-hash:sÆ0□Y6□□,
 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 50.6 ms-contend: whvw7-decrypted test4-hash:□□y¹]kÉ□Eìw'
 Data Integrity Ensured,newhash=86 98 5e 10 5f 79 b9 5d 6b c9 18 fb 45 ec 77 27
node0 received packet from  5 with trip-time 56.0 ms-contend: whvw7-decrypted test4-hash:□□y¹]kÉ□Eìw'
node1 received packet from  4 with trip-time 101.5 ms-contend: Message_Accepted-decrypted _-hash:□î□^x□Uq
node1 received packet from  4 with trip-time 107.0 ms-contend: Message_Accepted-decrypted _-hash:□î□^x□Uq
node4 received packet from  1 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:□u{ª□Ü□□□
node4 received packet from  1 with trip-time 102.9 ms-contend: Message_Accepted-decrypted _-hash:□u{ª□Ü□□□
node5 received packet from  0 with trip-time 99.3 ms-contend: Message_Accepted-decrypted _-hash:□□y¹]kÉ□Eìw'
node5 received packet from  0 with trip-time 104.8 ms-contend: Message_Accepted-decrypted _-hash:□□y¹]kÉ□Eìw'
 Data Integrity Ensured,newhash=86 98 5e 10 5f 79 b9 5d 6b c9 18 fb 45 ec 77 27 Sent message itisalongmessageIcansend with hash value 73 c6 30 e4 87 ab 07 c8 b2 36 9a 1
8 6a 9e 31 2c
Sent message Itisashotermessage with hash value 16 66 ee 94 ed 5e 78 14 95 cd 55 71 e4 55 0c 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-contend: lwlvdorqjphvvdjhLfdqvhqg-decrypted itisalongmessageIcansend-hash:sÆ0□Ŷ6□□,
 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 5d 6b c9 18 fb 45 ec 77 27

node4 received packet from  1 with trip-time 48.7 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:□î□^x□Uq
 Data Integrity Ensured,newhash=16 66 ee 94 ed 5e 78 14 95 cd 55 71 e4 55 0c 3b
node1 received packet from  4 with trip-time 48.7 ms-contend: whvw6-decrypted test3-hash:□u{ª□Ü□□□
node0 received packet from  5 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:sÆ0□Y6□□,
 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-contend: whvw7-decrypted test4-hash:□□y¹]kÉ□Eìw'
node1 received packet from  4 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:□î□^x□Uq
node4 received packet from  1 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:□u{ª□Ü□□□
node5 received packet from  0 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:□□y¹]kÉ□Eìw'
 Data Integrity Ensured,newhash=86 98 5e 10 5f 79 b9 5d 6b c9 18 fb 45 ec 77 27 arnav@arnav-desktop:~/ns-allinone-2.35/ns-2.35$
```

Applications Places System
Thu Aug 20, 5:50 AM arnav
arnav@arnav-desktop: ~/ns-allinone-2.35/ns-2.35
File Edit View Terminal Help
[ns-2.35 - File Browser]
ns-default.tcl (~/ns-alli...
[mtechproject - File Bro...
arnav@arnav-desktop: ...

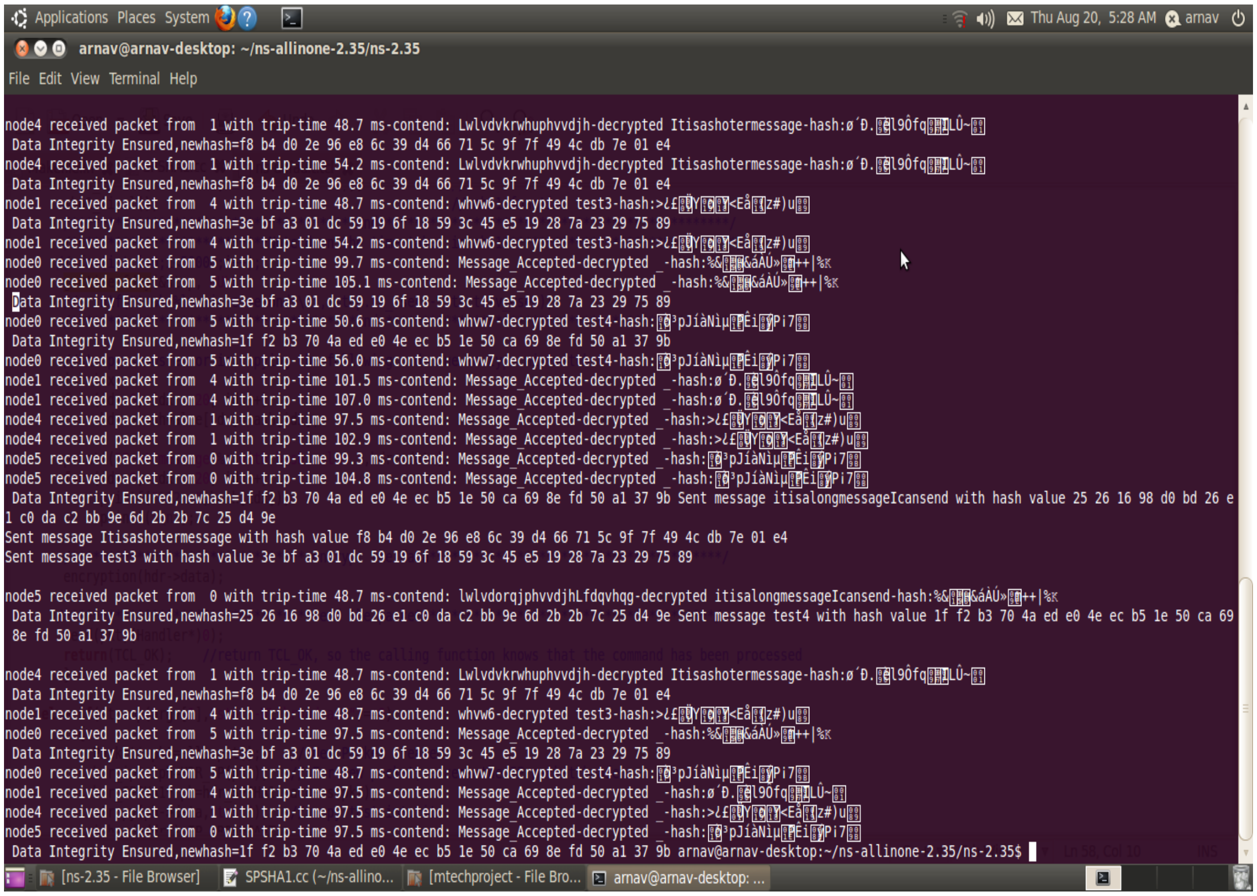
Fig 7.3 md5shot3



Applications Places System
Thu Aug 20, 5:27 AM arnav
arnav@arnav-desktop: ~/ns-allinone-2.35/ns-2.35
File Edit View Terminal Help
[ns-2.35 - File Browser]
SPSHA1.cc (~/ns-allino...
[mtechproject - File Bro...
arnav@arnav-desktop: ...

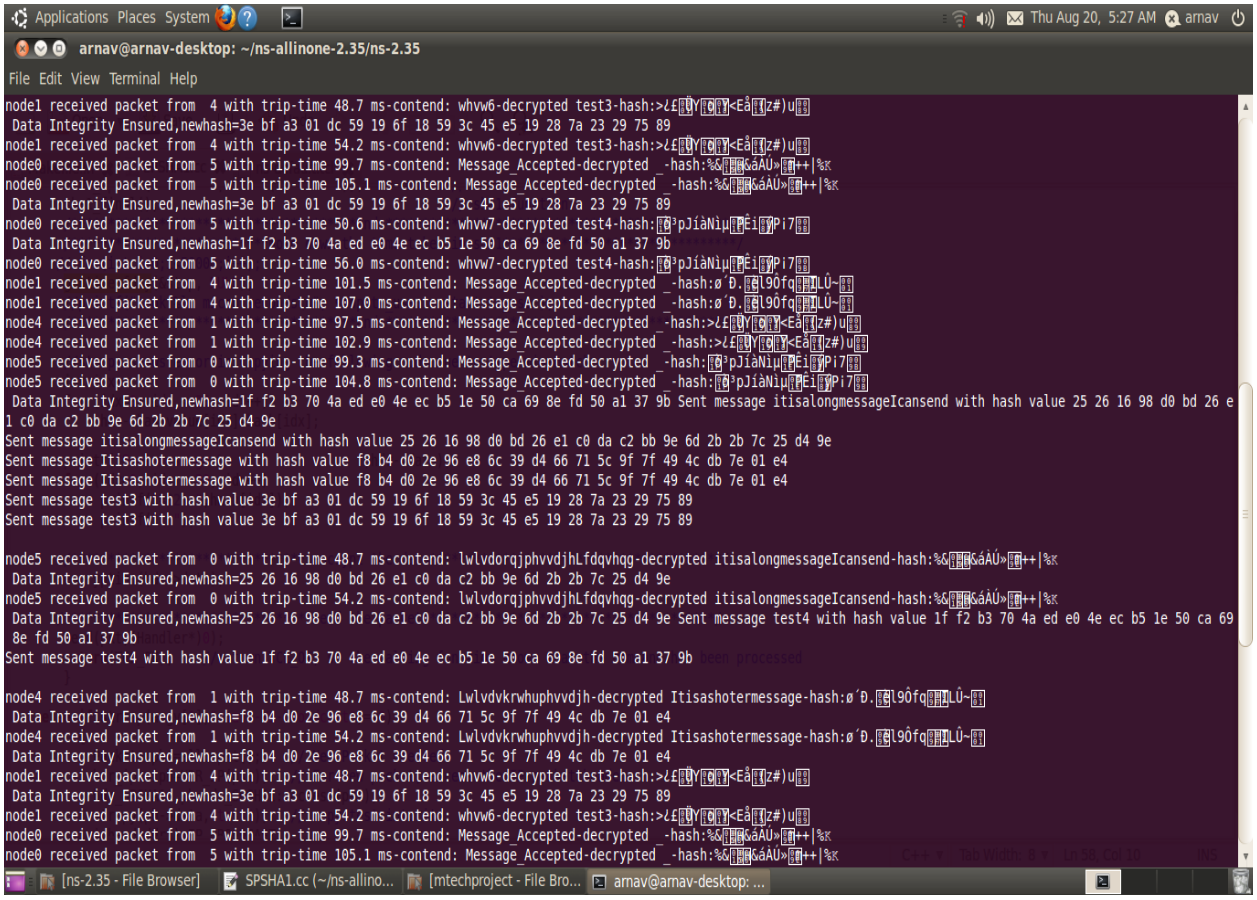
```
arnav@arnav-desktop:~$ cd ns-allinone-2.35
arnav@arnav-desktop:~/ns-allinone-2.35$ cd ns-2.35
arnav@arnav-desktop:~/ns-allinone-2.35/ns-2.35$ ns demosha1.tcl
Sent message itisalongmessageIcansend with hash value 25 26 16 98 d0 bd 26 e1 c0 da c2 bb 9e 6d 2b 2b 7c 25 d4 9e
Sent message Itisashotermessage with hash value f8 b4 d0 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 01 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  0 with trip-time 48.7 ms-contend: lwlvdorqjphvvdjhLfdqvhqg-decrypted itisalongmessageIcansend-hash:%&▯▯&áÀÚ»▯++|%K
 Data Integrity Ensured,newhash=25 26 16 98 d0 bd 26 e1 c0 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-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:ø´Ð.▯l9Ôfq▯LÛ~▯
 Data Integrity Ensured,newhash=f8 b4 d0 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 01 e4
node1 received packet from  4 with trip-time 48.7 ms-contend: whvw6-decrypted test3-hash:>¿£▯Y▯▯<Eå▯z#)u▯
node0 received packet from  5 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:%&▯▯&áÀÚ»▯++|%K
 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-contend: whvw7-decrypted test4-hash:▯³pJíàNìµ▯Êi▯Pi7▯
node1 received packet from  4 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:ø´Ð.▯l9Ôfq▯LÛ~▯
node4 received packet from  1 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:>¿£▯Y▯▯<Eå▯z#)u▯
node5 received packet from  0 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:▯³pJíàNìµ▯Êi▯Pi7▯
 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 d0 bd 26 e
1 c0 da c2 bb 9e 6d 2b 2b 7c 25 d4 9e
Sent message itisalongmessageIcansend with hash value 25 26 16 98 d0 bd 26 e1 c0 da c2 bb 9e 6d 2b 2b 7c 25 d4 9e
Sent message Itisashotermessage with hash value f8 b4 d0 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 01 e4
Sent message Itisashotermessage with hash value f8 b4 d0 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 01 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  0 with trip-time 48.7 ms-contend: lwlvdorqjphvvdjhLfdqvhqg-decrypted itisalongmessageIcansend-hash:%&▯▯&áÀÚ»▯++|%K
 Data Integrity Ensured,newhash=25 26 16 98 d0 bd 26 e1 c0 da c2 bb 9e 6d 2b 2b 7c 25 d4 9e
node5 received packet from  0 with trip-time 54.2 ms-contend: lwlvdorqjphvvdjhLfdqvhqg-decrypted itisalongmessageIcansend-hash:%&▯▯&áÀÚ»▯++|%K
 Data Integrity Ensured,newhash=25 26 16 98 d0 bd 26 e1 c0 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-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:ø´Ð.▯l9Ôfq▯LÛ~▯
 Data Integrity Ensured,newhash=f8 b4 d0 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 01 e4
node4 received packet from  1 with trip-time 54.2 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:ø´Ð.▯l9Ôfq▯LÛ~▯
 Data Integrity Ensured,newhash=f8 b4 d0 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 01 e4
```

Fig 7.4 shashot1


Applications Places System
Thu Aug 20, 5:28 AM arnav
arnav@arnav-desktop: ~/ns-allinone-2.35/ns-2.35
File Edit View Terminal Help
[ns-2.35 - File Browser]
SPSHA1.cc (~/ns-allino...
[mtechproject - File Bro...
arnav@arnav-desktop: ...

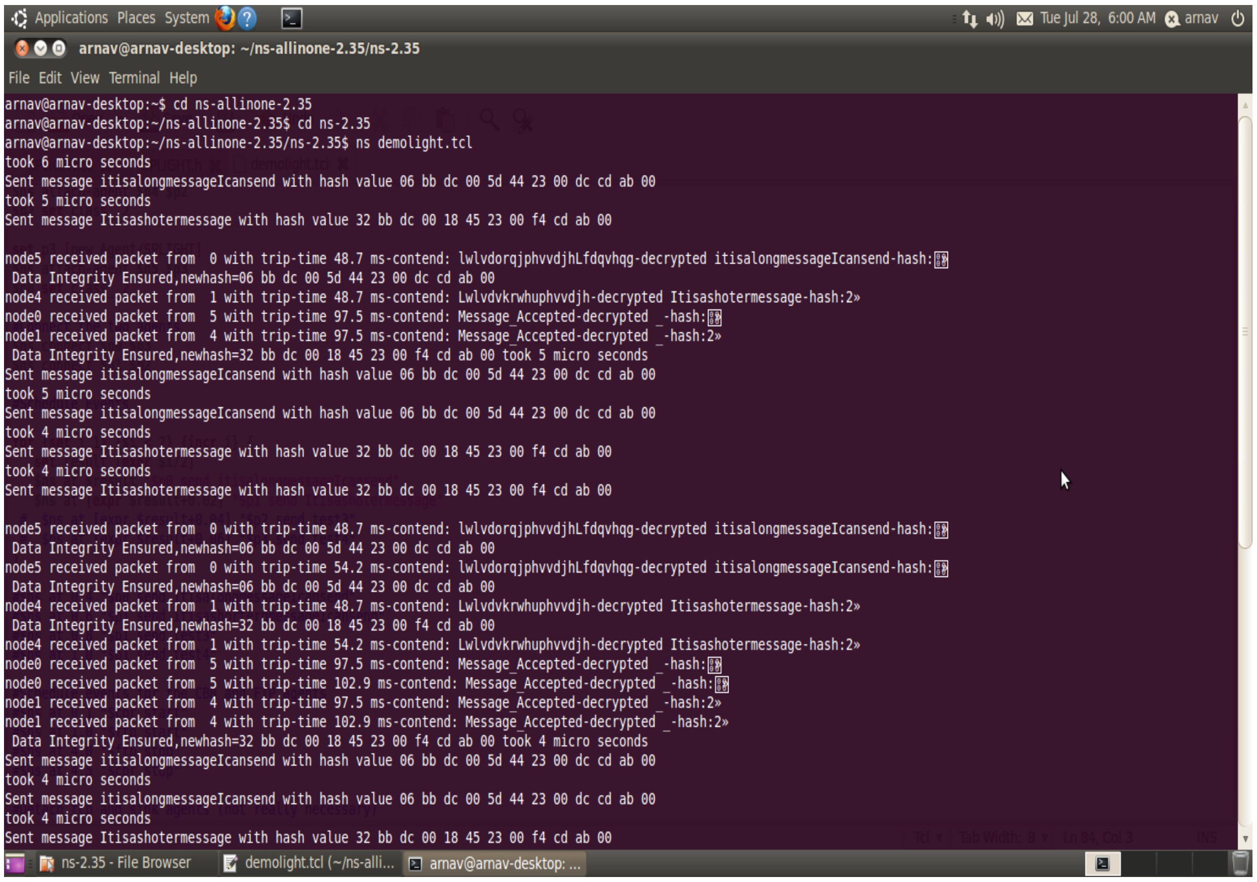
```
node4 received packet from  1 with trip-time 48.7 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:ø´Đ.▯l9Ôfq▯▯LÛ~▯
 Data Integrity Ensured,newhash=f8 b4 d0 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 01 e4
node4 received packet from  1 with trip-time 54.2 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:ø´Đ.▯l9Ôfq▯▯LÛ~▯
 Data Integrity Ensured,newhash=f8 b4 d0 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 01 e4
node1 received packet from  4 with trip-time 48.7 ms-contend: whvw6-decrypted test3-hash:>¿£▯Y▯▯<Eå▯z#)u▯
 Data Integrity Ensured,newhash=3e bf a3 01 dc 59 19 6f 18 59 3c 45 e5 19 28 7a 23 29 75 89
node1 received packet from  4 with trip-time 54.2 ms-contend: whvw6-decrypted test3-hash:>¿£▯Y▯▯<Eå▯z#)u▯
node0 received packet from  5 with trip-time 99.7 ms-contend: Message_Accepted-decrypted _-hash:%&▯▯&áAÚ»▯++|%
node0 received packet from  5 with trip-time 105.1 ms-contend: Message_Accepted-decrypted _-hash:%&▯▯&áAÚ»▯++|%
 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 50.6 ms-contend: whvw7-decrypted test4-hash:▯³pJíàNìµ▯Êi▯Pi7▯
 Data Integrity Ensured,newhash=1f f2 b3 70 4a ed e0 4e ec b5 1e 50 ca 69 8e fd 50 a1 37 9b
node0 received packet from  5 with trip-time 56.0 ms-contend: whvw7-decrypted test4-hash:▯³pJíàNìµ▯Êi▯Pi7▯
node1 received packet from  4 with trip-time 101.5 ms-contend: Message_Accepted-decrypted _-hash:ø´Đ.▯l9Ôfq▯▯LÛ~▯
node1 received packet from  4 with trip-time 107.0 ms-contend: Message_Accepted-decrypted _-hash:ø´Đ.▯l9Ôfq▯▯LÛ~▯
node4 received packet from  1 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:>¿£▯Y▯▯<Eå▯z#)u▯
node4 received packet from  1 with trip-time 102.9 ms-contend: Message_Accepted-decrypted _-hash:>¿£▯Y▯▯<Eå▯z#)u▯
node5 received packet from  0 with trip-time 99.3 ms-contend: Message_Accepted-decrypted _-hash:▯³pJíàNìµ▯Êi▯Pi7▯
node5 received packet from  0 with trip-time 104.8 ms-contend: Message_Accepted-decrypted _-hash:▯³pJíàNìµ▯Êi▯Pi7▯
 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 d0 bd 26 e
1 c0 da c2 bb 9e 6d 2b 2b 7c 25 d4 9e
Sent message Itisashotermessage with hash value f8 b4 d0 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 01 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  0 with trip-time 48.7 ms-contend: lwlvdorqjphvvdjhLfdqvhqg-decrypted itisalongmessageIcansend-hash:%&▯▯&áAÚ»▯++|%
 Data Integrity Ensured,newhash=25 26 16 98 d0 bd 26 e1 c0 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-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:ø´Đ.▯l9Ôfq▯▯LÛ~▯
 Data Integrity Ensured,newhash=f8 b4 d0 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 01 e4
node1 received packet from  4 with trip-time 48.7 ms-contend: whvw6-decrypted test3-hash:>¿£▯Y▯▯<Eå▯z#)u▯
node0 received packet from  5 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:%&▯▯&áAÚ»▯++|%
 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-contend: whvw7-decrypted test4-hash:▯³pJíàNìµ▯Êi▯Pi7▯
node1 received packet from  4 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:ø´Đ.▯l9Ôfq▯▯LÛ~▯
node4 received packet from  1 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:>¿£▯Y▯▯<Eå▯z#)u▯
node5 received packet from  0 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:▯³pJíàNìµ▯Êi▯Pi7▯
 Data Integrity Ensured,newhash=1f f2 b3 70 4a ed e0 4e ec b5 1e 50 ca 69 8e fd 50 a1 37 9b arnav@arnav-desktop:~/ns-allinone-2.35/ns-2.35$
```

Fig 7.5 shashot2



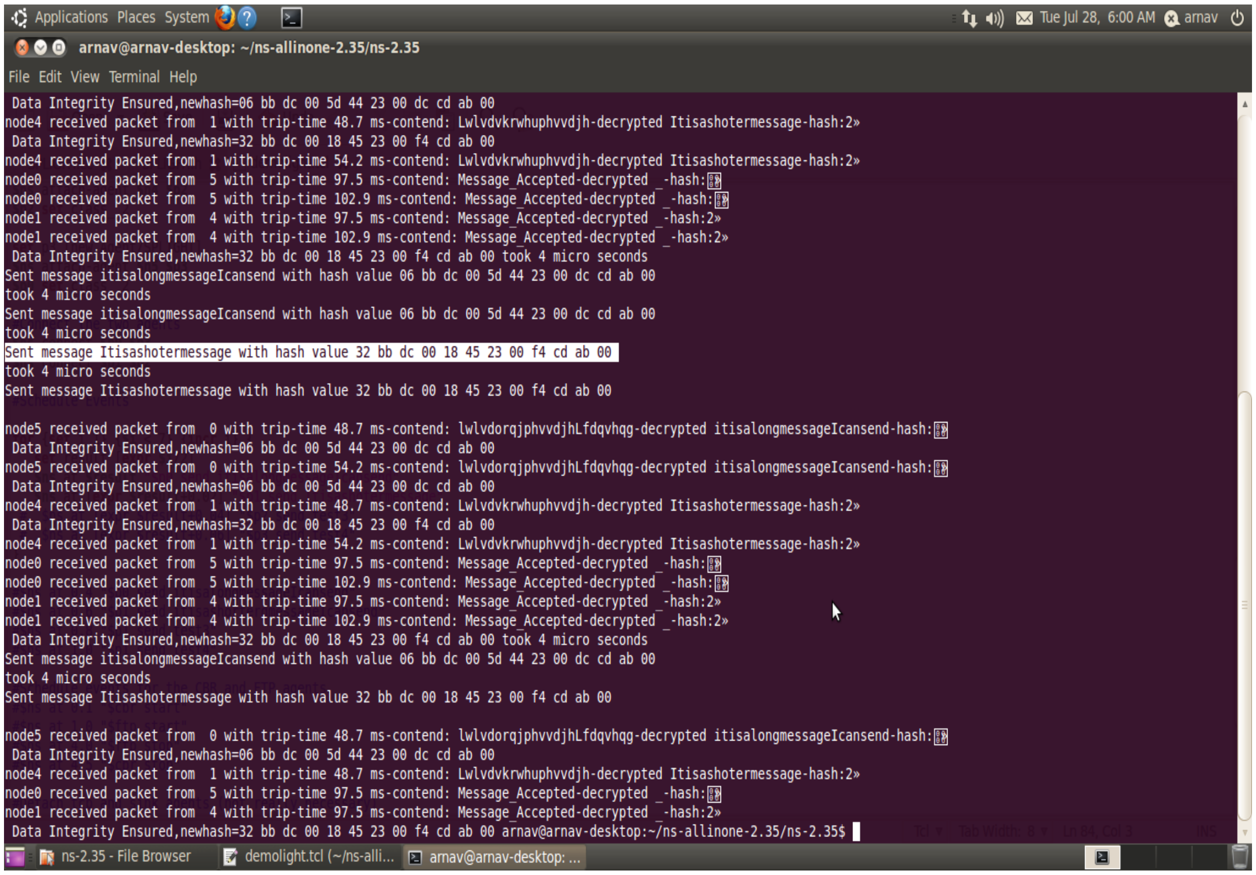
```
node1 received packet from  4 with trip-time 48.7 ms-contend: whvw6-decrypted test3-hash:>¿£□Y□□<Eå□z#)u□
 Data Integrity Ensured,newhash=3e bf a3 01 dc 59 19 6f 18 59 3c 45 e5 19 28 7a 23 29 75 89
node1 received packet from  4 with trip-time 54.2 ms-contend: whvw6-decrypted test3-hash:>¿£□Y□□<Eå□z#)u□
node0 received packet from  5 with trip-time 99.7 ms-contend: Message_Accepted-decrypted  -hash:%&□□&áAÚ»□++|%K
node0 received packet from  5 with trip-time 105.1 ms-contend: Message_Accepted-decrypted  -hash:%&□□&áAÚ»□++|%K
 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 50.6 ms-contend: whvw7-decrypted test4-hash:□³pJíàNìµ□Êi□Pi7□
 Data Integrity Ensured,newhash=1f f2 b3 70 4a ed e0 4e ec b5 1e 50 ca 69 8e fd 50 a1 37 9b
node0 received packet from  5 with trip-time 56.0 ms-contend: whvw7-decrypted test4-hash:□³pJíàNìµ□Êi□Pi7□
node1 received packet from  4 with trip-time 101.5 ms-contend: Message_Accepted-decrypted  -hash:ø´Ð.□l9Ôfq□LÛ~□
node1 received packet from  4 with trip-time 107.0 ms-contend: Message_Accepted-decrypted  -hash:ø´Ð.□l9Ôfq□LÛ~□
node4 received packet from  1 with trip-time 97.5 ms-contend: Message_Accepted-decrypted  -hash:>¿£□Y□□<Eå□z#)u□
node4 received packet from  1 with trip-time 102.9 ms-contend: Message_Accepted-decrypted  -hash:>¿£□Y□□<Eå□z#)u□
node5 received packet from  0 with trip-time 99.3 ms-contend: Message_Accepted-decrypted  -hash:□³pJíàNìµ□Êi□Pi7□
node5 received packet from  0 with trip-time 104.8 ms-contend: Message_Accepted-decrypted  -hash:□³pJíàNìµ□Êi□Pi7□
 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 d0 bd 26 e
1 c0 da c2 bb 9e 6d 2b 2b 7c 25 d4 9e
Sent message itisalongmessageIcansend with hash value 25 26 16 98 d0 bd 26 e1 c0 da c2 bb 9e 6d 2b 2b 7c 25 d4 9e
Sent message Itisashotermessage with hash value f8 b4 d0 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 01 e4
Sent message Itisashotermessage with hash value f8 b4 d0 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 01 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  0 with trip-time 48.7 ms-contend: lwlvdorqjphvvdjhLfdqvhqg-decrypted itisalongmessageIcansend-hash:%&□□&áAÚ»□++|%K
 Data Integrity Ensured,newhash=25 26 16 98 d0 bd 26 e1 c0 da c2 bb 9e 6d 2b 2b 7c 25 d4 9e
node5 received packet from  0 with trip-time 54.2 ms-contend: lwlvdorqjphvvdjhLfdqvhqg-decrypted itisalongmessageIcansend-hash:%&□□&áAÚ»□++|%K
 Data Integrity Ensured,newhash=25 26 16 98 d0 bd 26 e1 c0 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-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:ø´Ð.□l9Ôfq□LÛ~□
 Data Integrity Ensured,newhash=f8 b4 d0 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 01 e4
node4 received packet from  1 with trip-time 54.2 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:ø´Ð.□l9Ôfq□LÛ~□
 Data Integrity Ensured,newhash=f8 b4 d0 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 01 e4
node1 received packet from  4 with trip-time 48.7 ms-contend: whvw6-decrypted test3-hash:>¿£□Y□□<Eå□z#)u□
 Data Integrity Ensured,newhash=3e bf a3 01 dc 59 19 6f 18 59 3c 45 e5 19 28 7a 23 29 75 89
node1 received packet from  4 with trip-time 54.2 ms-contend: whvw6-decrypted test3-hash:>¿£□Y□□<Eå□z#)u□
node0 received packet from  5 with trip-time 99.7 ms-contend: Message_Accepted-decrypted  -hash:%&□□&áAÚ»□++|%K
node0 received packet from  5 with trip-time 105.1 ms-contend: Message_Accepted-decrypted  -hash:%&□□&áAÚ»□++|%K
```

Fig 7.6 shashot3


Applications Places System
arnav@arnav-desktop: ~/ns-allinone-2.35/ns-2.35
File Edit View Terminal Help
Tue Jul 28, 6:00 AM
arnav
ns-2.35 - File Browser
demolight.tcl (~/ns-alli...
arnav@arnav-desktop: ...

```
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 6 micro seconds
Sent message itisalongmessageIcansend with hash value 06 bb dc 00 5d 44 23 00 dc cd ab 00
took 5 micro seconds
Sent message Itisashotermessage with hash value 32 bb dc 00 18 45 23 00 f4 cd ab 00

node5 received packet from  0 with trip-time 48.7 ms-contend: lwlvdorqjphvvdjhLfdqvhqg-decrypted itisalongmessageIcansend-hash:▯
 Data Integrity Ensured,newhash=06 bb dc 00 5d 44 23 00 dc cd ab 00
node4 received packet from  1 with trip-time 48.7 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:2»
node0 received packet from  5 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:▯
node1 received packet from  4 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:2»
 Data Integrity Ensured,newhash=32 bb dc 00 18 45 23 00 f4 cd ab 00 took 5 micro seconds
Sent message itisalongmessageIcansend with hash value 06 bb dc 00 5d 44 23 00 dc cd ab 00
took 5 micro seconds
Sent message itisalongmessageIcansend with hash value 06 bb dc 00 5d 44 23 00 dc cd ab 00
took 4 micro seconds
Sent message Itisashotermessage with hash value 32 bb dc 00 18 45 23 00 f4 cd ab 00
took 4 micro seconds
Sent message Itisashotermessage with hash value 32 bb dc 00 18 45 23 00 f4 cd ab 00

node5 received packet from  0 with trip-time 48.7 ms-contend: lwlvdorqjphvvdjhLfdqvhqg-decrypted itisalongmessageIcansend-hash:▯
 Data Integrity Ensured,newhash=06 bb dc 00 5d 44 23 00 dc cd ab 00
node5 received packet from  0 with trip-time 54.2 ms-contend: lwlvdorqjphvvdjhLfdqvhqg-decrypted itisalongmessageIcansend-hash:▯
 Data Integrity Ensured,newhash=06 bb dc 00 5d 44 23 00 dc cd ab 00
node4 received packet from  1 with trip-time 48.7 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:2»
 Data Integrity Ensured,newhash=32 bb dc 00 18 45 23 00 f4 cd ab 00
node4 received packet from  1 with trip-time 54.2 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:2»
node0 received packet from  5 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:▯
node0 received packet from  5 with trip-time 102.9 ms-contend: Message_Accepted-decrypted _-hash:▯
node1 received packet from  4 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:2»
node1 received packet from  4 with trip-time 102.9 ms-contend: Message_Accepted-decrypted _-hash:2»
 Data Integrity Ensured,newhash=32 bb dc 00 18 45 23 00 f4 cd ab 00 took 4 micro seconds
Sent message itisalongmessageIcansend with hash value 06 bb dc 00 5d 44 23 00 dc cd ab 00
took 4 micro seconds
Sent message itisalongmessageIcansend with hash value 06 bb dc 00 5d 44 23 00 dc cd ab 00
took 4 micro seconds
Sent message Itisashotermessage with hash value 32 bb dc 00 18 45 23 00 f4 cd ab 00
```

Fig 7.7 lightshot1



```
 Data Integrity Ensured,newhash=06 bb dc 00 5d 44 23 00 dc cd ab 00
node4 received packet from  1 with trip-time 48.7 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:2»
 Data Integrity Ensured,newhash=32 bb dc 00 18 45 23 00 f4 cd ab 00
node4 received packet from  1 with trip-time 54.2 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:2»
node0 received packet from  5 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:▯
node0 received packet from  5 with trip-time 102.9 ms-contend: Message_Accepted-decrypted _-hash:▯
node1 received packet from  4 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:2»
node1 received packet from  4 with trip-time 102.9 ms-contend: Message_Accepted-decrypted _-hash:2»
 Data Integrity Ensured,newhash=32 bb dc 00 18 45 23 00 f4 cd ab 00 took 4 micro seconds
Sent message itisalongmessageIcansend with hash value 06 bb dc 00 5d 44 23 00 dc cd ab 00
took 4 micro seconds
Sent message itisalongmessageIcansend with hash value 06 bb dc 00 5d 44 23 00 dc cd ab 00
took 4 micro seconds
Sent message Itisashotermessage with hash value 32 bb dc 00 18 45 23 00 f4 cd ab 00
took 4 micro seconds
Sent message Itisashotermessage with hash value 32 bb dc 00 18 45 23 00 f4 cd ab 00

node5 received packet from  0 with trip-time 48.7 ms-contend: lwlvdorqjphvvdjhLfdqvhqg-decrypted itisalongmessageIcansend-hash:▯
 Data Integrity Ensured,newhash=06 bb dc 00 5d 44 23 00 dc cd ab 00
node5 received packet from  0 with trip-time 54.2 ms-contend: lwlvdorqjphvvdjhLfdqvhqg-decrypted itisalongmessageIcansend-hash:▯
 Data Integrity Ensured,newhash=06 bb dc 00 5d 44 23 00 dc cd ab 00
node4 received packet from  1 with trip-time 48.7 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:2»
 Data Integrity Ensured,newhash=32 bb dc 00 18 45 23 00 f4 cd ab 00
node4 received packet from  1 with trip-time 54.2 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:2»
node0 received packet from  5 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:▯
node0 received packet from  5 with trip-time 102.9 ms-contend: Message_Accepted-decrypted _-hash:▯
node1 received packet from  4 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:2»
node1 received packet from  4 with trip-time 102.9 ms-contend: Message_Accepted-decrypted _-hash:2»
 Data Integrity Ensured,newhash=32 bb dc 00 18 45 23 00 f4 cd ab 00 took 4 micro seconds
Sent message itisalongmessageIcansend with hash value 06 bb dc 00 5d 44 23 00 dc cd ab 00
took 4 micro seconds
Sent message Itisashotermessage with hash value 32 bb dc 00 18 45 23 00 f4 cd ab 00

node5 received packet from  0 with trip-time 48.7 ms-contend: lwlvdorqjphvvdjhLfdqvhqg-decrypted itisalongmessageIcansend-hash:▯
 Data Integrity Ensured,newhash=06 bb dc 00 5d 44 23 00 dc cd ab 00
node4 received packet from  1 with trip-time 48.7 ms-contend: Lwlvdvkrwhuphvvdjh-decrypted Itisashotermessage-hash:2»
node0 received packet from  5 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:▯
node1 received packet from  4 with trip-time 97.5 ms-contend: Message_Accepted-decrypted _-hash:2»
 Data Integrity Ensured,newhash=32 bb dc 00 18 45 23 00 f4 cd ab 00 arnav@arnav-desktop:~/ns-allinone-2.35/ns-2.35$
```

Applications Places System
Tue Jul 28, 6:00 AM arnav
arnav@arnav-desktop: ~/ns-allinone-2.35/ns-2.35
File Edit View Terminal Help
ns-2.35 - File Browser
demolight.tcl (~/ns-alli...
arnav@arnav-desktop: ...

Fig 7.8 lightshot2

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:

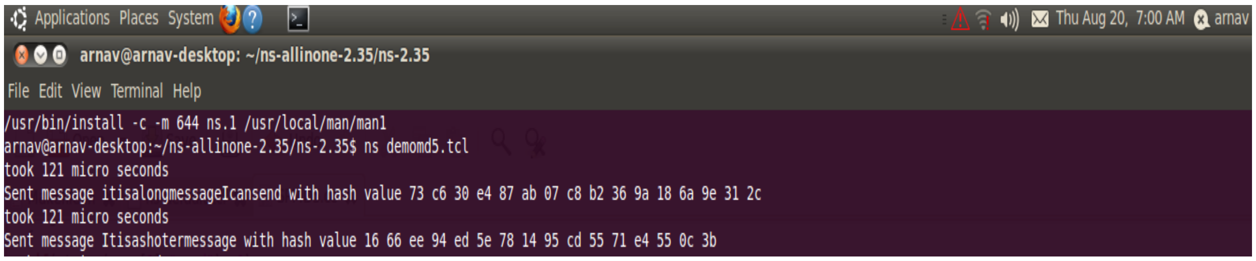
MD5 -- 16.25x128 (bit) = 16.25x16 (byte) = 260 µJ

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

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

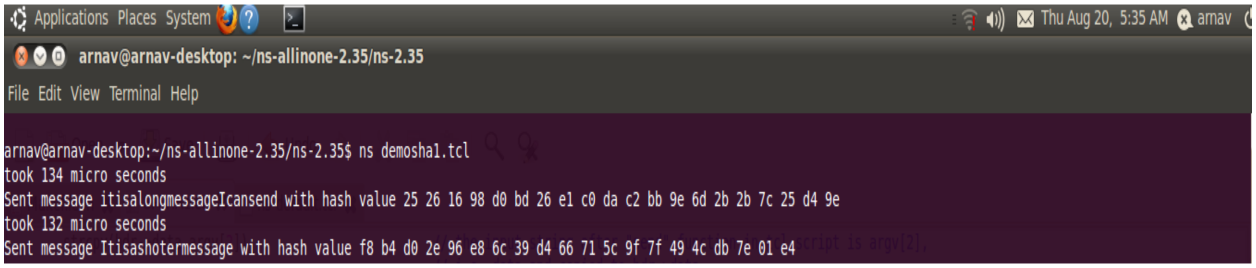
Comparison of Computation Speed:

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
Thu Aug 20, 7:00 AM
arnav
arnav@arnav-desktop: ~/ns-allinone-2.35/ns-2.35
File Edit View Terminal Help

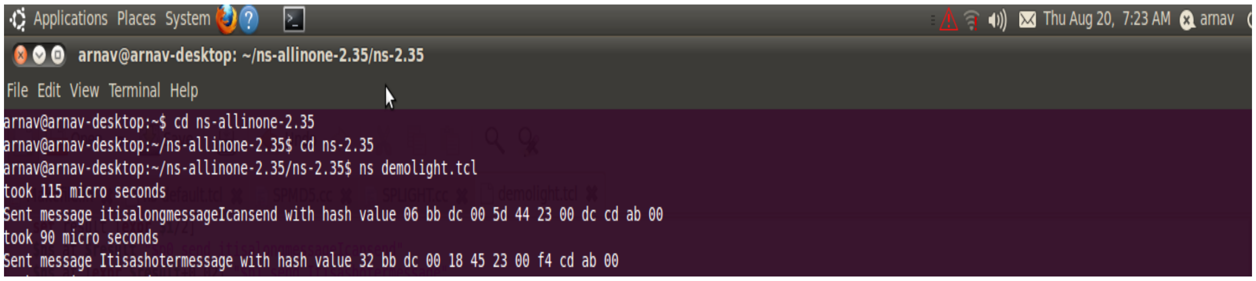
```
/usr/bin/install -c -m 644 ns.1 /usr/local/man/man1
arnav@arnav-desktop:~/ns-allinone-2.35/ns-2.35$ ns demomd5.tcl
took 121 micro seconds
Sent message itisalongmessageIcansend with hash value 73 c6 30 e4 87 ab 07 c8 b2 36 9a 18 6a 9e 31 2c
took 121 micro seconds
Sent message Itisashotermessage with hash value 16 66 ee 94 ed 5e 78 14 95 cd 55 71 e4 55 0c 3b
```

Fig 8.1 md5_time_shot


Applications Places System
Thu Aug 20, 5:35 AM arnav
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 demosha1.tcl
took 134 micro seconds
Sent message itisalongmessageIcansend with hash value 25 26 16 98 d0 bd 26 e1 c0 da c2 bb 9e 6d 2b 2b 7c 25 d4 9e
took 132 micro seconds
Sent message Itisashotermessage with hash value f8 b4 d0 2e 96 e8 6c 39 d4 66 71 5c 9f 7f 49 4c db 7e 01 e4
```

Fig 8.2 sha1_time_shot


Applications Places System
Thu Aug 20, 7:23 AM arnav
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 00 dc cd ab 00
took 90 micro seconds
Sent message Itisashotermessage with hash value 32 bb dc 00 18 45 23 00 f4 cd ab 00
```

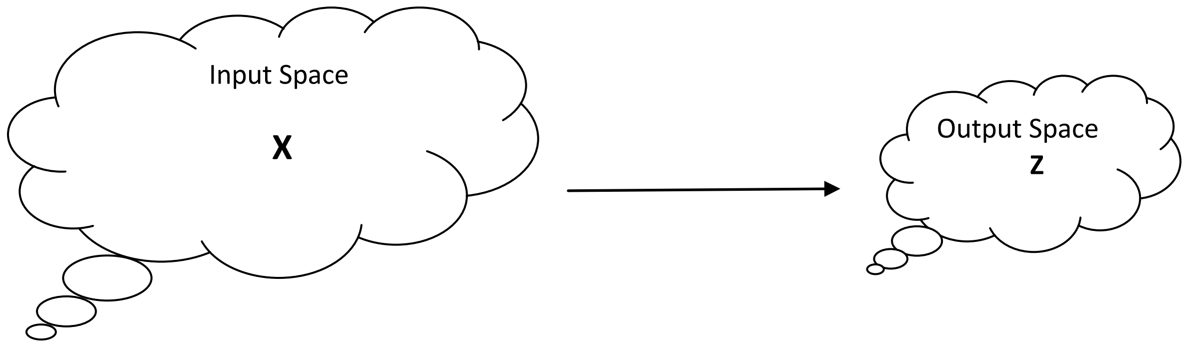
Fig 8.3 light_time_shot

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.

Appendix 1
Collision Attacks & the Birthday Paradox

"Collision attacks are much harder to prevent than 2nd preimage attacks"

Can we have hash functions without collision?



Input space is much bigger than output space. In SHA-1 the output space is 2^{160}. So whenever (2^{160} +1)th input value is considered, there will be a collision (assuming that first 2^{160} input values gives a unique hash output.

$$|x|>>|z| \implies \text{collision must exist}$$

Two more principles are also there-

***Dirichlet's Drawer Principle:** - 9 drawers are available for 10 pairs of socks, definitely there will be 1 collision.

*Pigeonhole Principle: - 19 holes for 20 pigeons are there. So there will be 1 hole in which 2 pigeons will be there.

We cannot get rid of the collision but the purpose is hard to find.

Birthday Paradox:-

If there are 23 or more people in a room, the odds are better than 50% that 2 of them **(any possible pair of 2 people)** will have the same birthday.

Let's calculate-

P (no collision between 2 people) = (1 – 1/365) [chance to collide=1/365]

P (no collision among 3 people) = (1 – 1/365) (1 – 2/365)

.

.

$$P(\text{no collision among } t \text{ people}) = \prod_{i=1}^{t-1} (1 - i/365)\ )$$

for t=23

$$P(\text{no collision among 23 people}) = \prod_{i=1}^{22} (1 - i/365) = 0.507 \approx 50\%$$

$$\approx \sqrt{365}$$

Another way to represent Birthday Paradox:-

If there are 23 people in a room, the odds are better than 50% that 2 of them will have the same birthday. Analyzing this parlor trick can give us some insight into hash cryptography.

Let's assume n inputs (Number of humans in birthday example) and k possible outputs (365 or 366). With n inputs there are n(n-1)/2 pairs of inputs.

Probability for each pair producing the same output= 1/k

About k/2 pairs will be required for the probability to be about 50%, that we will find a matching pair. That means if n is greater than $\sqrt{k}$ ($\sqrt{365}$), there is a good chance of matching pair.

Example:

With 23 people in a group there will be 253 pairs in total:

$$\frac{23 \cdot 22}{2} = 253$$

The chance of 2 people having different birthdays is:

$$1 - \frac{1}{365} = \frac{364}{365} = .997260$$

There's 364 out of 365 birthdays that are "OK".
Having all **253 pairs** be different is like getting heads 253 times in a row (sort-of)
let's assume birthdays are independent). We use exponents to find the probability:

$$\left(\frac{364}{365}\right)^{253} = .4995$$

99.7260% is really close to one, but when we multiply it by itself a few hundred times, it shrinks really fast. The chance that we have a match is: 1 – 49.95% = 50.05%, or just over half! If you want to find the probability of a match for any number of people n the formula is:

$$p(n) = 1 - \left(\frac{364}{365}\right)^{C(n,2)} = 1 - \left(\frac{364}{365}\right)^{n(n-1)/2}$$

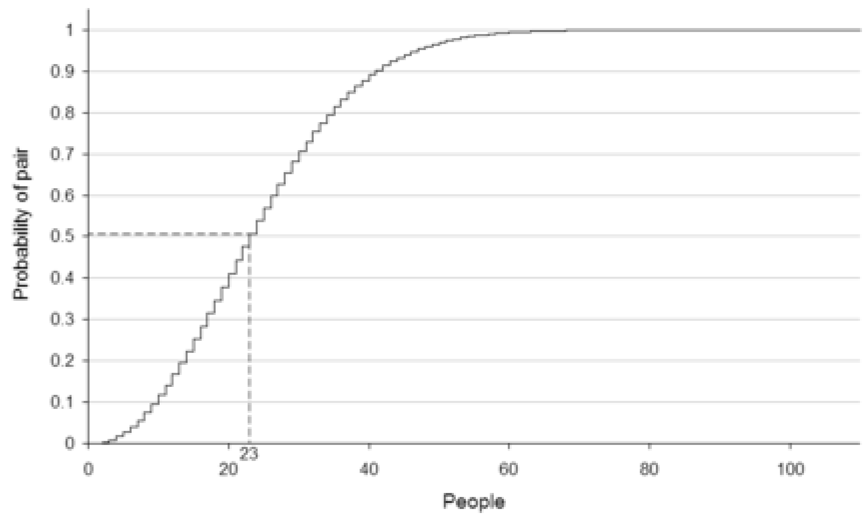
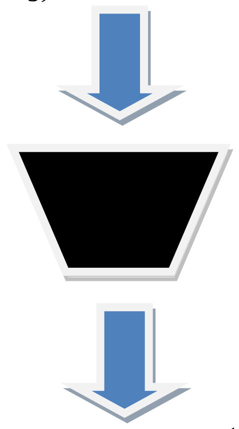


Fig A.1 Birthday Paradox

Birthday Paradox and Cryptographic Hash Functions

The mathematics of birthday paradox is used to guess a brute-force cryptographic attack against hash functions. This mathematical information may be utilised to speed up the looking up of hash digests stored in the database. As the hash space is limited in comparison of input space there can be many collisions of hashes. This is the how hash collision problem is similar to birthday paradox. The length of hash digest is fixed according to the algorithm used whereas the input text may be arbitrary long.

X_1, X_2, X_3,……………………..X_t (t strings)



2^n is the output space (n bit long)

$$P\text{ (no collision)} = \prod_{i=1}^{t-1} (1 - i/2^n)$$

(Output space is 2^n as it was 365 in previous example)

No. of inputs $t = 2^{(n+1)/2} \sqrt{\ln((1/(1-\lambda))}$

(λ = probability of at least 1 collision or likelihood of collision)

Example:

Suppose output space is 80 bits long and $\lambda = 0.5$

$t = 2^{(80+1)/2} \sqrt{\ln((1/(1-0.5))}$

$= 2^{(81)/2} \sqrt{\ln 2}$

$\approx 2^{40}$

(only 2^{40} input values are required to find a collision on an average)

Conclusion:

Keeping birthday paradox in mind, the input length must be almost half of the hash digest to avoid the hash collision. This discussion also concludes that the output length of almost all the hash functions is at least 128 bits long to protect from a second preimage attack. Most of the modern algorithms used to calculate hash digest produces longer outputs.

	Hash Output Length				
Λ	**128 bit**	**160 bit**	**256 bit**	**384 bit**	**512 bit**
0.5	2^{65}	2^{81}	2^{129}	2^{193}	2^{257}
0.9	2^{67}	2^{82}	2^{130}	2^{194}	2^{258}

Table1: Number of input hash values needed for a collision for different hash functions output length for 2 different collision likelihoods.

Note: λ does not seem to be very important as there is not a big difference in the values for probability of collision as 50% and 90%.

REFERENCES

[1] I. F. Akyildiz *et al.*, “A Survey on Sensor Setworks,” *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. Weger.: 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, Ventzislav 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 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