Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 32 (2014) 497 – 504

Procedia Computer Science

The 5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014)

LOCHA: A Light-Weight One-way Cryptographic Hash Algorithm for Wireless Sensor Network

Amrita Roy Chowdhury[a], Tanusree Chatterjee[b], Sipra DasBit[a] *

[a]*Dept. of Computer Sc.& Tech., Bengal Engineering &Science University,Shibpur,Howrah, India*
[b]*Dept. of Computer Sc. & Engineering, Regent Education & Research Foundation Group of Institutions, Barrackpore, Kolkata,India*

Abstract

Cryptographic hash functions are used to protect the authenticity of information. Some of the most popular and commonly used cryptographic hash algorithms are MD5, SHA1, RIPEMD. These hash algorithms are used in a wide variety of security applications e.g. securing node/message in traditional networks. However, the commonly used hash algorithms require huge computational overhead which is not affordable by applications in energy-starved network e.g. wireless sensor network (WSN). In these applications the major constraints are communication, computation and storage overheads; out of which communication and computation overheads consume high energy. Keeping this fact in mind, in this paper, a light-weight, one-way, cryptographic hash algorithm is developed with a target to produce a hash-digest with fixed and relatively small length for such an energy-starved wireless network. The primary focus is making the algorithm light-weight so that upon using it in application of network like WSN, the nodes can successfully run the algorithm with low energy. We claim the algorithm fulfils all the basic properties such as preimage resistance, collision resistance of a one-way unkeyed hash function. Finally the comparative usability of the hash algorithm in the said application domain is worked out and that shows the dominance of our scheme over two of the state-of-the-art hashing schemes.

Selection and Peer-review under responsibility of the Program Chairs.

*Keywords:*Cryptographic hash function;wireless sensor network; mote class attacker; avalanche effect

1. Introduction

Hash algorithms play an important role in modern cryptography. They are widely used in a variety of security applications such as node authentication[1, 2], message authentication[3], password protection[4], digital signature[5]

* Corresponding author. Tel.: +91-33-2668-5186; fax: +91-33-2668-5186.
E-mail address: sdasbit@yahoo.co.in

Selection and Peer-review under responsibility of the Program Chairs.
doi:10.1016/j.procs.2014.05.453

etc. The hash function uses a string of arbitrary length as its input and creates a fixed-length string as output. The fixed-length hash value is often called message digest. 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. Hash functions may be split into two classes[16]: *unkeyed hash functions*, whose specification dictates a single input parameter (a message); and *keyed hash functions*, whose specification dictates two distinct inputs, a message and a secret key. Two commonly used functions are MD5 and SHA-1. Both SHA-1 and MD5 are derived from MD4 which has been known[6] for its weaknesses. MD5 which uses a hash algorithm with 128-bit output has been designed in 1991 and in 2005 it was shown[7] how quickly random collisions for MD5 can be constructed. Also it is not suitable for applications that rely on the properties like SSL certificates or digital signatures. In [8] authors have shown that how a pair of X.509 certificates can be created that result in the same MD5 hash digest. Then cryptographers began recommending the use of other algorithms, such as SHA-1 which has since been found to be vulnerable[9] as well and most U.S. government applications now require the SHA-2 and SHA-3 family of hash functions[10, 11]. But most of these widely used hash functions are used in large conventional networks.

Unlike conventional network, in short-lived, energy-constrained network like WSNs, for many applications nodes are deployed[12] in outdoor environment without human monitoring and therefore, data authenticity and reliability becomes the main concern to deal with such kind of networks. Since WSN suffers from many constraints including low processing power, low battery life, small memory and wireless communication channel, it is not able to deal with traditional cryptographic algorithms. Due to these limitations, it becomes mandatory to devise lightweight security solutions for WSNs. Many works are reported so far towards hash-based security solutions and a few of these works[13, 14, 15] are briefed here. Here [13, 14] give solutions for WSNs whereas [15] is not typically meant for WSNs. However, to the best of our knowledge no attempt has been reported to make light weight hash algorithm customized for energy-starved network like WSN.

In one such work[13], authors have presented a hash-based signature scheme which can be used to authenticate messages both for unicast and broadcast communication. They have claimed that both the signature generation and verification are faster than other existing schemes e.g. ECDSA (elliptic curve digital signature algorithm). In security analysis, they have shown that the signature scheme is preimage and second preimage resistant**.** However, the authors have not claimed about collision resistance, the other important property.

The authors of [14] have designed an efficient and robust scheme against node capture attack using hash chain in WSN. The basic idea of the scheme is applying a hash function on the initial preloaded keys in the nodes before their deployment. The hashed key is used as communication key. The one-way property of hash function reduces the exposure of data due to node capture attack. It also shows an improvement compared to other existing schemes in terms of key exposal probability and resilience against node capture attack.

In [15] authors have presented a cryptographic hash function Whirlwind which can be efficiently implemented in software. It can be considered as an improvement in design compared to SHA3 hash functions. It employs large S-boxes which allow efficient and flexible implementations on a wide variety of platforms. The hash function produces 512-bit digest by incorporating a compression function, computing initialization vector etc. The digest is presented by an 8x4 array of 16-bit elements each. Though the cryptanalysis of the hash function shows an improvement but while implemented in software the performance is not very fast compared to the other competing schemes.

In this paper we aim to propose a light-weight one-way hash algorithm (LOCHA) which produces a hash-digest with fixed and relatively small length. The LOCHA satisfies all the properties of a one-way unkeyed hash functions. Moreover the algorithm is light-weight in nature thereby makes it suitable for energy-starved WSN.

The rest of the paper is organized as follows. Section 2 describes the proposed hashing scheme followed by the implementing algorithm. Comparative performance of the scheme including the justification of our claim about maintenance of basic cryptographic hash properties is evaluated in Section 3. We conclude our work along with the future scopes in section 4.

2. LOCHA: The proposed hashing scheme

In this section we describe the proposed light-weight hashing scheme along with the algorithm for generating the same. The target application may be securing (e.g. node authentication) energy-starved networks like WSN. The hashing scheme described here creates a relatively short-length, fixed hash digest from an input message of arbitrary

length. We assume that the input message consists of only those characters which belong to the set of the 96 printable ASCII characters (character code 32-127) only. The first 32 characters in the ASCII table are unprintable control codes used to control peripherals such as printers etc. and hence their exclusion is justified.

2.1. The Scheme

The input message is first preprocessed by converting it into binary representation of the respective ASCII codes of the constituent characters and employing unambiguous padding in the least significant position of the message to make it divisible by 512. Even if the length of the processed message is already a multiple of 512, an additional 512 0's are added to enhance the robustness of the algorithm. Now the preprocessed message is split 3 times in a nested manner where the first level nested split results in block size of 512-bit each. The second level and third level nested split result in block size of 64-bit and 8-bit respectively.

Subsequent to the above splits, 3 steps of transformations are employed on the input message. Firstly blocks of 512-bit are taken. Each of these 512-bit blocks is divided into 8, 64-bit blocks. Now, every 64-bit block is further subdivided into 8 numbers of 8-bit blocks. Substitution of each of these 8-bit inner most split blocks takes place at first by a prime number selected from a substitution table S-Table1. This table consists of 97 prime numbers (96 for the printable ASCII characters and 1 for padding; small prime numbers are chosen to reduce overheads). Now for every 64-bit block a number is computed using the aforementioned 8 substituted values and that creates the output of the first level transformation.

Once the first transformation is over, the second transformation takes place using another substitution table S-Table2 which consists of 67 prime numbers. Here, 67 numbers are chosen randomly to ensure uniformity in transformation and to reduce storage overhead. These values of S-Table2 are computed by the formula log(sin(index+128)), where index= 0,…,66. The third step transformation is done using the result of the first and second step transformations. The decimal result of the third step received from each 64-bit block, is then converted to the equivalent 3-digit hexadecimal number.

Now the 3-level swapping is applied to the 24-digit hexadecimal number of each 512-bit block. Due to lack of space the details of all the 3-level swapping are excluded here.

After all the swap functions are completed, we receive the final hash digest of 96 bits for each 512-bit message. All such hash digests for each of the 512-bit blocks are then added modular arithmetically to get the final hash digest for the complete message.

The two substitution tables viz. S-Table1 and S-Table2 are attached in the Appendix section. To keep the tables concise all the values of the tables are not shown.

2.2. Algorithm

Input: A message (X) of arbitrary length (consist of 96 printable ASCII characters only)
Output: A hash digest (H) of 96 bits

Initialize the followings-

i) H to 0 *// final hash digest*
ii) $1step\_conversion$ to 1 *// variable related to the 1st step conversion of the input*
iii) $2step\_conversion_{part3}$ to 7 *// variable related to the 2nd step conversion of the input*
iv) $hexdigit_{i(j-1)1}$ $hexdigit_{i(j-1)0}$ to 0 when j=0 *// hexadecimal digits*

Begin

1. Convert each of the character of X into the 8-bit binary *//* $X = X_0, X_1, ..., X_7$
2. Apply unambiguous padding
3. Obtain X' *// after appending 0's in the LSB of X to make it divisible by 512*
4. Split X' *// splitting in t number of 512-bit blocks*

5. for(i=0; i<t; i++) // *for t number of 512-bis block* $subblock_i$
6. for(j=0; j<=7; j++) // *for 8 number of 64-bit block* $subblock_{ij}$
7. $H_i = \emptyset$ //*variable storing the digest for* i^{th}*512-bit block is initialized as empty*
8. for(k=0; k<=7; k++) // *for 8 number of 8-bit block* $subblock_{ijk}$
9. Obtain $subblock_{ijk}$ // *result of Split X'*
10. if $subblock_{ijk}$ contains at least one 1
11. p_k =decimal($subblock_{ijk}$) – 31 // $p_k \in \{2, 3, ..., 97\}$
12. $subblock_{ijk}$ =S-Table1[p_k]
13. else // *when* $subblock_{ijk}$ *contains all 0's*
14. $subblock_{ijk}$ =S-Table1[1] // $p_k = 1$
15. end if
16. $1step\_conversion$ =S-Table1(p_k) x $1step\_conversion$
17. if($1step\_conversion > 2^{16} - 1$)
18. $1step\_conversion = 1step\_conversion \% 2^{16}$
19. end if
20. end for // *end of 8, 8-bit blocks*
21. Compute $2step\_conversion_{part1} = 1step\_conversion \% 67$
22. if ($subblock_{ijk}$ [1][7]==0) // $subblock_{ijk}$ *[1][7] denotes* 8^{th} *bit of* 2^{nd} *subblock* $subblock_{ij1}$
23. Compute $2step\_conversion_{part2} = 2step\_conversion_{part1}$
24. else
25. Compute $2step\_conversion_{part2} = 67 - 2step\_conversion_{part1}$
26. end if
27. Compute $2step\_conversion_{part3} = [2step\_conversion_{part3} + (2step\_conversion_{part1} + 1step\_conversion) \%256]$
28. Generate $after2step\_conversion = (1step\_conversion \% 2step\_conversion_{part3}) + 1step\_conversion$ + S-Table2 [$2step\_conversion_{part2}$]
29. Compute $after3step\_conversion = (after2step\_conversion \% 255) + p_2$ + decimal equivalent ($hexdigit_{i(j-1)1}$ $hexdigit_{i(j-1)0}$) + p_0 % 127 // p_0, p_2 *are* p_k *values when k=0, k=2 respectively*
30. Convert $after3step\_conversion$ to $hexnumber_{ij}$ // $hexnumber_{ij} = hexdigit_{ij0} hexdigit_{ij1} hexdigit_{ij2}$
31. Apply intra-hexnumber hexdigit swapping on each hexnumber // 1^{st} *level swapping on hexdigits of each 64-bit block*
32. Compute H_i = concat(H_i , $hexnumber_{ij}$)

33. Apply inter-hexnumber hexdigit swapping on hexdigits of j 64-bit blocks *// 2nd level swapping*
34. end for *// end of 8,64-bit blocks*
35. Compute $H_i = hexnumber_{i0} hexnumber_{i1} ... hexnumber_{i6} hexnumber_{i7}$
36. Apply inter-hexnumber swapping on H_i *// 3rd level swapping*
37. Compute final hash digest, H=H+ H_i *// ignoring the carry bits*
38. end for *// end of t, 512-bit blocks*

End

3. Performance Analysis

In this section primarily we focus on the strength of the proposed hash algorithm, LOCHA. Subsequently we provide comparative performance through qualitative analysis.

3.1. Strength of LOCHA

We evaluate the strength of the algorithm by showing the extent of maintaining the following basic cryptographic properties[16] of a one-way hash function. Breaking of the proposed hashing scheme requires breaking these cryptographic properties.

i. **Preimage Resistance** - For a given output (hash digest) H corresponding to a unknown input, it is computationally infeasible to find a input (message) m such that h(m) =H where h(m) is the hash digest of m. It represents the one-way property of a hash function.

ii. **Collision Resistance** - It is computationally infeasible to find any two distinct input (messages) m_1, m_2 which hash to the same output, such that $h(m_1) = h(m_2)$.

iii. **Second Preimage Resistance** - given an input m_1, it is computationally infeasible to find any second input m_2 which has the same output i.e., $h(m_1) = h(m_2)$.

We claim that the hash algorithm LOCHA maintains all the above basic properties. We also provide the justifications of our claims of maintaining preimage resistance, collision resistance and second preimage resistance. Due to space limitation the complete justification of Collision Resistance cannot be provided.

Preimage Resistance:

As demonstrated in the algorithm LOCHA, the final hash digest is the sum of t number of 24-digit hexadecimal numbers where the value of t depends on the size of the input. Let H be the final hash digest. So $H = \sum_{i=0}^{t-1} H_i$ (modular arithmetic) where H_i is the 24-digit (hexadecimal) hash digest from the $(i+1)^{th}$ 512-bit block, so there are ${}^{H+t-1}C_{t-1}$ possible ways to calculate H_i. For the simplest case, let us consider that the input size is only 512 bits i.e. t =1. Therefore, number of solutions for the above becomes 1. We also consider the mote class attacker which tries to break the hash digest. In the average case for the attacker, upon capturing the digest H, while it attempts to find a message m such that h(m) = H(given), all the possible numbers of tasks the attacker has to perform are as follows.

i. In the algorithm, 3-level swapping (steps 31, 33, 36) requires 2^4 x 4, 4 and 2^8 x 8 x 2 possible number of operations respectively.

ii. For the calculation of the variable $after3step\_conversion$ in step 29 the values of p_0, p_2 which directly influence the input m have to be guessed. Whatever the values chosen for them, it should satisfy the loop iterations k=0 and 2(step 8). Possible values of p_0, p_2 and $after2step\_conversion$ in this step are 97, 127 and 255 respectively. So computation of $after3step\_conversion$ requires 97 x 128 x 255 x 16^2 operations, where 16^2 is for the hex digits.

iii. In step 28 for the computation of $after2step\_conversion$ the number of operations required are as follows, 67 x 2 possible ways of selecting values from S-Table2, for $2step\_conversion_{part3}$ there are 256x4 (average case) and for selecting the value of $1step\_conversion$, 2^{15} (reducing range considering that some numbers can never be chosen) possible number of operations are required (steps 17 and 18).

iv. For the steps 8-20 total possible operations are 8 x 97 x 2^8

Summing up the total number of possible operations from all the above we get,

2^4 x 4 x 4 x 2^8 x 8 x 2 x 97 x 127 x 255 x 16^2 x 67 x 2 x 256 x 4 x 2^{15} x 8 x 97 x 2^8

$\approx 2^{96}$ [under-estimating 127, 97 and 67 as 2^6 and 255 as 2^7]

Hence the only method to find a valid message satisfying the aforementioned conditions is using brute force method and by definition, an ideal hash function is such that the fastest way to compute a preimage is through a brute force attack. Hence the algorithm is preimage resistant. For an *n*-bit hash digest, the attacker has to perform operations[17] of the order of 2^n. Hence, for LOCHA, it is of the order of 2^{96}operations. Time to perform 2^{96} operations for Mica2 mote using ATMega 128 processor[18] clocked at 7.37MHz is 3.4×10^{14} years.

Collision Resistance:

In LOCHA, to find out two different messages m1 and m2 with hash digests h1 and h2 respectively such that h1=h2, the followings need to be true

i) $hexnumber_{ij}(m1) = hexnumber_{ij}(m2)$ $\forall$ i, j where i $\in$ {0,1,…,t-1} and j $\in$ {0,1,…7} and

ii) $\text{subblock}_{ij}(m1) \neq \text{subblock}_{ij}(m2)$ $\exists$ i, j where i $\in$ {0,1,…,t-1} and j $\in$ {0,1,…7}

For the aforementioned conditions to be true at least one of the following conditions need to be satisfied:

a) $1step\_conversion_{ij}(m_1) = 1step\_conversion_{ij}(m_2)$ and $sub\_block_{ij}(m_1) \neq sub\_block_{ij}(m_2)$
b) $after2step\_conversion_{ij}(m_1) = after2step\_conversion_{ij}(m_2)$ and $sub\_block_{ij}(m_1) \neq sub\_block_{ij}(m_2)$
c) $after3step\_conversion_{ij}(m_1) = after3step\_conversion_{ij}(m_2)$ and $sub\_block_{ij}(m_1) \neq sub\_block_{ij}(m_2)$
d) The values $H_i(m_1)$ and $H_i(m_2)$ are permutations of each other before swapping occurs and the respective swap logic should ensure that after steps 31-36, $H_i(m_1) = H_i(m_2)$ and $sub\_block_{ij}(m_1) \neq sub\_block_{ij}(m_2)$

Now, it can be proved that in our proposed algorithm, brute force method is the only possible way to satisfy at least one of the above conditions. Hence our algorithm is collision resistant and the difficulty of coming up with two messages having the same message digest is the order of 2^{48} operations[17] by the brute force method and the time to perform 2^{48} operations for Mica2 mote using ATMega 128 processor[18] clocked at 7.37MHz is 1.21 years.

Second Preimage Resistance:

Second preimage resistance is considered as an easier or weaker version of collision resistance[16]. So, a collision resistant function is also a second preimage resistant. It is conjectured that the difficulty of coming up with a

message having the same digest as a given message is of the order 2^{96} operations which takes the same time as preimage resistance in Mica2 mote.

Further, referring the algorithm**,** for each iteration, the values of $2step\_conversion_{part3}$ and $1step\_conversion$ depend on the respective values computed in the immediately preceding iteration (except for the very first iteration).This enforces the avalanche effect.

3.2. Performance Analysis

In this section our hash algorithm is evaluated (Table 1) in terms of communication, computation and storage overheads. The performance is also compared with two commonly used hash algorithms viz. MD5 and SHA-1.

Table 1. Comparative Performance.

Scheme	Communication Overhead (Hash Digest in bits)	Computation Overhead (clock cycles)	Storage Overhead	
			RAM/ROM	Number of Registers
MD5	128	36360	32 RAM of 32-bit block data and ROM of 2368 bits/296 byte	12 registers of 32 bits
SHA-1	160	84272	32 RAM of 32-bit block data	12 Registers of 32 bits
LOCHA	96	2952	1 ROM of 804 bits and 1 ROM of 970 bits	4 registers of 16 bits, 18 registers of 8 bits

As per Mica2 specification, energy consumption[19] for transmitting one byte of data for ATMega 128 processor[18] is 16.25μJ and energy required per clock cycle[20] 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
LOCHA -- 16.25x96 (bit) = 16.25x12 (byte) = 195 μJ

Computation overhead:
MD5 -- 0.0032 x 36360 = 116.352μJ
SHA1-- 0.0032 x 84272 = 269.6704μJ
LOCHA -- 0.0032 x 2952 = 9.4464μJ

So, in terms of communication, computation and storage overhead our proposed hash algorithm outperforms the other popular schemes. Moreover for energy-starved network like WSN this scheme has been proved to be much more suitable as well than the other strong hash algorithms such as MD5, SHA1.

4. Conclusion and Future Works

In this paper we propose a lightweight, one-way hash algorithm which produces a hash digest with fixed and relatively small length and applicable for securing energy-starved wireless network e.g. WSN. The algorithm is made lightweight by using low overhead operations such as MOD, SWAP etc. as much as possible. We claim that our scheme fulfils all the basic properties such as preimage resistance, collision resistance and second preimage resistance of a one-way unkeyed hash function. Our claims of maintaining preimage resistance and second preimage resistance are justified. We also show that the proposed algorithm is light-weight in terms of communication, computation and storage overhead. The comparative performance also show our scheme's energy efficiency

compared to MD5 and SHA-1.

As a future extension, attempt will be made to use the generated hash digest in node/message authentication in WSN. Finally the entire scheme would be simulated using a real-time simulator e.g. Contiki to crosscheck our claim made in performance analysis.

Appendix

S-Table 1

[1-20]	521	997	983	733	…	53	59	61	67	71
[21-40]	73	79	83	89	…	151	157	163	167	173
…					…					…
[61-80]	809	293	307	311	…	383	389	397	401	409
[81-97]	863	421	431	433	…	503	509			

S-Table 2

[1-10]	2895	2887	2879	2871	…	2846	2837	2828	2818
[11-20]	2809	2799	2788	2778	…	2746	2734	2723	2711
…					…				…
[51-60]	3127	2323	2255	2240	…	2195	2180	2164	2149
[61-67]	2134	2120	2105	2090	…	2045			

References

1. Youtao Zhang, Jun Yang, Weijia Li, Linzhang Wang, Lingling Jin: An authentication scheme for locating compromised sensor nodes in WSNs, Journal of Network and Computer Applications, vol.33, pp.50-62, 2010.
2. Xiaomei Dong, Xiaohua Li: An Authentication Method for Self Nodes Based on Watermarking in Wireless Sensor Networks, International conference on Wireless Communication (WiCOM), pp.4529-4532, 2009.
3. Chia-Mu Yu, Yao-Tung Tsou, Chun-Shien Lu, Sy-Yen Kuo: Constrained Function-Based Message Authentication for Sensor Networks, IEEE Transactions on Information Forensics and Security, vol.6, no.2, pp. 407-425, 2011.
4. Manik Lal Das, Ashutosh Saxena and V. P. Gulati: A Dynamic ID-based Remote User Authentication Scheme, IEEE Transactions on Consumer Electronics, vol. 50, No. 2, 2004.
5. Fuh-Gwo Jeng , Tzer-Long Chen, Tzer-Shyong Chen: An ECC-Based Blind Signature Scheme, Journal of Networks, vol.5, no.8,pp.921-928, 2010.
6. H. Dobbertin: Cryptanalysis of MD4, Journal of Cryptology, vol. 11, No. 4, pp. 253-271, 1998.
7. X. Wang and H. Yu: How to Break MD5 and Other Hash Functions, EuroCrypt 2005, Springer LNCS 3494, pp. 19–35, 2005.
8. M. Stevens, A. Lenstra and B. Weger.: Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities, EUROCRYPT 2007: 1-22.
9. Rijmen, V.Oswald: Update on SHA-1, RSA 2005, LNCS 3376, pp. 58-71.
10. 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.
11. 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.
12. I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci: Wireless sensor network: a survey, Computer Networks, vol. 38, pp. 393-422, 2002.
13. 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.
14. 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.
15. Paulo Barreto, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, Elmar Tischhauser: Whirlwind: a new cryptographic hash function, Springer, vol. 56, pp.141–162, 2010.
16. Handbook of Cryptography by A. Menezes, P. van Oorschot, and S. Vanstone, CRC Press, 1996.
17. P.Hoffman, B.Schneier: Attacks on Cryptographic Hashes in Internet Protocols, November, 2005 (http://tools.ietf.org/html/rfc4270).
18. Atmel AVR8-bit Microcontroller ATmega 128 processor datasheet (http://www.atmel.in/Images/doc2467.pdf).
19. 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.
20. Mark Hempstead, Michael J. Lyons, David Brooks, and Gu-Yeon Wei: Survey of Hardware Systems for Wireless Sensor Networks, Journal of Low Power Electronics, vol.4, pp 1-10, 2008.