Hardware Keys Exchange Protocol in Wireless Sensor Networks

Kahina CHELLI

***Abstract*—Security in wireless sensor networks is an emerging field of research. Networked sensors have tremendous potential to provide very attractive, low-cost solutions to a variety of real-world problems. As sensors nodes edge closer towards wire-spread deployment, security issues become a crucial concern. However, sensors nodes have inherently constrained characteristics; those incur unique constraints to wireless sensor networks, such as low-computational capabilities, small memory, and limited energy resources. An appropriate cryptographic keys exchange techniques is, therefore, the linchpin of good security in wireless sensor networks. In this paper, we describe the design of cryptographic public-key exchange protocol that allow key agreement between sensor nodes implemented in an FPGA-type embedded architecture based on modified Diffie-Hellman protocol adapted to the specific context of wireless sensor networks.**

***Index Terms*—Wireless sensor network, security, keys exchange, VHDL, FPGA**

I. Introduction

WIRELESS sensor networks are exciting emerging domain of deeply networked systems of low-power wireless sensor nodes with a tiny amount of CPU and memory, and large federated networks for high-resolution sensing of the environment phenomena, signal processing, embedded computing, and communicating to other nodes [1] [2] [3].

This promising technology has a variety of purposes, functions, and capabilities, advancing under the push of recent technological advances and pull of myriad of potential solutions for both military and civilian real-world problems. In many applications, sensor nodes may interact with sensitive data and/or operate in hostile unattended environments. As networked sensors grow in applications area, the need for security in them becomes vital. However, sensors nodes have severe resource constraints. These constraints make wireless sensor networks different from traditional networks, and the security approaches used in these conventional networks cannot be directly applied to wireless sensor networks, because they require much more resource for their extensive computations. As well as the unreliable communication channel, and collaborative nature of sensor nodes in an infrastructure-less network make security defenses even harder [3] [4] [5] [6] [7].

Manuscript received March 04, 2014; revised March 19, 2014.
Kahina CHELLI, University of Mouloud MAMMERI Tizi-Ouzou 15000 Algeria; e-mail kahinachelli@gmail.com

II. Problem and Motivation

Wireless sensor nodes are vulnerable to resource consumption attacks. Energy is the biggest constraint to wireless sensor capabilities due to their autonomy, physical size, and low-cost of production. Those involve the use of low-computing power. For example, one common sensor type TelosB has a 16-bit, 8 MHz RISC CPU with only 10K RAM, 48K program memory, and 1024K flash storage. This microcontroller low-timing is optimized to respond to the energy economy constraint of sensor nodes. Thus, a wireless sensor networks will execute multiple applications concurrently. Adversaries can repeatedly send packets to drain the node's batteries and waste network bandwidth. As well, the mobility and the size of sensor nodes coupled with their variety of the deployment environments physically insecure make security mechanisms an absolute necessity to the defense of wireless sensor networks [1] [4] [5] [6] [8] [10] [11].

One challenging security aspect that receives a great deal of attention in a wireless sensor networks is cryptographic keys exchange, that is an important cryptographic primitive upon which the security of the network is built. In this context, we propose a hardware protocol based on the efficiency of Diffie-Hellman key exchange protocol and Montgomery algorithms.

III. Presentation of our Hardware Protocol

In this section we describe our protocol. Diffie-Hellman algorithm is used and modified to reduce hardware resources.

A. The Diffie-Hellman protocol

Diffie-Hellman protocol [12] is the first public key algorithm which was invented by Whitfield Diffie and Martin Hellman in 1976. Its security relies on the difficulty of computing discrete logarithms in a finite field. It allows two entities to establish a secret key over an insecure channel. Its principle is as follows:

To share a secret cryptographic key among two entities A and B:

--The two entities agree on two primes numbers M and g such that M > g > = 2 and (M-1) / 2 is also prime. The two numbers M and g are public and can be common to a group of entities. The choice of M can have a significant impact on the security of this protocol. More importantly, M must be large and g is a primitive root modulo M;

--The entity A chooses a secret random number x in [2, M-2], and sends to the entity B the calculation result: $X = g^x$

ISBN: 978-988-19252-7-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

mod M;

--The entity B chooses a secret random number y in [2, M-2], and sends to the entity A the calculation result $Y = g^y$ mod M;

--The entity A computes the secret key $Ka = Y^x$ mod M, and the entity B computes the secret key $Kb = X^y$ mod M.

The values Ka and Kb are both equal to $g^{(x\,y)}$ mod M. An intruder that listens to traffic cannot calculate this value, because the interceptor knows only the values M, g, X, and Y. To calculate the secret parameters, the intruder must solve the discrete logarithms x = ln (X) / ln (g), and y = ln (Y) / ln (g), which are very complex to achieve within a reasonable time.

Using this key exchange protocol in wireless sensor networks has many advantages. Sensor nodes may exchange their secret key on an insecure channel. Thus, it ensures retroactive security. That is to say, in case of disclosure the secret key of the system, only the current exchange is affected, but not the future. It allows also scalability insofar as the exchanges are peer to peer without central authority. Unfortunately, the direct application of this protocol is difficult for wireless sensor networks, because it is typically too computationally intensive for tiny sensor nodes. In other words, the arithmetic operations on this protocol are complex because they are on cyclic multiplicative groups of prime order on numbers of several hundred-bit. However, it is feasible with the rights selection of algorithms.

The solution that we propose is FPGA-type embedded architecture that allows discharging the microprocessor of cryptographic keys computational tasks, so optimizes the resource allocation. It's based on Montgomery multiplication and exponentiation algorithms which are very smart, fast and efficient algorithms. They replace the division by a shift and modulus-addition operation that reduce the computational complexity and generate a tremendous gain in energy. This solution tolerates the use of small keys combined with an appropriate keys exchange frequency.

B. *Network model and attacker*

Our protocol is based on the following assumptions: We consider a large-scale distributed architecture, where nodes are deployed in a random manner and may be mobile or static. The sensor nodes can be homogeneous or heterogeneous. We assume that the attacker can be passive (network traffic analysis, etc.) or active (injecting data, compromised sensors, etc.). We also assume that the information previously transmitted via the network is not interesting for the adversary. That is to say, the adversary is interested to the transmitted data at the present time.

C. *Protocol design*

Our hardware key exchange protocol is a digital circuit which is responsible of cryptographic keys computational tasks. The Fig.1 shows the position of our circuit in a sensor module. The description of the circuit is given in the following section.

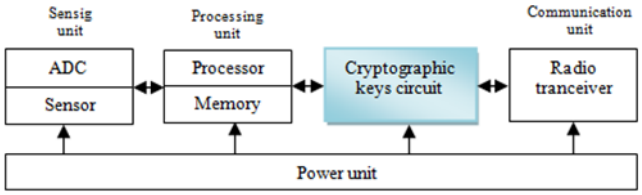

Fig. 1. Position of our circuit in a sensor node.

The particularities of the novel protocol

--Reducing the number of bits of all the parameters of Diffie-Hellman protocol, the public parameters M, g, X and Y, and also the private parameters x and y. We set the size of all these parameters on 160-bit. We use g = 2 as primitive root. Nothing prevents to take g the smallest value appropriate. In addition, it reduces the cost of the modular exponentiation, which is a costly operation for tiny sensor nodes.

--Increasing the keys exchange frequency among the sensor nodes. The security fault induced by reducing the number of bits is filled by an appropriate frequency of keys exchange, so that the attacker may not have the prescribed time to crack the secret key.

Our key exchange protocol

Two nodes A and B want to establish a secret cryptographic key agree on the big prime number M that can represent in 160-bit:

--The module of node A generates a random secret number x in [2, M-2], and sends to node B the calculation result $X = 2^x$ mod M;

--The module of node B generates a random secret number y in [2, M-2], and sends to node A the calculation result $Y = 2^y$ mod M;

--The module of node A calculates the secret key $Ka = Y^x$ mod M, and the module of node B calculates the secret key $Kb = X^y$ mod M.

For the generation of the secret numbers x and y, we use Blum-Blum-Shub Pseudorandom number Generator (BBS) [13]. It is one of the most efficient pseudorandom number generators. This generator works as follows:

--Choose two large prime numbers p and q that are congruent to 3 modulo 4. The product of these numbers is n = p * q, called Blum integer.

--Choose another random integer x that is prime to n and compute $x_0 = x^2$ mod n, this number is the seed of the generator.

--The i^{th} pseudorandom bit is the least significant bit of x_i where: $x_i = (x_{i-1})^2$ mod n.

The security of this generator is based on the mathematics underlying factoring large integer that is intractable.

Our protocol is mainly performed by modular exponentiation which is a succession of modular multiplication. Among the most widely used algorithms for these arithmetic operations are Montgomery's algorithms which are efficient methods to perform modular multiplication and exponentiation in hardware [14].

Montgomery modular multiplication

Montgomery algorithm for modular multiplication allows multiplying two integers modulo m avoiding division by m.

ISBN: 978-988-19252-7-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Algorithm.1: Montgomery modular multiplication
Inputs: *integers* $m = (m_{n-1}...m_1m_0)_2$, $x = (x_{n-1}...x_1x_0)_2$, $y = (y_{n-1}...y_1y_0)_2$, *with* $0 \le x, y < m$ and $R = 2^n$ with *gcd (m, 2) = 1,* $m' = -m^{-1} \bmod 2$.
Output: $x*y*R^{-1} \bmod m$.
1. A←0. (Notation: $A = (a_n a_{n-1} \ldots a_1 a_0)_2$)
2. For i from 0 to (n − 1) do the following:
   2.1 $u_i \leftarrow (a_0 + x_i y_0)\, m' \bmod 2$;
   2.2 $A \leftarrow (A + x_i y + u_i m) / 2$;
3. End for;
4. If A ≥ m then A←A – m;
5. Return (A).

The radix-2 Montgomery modular multiplication presented in algorithm.1 computes $x*y*R^{-1} \bmod m$, where x and y are the operands, m is the modulus and R is a power of two.

Before performing a modular multiplication using the Montgomery algorithm, the operands need to be transformed into Montgomery representation. The Montgomery representation of two integers x and y denoted by x_mont and y_mont can be computed as follow:

$$X\_mont = montgomeryMultiplication\ (x, R^2)$$
$$= x*R^2*R^{-1} \bmod M$$
$$= x*R \bmod M \quad (1)$$
$$Y\_mont = montgomeryMultiplication\ (y, R^2)$$
$$= y*R^2*R^{-1} \bmod M$$
$$= y*R \bmod M \quad (2)$$

After computing the Montgomery multiplication of two operands in Montgomery representation, the result is also in Montgomery representation and can be converted back by multiplication with R^{-1}, which comes down to Montgomery multiplication with 1. This can be illustrated as follow:

$$Result = montgomeryMultiplication\ (result\_mont, 1)$$
$$= x*y*R*1*R^{-1} \bmod M$$
$$= x*y \bmod M \quad (3)$$

Montgomery modular exponentiation

When using the Montgomery multiplier for modular exponentiation (see algorithm.2) P_0 and Z_0 have to be converted to M-residue before running the loop and Z has to be converted back at the end of the algorithm.

Algorithm.2: Montgomery modular exponentiation
Inputs: integers $x = (x_{n-1}...x_1, x_0)_2$, $e = (e_{n-1}...e_1, e_0)_2$, $m = (m_{n-1}...m_1, m_0)_2$;
Output: $P = x^e \bmod m$
1. P_0:= montgomeryMultiplication (1, R^2);
2. Z_0:= montgomeryMultiplication (x, R^2);
3. For i in 0 to n-1 loop
   3.1. Z:= montgomeryMultiplication (Z, Z);
   3.2. If e(i) = 1 then
   3.4. P:= montgomeryMultiplication (P, Z);
4. End for;
5. P:= montgomeryMultiplication (1, P);
6. End.

IV. Hardware Implementation of our Key Exchange Protocol

A. Synoptic diagram of the circuit

Fig.2 gives the synoptic diagram of our circuit. It composed of three modules, one module to generate pseudorandom number and two modules to perform modular exponentiation.

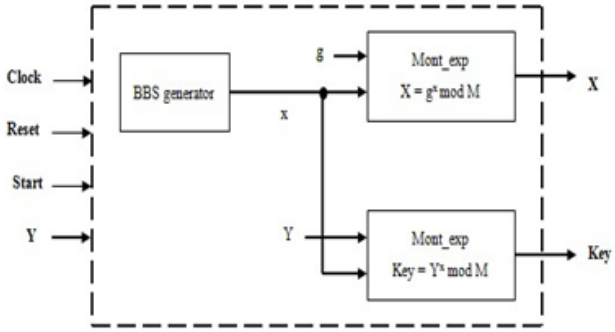

Fig. 2. Synoptic diagram of the circuit.

Our circuit has been described in VHDL. Each module is described independently using Algorithmic State Machine approach (ASM). We present the generator module, the modular multiplication module and the modular exponentiation module.

The pseudorandom number generator module

This module is responsible for generating the private parameter for each sensor node. It has two inputs signals clock and reset, and 160-bit output that represent the pseudorandom number. The diagram and interface signals of the generator are given in Fig.3.

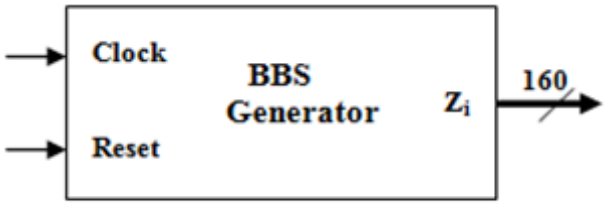

Fig. 3. The BBS Generator module.

The ASM corresponding to BBS generator is shown in fig.4.

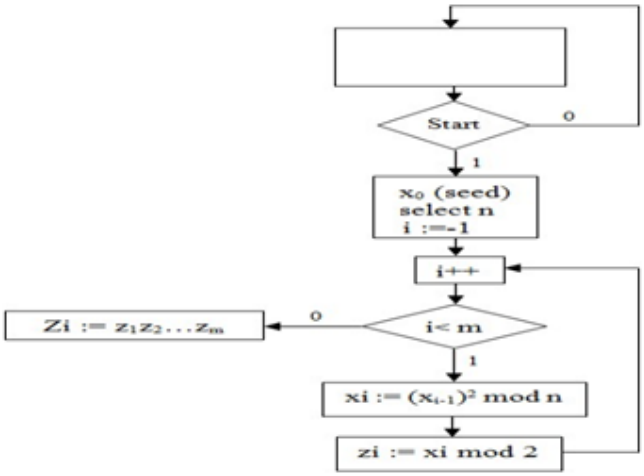

Fig. 4. ASM of the BBS generator.

ISBN: 978-988-19252-7-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Montgomery modular multiplication Module

Fig.5 shows the diagram and interface signals for the Montgomery modular multiplier. It has two 160-bit inputs X and Y, signal clock, two signals of reset command and start, and 160-bit output that represent the result of the modular multiplication in Montgomery representation.

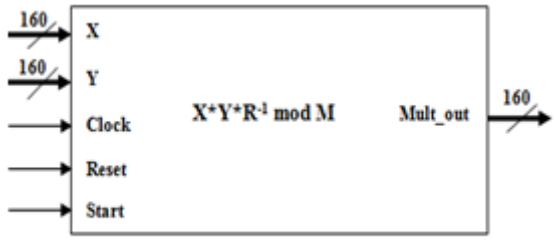

Fig. 5. Montgomery modular multiplication module.

The corresponding ASM of Montgomery modular multiplier is shown in fig.6.

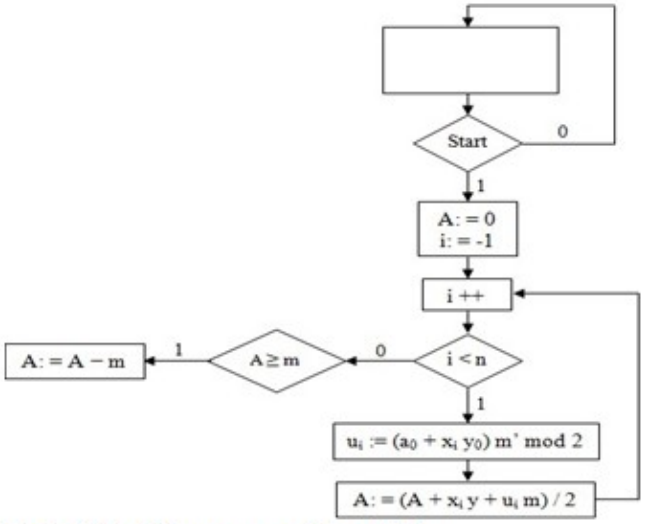

Fig. 6. ASM of Montgomery modular multiplier.

Montgomery modular exponentiation Module

Fig.7 gives the schema and interface signals for the modular exponentiation module. It has two 160-bit inputs x and e, signal clock, two signals of reset command and start, and 160-bit output that represent the result of the modular exponentiation in natural representation.

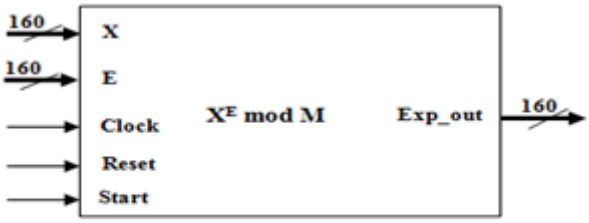

Fig. 7. Montgomery modular exponentiation module.

The corresponding ASM of Montgomery modular exponentiation is given in fig.8.

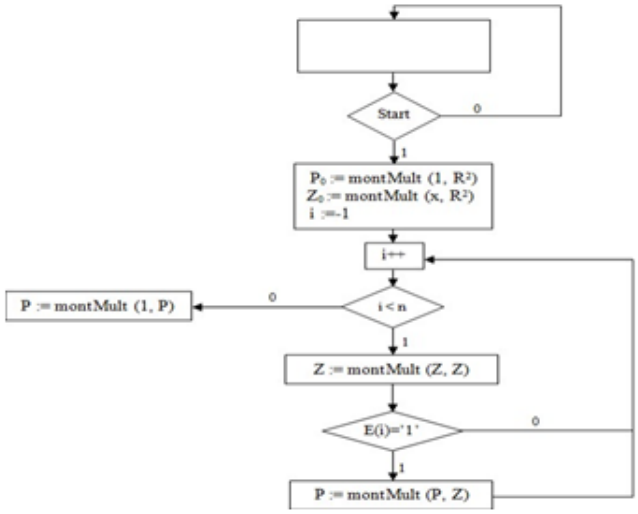

Fig. 8. ASM of Montgomery modular exponentiation.

According to the ASM shown in Fig.9, two temporary registers are initially loaded by P_0 and Z_0. Then the algorithm loops from the least significant bit of e to the most significant bit. Each iteration i, the new value of Z is computed. If e (i) = '1' the register is updated with the new value P = P * Z. Otherwise the register remains unchanged.

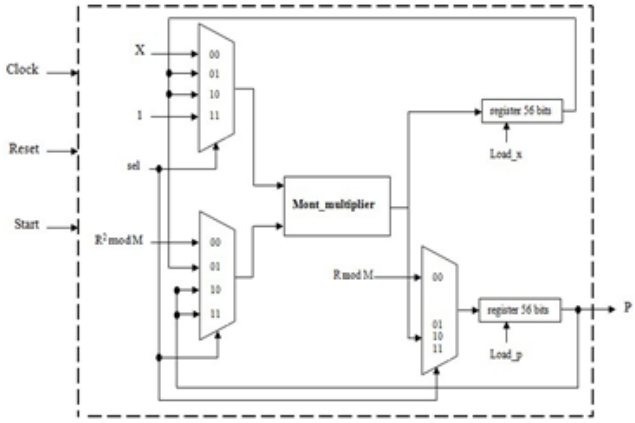

Fig. 9. Data path of Montgomery modular exponentiation.

V. VHDL SIMULATION

In this part, we have simulated the various modules using 8-bit.

Fig.10 gives the VHDL simulation of pseudorandom number generator module. For functional simulation, we created an instance of this generator having the following parameters:

N = p*q = 7*19 = 133, x = 100, $x_0 = 100^2 \bmod 133 = 25$, $x_1 = 25^2 \bmod 133 = 93$, $x_2 = 93^2 \bmod 133 = 4$, $x_3 = 4^2 \bmod 133 = 16$, $x_4 = 16^2 \bmod 133 = 123$, $x_5 = 123^2 \bmod 133 = 100$, $x_6 = 100^2 \bmod 133 = 25$, $x_7 = 25^2 \bmod 133 = 93$, $x_8 = 93^2 \bmod 133 = 4$.

The pseudorandom number is: $10010110_2 = 150_{10} = 96_H$.

After a reset, the circuit is initialized and starts processing. The end of processing is indicated by the passage of signal clock at 1. The random value generated is supplied to the bus Z_i as shown in Fig.10.

ISBN: 978-988-19252-7-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

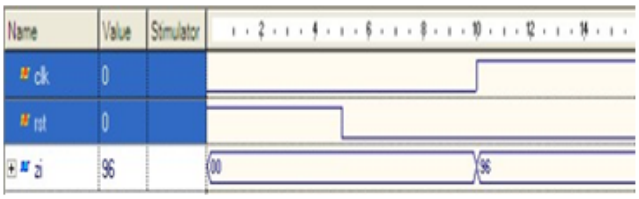

Fig. 10. VHDL simulation of the BBS generator module.

Fig.11 illustrates the VHDL simulation of Montgomery modular multiplication module. To instantiate the multiplier, we consider parameters in Montgomery representation as follow:

$X = 08_H$, $Y = 1B_H$ and the final result $mult\_out = A4_H$.

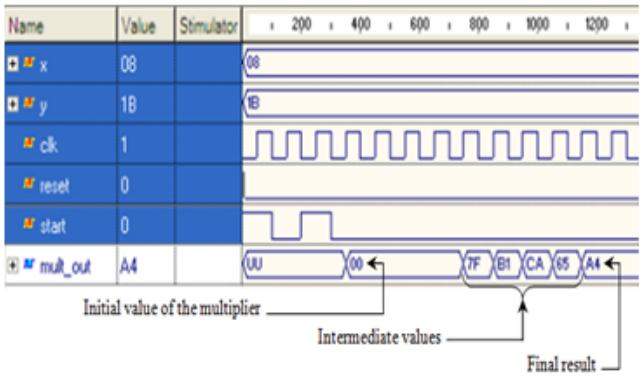

Fig.11. VHDL simulation of the Montgomery modular multiplication.

Fig. 12 gives the VHDL simulation of Montgomery modular exponentiation. We instantiate the module with the following parameters:

$X = 10_H$, $E = 0C_H$ and the final result $exp\_out = 0C_H$.

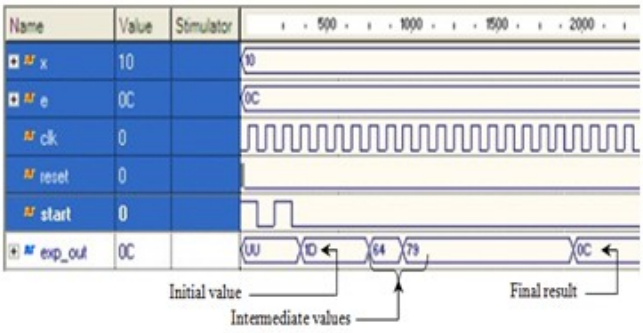

Fig. 12. VHDL simulation of the Montgomery modular exponentiation.

VI. Conclusion and Perspective

Wireless sensor networks remain one of the most exciting and challenging research domains of our time. Among the problems posed at present in this unique type of networks is the security problem. A dedicated solution depends on the capabilities of sensors nodes, indicating that there is no unified solution. Instead, security mechanisms are highly applications-specific

Our contribution consists of a hardware protocol which aims to bring a suitable solution to the problem of key exchange in wireless sensor networks.

As perspective, we envisage to study a suitable key size for a reasonable frequency of change. We also envisage quantifying the consumed energy by the synthesized circuit in a given technology (e.g. FPGA). Our protocol can be combined with an encryption algorithm, which can also be achieved by a hardware solution.

Reference

[1] K. Sohraby, D. Minoli, and T. Znati, WIRELESS SENSOR NETWORKS: Technology, Protocols, and Applications, USA, 2007, pp. 1-71.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “A survey on sensor networks,” IEEE Communications Magazine, 40(8): pp.102–114, August 2002.

[3] R. Vishal, M. Mrudang, “Security in Wireless Sensor Network: A survey,” Ganpat University Journal of Engineering & Technology, vol.1, issue.1, jan-jun-2011.

[4] Jaydip Sen, “A Survey on Wireless Sensor Network Security,” International Journal of Communication Networks and Information Security, Vol.1, No.2, August 2009.

[5] N. Gura, A. Patel, A. Wander, H. Eberle, and S. Shantz. Comparing elliptic curve cryptography and rsa on 8-bit cpus. In 2004 Workshop on cryptography Hardware and Embedded Systems, August 2004.

[6] R. Watro, D. Kong, S. Cuti, C. Gardiner, C. Lynn, and P. Kruus. Tinypk: securing sensor networks with public key technology. In Proceedings of the 2nd ACM Workshop on Security of Ad hoc and Sensor Networks (SASN ’04), pages 59–64, New York, 2004. ACM Press.

[7] T. Arampatzis, J. Lygeros, and S. Manesis, “A Survey of Applications of Wireless Sensors and Wireless Sensor Networks,” In Proceedings of the 13th Mediterranean Conference on Control and Automation, 2005, pp 719–724.

[8] S. Zhu, S. Setia, and S. Jajodia. Leap: Efficient security mechanisms for large-scale distributed sensor networks. In CCS ’03: Proceedings of the 10th ACM Conference on Computer and Communications Security, pages 62–72, New York, NY, USA, 2003. ACM Press.

[9] D. J. Malan, M. Welsh, and M. D. Smith. A public key infrastructure for key distribution in tinyos based on elliptic curve cryptography. In First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON, 2004.

[10] H.C. Chaudhari and L.U. Kadam, “Wireless Sensor Networks: Security, Attacks and Challenges,” International Journal of Networking, Vol.1, Issue.1, pp-04-16, 2011.

[11] L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks. In Proceedings of the 9th ACM Conference on Computer and Communications Security, pages 41–47. ACM Press, 2002.

[12] B. Schneier, applied cryptography: algorithms, protocols and source codes in C, 2nd edition.

[13] P. Junod, “Cryptographic Secure Pseudo-Random Bits Generation: The Blum-Blum-Shub Generator,” August 1999.

[14] A. J. Menezes, P. C. van Oorschot, and S. A Vanstone, HANDBOOK of APPLIED CRYPTOGRAPHY, USA, august 1996, pp. 602-620.

ISBN: 978-988-19252-7-5
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)