Establishing Symmetric Pairwise-keys Using Public-Key Cryptography in Wireless Sensor Networks (WSN)

Ibrahim Nadir, ibrahimnadir@gmail.com
Politecnico di Torino, Torino, 10129 Italy
Wondimu K. Zegeye, Farzad Moazzami, Yacob Astatke
{wozeg1, farzad.moazzami, yacob.astatke}@morgan.edu
Morgan State University, Baltimore MD, 21251 USA

***Abstract*—Wireless Sensor Networks (WSN) are becoming an increasingly popular technology for data collection. Since data is of significant importance in the current era, security of the data is essentially indispensable. Because the nodes deployed are usually on an unattended environment, security of the data communicated between the wireless sensor nodes is of utmost importance. A secure established connection between two nodes in a WSN promises a number of preventive measures against different threats but the main issue is to establish the secure connection between any two nodes in a wireless sensor network. With limited resources on a wireless sensor node in mind, this research focuses on establishing Symmetric-Pairwise keys between any two WSN nodes using Public-Key Cryptography (pkc) – which in this case is Elliptic Curve Cryptography.**

***Keywords*—wireless sensor network, Elliptic Curve Cryptography, Symmetric-Pairwise Keys, hardware implementation.**

I. Introduction

Wireless sensor networks are low-powered, low memory, self-organizing and fault tolerant networks as compared to traditional networks. The number of nodes in a WSNs ranges from several to a few hundreds or even thousands. Having a number of positive aspects like longer lifetime, ease of deployment and cost efficiency, WSNs like all other networks, faces security issues. The need to maintain tight security in WSNs is an important aspect of modern wireless communication [1]. Asymmetric and Symmetric cryptography are being used to create encrypted data in general. However, the use of these cryptographic keys and their management in dealing with the sensor nodes is a challenging task. To resolve the issue of such resource constrained networks, different key management schemes have been introduced. Symmetric pairwise-keys have been proved to work much efficiently and faster than asymmetric [2]. Therefore, a lot of key management schemes using symmetric algorithms have been devised [3].The issue is how to establish these pairwise-keys. Previously, asymmetric cryptography was considered infeasible for wireless sensor networks [4]. Based on some studies [5], the use of public-key cryptography is considered possible to establish symmetric pairwise-keys and data is later encrypted using the established symmetric keys.

Cryptographic keys are of vital importance in network communication because they provide the basic building blocks to a number of security parameters. Cryptographic keys make sure that authentication, privacy, non-repudiation and integrity is achieved. Choosing from the state-of-the-art algorithm to establish secure communication between the network nodes is an important decision because the sensor nodes are strictly resource constrained. As compared to Asymmetric, Symmetric pairwise-keys are considered still the fastest and memory efficient among different methods of encryption.

II. INTRODUCTION TO OUR SCHEME

In this paper, we will introduce a new security scheme that establishes a symmetric pairwise-key using an asymmetric cryptography between any two network nodes. We will use Elliptic curve cryptography (ECC) as the Public-key cryptography scheme an Advanced Encryption System (AES) as the symmetric pairwise-key algorithm because AES was designed for efficiency and small code size on low-end processors .

A. *Elliptic Curve Cryptography*

Elliptic curve cryptography (ECC) is a rather new way of implementing the Public-key cryptography as compared to the RSA [6][7]. ECC is based on elliptic curves over finite fields and has a higher level of complexity. Elliptic curve cryptography is a much efficient way to carry out public key cryptography because it uses less computation and memory to provide a higher security level. The main advantage to using ECC is that it can provide the same level of security with a smaller key size. For instance, the RSA key size 1024 provides the same security level as that of an ECC of 164 bits size key. Based on the elliptic curve discrete log problem, ECC is a much harder problem than simply factoring integers. The fact that ECC is based on algebraic structure of elliptic curves over finite fields makes it computationally infeasible to solve the problem and hence the security is of a higher level. Before ECC, the implementation of public key cryptography was almost impossible [4]. Thanks to ECC, public key cryptography can now be implemented easily because of the lower memory and computational requirement since the key size is much smaller and similar other primitives.

The tradeoff with ECC is its complex computation. A user can multiply a point by a number to produce another point on the curve. Even with the knowledge of what the number was, it is very hard to point out the original point and hence the result. The computation of ECC may be very tough to come up with but it can also be the charm to the method because this complexity is what makes the system difficult to crack. This is the reason we will choose ECC to use public key infrastructure in our algorithm. It should be noted that ECC is faster than RSA in decryption but slower in encryption.

B. *Assumptions*

The following assumptions are made based on the design of our algorithm:

- Each node has a private and a corresponding public-key loaded into it before deployment by the network administrator.
- The current version of the algorithm assumes that no new nodes are arriving in the network and only the nodes already considered in the network will establish a symmetric-key among themselves.
- All nodes are assumed equal in terms of their attributes. No node is considered powerful or weak as compared to any other.
- If there is a possible attack, and a node is compromised. All the data inside the node is considered compromised and known to the attacker.
- Each two nodes will have a different pairwise key established among them.

C. *Algorithm Design*

Each node in the network has a node id of itself, a private and public-key. Apart from that, every node has a table of the node ids' of all other nodes, a bit mask and a hash against its corresponding node id.

We use the following notations to simplify explanation:

k : Public-key of a node

k_l : length of Public-key

i: Node ID(from $1 \ldots N$ where N is the total number of nodes in the network)

j : Bits randomly chosen to be removed from the public-key(k) of the nodes.

M : Calculated hash of the public-key after j bits are removed from k.

Node Id (i)	Bitmask (j)	Hash ($m = k - j$)
1	*101001*	*A89B0*
2	*101101*	*320C9*
...	...	...
N	*101010*	*001D3*

Table 1: Data Format of a Node

The Table 1 shows an example of how the data will be contained inside a node in the network.

The hash function used to compute the hash is SHA-1.

III. Algorithm Phases

The algorithm is divided in two phases:

- Authentication
- Key Generation

A. *Authentication*

Each node starts broadcasting its public-key (k_i). Any other node after receiving the public-key of the broadcasting node will authenticate the node using its authentication table as follows:

- Receive the public-key (k) of length k_l
- Remove the j bits from the public-key (k)
- Take the hash (m) of the remaining $k_l - j$ bits
- Compare the computed result with the hash in the authentication table.

If the result obtained from the computation is the same as that in the authentication table Hash column, the node is authenticated and the node will start the generation of symmetric key with that node. Otherwise, the node will be discarded and considered a threat to the network.

B. *Key Generation*

After authentication, a random key is generated by both the nodes. Since both nodes have each other's public-key, the key will be used by the nodes to encrypt the secret key for the opponent node. On reception of the encrypted secret key, each node will decrypt the message using their corresponding private-key. After that an XOR is calculated from the secret message received from the other party and the secret message generated locally and hence the key between the two parties is established. Eventually, this key is used as a symmetric key to encrypt data between the two parties. The communication between the two nodes is shown in the Figure 1:

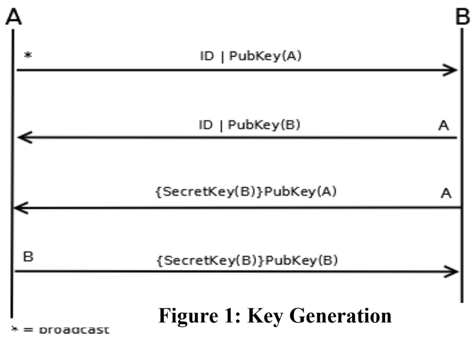

Figure 1: Key Generation

The above graphical representation shows the handshake to establish protocol between any two nodes *A* and *B*. In case, where there are more than two nodes and assuming all nodes deployed at the same time. Consider, node *A* broadcasts its public-key and other nearby nodes receives the broadcast will authenticate and respond with its public-key. Let's say, node *C* and *D* be two other nodes that respond to node *A* along with node *B*. Node *A* can start handshake with only one of the three nodes available for handshake. Therefore, as soon as node *A* receives a response, it will broadcast an *ack* stating the *id* of the node it is doing handshake with. The *ack* will tell other nodes to discard the key-generation with node *A* and hence node *C* and *D* will start a mutual handshake. This *ack* will help in improving the overall performance of the algorithm in the following ways:

- It will save communication and hence node *C* and *D* won't send encrypted messages.
- Save time because node *C* and *D* won't have to wait for Node A and instead they can start handshake with each other.
- Save computational energy by not calculating any encrypted secret messages.

With the addition of an *ack*, the *Figure 1* would be evolved into *Figure 2* as shown below.

We considered saving the HELLO packet for the moment and do the handshake once the node finishes off its current key-generation phase but that did not seem like a feasible solution. The reason for that would be:

- More memory is required to save the HELLO packets from different nodes.
- The algorithm would become more complex.

IV. Analysis of algorithm

This section describes the feasibility of the algorithm from various points of views. The most important of which are the following:

A. Security Analysis

The proposed algorithm can provide good security and promises the following security goals:

- Authentication
- Confidentiality
- Integrity
- Non-repudiation
- Data availability

A number of issues in WSN regarding security have been notified [8] and here is how the proposed algorithm mitigates them:

False Node ID: Fake nodes can't authenticate themselves. The reason is that every node in the network stores a list of all the nodes in the network in its authentication table. Hence any new node which doesn't know the ID and the corresponding public-key of that node, can't authenticate itself with any node of the network. However, any threat after authentication can not establish a key with a valid node in the network unless it knows the corresponding private-key to any public-key. The reason is that the threat will not be able to decrypt the secret message received after authentication.

Node replication/Duplication: In the proposed algorithm, any node that has correct information regarding the Node *Id* and the corresponding Public-key of that node can send it to any neighbour node and authenticate itself. This attack can be possible duplication/replication of a node. The reason that such attacks can be successful is because the other node will simply be authenticating the node using its public-key and node id. This attack can be dealt with pretty quickly with the help of asymmetric cryptography. Since any data sent to a node will be encrypted with its public-key. On the receiving end, if the fake node has a corresponding private-key, it can do more damage.

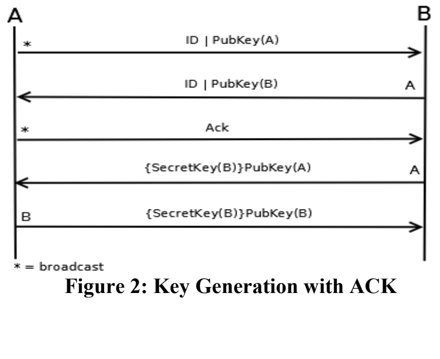

Figure 2: Key Generation with ACK

Since the private-key is always kept secret, there is no way a fake node can get the private-key of a node it is masquerading as. Hence the fake node can't be a successful attack after all.

DoS attack: Denial of service attacks can be performed by continuously sending HELLO messages to any node [8] of the network. But we can simply check if the node exists in the authentication table and if not, any future HELLO messages will be discarded from that node. DoS attacks are still energy consuming and they can always drain a node's battery.

Node Compromise: In node compromise attack, all the data of a node is stolen. This could be a serious attack once the secret keys are compromised by the attacker. The attacker can get data from and inject false data in the network. But that's all it can do. It can't get access to other nodes or take control of them. Only data sent to and received from that node is falsified by the attacker. Our proposed algorithm can save this attack from spreading to other nodes. The reason is, an attacker can't multiply the attack in some way by reaching the data of other nodes. It may inject some erroneous packets in the network but there are other methods to identify such threats and mitigate such nodes. Once recognized, the data related to these nodes will be removed and other nodes will be notified of this attack.

Man in the middle attack: Man in the middle attack can be launched whenever there is not enough authentication mechanism. The proposed algorithm provides strong authentication and it can be considered a two-step authentication. In the first step, a node is authenticated using the authentication table. In the second step, the public-key cryptography is an authentication in itself. Whenever data is encrypted with the public-key of a node, the decryption with the corresponding private-key is an authentication of the node itself. This attack is mitigated because man in the middle attack uses a false Public-key and since each node has hashed public-key of all of nodes stored in its authentication table, the result will be different, hence the attack will be identified and tackled.

Eavesdropping: Eavesdropping is listening to the communication channel and reading the packets data. The only information gained using eavesdropping can be the node ID and the public keys during the authentication phase. After that, all the data exchange is secured using asymmetric or symmetric cryptography. Therefore, it is impossible to gain anything out of the communication channel that can be a threat to the network.

B. Computation Analysis

The algorithm is pretty efficient in terms of computation. For each pairwise-key establishment, there is a hash computation, an encryption and a decryption. There isnt much computation needed when we are calculating the digest (Hash) of (*k-j*) bits of a node's public-key that is done during the authentication phase. Since the entire communication would be an encrypted one, there isn't much computational overhead required to establish a symmetric key. Specially, when it is done using ECC. We know that ECC provides higher security with less bits - Almost 1/10th times better security with the same number of bits as compared to other asymmetric key encryptions.

C. *Storage Analysis*

We need to consider the following notations in order to do the storage analysis:

k: Public-key

n: Number of nodes in network

i : Node id starting from *1…N*

j: Bits removed by the bit mask from the public key(*k*)

m: Hash calculated on the(*k-j*) bits

v: Number of neighbor nodes

Sym_key: Symmetric key after key-establishment

PubKey: Public-key of the node

PriKey: Private-key of the node

Then the storage is given in the Table 2(a).

Pre storage	*(n-1)(i + j + m) + (PriKey + PubKey)*
Working storage	*(v)(i + SymKey) + (PriKey + PubKey)*

Table 2(a): Storage

v	1	5	10	15	20
Bytes required	4292	4372	4472	4572	4672

Table 3(b): Storage

The algorithm assumes that after having established pairwise keys with neighbours, a node will delete the data (Bit mask and Hash) from the authentication table of its neighbouring nodes with whom it has established a pairwise-key already and just store the symmetric keys for secure communication. This way, memory is freed up after key establishment.

With the size of i being *4 bytes*, *SymKey 16 bytes*, *PriKey* and *PubKey* each *20 bytes*, the memory requirements scales good according to the Table 2(b).

D. *Communication Analysis*

The algorithm behaves well in terms of communication. A HELLO packet contains the public-key of a node and it is the first step for authentication. The next packet contains the encrypted secret message and hence the symmetric-key is established between the nodes. It takes four packets to establish the symmetric pairwise key between two nodes, two packets from each node. The whole scenario is given in Figure 1 already. Node *A* broadcasts its public-key and any neighboring node near-by will hear. The node, say *B*, after receiving *A*'s public-key will respond with its public-key after authenticating node *A*. Node *B* will also send its secret message encrypted with *A'*s public-key. On the other hand, node *A* will do the same. Hence, both the secret messages will be exchanged and the symmetric key established.

We compare our algorithm with LEAP+ [9] using the following notations as shown in Table 3:

i : Node identifier

PubKey : Public key of node

ESecMsg : Encrypted secret message

l_k: Length of key

Algorithm	LEAP+	Our Algorithm	Our algorithm with *ack*
Hops	*1*	*1*	*1*
Number of Messages	*2*	*4*	*5*
Size of transmitted data	$2(i + l_k)$	*2(i + PubKey + ESecMsg)*	*2(i+PubKey+ESecMsg) + ack*

Table 3: Comparison with LEAP+

We also introduced another flavor of the algorithm in which we added an *ack* to improve performance. With the addition of just an *ack* we are able to save a lot of time as stated before. The communication comparison is given in table.

E. *Time Analysis*

As compared to the negotiation done using symmetric-key encryption, asymmetric-key encryption used to negotiate for key establishment takes much more time. For the proposed algorithm, since ECC is used which is much quicker than any other asymmetric-key encryption methods, therefore it is feasible to implement the algorithm. In our experiments, a key is established between any two nodes in between 21-22 seconds assuming the time right from the point where the node is powered on. Asymmetric-key encryption is notorious for being slow and almost commercially impossible to implement but our algorithm implements it using ECC which makes it looks quite possible to implement. We have shown that within few minutes we are able to establish key between any two nodes in a network. The proposed algorithm comes in two flavours which gives different time consumption results as we will show in the section below. In the next section, we will show results of the time taken by our algorithm in both the flavours.

V. Experimental Environment And Results

We used the Crossbow's Telosb mote (TPR2400) in lab to carry out the experimental procedures. The mote has a Universal Serial Bus (USB) programming capability, 8 MHz TI MSP430 microcontroller with 10kB RAM and a data rate of up to 250 kb/sec. It runs TinyOS 1.1.10 or higher. Hence we used TinyOS which is a free and open source operating system. We chose eclipse as the Integrated development environment (IDE) and installed YETI plugin to support network embedded systems C (nesC) language for programming. The working environment was Ubuntu 12.04.

For the encryption and decryption of data using ECC, we used a library that is implemented by the Computer Sciences department of the North Carolina State University called TinyECC. TinyECC implemented Public-Key Cryptography operations like digital signature, encryption and decryption. The implementation is configurable and quite flexible according to the hardware devices available [10].

The code written was of an average size and covers:

- 31066 bytes in ROM
- 4292 bytes in RAM

We have compiled the results for the following three scenarios:

- Nodes deployed at the same time
- Nodes deployed at different times
- Nodes deployed with same time with *ack*

As can be seen from Figure 3, the percentage of keys established when the node density decreases slowly for all of the three scenarios. After a certain number of nodes (in this case 7), when the nodes are deployed at the same time, the percentage of keys established is slightly better than the other two scenarios. The percentage of keys established in the case of nodes deployed at the same time with ACK is slightly lower than the other two scenarios.

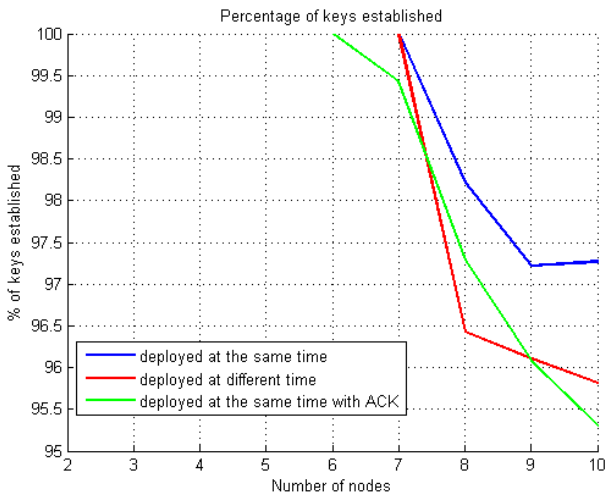

Figure 3: Percentage of Keys Established

Figure 4 shows that, for the three scenarios as the node density increases, the average time it takes to establish keys decreases. In addition, the average time taken by the algorithm to establish keys by the nodes deployed at the same time with ACK is better than the other two cases after 5 nodes.

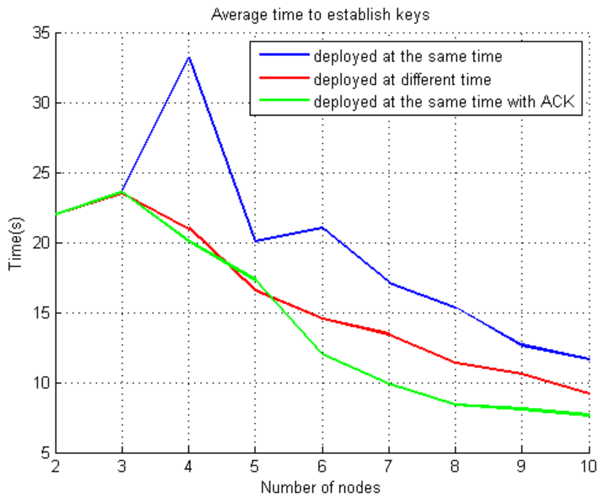

Figure 4: Average Time to Establish Keys

VI. Conclusion And Future Work

The need to maintain tight security in WSNs is an important aspect of modern wireless communication. Once considered almost impossible, with ECC, the establishment of a pairwise-key has been made possible on the telosb from Berkeley labs. Since the use of ECC has been recently initiated because of its quicker nature and higher security, therefore the results obtained from this work are significant in terms of new research performed. A number of tests and results have been obtained to show in the first place the possibility of implementing symmetric pairwise-keys using public key cryptography. The study depicts the possibility of having a higher density of nodes in a neighborhood and still key exchange percentage is very high.

From the results of the algorithm, it is observed that the memory required to establish pairwise-keys is not too high as compared to other such algorithms but it is much more secure method of ensuring authentication, confidentiality and non-repudiation at the same time. The time taken to establish the key is almost same as other key establishment techniques using asymmetric encryption but nodes deployment time largely affects the key establishment time among nodes in a neighborhood. The computational and communicational requirements for key establishment are average as well. With an excellent key establishment percentage, the algorithm proposed is overall applicable and can be adopted for any real network.

Keeping in view the significance of the algorithm, future work could be performed in the same context. A possible work could be addition of new nodes in the network. In that case, any new node entering the network will be authenticated by the nodes already deployed in the network. After authentication, secret messages could be exchanged with the node hence a symmetric pairwise-key will be established with the node.

Another possible future work could be the modification of the algorithm to work when the nodes are mobile. Therefore, even if nodes are dynamic in their movement, the algorithm will be able to establish symmetric pairwise-keys between them. But as for now this seems difficult to be possible since the time taken to establish the keys is high. With the evolution of hardware resources and efficient implementation of TinyECC library, it can still be considered a possibility.

We could also consider the possibility of implementing the algorithm using a new scheme called multivariate quadratic public key near group (MQQ). MQQ has been proved to work much faster than RSA and ECC [11].

ACKNOWLEDGMENT

The authors acknowledge the support of Prof. Maurizio Rebaudengo and Prof. Filippo Gandino from Politecnico di Torino.

VII. References

1. Xiaojiang Du, North Dakota State University and Hsiao-Hwa Chen, National Cheng Kung University "Security in Wireless Sensor Networks" *IEEE Wireless Communication August 2008*

2. U.Senthil Kumaranilango, “Evolution of key management and variations of random key pre-distribution in wireless sensor networks: survey”, *International journal of Applied engineering and research –volume 9* number (21) (2014) pp 11681 – 11688.
3. G. J. Pottie and W. J. Kaiser, “Wireless Integrated Network Sensors”, *Communications of the ACM*, 43(5):51-58, 2000.
4. A. Perrig, R. Szewczyk, V. Wen, D. Cullar, and J. D. Tygar, “Spins: Security protocols for sensor networks,” *in Proceedings of the 7th Annual ACM/IEEE Internation Conference on Mobile Computing and Networking (MobiCom)*, Rome, Italy, July 2001, pp. 189–199.
5. L. Yuan and G. Qu, “Design Space Exploration for Energy-Efficient Secure Sensor Network”, *ASAP* 2002.
6. R. L. Rivest, A. Shamir, and L. Adelman, “A method for obtaining digital signatures and public-key cryptosystems”, *Communications of the ACM*, 21 (1978), pp. 120–126.
7. A. Shamir, Identity-based cryptosystems and signature schemes, in On Advances in cryptology, proceedings of CRYPTO 84, New York, NY, USA, 1985, Springer-Verlag New York, Inc., pp. 47–53.
8. Sushma, D. Nandal, V. Nandal, “Security Threats in Wireless Sensor Networks” *in IJCSMS International Journal of Computer Science & Management Studies*, Vol. 11, Issue 01, May 2011.
9. S. Zhu, S. Setia, S. Jajodia, “LEAP+: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks” *in 10th ACM Conference on Computer and Communications Security (CCS 03)*, pages 62–72, October 2003.
10. A. Liu, P. Ning, “TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks”, *International Conference on Information Processing in Sensor Networks*, 2008
11. M. El-Hadedy, D. Gligoroski, and S.J. Knapskog. “High performance implementation of a public key block cipher-mqq, for fpga platforms”, *International Conference on Reconfigurable Computing and FPGAs, 2008. ReConFig’08.,* pages 427–432.