2008

# A scalable security protocol for wireless sensor networks

Mohamed Watfa
*University of Wollongong*, mwatfa@uow.edu.au

Hiba Halabi
*American University of Beirut*

Marwa El-Ghali
*American University of Beirut*

## Publication Details

Watfa, M., El-Ghali, M. & Halabi, H. 2008, 'A scalable security protocol for wireless sensor networks', The 2008 International Conference on security and management (SAM '08), 8p.

# A Scalable Security Protocol for Wireless Sensor Networks

**Mohamed Watfa[1] , Marwa El-Ghali[1], and Hiba Halabi[1]**
[1]Department of Computer Science, American University of Beirut, Beirut, Lebanon

**Abstract -** *Since sensor nodes suffer from limited resources, in memory storage, computing power, energy capabilities, and transmission rates, available network security protocols are inadequate. Symmetric algorithms cannot provide the same degree of security as public-key algorithms, the fact of which has led us to devise a new algorithm, SHESP, which uses public keys within the limitations of sensor nodes. This paper aims to present a way to utilize existing public-key algorithms in the field of wireless sensor network security by dividing the network into clusters. Our algorithm supplies data confidentiality, node authentication and data integrity while remaining within acceptable memory, time and energy constraints. Also, an important feature we opted to establish, which was lacking in most security protocols, was enabling secure node-to-node communication, without the need to route through a distant base station. We provide theoretical as well as experimental evidence to validate our algorithm.*

**Keywords:** Sensor Networks, Energy Efficiency, Security Protocol, Public Keys, Clustering

## 1 Introduction and Related Work

Sensor nodes are deployed in harsh physical conditions, and are required to transmit sensitive data, so securing the data transmitted through WSNs is important to maintain the confidentiality of the data and authenticate the sensor nodes participating in the network. Most protocols opt to use symmetric cryptography, because it is less expensive than public key algorithms. It is possible to use a single global secret key that is shared by the base station and all the sensor nodes, but that becomes useless if a node is compromised by a malicious adversary, and the entire network can become vulnerable to attacks. In other scenarios, the base station shares a distinct pair-wise key with every sensor node in the network, but that requires a lot of overhead and usually does not scale well. Also, to enable node-to-node communication, nodes would have to resort to a pair-wise key sharing, which is not efficient since it requires the storage of (*n-1*) keys in each node in an *n*-node network, and the addition (or deletion) of nodes dynamically requires establishing (or removing) shared keys with all the nodes in the network. The main problem with using symmetric keys is their secure establishment between the nodes without being detected by an outside attacker, which would often require either pre-deployment actions to bootstrap the keys or the involvement of a base station to distribute the keys. Thus, symmetric algorithms cannot provide the same degree of security as public-key algorithms, the fact of which has led to a new trend in finding ways to use public keys within the limitations of sensor nodes. Our research aims to present a way to utilize existing public-key algorithms such as RSA, Diffie-Hellmann, and Elliptic Curve in the field of wireless sensor network security. By dividing the nodes in the network into clusters, asymmetric encryption and decryption can be applied in the nodes chosen to be clusterheads. The use of asymmetric cryptography permits the establishment of symmetrically secure channels between nodes and their clusterheads, as well as allowing node-to-node communication within the network. Our algorithm provides confidentiality, node authentication, and data integrity while remaining within acceptable memory, time and energy constraints. Most existing protocols utilize the computational simplicity of symmetric algorithms to implement security in Wireless Sensor Networks. The most prominent algorithm is SPINS [1], which was developed at Berkeley; it uses RC5 for encryption and depends on two building blocks: SNEP, to provide data confidentiality through encryption, authentication with Message Authentication Codes (MAC) and freshness by use of counters, and µTESLA, for authenticated broadcast. However, such protocols require pre-deployed keys which affect scalability and node-to-node communication can only be achieved by going through the base station. Another protocol was built at Berkeley to be used on top of TinyOS called TinySec [2], which is based on implementing link-layer security on the level of packets. This protocol provides a way for authenticating and encrypting packets, focusing on the use of an Initialization Vector to enhance the encryption technique, but does not put forth a new keying mechanism and instead implements security on a one-hop level. In the provided implementation, the researchers use a pre-deployed network-wide key. In terms of our research, this mechanism is not viable or efficient, and the level of security it can assure is questionable since the capture of a single node can break down the entire network.

Research attempts into utilizing public key cryptographic systems in wireless sensor networks began, and one suggested protocol to utilize public keys is built on top of TinySec under the name TinyPK [3]. It necessitates the presence of two entities other than the sensor nodes, a Certification Authority (CA) and an External Party (EP), having more powerful resources, and having a pair of public/private keys, while the CA’s public key is preconfigured into the nodes. The EP signs its public key with the CA's private key and sends it to the nodes, which can verify it and obtain the EP's public key. Nodes can then generate a session key, encrypt it with the EP's public key and share the session key with the EP, establishing symmetric security with the exchanged key. The issues with regard to this protocol is the need for two extra entities that have more functionalities to be able to maintain all the node's keys and manage the public key system, as these entities are not always available in regular sensor networks. Also, the protocol does not appropriately handle compromised private keys and node-to-node communication also relies on the EP as a middle-man. The most recent work in wireless sensor network security is [4] which is able to provide a better level of security than TinySec with lower energy consumption. PIKE [5] employs the use of intermediary nodes as trusted sources to securely establish shared keys. As for [6], the authors provide a secure and energy-efficient way of performing data aggregation in sensor networks.

The rest of this paper is divided as follows. Section 2 provides a detailed explanation of our algorithm, including our intended goals and design. The mathematical analysis of our protocol is in Section 3. Section 4 presents our simulation results and a comparison with the security protocol in TinyPK [3], and we conclude this paper in Section 5.

## 2 Design and Framework

We start with a brief abstract idea of the algorithm before going into the details of every phase. The protocol identifies three types of possible entities participating in the network: a Base Station, clusterheads, and regular nodes. The Base Station is assumed to be secure against all kinds of attacks and has much more powerful resources than other nodes in the network, which is a valid assumption used in other references such as [4][6]. Clusterheads are identical in hardware formation to all other nodes in the network, but their selection process depends on them having the most capabilities at that time (Section III.C.4). Each node is identified by a 2-byte ID, stored in it prior to deployment, along with a 4-byte authenticator, which is stored in the Base Station as well, to allow integrity checking for nodes.For the cryptography aspect of the protocol, we summarize here the general algorithms used, whereas the detailed analysis involved with each mechanism is analyzed later. As an overview, for symmetric operations, we adopted RC2, as it is fast and requires a 56-bit key only. As for encrypting or decrypting with the node's authenticator, the authenticator is doubled into 8 bytes and used as the key within an implementation of the DES mechanism. For the asymmetric encryption/ decryption processes, we opted to use Elliptic Curve algorithms, as opposed to RSA, because analysis showed that it does faster computations and can use a smaller-size key (112 bits) [2][7]. As for integrity checking, we use a MAC-like checksum for every transmitted packet, which amounts to 4 bytes; the data that this checksum is applied to varies with each type of packet depending on the required validation. This type of MAC is chosen because it is capable of providing the needed data validation without large packet overhead or increased source code requirements. Note that keys in the whole protocol are generated only once, and used for all transmissions after that. However, the algorithm might decide that new keys must be generated because of the change in a clusterhead, for example.

***1- Establishing the Key between the Clusterhead and Nodes in the cluster***: The agreed-upon fact is that public key algorithms are computationally expensive, and so energy-draining; thus, not all nodes can be expected to perform such operations, which would lead to great energy requirements and the death of nodes in relatively short time. Therefore, the communication within clusters is done by way of symmetric keys, known to be secure and require fewer computations. The restrictions imposed by regular symmetric protocols are the initial establishment of secret keys between any two nodes in a secure manner, as well as the need to store several keys in a single node to enable its communication with other nodes. The algorithm in this phase aims to solve these issues. Initially, the Base Station has a pair of public/private keys, and assuming that the clusters are established in the network, it broadcasts its public key to all the clusterheads. Then, each clusterhead has to establish a set of keys with the member nodes of its cluster. For each node, the clusterhead generates a random secret key, includes its authenticator, its own ID, and the ID of the node it wishes to use this secret key with. All this data is encrypted with the Base Station's public key, and if this key is not available in it, it can request the public key from the Base Station. The packet then is appended with the Base Station's ID, to allow intermediate nodes to forward it, and the necessary MAC, computed over the symmetric key, the CH's ID and authenticator, and the ID of the intended node (all data before encryption). The packet is broadcasted, since no routing protocol is in use here (even though a routing protocol can be added), and recipient nodes continue to broadcast until the packet reaches the Base Station. The Base Station then decrypts the data, using its private key, and computes the checksum to validate that the received data has not been tampered with. If the integrity check passes, the BS compares the received authenticator with the one listed with the sender's ID in its list of nodes. If the authenticator does not match, the message is discarded, as the sender is assumed to be insecure. When the sender is assured to be reliable, the Base Station forms a packet consisting of the key, its ID, and the sending CH's ID, encrypted with the receiving node's authenticator, and appends the recipient node's ID, also to allow the proper forwarding. A MAC is computed over this entire data and added to the packet. When the packet reaches the intended node, it decrypts it using its authenticator, and recognizes the ID of its clusterhead and the secret key that is to be used for all later communications with this clusterhead. Note that a node cannot communicate securely with other nodes unless the data is routed through the clusterhead; this will be further explained in a later section. By repeating these same steps for all the nodes declared to be members of a clusterhead's group, secret symmetric keys will be established between the clusterhead and these nodes, and are maintained in the clusterhead. The node, on the other hand, only has to maintain this single key to enable its communication with its clusterhead. For this phase, both asymmetric and symmetric algorithms are used. The CH encrypts the generated key with the Base Station's public key, and uses Elliptic Curve encryption on the data packet, whereas the Base Station decrypts using its private key and Elliptic Curve decryption. The generated key in the clusterhead is a symmetric key established by RC2 mechanism, which the clusterhead and the corresponding node can use for communication from that point on. As for the transmission of this key from the Base Station to the node, it is encrypted with the node's authenticator, listed in the Base Station and in the node itself since deployment, providing a symmetric mechanism that the node can decrypt with. The authenticator is used as a symmetric key using the DES encryption/decryption algorithm, thereby allowing both the Base Station and the node to apply it for the needed cryptographic operations.

***2- Establishing Communication between Two Clusterheads***: Clusterheads are the designated leaders in the disjoint groups within the network, so transmissions from and to the nodes in any cluster have to pass through the corresponding clusterheads. Thus, to enable secure communication between different clusters, we must enable secure communication between their clusterheads. For this case, symmetric keys cannot be used, because clusterheads are established dynamically, and may be rotated throughout the lifetime of the network. Because of the choice of clusterheads in the network, we allow these leaders to perform public key operations, not only with the Base Station (as in the first part of the protocol), but also with fellow clusterheads. Upon election, clusterheads generate a pair of public/private keys to use in the EC system, so when a clusterhead wishes to communicate with another, it requests the recipient's public key, encrypts any data to be sent with it, and broadcasts it on route to the receiving clusterhead. This latter can asymmetrically decrypt the data, and in doing so can now capture secure data from another Clusterhead.

***3- Establishing Communication between Two Nodes***: Now that the previous part enabled uncompromised exchange between clusterheads, this phase can explore the steps of putting forth a safe technique for node-to-node communication. For any two nodes $N_i$ and $N_{j,}$ wishing to share data, regardless of whether the two nodes are in the same cluster or not, the nodes have to transmit the data through their assigned clusterheads. First, $N_i$ encrypts its message with the symmetric key it shares with the clusterhead (determined in 1st part); note that this operation is computationally simple and non-expensive for a regular node. $N_i$ also encrypts its own ID and the ID of the recipient node, appends the clusterhead's ID, and computes a MAC over the entire packet. It then sends the encrypted message to the clusterhead – say $C_i$ – which checks to determine if the receiving node is part of this cluster or not. In case it is, the clusterhead must decrypt the message and re-encrypt it with the symmetric key appointed to the node that is to obtain this data. Otherwise, the clusterhead has to forward it to the leader of another group that contains the sought-after node. The sender side's clusterhead decrypts the data symmetrically, then sends an initial message containing the recipient node's ID. As in all broadcast transmissions, nodes continue to forward this message until it is stopped by a relevant node, which in this case is the clusterhead of the group including $N_j$. Once the remote clusterhead – call it $C_j$ – detects that the message is meant for a node in its vicinity (from the list of nodes in its cluster), it sends back its public key to $C_i$, allowing them to communicate. $C_i$ then encrypts the message from $N_i$ with $C_j$'s public key and transmits it in the same way. When it is delivered to $C_j$ , it decrypts it with its local private key, then encrypts it with the required node shared key and forwards the message to it, along with the initial sender's ID. Naturally, $N_j$ is now able to acquire the message by decrypting with its secret key.

***4- Clusterhead Hand-Off***: Several existing clustering protocols - e.g. LEACH [7] - advise that clusterheads should be rotated to maintain the highest level of battery power possible. For our protocol, any clustering algorithm can be used, but our concern is maintaining the security even during the clusterhead rotation. In general, for a clusterhead to give over its responsibility to another node in its neighborhood, it has to deliver its shared keys to the newly-chosen node. However, to keep up the level of security, these shared keys cannot be continuously used by all clusterheads as that implies the exposure of the keys to several nodes in the neighborhood endangering confidentiality. So, for safe clusterhead rotation to take place, we propose that the resigning CH send its list of node shared keys to the newly-chosen CH, encrypted with that CH's public key (CHs are required to generate public/private key pairs as soon as they are elected). The new clusterhead can decrypt this list but it cannot use them for its communications; instead, it generates a new list of secret keys for the nodes in the cluster (including the old CH), and then delivers them to their corresponding nodes encrypted with the old shared keys. Thus, the keys established by the resigning CH are used once by the new CH, only to inform the nodes of the new shared keys. Upon reception of these messages, the nodes can remove the old keys and replace them with the new, for these will be the ones used for communication from that point on. As for the old CH, which does not have a previous symmetric key, the new CH can send it a generated shared key encrypted with its public key, so it performs an asymmetric operation to get its new symmetric key, and proceeds to behave as a regular node in the group. In this way, the clusterhead has been rotated to maximize energy usage, and the rotation was done locally without any interference from the Base Station. The flowchart of SHESP is presented in ***Fig. 1***.

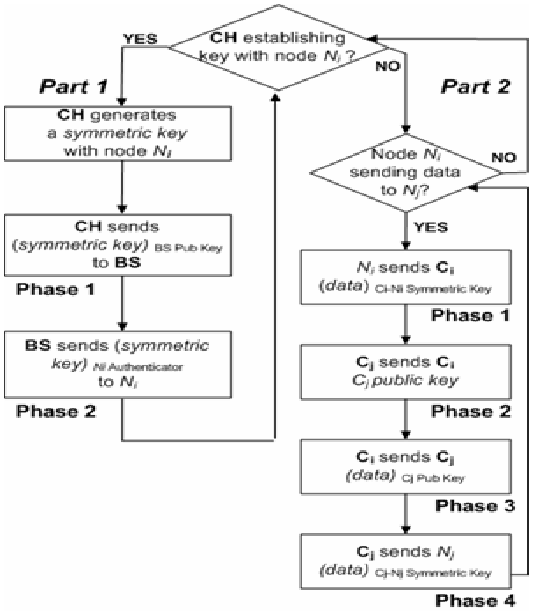
CH establishing key with node $N_i$ ?
YES
NO
Part 1
Part 2
CH generates a *symmetric key* with node $N_i$
CH sends (*symmetric key*) BS Pub Key to BS
Phase 1
BS sends (*symmetric key*) Ni Authenticator to $N_i$
Phase 2
Node $N_i$ sending data to $N_j$?
NO
YES
$N_i$ sends $C_i$ (*data*) Ci-Ni Symmetric Key
Phase 1
$C_j$ sends $C_i$ $C_j$ *public key*
Phase 2
$C_i$ sends $C_j$ (*data*) Cj Pub Key
Phase 3
$C_j$ sends $N_j$ (*data*) Cj-Nj Symmetric Key
Phase 4

**Fig 2.** Flowchart of the protocol–entities between brackets are encrypted with the subscript mechanism

# 3 Analysis of SHESP

In this section, we go through the phases of the security protocol, presenting detailed analysis of the algorithms used and the induced expenses in terms of storage, computational overhead, time, and energy. Basically, our aim is to evaluate the energy required for each of the primary operations: send, receive, and compute. The energy overhead is rated (in Mica2 motes) at 1μJ for sending one bit, 0.5μJ for receiving one bit, and 5 nJ (0.005 μJ) for processing a single instruction [8]. We calculate the memory, time, and thereby energy needed for each of the basic operations described in the phases above.

## Analysis of the Storage Requirements

Here, we evaluate what must be maintained in each node for use in the protocol, i.e. keys and information about other nodes in the network. The Base Station is the focus point of the network, and considering that it is capable of holding large amounts of information and has enough energy to maintain it for long periods of time, we're not concerned with minimizing expenditures on the Base Station's side. The Base Station saves the IDs of all the nodes in the network and their corresponding authenticators, but it is not required to store any keys related to the nodes, since it uses their authenticators as symmetric keys when it needs secure communication with them. So, for an *n*-node network, it will need to maintain (2+4) *n* bytes of IDs and authenticators (2-byte IDs, 4-byte authenticators), in addition to its pair of public/private keys, which are 44 bytes each (88 bytes in ECC). The clusterhead is also required to maintain information about the nodes in its cluster, namely their IDs and the generated secret keys. The secret keys generated by RC2 are 56 bits long, so for a neighborhood of *k* nodes, each clusterhead has storage of (2+7) *k* bytes (2-byte IDs, 7-byte keys), in addition to its own ID, authenticator and public/private keys for a total of 94 bytes (2 + 4 + 88). As for the nodes, they are only requested to maintain their own IDs, authenticators, and shared key with their respective clusterheads, resulting in a total of 13 bytes (2 + 4 + 7) in each node. Obviously, the storage requirements are within reason and capabilities of the nodes, even in the case of clusterheads, because of the assumption that the number of neighbors in a single cluster is limited.

## Analysis of the Security Implications

The algorithm as applied ensures data confidentiality, authenticity, and integrity as well as resilience to some attacks against the security of the network.

1- ***Data Confidentiality***: This metric is ensured through the keying mechanism used and the use of strong encryption algorithms. In the first phase, the symmetric key established between every node and its clusterhead cannot be detected. The packet transmitted from a clusterhead to the base station is encrypted with the base station's public key and the only way to recover the packet is through the base station's private key which is only maintained by the base station itself. The packet is then delivered from the base station to the node encrypted with the node's authenticator, which is pre-deployed and never transmitted across the network. As for the packet transmission from one clusterhead to another, it is also achieved with public-key cryptography using the receiving clusterhead's public key. Also, for a clusterhead to communicate with a node in its own cluster, the established symmetric key is used to encrypt messages, and since the key was exchanged safely, then this too is a secure operation. Data confidentiality is also ensured through the use of strong encryption algorithms combining both symmetric and asymmetric key cryptography.

2- ***Data Integrity***: Our algorithm ensures data integrity. Data integrity means that data has not been tampered with. This is ensured through the use of message authentication codes (MAC). The receiver drops a packet if the MAC generated over the packet is different than the MAC appended to the message. We choose a 4-byte MAC, which is proven to be robust against brute force attacks. With a 4-byte MAC, an attacker has to generate $2^{32}$ packets in the worst case, if he is to attempt to pose as a secure node and send data with a forged MAC. On average, he needs to send $2^{31}$ packets. It is important to realize that brute forcing a MAC cannot be done offline, since the only way to decide whether the brute force attack is successful is by sending the message to the node. With conventional networks, where the bandwidth can reach up to 1Mbps, this method is not a problem. However, with an average of 19.2 kbps in a sensor network, sending $2^{31}$ packets would require more than 20 months, a period during which battery-operated nodes would already be dead [2].

3- ***Data Authenticity***: Data authenticity requires that the data come from a proper and secure source. An attacker who would want to fabricate its own message should fabricate its associated MAC as well. However, the attacker lacks the required MAC-generating function. Also, considering that the MAC is generated over the plain-text data (prior to encryption), the MAC function cannot be deduced from inspecting messages during transit, making the MAC forgery even more difficult. Messages with invalid MACs would thus be exposed and dropped. We already argued that brute-forcing a MAC is not feasible.

4- ***Protection against Routing Attacks***: A common attack is for the attacker to change the destination of a packet in transit or its source. We already encrypt the id of the source node, so having an attacker alter it is not possible. Though the id of the destination node is not encrypted, yet a MAC is applied to the whole packet including the id of the destination node. Thus, if an attacker alters this field, the receiver would not accept the packet since the regenerated MAC and the one appended would not match.

5- ***Protection against Sybil Attacks***: In a Sybil attack, an attacker sends messages with different virtual identities and locations, thus claiming to be multiple nodes. Without cryptographic authentication, a receiver of a message cannot determine the true identity of its originator, and does not know how many of the claimed identities are truly existent and unique [9]. Our algorithm is resilient to Sybil attacks only after the clusters have been formed, since each node would already know its clusterhead, the only node with which it communicates, and each clusterhead has a list of the nodes that belong to it. Still an attacker can pose itself as a neighbor node since the node ids are public. However, no packet is sent from a node without a form of authentication. Since the attacker is neither aware of the key to be used for encryption nor the authentication mechanism used, all packets it sends would be neglected.

6- ***Protection against Selective Forwarding Attacks***: An attacker can use compromised nodes to launch selective forwarding attacks, in which compromised intermediate nodes selectively drop data traffic passing through them and thus severely jeopardize data availability. Our approach adopts a one-to-many data forwarding approach through the use of broadcasting. Thus, if an attacker drops a packet by way of selective forwarding, this same packet would still reach its destination via other routes passing through non-compromised nodes.

7- ***Analysis of the Mobility Implications***: Our protocol continues to work as expected in case of mobility. The only effect mobility imposes on our algorithm is the changes introduced for the clustering. In case a node changes its position and moves out of range from its assigned clusterhead, it should inform its clusterhead (before moving) of its decision, and the clusterhead can accordingly remove its shared key from the list. Afterwards, the node must join another cluster. With an advertisement message, it can alert another clusterhead to its presence, and then the clusterhead can simply generate a new shared key for it and establish the symmetric connection via the Base Station. Another case occurs if the clusterhead becomes mobile, which should not happen often because of the mobility metric integrated into the clusterhead election process, and this is handled in the same manner with which the clusterhead handles the reduction in energy. The node with the most capabilities within the cluster is selected to be the new clusterhead and can generate new keys and distribute them securely to all member nodes using their old symmetric keys. The original clusterhead becomes a regular node and can join any other cluster.

# 4 Simulation Results

To evaluate the performance of our security protocol, we implemented SHESP in JProwler, a Java-based simulator intended for sensor networks. We created a network of 100 nodes, divided into 10 clusters, and a base station, where each cluster contains a clusterhead and 9 non-clusterhead nodes. In our experiments, we simulate sending data between a source and a target node and study the effect of increasing the spatial separation between them on the incurred energy expenses and the end-to-end time delay. We assume only one sender transmitting a single data packet. The basic data message is 11 bytes long, and it is expanded into a packet of different sizes. We measure the energy consumed by the entire network till the data reaches the target node. In both graphs shown below, we do not consider the energy needed for the clusterheads to generate their public and private keys since this is done only once for every clusterhead. We consider five test cases. In the first case, the sending and receiving nodes belong to the same cluster. The remaining four have the nodes belonging to two different clusters with increasing spatial separation. The protocols used for comparison purposes are: (1) a network applying no security protocol, i.e. pure broadcasting, (2) a network applying the TinyPK security protocol [3].

The first no-security protocol helps in showing the overhead incurred purely from enabling encryption/decryption operations. The messages exchanged are exactly the same as those done in SHESP, so the forwarding overhead is similar except in packet sizes, which are also affected by cryptographic operations. As for TinyPK [3], we chose to compare this security protocol with ours since it also employs public-key operations to establish a symmetric secure channel between any node in the network and an external party (or a base station).

***Fig. 2*** displays the energy expenditure over all the nodes in the network relative to the spatial separation between the source and target nodes. The figure shows the variation in energy consumption between the different network setups. Our protocol induces a decrease in energy usage due to the optimized required cryptographic operations, so the witnessed decrease in energy relative to TinyPK will prolong the overall lifetime of the system. Compared to the non-security protocols, the minor increase in energy usage is expected due to the extra computational usage of encryption. We notice that the energy expenditure in SHESP over the entire network does not exceed 0.1J for a single data exchange. Considering that our network is made up of 100 nodes and each node has around 2.5J of energy as it starts out, then the network as a whole has 250J of available energy. If we were to measure the life expectancy of the sensor network based on these results, we can estimate that it can provide on the order of a thousand data exchanges, taking into consideration the energy needed to establish the clusters and the symmetric keys. Note that the energy results given are excluding those needed for establishing the public keys for the CHs and corresponding clusters, which is done once upon deployment. ***Fig. 3*** shows the total time delay incurred between the sending of the message at the source node and its reception at the target node. The results show that the delay is comparable to that induced in the non-secure approach. The difference in the delay can be accounted to the time needed to perform the cryptographic operations. However, the difference is greater compared to TinyPK and that shows the strength of SHESP. To measure the maximum throughput when using SHESP, we computed the total number of packets that could be sent in a 60 second time period. In this experiment, we configured a network of nodes so that multiple nodes would simultaneously transmit as rapidly as possible. Since the number of senders affects the channel utilization, we varied the number of senders. This allows us to characterize the throughput at different regimes. We sent 11 bytes of application data using the SHESP, TinyPK, and with no security protocol. We measured the number of packets that were successfully received. The results are in ***Fig. 4***. SHESP bandwidth characteristics are nearly identical to those without using a security protocol considering that the overhead resulting from clustering and establishing the keys are neglected which will only be done once upon deployment. With fewer senders, channel contention is less of an issue, so the packet length overhead does not affect the throughput. Comparing with TinyPK, the resulting bandwidth is better in SHESP due to the resulting overhead encountered in using TinyPK.

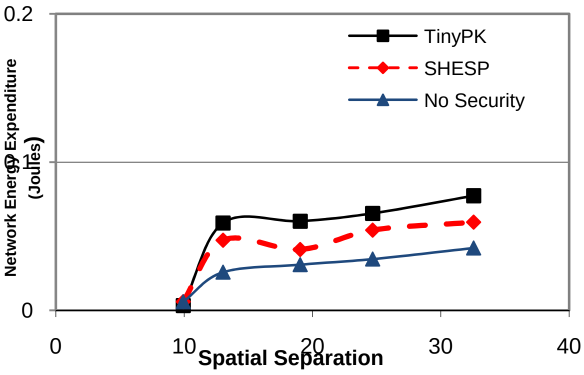

**Fig 2.** Total energy expenditure in the network relative to spatial separation between the source and target nodes.

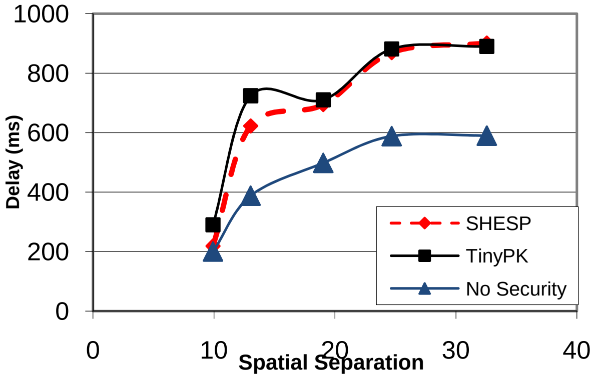

**Fig 3.** Total delay incurred between sending and receiving at the target node relative to the spatial separation between source and target nodes.

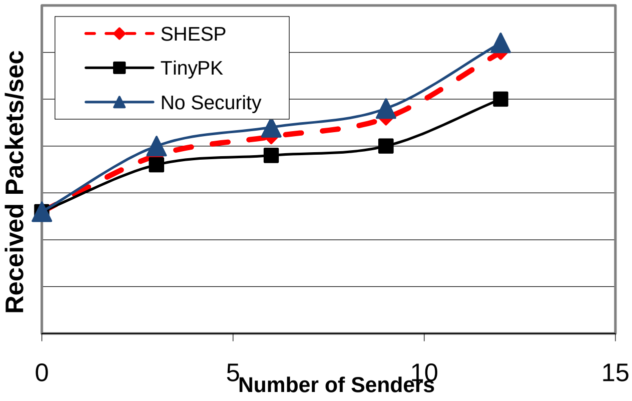

**Fig 4.** Bandwidth as a function of send-receive pairs

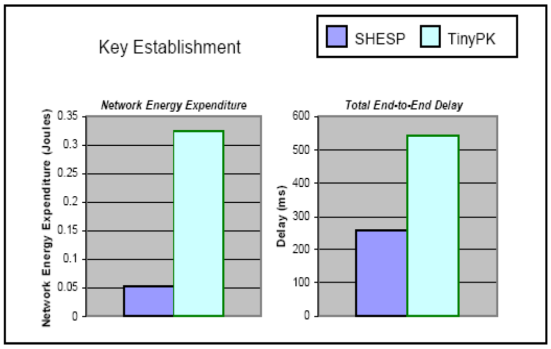

**Fig 5**. Total energy and delay incurred due to establishing the symmetric key in a single node in the network (in SHESP and TinyPK). The results are an average of 5 simulation runs.

Both SHESP and TinyPK undergo a key establishment phase during the initial network deployment, where each node is given a symmetric key to share either with its clusterhead (in SHESP) or the External Party (in TinyPK). We present in ***Fig. 5*** the energy and time consumed to establish the symmetric key in a single node in the network. Here, we notice the difference between our protocol and TinyPK: SHESP shows dramatic savings both in energy consumption and time. This is mostly due to the adopted asymmetric algorithm in our protocol, where Elliptic Curve has been proven to be superior to RSA (algorithm used in TinyPK), in both computation time and energy.

TinyPK establishes node-to-node communication by routing the packet through the External Party, which introduces unnecessary delay when the communicating nodes are in the same vicinity. An important primitive in securing sensor networks is data integrity, which ensures that data has not been tampered with. Security practices have shown that using encryption without authentication is insecure [2]. SHESP provides high data integrity in all cases and for all packets in transit. However, TinyPK does not ensure data integrity in packets sent from node to node (through the EP), nor in packets from a node to the EP informing it of the symmetric key to be established, since it does not enforce the use of any authentication code. Thus, the symmetric key can be easily altered by an attacker, and the symmetric keys maintained at the node and the EP would no longer match. This would impose time delays and energy waste before the error can be caught, and the node would have to send the packet all over again to the EP which is always exposed to the same attack. The only packet by which TinyPK includes a MAC to enforce data integrity is when the EP sends a node its public key. This clearly violates what most papers in the field of security argue as the most important primitive in sensor networks. SHESP provides data authenticity (ensuring secure source) by applying a MAC on the packet before encryption, making it more efficient. However, the current implementation of TinyPK does not ensure data authenticity.

# 4 Conclusions

The protocol proposed in this paper provides the ability to employ the benefits of both symmetric and asymmetric cryptographic operations to establish an acceptable level of security in wireless sensor networks. The main purpose was to make use of public key cryptography to put in place a secure symmetric exchange of data, thus providing security at a reasonable cost. The division of the network into clusters is also essential in making the protocol more energy-efficient, more scalable, and more resilient to attacks.

# 5 References

[1] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J.D. Tygar. SPINS: Security Protocols for Sensor Networks. Proceedings of *MobiCom '01*, Rome, Italy, July 2001, Vol. 8 pp. 189-199.

[2] C. Karlof, N. Sastry, D. Wagner. TinySec: A Link Layer Security Architecture for Wireless Sensor Networks. Proceedings of the *2nd ACM Conference on Embedded Networked Sensor Systems (SenSys '04)*, Baltimore, Maryland, November 2004, pp. 162-175.

[3] R. Watro, D. Kong, S. Cuti, C. Gardiner, C. Lynn, P. Kruus. TinyPK: Securing Sensor Networks with Public Key Technology. Proceedings of the *2nd ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '04)*, Washington, DC, USA, October 2004, pp. 59-64.

[4] M. Luk, G. Mezzour, A. Perrig, V. Gligor. MiniSec: A Secure Sensor Network Communication Architecture. Proceedings of the *6th international conference on Information Processing in Sensor Networks* (***IPSN '07***), Cambridge, Massachusetts, USA, April 2007, pp. 479-488.

[5] H. Chan, A. Perrig. PIKE: Peer Intermediaries for Key Establishment in Sensor Networks. Proceedings of *INFOCOM 2005, the 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, Miami, Florida, USA, March 2005, Vol. 1 pp.524-535.

[6] H. Chan, A. Perrig, B. Przydatek, D. Song. SIA: Secure Information Aggregation in Sensor Networks. Proceedings of the *1st International Conference on Embedded Networked Sensor Systems (SenSys '03)*, Los Angeles, California, USA, November 2003, pp. 255-265.

[7] W. Heinzelman, A. Chandrakasan, H. Balakrishnan. An Application-Specific Protocol Architecture for Wireless Microsensor Networks. In *IEEE Transactions on Wireless Communications*, Vol. 1, No. 4, 2002, pp. 660-670.

[8] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister. System Architecture Directions for Networked Sensors. Proceedings of the *9th International Conference on Architectural Support for Program Languages and Operating Systems (ASPLOS '00)*, Cambridge, USA, pp. 93-104.

[9] D. Wood, L. Fang, A. Stankovic, T. He. SIGF: A Family of Configurable, Security Routing Protocols for Wireless Sensor Networks. Proceedings of the *4thACM Workshop on Security of ad hoc and sensor networks*, Alexandria, Virginia, USA, October 2006, pp. 35-48.