

# Practical Algorithm for Data Security (PADS) in Wireless Sensor Networks

Julia Albath and Sanjay Madria  
Department of Computer Science  
University of Missouri-Rolla  
jgadkc[at]umr.edu, madrias[at]umr.edu

## Abstract

*When data are generated in sensor networks, high-speed data streams travel through the network. Traditional security approaches are often unable to keep up with the rates of the streams or they introduce overhead, which shortens the life of the network. The approach proposed in this paper is one that solves the problems posed above. By embedding a one-time pad, the actual value is distorted enough to make any information gleaned from eavesdropping useless to an attacker. The use of the one-time pad ensures that the data were indeed received from a particular sensor, and it gives adequate protection against injected messages. The simulation shows this approach provides security with negligible overhead while the throughput is similar to the same network without security.*

## 1 Introduction

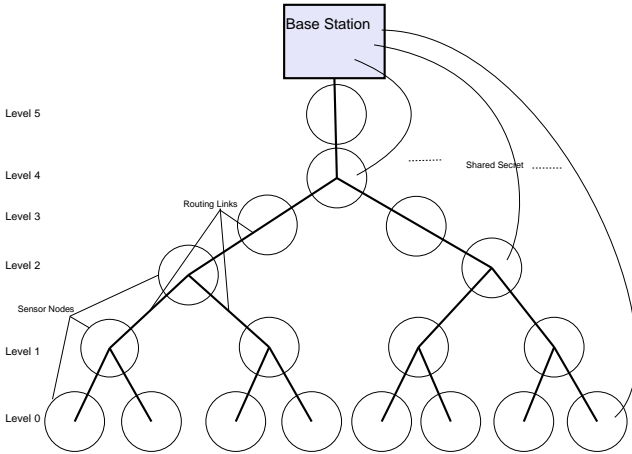
There are many possible applications of sensor networks from environmental monitoring to applications in military and homeland security [3]. Because radio communication is the most expensive operation in terms of energy usage, research has focused on finding ways to conserve energy [6]. Until recently, data security has received little consideration. In order for ubiquitous computing to become a reality, system security and privacy protection need to be assured. Securing data streams in sensor networks is important because traditional encryption and authentication protocols such as TinySec are often unable to keep up with high stream rates, and they deplete the network of energy too quickly [8].

The challenge in applying security to wireless sensor networks lies in the need to balance data integrity, confidentiality, and availability as well as preserving constraint energy resources. In sensor networks where radios are used for communication, an attacker can easily acquire data by eavesdropping anywhere the signal is transmitted. New security devices, methods, and approaches that safeguard sen-

sitive data are needed. Among the current research directions are efficient routing protocols [2], security schemes [8], and data management tools such as aggregation and data fusion.

Previous works in data security include implementing link layer security for sensor networks and public key cryptography using elliptic curves [8, 13]. These existing security protocols must be adapted to work in sensor networks because sensor networks have constraints on energy, communication and computation. Many other security protocols cannot be adapted for sensor networks and thus our society requires new ways of providing security. In [17], the authors consider the case of resilient rights protection of data streams through watermarks. The protocol provides a way identify the original owner of a data stream. Given that the stream is available, this technique can be used to determine whether the integrity of the data has been compromised; however no protection against eavesdropping is provided. TinySec provides security through traditional linklayer encryption and authentication schemes using the RC5 or SkipJack ciphers [8]. As with any cryptographic security scheme, TinySec increases the payload size of the packet, which results in increased energy consumption. In [14], the authors introduce SPINS, a three-part approach that provides authenticated streaming broadcast, data confidentiality, two-party data authentication, and data freshness as well as an authenticated routing protocol. The SNEP protocol of the SPINS suite of protocols provides data confidentiality. SNEP has low communication overhead by only adding 8 bytes to the packet. The rates at which data are produced may be much faster than the rates at which a security scheme can process and secure the data. While there is a need to tightly encrypt some type of messages, in which case a reduction in availability is often acceptable, other types of data may only require a light security scheme but demand high availability. Many security schemes have considered different approaches for messages in different levels in the network, but have not considered different levels of security depending on the message type.

This paper presents a method to provide protection



**Figure 1. Network Architecture**

against passive eavesdropping by employing confidential transmissions of data messages. For each transmission, a one-time pad (OTP) is created. A secret key and the MAC calculated over the data are used to create the OTP. When the OTP is combined with the data, data encryption and data integrity are achieved at the same time. The complete algorithmic details are given in Section 4. The PADS protocol only adds 4 bytes, making it even more efficient than SNEP. Just as with SNEP, PADS also provides data confidentiality and semantic security. Semantic security implies that eavesdroppers cannot infer the data even if they see the same data encrypted several times. The PADS protocol has dynamic key sizes and the ability to change the number of bytes to be encrypted. By using one-to-one routing and pair-wise shared keys, data integrity is guaranteed.

At the worst, the algorithms run in linear time on the size of the reading, regardless of the size of the network. The correctness of the algorithms was proven and the theoretical analysis of the energy usage shows that the application is appropriate for sensor networks. Simulations have shown this protocol to be suitable for sensor networks. Refer to Section 6 where the simulation results are provided. The simulation shows that the overhead due to the algorithm is negligible, and the performance of a network with the protocol is similar to a network without additional security. Compared with a network using encryption and authentication using TinySec, the performance of a network using this study's protocol is much better in terms of throughput, delay and energy consumption per message received at the base station.

## 2 Related Work

Researchers have investigated securing data in wireless sensor networks in order to ensure that the basestation can

trust the answers it receives. The authors in [15] propose the use of interactive proofs to force the aggregator to show that the answer previously provided is a good approximation of the true value. An aggregator is a node that applies an aggregate function such as a sum or average to all the data received from its child nodes. The authors make the assumption that each node has a unique identifier and shares a secret cryptographic key with the base station as well as with the aggregator. All types of stealthy attacks by the aggregator are considered. A stealthy attack is an attack where the aggregator wants the base station to accept results that are different from the true value, while at the same time evading detection. The authors want to guarantee that if the base station accepts a value from the aggregator, then the value is close to the actual value with high probability. The aggregate-commit-prove (ACP) protocol works as follows. The aggregator collects the data and locally computes the aggregation results. The aggregator can verify the authenticity of each value because of the key shared with each node. This prevents impostors, but will not prevent falsified results by a corrupt sensor. Next, the aggregator calculates a commitment, which during the proving phase will show that the aggregator used the values provided by the sensor to calculate the aggregation value. Whenever the home server requests, the aggregator proves to the home server that the result is valid. The home server checks to see if the aggregator is cheating, in the sense that the aggregation result is or is not close to the correct result aggregated from the committed data. In contrast, our work concentrates on securing the data on the routing path. PADS provides security against an eavesdropper and would work well in conjunction with ACP to provide confidentiality and integrity in wireless sensor networks.

The work in [17] is a novel idea to provide copyright protection to data stream owners and authorized users. Consider the case where a stream is generated and safely transmitted from the sensors to the base station. A watermark is applied to the stream at the base station. The data are then transmitted to an authorized user. The owner and authorized users need a way to show that the data were generated by them and they want to prove that the stream was illegally obtained by the attacker. One commonly accepted way to prove ownership is the use of embedded watermarks. This technique works by embedding a watermark bit into major extremes, which are extremes that will survive any uniform sampling. Because the watermark bits are embedded in the major extremes, they can then be extracted and used to show copyright and ownership can be established. Extremes are chosen because even after alteration and aggregation, most extremes will be recoverable and able to ensure an overlap when rebuilding the watermark. During detection, all extremes, not just the major ones, are identified, and the same selection criteria as during embedding are used to identify

the potential watermark recipients. For each selected extreme, its corresponding 1-bit watermark is extracted and the global watermark is gradually re-constructed. Similarly, PADS could be considered a watermark, which is embedded in the data, the difference being that a new watermark is generated for each data item.

Another example of stream security is [18]. The author explores message authentication in sensor networks. He compares several authentication possibilities: end-to-end, hop-by-hop, and physical and virtual multipath authentication. While end-to-end authentication provides the greatest level of security, hop-by-hop authentication can be implemented with relatively little overhead, but it provides security only against a very restricted attacker. The author shows that physical multicast authentication provides a good intermediate level of security, and that virtual multicast authentication retains similar properties as physical multicast authentication but also has reduced energy demands.

In contrast to our work on securing messages Zhu et al. introduce a protocol that provides protection against messages injected by an attacker with the goal to deceive the base station or to shorten the lifetime of the network by depleting the resources of the sensors in the network [21]. The authors present an interleaved hop-by-hop authentication scheme, which guarantees that the base station will detect injected messages as long as fewer than  $t$  nodes are compromised. When nodes are compromised, false messages are created and injected in the network.

### 3 System Model and Architecture

Sensor networks can be modeled as graphs. Let  $G = (V, E)$  be an undirected graph with a set of  $n$  nodes  $V \subset \mathbb{R}^2$  in the Euclidean plane and a set of  $m$  edges  $E \subset V^2$  [16]. The vertices in this model represent the nodes in the sensor network and the edges represent communication links between the nodes. While the network could be modeled as a directed graph, only an undirected graph is considered and so only bi-directional links are modeled. This study defines a round as a time interval in which each node is to send its gathered data to the base station. In some cases, a sensor will continuously generate, process, and send data as it monitors its environment.

A data stream is defined as an infinite set of values generated by a sensor. Furthermore, each sensor node generates one data packet each round. While in many cases the timestamp will be preserved, the notion of time is only introduced to show the sequential nature of such data streams. The timestamp information may or may not be routed with the data. In many cases the timestamps lose their meaning and as such, should not be considered as important.

A sensor network most often generates numeric data, but at times may produce data from other domains. However,

this study consider all data only at the level of a string. All data, whether numeric or not, are just a sequence of ones and zeros, and thus this research technique will work equivalently on numeric or other data.

The following assumptions are made. Each node shares a secret key with the base station. This key is used to generate a new key for every transmission using a key derivation function. This approach provides protection against an attacker who can overhear transmissions and who can also inject false messages and nodes. The sensor nodes can communicate with a powerful base station via relay nodes. A simple multi-hop routing protocol is used [12] as shown in Figure 1. This study assumes that the base station and the sensors are time synchronized, which can be handled as described in [4]. Here sensor nodes synchronize with their parent nodes in a hierarchical structure until a synchronization chain has been build up to the root or base station. In cases where the sensor send data at periodic intervals, an explicit synchronization exists given that each packet has a sequential packet number. In aperiodic sensing of events the protocol in [4] can be use to keep the network synchronized.

**Definition 3.1** *A one-time pad is a sequence of bits that is XOR to the reading in order to provide protection against eavesdropping.*

**Definition 3.2**  $k_i$  is the secret master key shared between node  $i$  and the base station. The value of  $k_i$  can be updated similar to updating keys in [20] and [14].

**Definition 3.3**  $k_{ij}$  is the  $j^{\text{th}}$  secret key shared between node  $i$  and the base station. The key is periodically updated to guarantee freshness.

**Definition 3.4**  $x_{ij}$  is the data value  $x$  generated at time  $j$  by node  $i$ .  $x_i$  is the current value of  $x$  generated by node  $i$  and  $p(x_{ij})$  is the packet generated by node  $i$  at time  $j$ , which contains the sensor value  $x$ .

**Definition 3.5**  $b(x)$  is the bit size of the value  $x$ .  $b(MAC)$  is the bit size of the Message Authentication Code (MAC).

**Definition 3.6**  $\alpha$  determines the length of the subsequence of the MAC.  $\beta$  determines the starting location of the aforementioned subsequence of the MAC.  $\alpha, \beta \in \mathbb{Z}$ .

### 4 Algorithms

The one-time pad is constructed by the nodes using only information contained within the data packet and the secret key shared between the sender and the basestation. First, the MAC (message authentication code) is calculated based on the sensor reading. Next, a one-time pad is constructed from the MAC and the key shared with the receiver. The MAC

is attached to the sensor reading. The variables  $\alpha$  and  $\beta$  are calculated based on the  $MAC$ , and a subsequence of the  $MAC$  is extracted and used to secure the sensor reading. The receiving node uses the attached  $MAC$ , removes the encryption, and calculates its own  $MAC$ . To show that the message has not been tampered with, the two  $MAC$ s are compared for equality.

#### 4.1 Embedding of a One-time Pad

---

##### Algorithm 1 Basic Embedding Algorithm: **embed**

---

**Require:**  $k_{ij}$ , packet  $p(x_i)$  of sensor value  $x_i$

**Ensure:**  $p(x_i)$  XOR with the one-time pad

- 1: calculate  $MAC$  over  $p(x_i)$  (excluding routing information, see Figure 2) and save in  $\widetilde{p}(x_i)$
  - 2:  $k_{time} = k_{ij}$  modified and time synched
  - 3:  $\alpha = k_{time} \bmod b(MAC) + 1$
  - 4:  $\beta = k_{time} \bmod (b(MAC) - \alpha - 1)$
  - 5:  $\widetilde{temp\_pad} =$  substring of  $MAC$  starting at  $\beta$  for  $\alpha$  bits
  - 6:  $\widetilde{pad} =$  Append  $\widetilde{temp\_pad}$  to  $\widetilde{pad}$  until  $\widetilde{pad}$  is of the same length as the data
  - 7:  $\widetilde{x}_i = x_i \text{ XOR } \widetilde{pad}$
  - 8: Replace  $x_i$  with  $\widetilde{x}_i$  in  $p(x_i)$  and send
- 

Refer to Algorithm 1 for details on the embedding algorithm. This algorithm calculates a  $MAC$  over the static part of the packet. Multi-hop routing needs to change the packet header to properly route the packet, so only the part of the packet that does not change is used, as shown in Figure 2. The calculated  $MAC$  is appended to the data and the secret key shared between the sender and the receiver is used to create a time synched key. For example, at time  $t$ , every  $t^{th}$  bit of the key is dropped. This ensures that an attacker has to be time synched with the network in order to break the encryption. Time synchronization is handled similarly to [19]. It is important to note that our algorithm will work without time synchronization, but time synchronization provides for dynamic adjustment to the keys, thus providing added security. The next step is to calculate the length of the substring and the starting position of the substring. This study calculates  $\alpha$  by taking the secret key modulo the length of the  $MAC$  in bits.  $\beta$  is calculated by taking the key modulo the length of the  $MAC$  minus  $\alpha$  plus 1; this ensures that the substring will exist. The substring is of length  $\alpha$  bits, so the last  $\alpha$  bits for the source string cannot be the starting point of the substring. The final steps involve generating the one-time pad by repeatedly concatenating the substring to a target string until the target string is of the same length as the static portion of the packet to be secured. In most cases that will be the payload plus some portion of the packet header as shown in Figure 2. The study then encrypts the data that

are transmitted by XORing the generated one-time pad to the value  $x_i$  and then transmit the packet.

#### 4.2 Detection of a One-time Pad

The detection process works very similarly to the embedding process. This is necessary in order to find the embedded pad, remove it, and return to the original sensor value. The detection and removal of the one-time pad is done by the base station, because only the base station shares the secret key with the embedding sensor node.

---

##### Algorithm 2 Basic Detection Algorithm: **wm.detect**

---

**Require:**  $k_{ij}$  packet  $p(\widetilde{x}_i)$  of sensor value

**Ensure:**  $p(\widetilde{x}_i)$  the original packet with the original sensor value  $x_i$

- 1: Take the  $MAC$  calculated by the sender
  - 2:  $k_{time} = k_{ij}$  modified and time synched
  - 3:  $\alpha = k_{time} \bmod b(MAC) + 1$
  - 4:  $\beta = k_{time} \bmod (b(MAC) - \alpha + 1)$
  - 5:  $\widetilde{temp\_pad} =$  substring of  $MAC$  starting at  $\beta$  for  $\alpha$  bits
  - 6:  $\widetilde{pad} =$  repeatedly concatenate  $\widetilde{temp\_pad}$  to  $\widetilde{pad}$  until  $\widetilde{pad}$  is of the same length as the static portion of the packet.
  - 7:  $x_i = \widetilde{x}_i \text{ XOR } \widetilde{pad}$
  - 8: calculate  $MAC$  over  $p(x_i)$  and compare to  $MAC$  received with packet
  - 9: **if** The two  $MAC$ 's match **then**
  - 10:     Packet was not altered
  - 11: **else**
  - 12:     Packet was altered
  - 13: **end if**
- 

Because sensor networks communicate via wireless mediums, there exists a natural probability of collisions, dropped packets, or corrupted packets. Any of these problems will be detected by this protocol. If the  $MAC$  or the payload of a packet is corrupted, then the  $MAC$  calculated by the receiver will not match the one from the sender.

In order to retrieve the original data, the detection process described in Algorithm 2 is used. This study uses the  $MAC$  calculated by the sender and stored in the packet to calculate the  $\widetilde{pad}$  and remove it from the packet. Then the  $MAC$  is calculated just like the sender, and the two  $MAC$ s are compared. If they match, then processing the packet continues.

#### 4.3 Example

The following is an example of Algorithm 1 executed at node X and Algorithm 2 executed at the base station, along with the intermediate steps. X senses an event, does the

Address (2) Bytes	Msg Type (1) Byte	Group ID (1) Byte	Data Length (1) Byte	Source Address (2) Bytes	Original Address (2) Bytes	Sequence Number (2) Bytes	Hop Count (1) Byte	Type (1) Byte	Reading (2) Bytes	Parent Address (2) Bytes	MAC (4) Bytes	CRC (2) Bytes

**Figure 2. Multihop packet structure. The fields shaded gray are protected by the MAC calculation.**

embedding, and transmits the data to the base station. Then the base station does the detection.

**Step 1: Embedding at Node X.**  $X$  senses an event and generates the value  $x_X = 00011110\ 10011011$ .  $X$  calculates the  $MAC$  with the secret key  $k_{ij}$  using SkipJack in the CBCMAC mode, as implemented by [8] over  $x_X$ .  $MAC_{x_X, k_{ij}} = 01100000011111001101111111001010$ .  
*Algorithm1:Line2*  $\alpha = MAC \bmod 32 + 1 = 00101001$

*Algorithm1:Line3*  $\beta = MAC \bmod (32 - \alpha + 1) = 2$

*Algorithm1:Line4*  $temp\_pad =$  subsequence of  $MAC$  starting at 29 for  $2 = 11$

*Algorithm1:Line5*  $\widetilde{pad} = 11111111\ 11111111$

*Algorithm1:Line6*  $x_i(old) = 00011110\ 10011011$   
 $x_i(new) = 11100001\ 01100100$

**Step 2: Detection at the base station.** BS receives the value of  $x_i = 11100001\ 01100100$  from node  $X$  with the stored  $MAC = 01100000\ 01111100\ 11011111\ 11001010$ .

*Algorithm2:Line2*  $\alpha = MAC \bmod 32 + 1 = 00101001$

*Algorithm2:Line3*  $\beta = MAC \bmod (32 - \alpha + 1) = 2$

*Algorithm2:Line4*  $temp\_pad =$  subsequence of  $MAC$  starting at 29 for  $2 = 11$

*Algorithm2:Line5*  $\widetilde{pad} = 11111111\ 11111111$

*Algorithm2:Line6*  $x_i(old) = 11100001\ 01100100$   
 $x_i(new) = 00011110\ 10011011$

*Algorithm2:Line7* BS calculates the  $MAC$  over  $x_i(new) = 01100000\ 01111100\ 11011111\ 11001010$ .

*Algorithm2:Line9*  $MAC$ 's match, data was not altered.

## 5 Security Analysis

### 5.1 Message Integrity and Authenticity

The message integrity of the one-time pad protocol is based on the security of the MAC. The MAC used in the simulations is a CBC-MAC implemented by TinySec. It uses a MAC of 4 bytes. Using a 4 byte MAC, there is a 1 in  $2^{32}$  chance that an attacker is able to create a MAC and have it be an exact match. Overall, after about  $2^{31}$  such attempts and attacker should have made a match. The assumption is that the MAC can only be verified by sending it to an authorized receiver and noting if the message is validated, the attacker would have to send  $2^{31}$  messages. In traditional networks such a number isn't much, however in sensor networks it will provide a level of security that is sufficient. Most radios channels in sensor networks operate at 19.2kb/s and it would take an attacker about 8 months to

send  $2^{31}$  messages of 23 bytes each. This exceeds the lifetime of most sensor networks. Additionally, our protocol uses a new key for each transmission, therefore an attacker cannot learn anything about future transmissions from the past.

### 5.2 Confidentiality

The security of the OTP depends on the security of the key  $k_{time}$ . Since a new key is generated at each transmission the security of the protocol depends on the security of the key derivation function (KDF) as describe in the IEEE Standard Specifications for Public-Key Cryptography [1]. Under the assumption that the KDF is secure the problem reduces to the ability of an attacker to randomly create a one-time pad and have it match. The size of the one-time pad is equal to the size of the data to be protected, we assume 5 bytes throughout this work. That is equal to a 1 in  $2^{64}$  chance of randomly having the right string or an average of  $2^{32}$  tries until a correct one is found. At almost 16 months that is well over the lifetime of the typical sensor network.

## 6 Analytical Analysis

### 6.1 Cost Analysis

The embedding and the detection algorithms are very similar. The calculation of the MAC using an appropriate algorithm takes  $O(b(x_i))$  time, where  $b(x_i)$  is the bit size of the input, not the size of the network. This study uses the Skipjack algorithm in the analysis. All other lines have complexity  $O(1)$ . Thus, the running time of the complete embedding and detection algorithms is  $O(b(x_i))$ .

While the program space is of concern in sensor programming, this research is primarily concerned with the size of the memory necessary to perform the computation depending on the input size  $b(x_i)$ . The algorithm neither increases nor decreases the size of the measured value. The memory requirements for any of the variables used in the algorithm is rather small; however, the requirements of the Skipjack algorithm are large in comparison. The memory requirements of the Skipjack algorithm are independent of the size of the input, but because it has been implemented for TinyOS as part of the TinySec project, it is known that the size is small enough to work [8].

## 6.2 Correctness

**Claim 6.1** *The algorithm correctly embeds a one-time pad in the sensed value.*

**Proof** A sensor value  $x_i$  of  $b(x_i) = 1$  is given. The calculation of the  $MAC$  in line 1 of Algorithm 1 results in a  $MAC$  of 4 bytes. The calculation of  $\alpha$  and  $\beta$  on lines 2 and 3 of algorithm 1 are as follows. The calculation of  $\alpha$  will result in a value of  $[1, b(MAC))$ . The calculation of  $\beta$  will result in a value of  $[0, b(MAC) - \alpha + 1)$ . Because  $\alpha$  is subtracted from the bitsize of the  $MAC$ , it is confirmed that the substring  $temp\_pad$  will in fact exist. Then  $temp\_pad$  is repeated until  $b(temp\_pad) = b(x_i)$ . Thus, the one-time pad  $\widetilde{pad}$  is the same length as the data, and by XOR  $\widetilde{pad}$  with the reading, the data are securely encrypted.

It is hypothesized that for all  $l$  less than  $m$ , such that  $|x_i| = l$  it implies that the one-time pad used to secure the reading is of the same length as the reading. In other words,  $\forall l, l \leq m, |x_i| = l \Rightarrow$  successful embedding of the one-time pad.

Now, it will be shown that  $|x_i| = m$  has proper embedding  $\Rightarrow |x_i| = m + 1$  has proper embedding. By adding an additional bit to  $x_i$ , the  $MAC$  calculated will be different, but the values of  $\alpha$  and  $\beta$  still result in a value of  $[1, b(MAC))$  and  $[0, b(MAC) - \alpha + 1)$ , respectively. The calculation of  $temp\_pad$  on line 4 of algorithm 1 will still result in a proper substring of the  $MAC$  and thus a proper one-time pad, and hence the embedding is done properly.

**Claim 6.2** *The algorithm detects the embedded one-time pad in a received value and restores the sensed value.*

**Proof** Again, the smallest permissible length of the values of  $x_i$  is one. A sensor value  $x_i$  with  $|x_i| = 1$  is given. The calculations of  $\alpha$  and  $\beta$  on lines 2 and 3 of Algorithm 2 using the  $MAC$  received as part of the packet result in the values of  $[1, b(MAC))$  and  $[0, b(MAC) - \alpha + 1)$ , respectively. If the  $MAC$  was not modified during transmission, then this guarantees that the same  $\alpha$  and  $\beta$  will be calculated. Naturally, it also implies that the same  $temp\_pad$  will be extracted. Because the size of the reading is fixed in the network, the same  $\widetilde{pad}$  will be generated and thus can successfully be removed from the packet. The calculation of the  $MAC$  is over the same data as at the embedding location, and it will only result in a successful outcome if the data were not modified during transmission.

It is hypothesized that for all  $l$  less than  $m$ , such that  $|x_i| = l$  it implies that the embedding can be found during detection and the original value can be restored. In other words,  $\forall l, l \leq m, |x_i| = l \Rightarrow$  successful detection and removal of the one-time pad.

Next it will be shown that  $|x_i| = m$  has proper embedding  $\Rightarrow |x_i| = m + 1$  has proper embedding. By adding

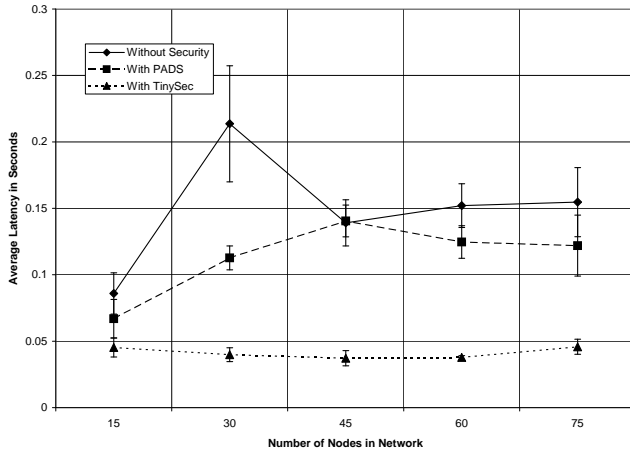
an additional bit to  $x_i$ , the  $MAC$  calculated will be different, but the values of  $\alpha$  and  $\beta$  will still result in a value of  $[1, b(MAC))$  and  $[0, b(MAC) - \alpha + 1)$ , respectively. The calculation of  $temp\_pad$  on line 4 of Algorithm 1 will still result in a proper substring of the  $MAC$  and thus a proper one-time pad. Thus, the embedding is done properly.

## 6.3 Energy Use Analysis

Law, Doumen, and Hartel showed that while there are some difference in the number of clock cycles an instruction will take, the differences are not statistically significant enough to matter [9]. Therefore, the computational complexity of an algorithm can be directly translated to energy consumption, assuming that the energy per CPU cycle is fixed. In [5] the authors showed that the Skipjack cipher uses 15925 cycles. The algorithms in the current research add an additional 20 instructions per algorithm. Because the average instruction takes 400 cycles, these algorithms add an additional 8000 cycles for a total of 23925 cycles for the application [11]. The MICA2 motes use 4 nJ per cycle [7]. Thus, the expected energy consumption for the embedding and detection algorithms is 95700 nJ. The average AA battery contains 1000 Joules. Because the MICA2 motes operates on two AA batteries, the energy consumption of both the embedding and detection algorithms is appropriate for sensor networks.

## 7 Simulation

PADS was simulated using the TinyOS operating system and its simulator TOSSIM [10]. It was compared to other protocols using a fixed topology for each simulation. In all simulations, every node in the network generates and sends a packet about every five seconds. The comparison included routing with non-secure AODV, with the PADS technique, and with TinySec [8]. Each simulation ran for a simulated 10 minutes, and for each network size the simulation was repeated five times. The average of the five simulations was used for the evaluation. For each set of simulations, five sizes of networks were compared: 15 nodes, 30 nodes, 45 nodes, 60 node and 75 nodes. Three different sets of simulations were performed. The first set compared same sized messages; all messages had a size of 23 bytes no matter the protocol. We ran simulations using the same size message to study the performance, as communication costs mainly depend on the size of the message. Since larger messages would require more energy to transmit any difference in performance can be attributed to sources other than message size. This allowed us to remove any doubt regarding what contribution the message size has on the performance. In order to achieve this, messages in non-secure AODV had payloads of seven bytes, PADS had payloads of three bytes



**Figure 3. Average Latency for Messages of 23 Bytes for Various Protocols**

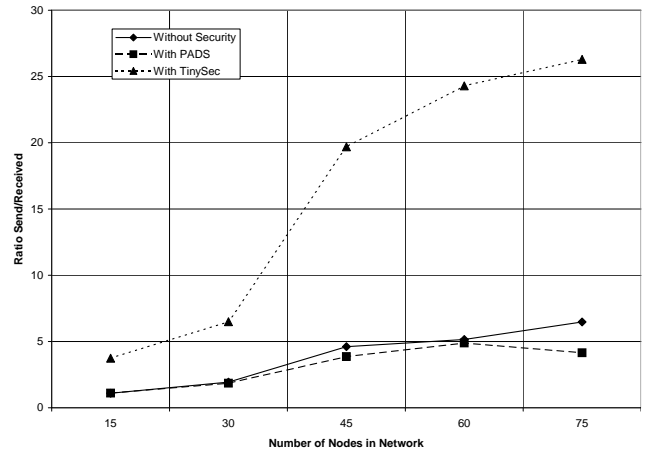
and TinySec used payloads of two bytes. In the second set of simulations we are using the three protocols using messages with a two byte payload resulting in messages of 18 bytes, 22 bytes and 23 bytes for non-secure AODV, PADS and TinySec, respectively. In reality, an application will dictate the size of the payload and any underlying protocols such as AODV or TinySec/PADS will add its necessary fields, adjusting the message size accordingly. Hence the seconds set of simulations correspond to an application of the protocols. The last set of experiments compared PADS against itself at different payload sizes, 2 bytes, 4 bytes and 8 bytes. This allowed us the measure the effect of payload size on the performance of the protocol. The latency (average time it takes a packet to reach the base station), the throughput of bits per second (bps), and the average power used per node were evaluated. We calculated the standard error for each measure. Two measures are considered to be statistically different if their error bars do not overlap.

## 7.1 Comparison of Protocols with total message size of 23 bytes

### 7.1.1 Latency

It is important to know what delay the addition of creating and applying the one-time pad adds. Each packet was timestamped when sent in the application layer at the sending node and again when received in the application layer at node 0. The time it took for a packet to travel from the application layer in the sending node to the receiving point in the application layer was measured. The difference is the travel time of a packet.

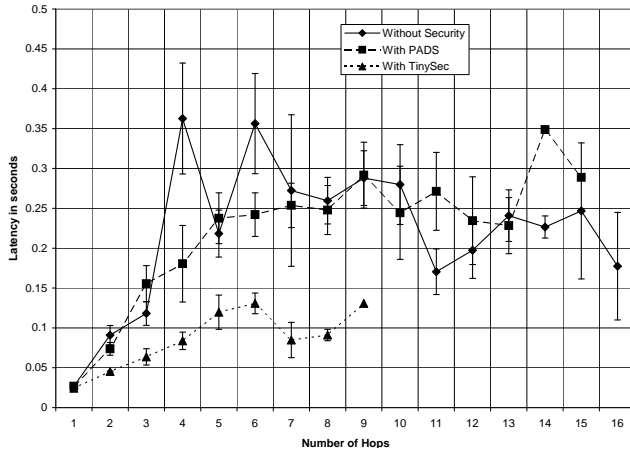
Figure 3 shows the average latency per packet for various network sizes. It is easy to see that the latencies for non-secure AODV and the PADS protocol are similar. The



**Figure 4. Ratio Send/Received Messages for Messages of 23 Bytes for Various Protocols**

only statistically significant difference between PADS and non-secure AODV is at 30 nodes. One of the simulations for the non-secure network had congestion, resulting in several hundred messages with latencies well above 1 second, while the average for the rest of the simulations was closer to 0.14 seconds. This caused the average latency to be so high. However, it seems that TinySec has an average latency that is much better than either of the two other protocols. A closer look at the data revealed that TinySec, on average sends 16 times as many messages as are being received. Figure 4 shows the ratio of Sent Message to Received Messages, the failure rate, for the three protocols. It is worthy to note, that the ratio is similar for PADS and for networks without security. TinySec, on the other hand quickly reaches ratios of more than 5 messages sent for each message received. The reason is that TinySec, due to the lengthy encryption/decryption processes, at each node drops messages, especially at nodes which are utilized as relay nodes.

Another important metric is the average latency per total number of hops traveled. It is expected that the latency increases as a message has to travel multiple hops. Figure 5 shows the average latency per hop. Note how the slope of the increase is similar for PADS and networks without security. At 4 and 6 hops, respectively, the simulations without security resulted in a lower number of received messages, which inflates the average latency for those hop counts. At 11 the network without security had very little congestion resulting in about 15% of the messages to have average latencies below 0.1 seconds. This reduced the overall average latency for messages with that hop count. At 14 hops, PADS had a particularly congested network, resulting in a higher than expected average latency. TinySec seems to have an average per hop latency much better than either of the other proto-



**Figure 5. Per Hop Latency for Various Protocols**

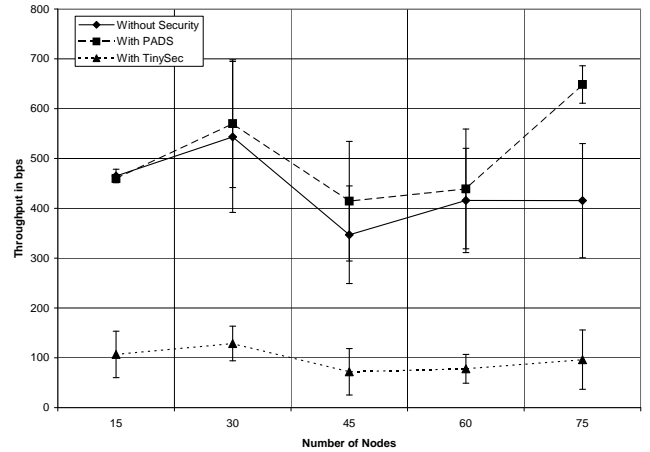
cols. Considering the low success rate of received messages as well as the fact that despite using a fixed topology in each simulation, there are no simulations using TinySec with hop counts greater than 9 hops, the overall performance of TinySec is much worse than PADS.

### 7.1.2 Throughput

Another important consideration in a sensor network is the throughput. Because the usefulness of sensor networks lies in independently sensing and sending large amounts of data, any technique must be able to sustain high data rates. Again, PADS is compared to no security and TinySec encryption and authentication. As is evident from Figure 6, the throughput of PADS is close to that of a network without security. The only noteworthy difference is for networks of 75 nodes. Of the 5 simulations, 2 had very high failure rates, resulting in only a few hundred received messages when almost 9000 were sent. This causes a lower than expected throughput. On the other hand, TinySec encryption and authentication results in greatly reduced throughput.

### 7.1.3 Energy Cost

The amount of energy available in sensor networks is very limited. Any sensor network protocol needs to be energy aware. Since the energy available to sensor networks is limited, the energy consumption of any application is critically important. PADS is compared to sensing without security and to TinySec using encryption and authentication. During simulation each network sends different amounts of messages. In order to have an accurate picture of the energy usage in a network and what the true cost of sending data message is, we calculate the average amount of energy used per data message received. Figure 7 shows the average amount



**Figure 6. Average Throughput in bps for Messages of 23 Bytes for Various Protocols**

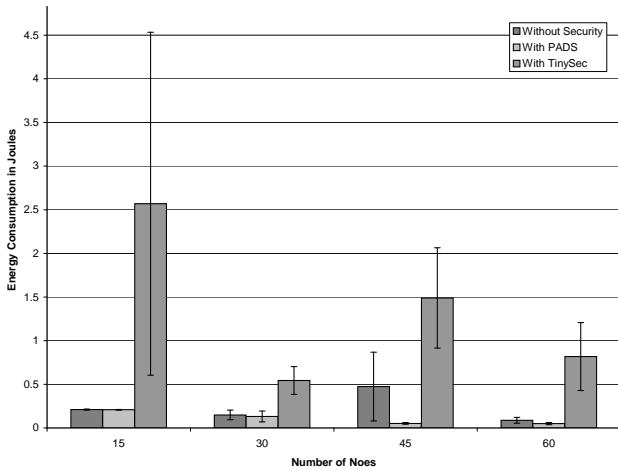
of energy used per node per message send for various network sizes. The difference for networks with 45 nodes is because the networks without security had high failure rates on average resulting in a high energy consumption per message received. It is important to note, that TinySec always uses significantly more power than the other protocols no matter the network sizes. The reason for the much higher power consumption per message for TinySec is that we are measuring the power used per message received at the base station and the failure rate for TinySec is much higher than the other networks resulting in a higher energy consumption per message received.

## 7.2 Comparison of Protocols with 2 Byte Payload

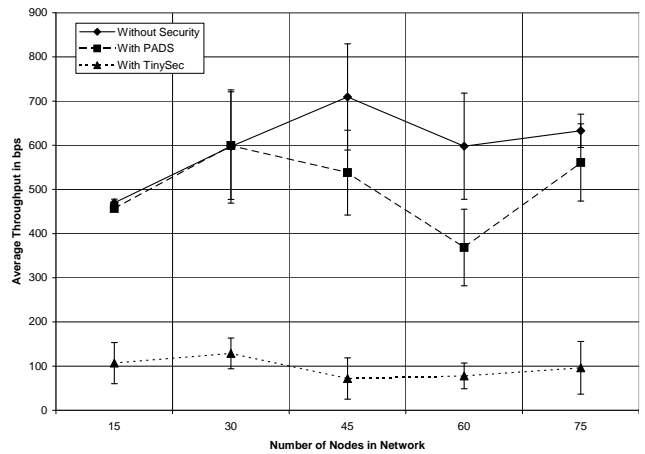
In most applications of sensor networks, the payload remains stable and depending on other protocols used, such as AODV for multi-hop routing and TinySec for security, the total message size may vary. In this set of experiments, each message generated has a payload of 2 bytes, resulting in message sizes of 18 bytes for networks without security, 22 bytes for PADS and 23 bytes for TinySec.

### 7.2.1 Latency

The average latency for the three protocols is shown in Figure 8. The average latency increases at a similar pace for the PADS protocol as well as for non-secure AODV. On average the PADS networks with 30 nodes experienced more congestion which lead to a higher latency. Again TinySec seems to outperform both networks without security as well as the proposed PADS protocol. However, when considering the low success rate for TinySec, performance is no



**Figure 7. Average Energy Consumption Per Node Per Message Received for Messages of 23 Bytes**

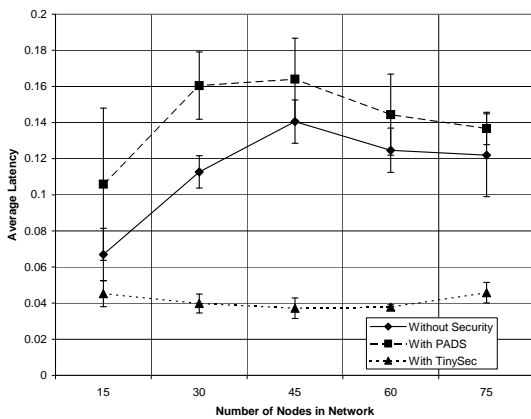


**Figure 9. Average Throughput in bps for Messages with 2 Byte Payload for Various Protocols**

longer adequate.

### 7.2.2 Throughput

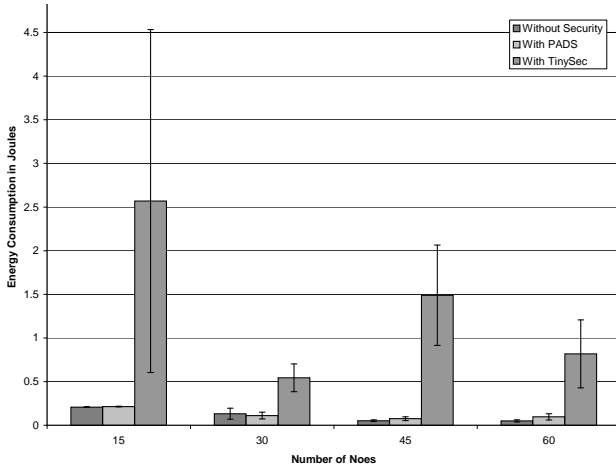
Figure 9 shows the comparison of the throughput in bps. As expected, the PADS protocol and non-secure AODV behave in a similar fashion. The only statistically significant difference between PADS and non-secure AODV is at 60 nodes. This is because two simulations produced less than 800 messages while the three other simulations produced more than 1800 messages. Hence the congestion decreased the average bps for PADS networks with 60 nodes. The performance of TinySec is much worse than either of the other two protocols since TinySec received many fewer messages than either of the other two protocols.



**Figure 8. Average Latency for Messages with 2 Byte Payload for Various Protocols**

### 7.2.3 Power Consumption

The average power consumption per message sent was measured and compared for different size networks for each of the three protocols where each message had a 2 byte payload. Figure 10 shows the result of this simulation. TinySec's power consumption per message is much higher than the other two protocols, this stems from the fact that in TinySec much fewer messages are received. There are no statistically significant differences between the PADS protocol and non-secure AODV.



**Figure 10. Average Energy Consumption Per Node Per Message Received for Messages with 2 Byte Payload**

### 7.3 Comparison of Protocol for different payloads

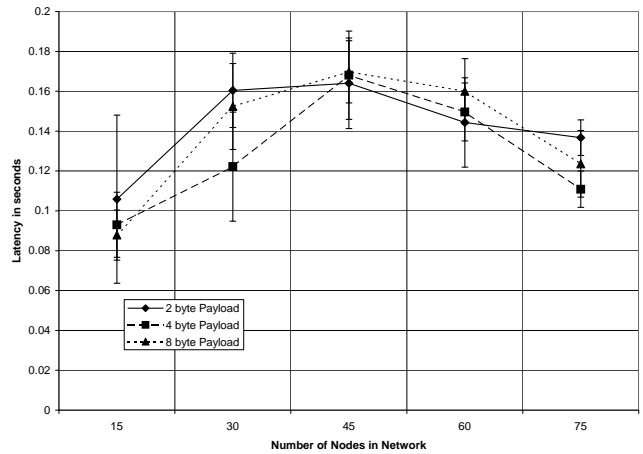
Depending on the need of an application, the size of the payload may vary from implementation to implementation. Experiments were conducted using the proposed PADS protocol with varying payload sizes in order to study the effects of the payload size on the protocol. Payload sizes of 2 bytes, 4 bytes and 8 bytes were used in these experiments.

#### 7.3.1 Latency

Figure 11 shows the different latencies for the PADS protocol using payload sizes of 2, 4 and 8 bytes. Overall, the protocol behaves similarly regardless of the payload size. Additionally, statistically, there are no differences between the simulations.

#### 7.3.2 Throughput

As shown in Figure 12 the PADS protocol behaves similarly regardless of payload size. There is statistical difference between PADS with 2 byte payload and PADS with 4 byte payload at 15 nodes. This is because one simulation for PADS with 4 byte payload had a very high failure rate due to congestion resulting in only 40 received messages. The next statistical difference is between PADS with 4 byte payload and PADS with 8 byte payload at 45 nodes. This time the simulations for PADS with 8 byte payload had high congestions and subsequently high failure rates, this resulted in a lower than expected bps. The difference at 60 nodes between PADS with 4 byte payload and PADS with 8 byte payload due to the fact that the simulations for PADS with



**Figure 11. Average Latency for Different Size Payload**

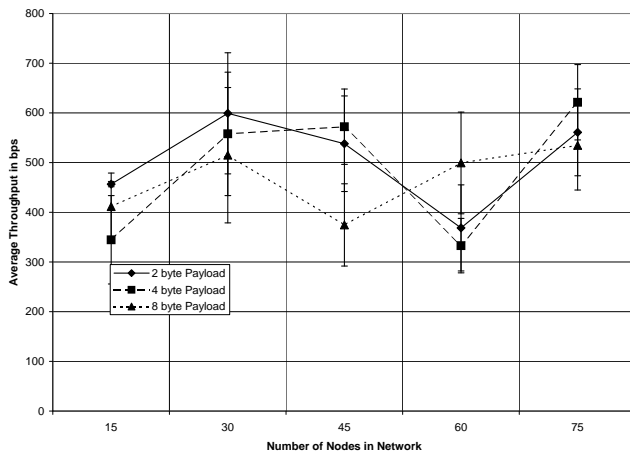
4 byte payload experienced high failure rates resulting in low throughput.

#### 7.3.3 Power Consumption

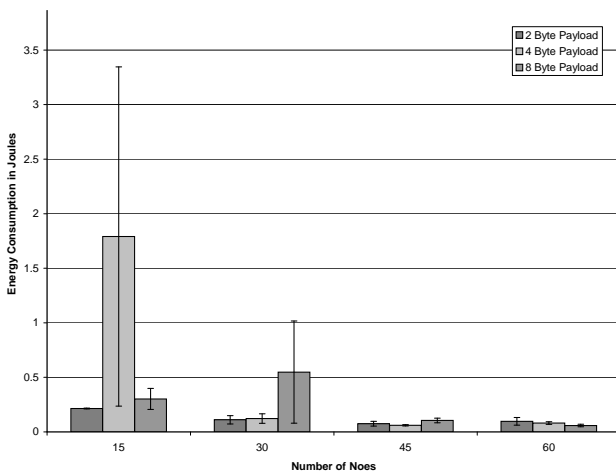
Figure 13 show the energy consumption per message received for the proposed PADS protocol with different payloads. For simulations with networks that had 15 nodes each, the high average energy consumption per message for PADS with 4 byte payload is because of a highly congested network. This resulted in only 40 received messages and a very high average energy consumption. This can also be seen by the size of the error bars. Another significant difference is at 45 nodes. Here the simulations using PADS with 8 byte payload had a very congested network which led to a high failure rate and a high average energy consumption per message received.

## 8 Conclusion and Future Work

This paper has shown that data in sensor networks can be securely routed through the network. By embedding a one-time pad to encrypt the payload of a message, the actual value transmitted is distorted so that any eavesdropper will not be able to use the information. Additionally, it is possible to ensure that the data were generated by a trusted source because the construction of the one-time pad will fail if any messages have been corrupted or were injected. The correctness of the two algorithms presented has been proven and an analysis of the time and space complexity required by the algorithms has been given. The simulations have shown that the additional work required by these algorithms is negligible and the performance rivals that of a



**Figure 12. Average Throughput for Messages with Different Size Payload**



**Figure 13. Average Energy Consumption Per Node Per Message Received Various Payloads**

network without security. The one-time pad protocol outperforms a network with encryption and authentication using TinySec. The throughput of this approach is as good as one without security and better than TinySec. Future work will include applying the one-time pad protocol to data aggregation and data fusion.

## References

- [1] Ieee standard 1363–2000. Standard Specifications for Public Key Cryptography, August 2000. Available from <http://grouper.ieee.org/groups/1363>.
- [2] B. Deb, S. Bhatnagar, and B. Nath. A topology discovery algorithm for sensor networks with applications to network management. DCS Technical Report DCS-TR-441, Rutgers University, May 2001.
- [3] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Mobile Computing and Networking*, pages 263–270, 1999.
- [4] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 138–149, New York, NY, USA, 2003. ACM Press.
- [5] G. Guimaraes, E. Souto, D. Sadok, and J. Kelner. Evaluation of security mechanisms in wireless sensor networks. *icw*, 00:428–433, 2005.
- [6] W. R. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 174–185. ACM Press, 1999.
- [7] C. T. Inc. MPR400/410/420 MICA2 Mote. Datasheet 2005.
- [8] C. Karlof, N. Sastry, and D. Wagner. Tinysec: A link layer security architecture for wireless sensor networks. In *Second ACM Conference on Embedded Networked Sensor Systems (SensSys 2004)*, November 2004.
- [9] Y. Law, J. Doumen, and P. Hartel. Benchmarking block ciphers for wireless sensor networks (extended abstract). In *1st Int. Conf. on Mobile Ad-hoc and Sensor Systems*, page electronic edition, Fort Lauderdale, Florida, Oct 2004. IEEE Computer Society Press, Los Alamitos, California.
- [10] P. Levis. Tossim: A simulator for tinyos networks.
- [11] P. Levis, D. Gay, and . Culler. Active sensor networks. In *Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI '05)*, Boston, MA, USA, May 2005.
- [12] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 126–137, New York, NY, USA, 2003. ACM Press.
- [13] D. Malan, M. Welsh, and M. Smith. A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography, 2004.

- [14] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. Tygar. SPINS: Security protocols for sensor networks. In *Seventh Annual International Conference on Mobile Computing and Networks (MobiCOM 2001)*, Rome, Italy, July 2001.
- [15] B. Przydatek, D. Song, and A. Perrig. Sia: secure information aggregation in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 255–265. ACM Press, 2003.
- [16] P. V. Rickenbach and R. Wattenhofer. Gathering correlated data in sensor networks. In *DIALM-POMC*, pages 60–66, October 2004.
- [17] R. Sion, M. Atallah, and S. Prabhakar. Resilient rights protection for sensor streams. In *VLDB*, pages 732–743, 2004.
- [18] H. Vogt. Exploring message authentication in sensor networks. In *Proceedings of ESAS 2004 (1st European Workshop on Security in Ad Hoc and Sensor Networks)*, LNCS, Heidelberg, Germany, Aug. 2004. Springer-Verlag.
- [19] H. Xu, L. Huang, Y. Wan, and B. Xu. Accurate time synchronization for wireless sensor networks. In X. Jia, J. Wu, and Y. He, editors, *MSN*, volume 3794 of *Lecture Notes in Computer Science*, pages 153–163. Springer, 2005.
- [20] 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. ACM Press, 2003.
- [21] S. Zhu, S. Setia, S. Jajodia, and P. Ning. An interleaved hop-by-hop authentication scheme for filtering of injected false data in sensor networks. In *IEEE Symposium on Security and Privacy*, pages 259–271. IEEE Computer Society, 2004.