Secure SOurce-BAsed Loose Synchronization (SOBAS) for Wireless Sensor Networks

A. Selcuk Uluagac, *Senior Member*, *IEEE*, Raheem A. Beyah, *Senior Member*, *IEEE*, and John A. Copeland, *Fellow*, *IEEE*

Abstract—We present the **S**ecure S**O**urce-**BA**sed Loose **S**ynchronization (SOBAS) protocol to securely *synchronize the events* in the network, *without* the transmission of explicit synchronization control messages. In SOBAS, nodes use their local time values as a one-time dynamic key to encrypt each message. In this way, SOBAS provides an effective *dynamic en-route filtering* mechanism, where the malicious data is filtered from the network. With SOBAS, we are able to achieve our main goal of synchronizing events at the sink as quickly, as accurately, and as surreptitiously as possible. With loose synchronization, SOBAS reduces the number of control messages needed for a WSN to operate providing the key benefits of reduced energy consumption as well as reducing the opportunity for malicious nodes to eavesdrop, intercept, or be made aware of the presence of the network. Albeit a loose synchronization per se, SOBAS is also able to provide $7.24\ \mu$s clock precision given today's sensor technology, which is much better than other comparable schemes (schemes that do not employ GPS devices). Also, we show that by recognizing the need for and employing loose time synchronization, necessary synchronization can be provided to the WSN application using half of the energy needed for traditional schemes. Both analytical and simulation results are presented to verify the feasibility of SOBAS as well as the energy consumption of the scheme under normal operation and attack from malicious nodes.

Index Terms—Secure loose synchronization, secure time synchronization for wireless sensor networks, SOBAS, wireless sensor networks, sensor-based cyber-physical systems

1 Introduction

In this paper, we address the secure synchronization problem for WSNs. Current secure time synchronization protocols for wireless sensor networks (WSNs) send separate synchronization messages and utilize reference points, global positioning systems (GPSs), and static pairwise key-based cryptographic mechanisms to ensure that the clocks of each sensor device are securely globally synchronized (i.e., every device has the same clock value). In addition to being costly in terms of energy consumption, these control messages (albeit necessary for their schemes) used in traditional protocols make them difficult to deploy in situations where it is required that the radio frequency (RF) footprint of the communicating devices be minimal (e.g., military sensor networks). In fact, many wireless sensor network applications are event-based and centrally controlled (e.g., critical infrastructure monitoring, video-surveillance, and patient-data collection), and do not necessarily need to maintain perfect synchronization of all the nodes in the network. Instead, it must be guaranteed that the event reports generated by the nodes are ordered properly prior to exiting the sensor network (i.e., loose synchronization). Assume, for instance, the military surveillance application in Fig. 1. The most valuable piece of information to the headquarters is that the enemy unit intruded the protected zone and is advancing South-West. In this case, the proper ordering of events can be achieved by synchronizing the sink with each source, and does not require that each sensor's clock be globally synchronized. In fact, an accurate knowledge of event ordering may generally suffice for many WSN applications, where the centralized decision authority (not sensors) acts swiftly on the information collected from the network. Second, because the communication cost is the most dominant factor in a sensor's energy consumption [1], [2], if the synchronization control messages in the network are eliminated as opposed to current "chatty" schemes, some of the energy savings from transmission cost can be utilized for the computation of local security operations.

Therefore, motivated by the downside of current schemes and considering the event-based characteristics of WSN applications [3] and the resource-limited nature of sensors [4], we propose the **S**ecure S**O**urce-**BA**sed Loose **S**ynchronization protocol. Essentially, SOBAS is an energy-efficient protocol for WSN applications that do not need perfect synchronization. SOBAS presents an effective technique to securely synchronize the data path in the network, *without* the transmission of explicit synchronization control messages. Instead of synchronizing each sensor globally as opposed to approaches providing perfect synchronization, we focus on ensuring that each source node is synchronized with the sink and nodes along the data delivery path such that event reports generated by the sink are ordered properly.

With SOBAS, we are able to achieve our main goal of synchronizing events loosely at the sink and at the data

- *A.S. Uluagac and J.A. Copeland are with the Communications Systems Center, School of Electrical and Computer Engineering, Georgia Institute of Technology, 266 Ferst Drive, Atlanta, GA 30332-0765. E-mail: selcuk@gatech.edu, john.copeland@ece.gatech.edu.*
- *R.A. Beyah is with the Communications Assurance and Performance Group, School of Electrical and Computer Engineering, Georgia Institute of Technology, Klaus Advanced Computing Building, 266 Ferst Drive, Atlanta, GA 30332-0765. E-mail: rbeyah@ece.gatech.edu.*

Manuscript received 12 May 2011; revised 18 Nov. 2011; accepted 26 May 2012; published online 6 June 2012.
Recommended for acceptance by S. Gupta.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2011-05-0301.
Digital Object Identifier no. 10.1109/TPDS.2012.170.

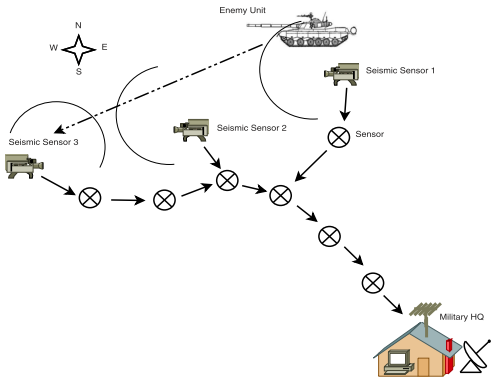


Fig. 1. An example military surveillance application.

delivery path as quickly, as accurately, and as surreptitiously as possible. SOBAS is perfectly suitable for WSN applications that do not need perfect synchronization and it is able to provide 7.24 μs clock precision on the data delivery path given today's sensor technology. Simulation results show that SOBAS is an energy efficient scheme under normal operation and attack from malicious nodes. In addition to being suitable for the applications where the centralized decision authority acts on the information collected from the network, our novel approach to synchronization with dynamic en-route filtering is also well suited for both WSNs and sensor-based CPS applications where utmost silence is necessary (e.g., military scenarios), as SOBAS is not "chatty." For instance, radio silence is very important for military operations as any radio transmission may reveal troop positions; so, restrictive EMCON[1] orders may be in effect [5], [6].

The paper proceeds as follows: an overview of SOBAS is given in the following section. The protocol architecture is presented in Section 2. Time uncertainty associated with using local clocks is analyzed in Section 3. A performance evaluation with simulations and a comparison with other schemes is presented in Section 4 and in Section S1, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2012.170, respectively. Related work is discussed in Section S2, which is available in the online supplemental material. Finally, Section 5 concludes the paper and gives the future work. Note that a preliminary conference version of this manuscript was presented at [7] and major differences from it are discussed in Section S2, which is available in the online supplemental material.

2 Protocol Architecture

There are three main components of the SOBAS protocol: Time-Based Key Management (TKM), Crypto (CRYPT), and Filtering-Forwarding-Synch (FFS) Modules. The relevant modules are explained below in the order they function in the SOBAS and the notations used are tabulated in Table 1, which is available in the online supplemental material.

1. EMCON: "The selective and controlled use of electromagnetic, acoustic, or other emitters to optimize command and control capabilities while minimizing, for operations security: 1) detection by enemy sensors; 2) mutual interference among friendly systems; and/or 3) enemy interference with the ability to execute a military deception plan" [5].

2.1 Threat Model and Assumptions

Source nodes are synchronized and loaded with a network-wise initialization vector (IV) predeployment. The IV and local time information will be used to generate the initial and subsequent dynamic keys. Note that the sensor nodes *do not* have perfect clocks and over time the sensors' clocks gradually diverge from the real clock value due to changes in the environmental conditions such as temperature, humidity, pressure, and vibration. In the worst case, they can accumulate up to several seconds of error per day [8]. Thus, the dynamic keys generated in SOBAS will change as a function of time and random drift. As such, the same event reported by different sources located nearby or separate events reported by different sources located elsewhere in the deployment region use different keys. In fact, this is an instrumental property, which we refer to as a *Spatio-Temporal* Security Property for WSN applications. Also, SOBAS does not incur a cost to discover which keys are shared between any two neighboring node (shared-key discovery phase [9]) via explicit messages because the nodes use the local time information to create the dynamic keys. Nonetheless, using real clocks requires designing both a flexible and an error-cognizant scheme that would compensate for drifting clocks. This issue is investigated more in Section 3. Similar to [10], [11], [12], [13], [14], [15], we mainly consider the false injection and eavesdropping of messages from an outside malicious node; hence, the insider attacks are outside the scope of this work (i.e., compromise of nodes is not considered). The security mechanisms against such attacks are referred to as *dynamic en-route filtering* systems in the literature [16]. Our rationale for considering this type of security mechanism is that they are effective methods for resource-constrained devices like wireless sensor networks as the malicious data is immediately filtered out from the network before propagating too much in the network, hence also helping save in energy. Moreover, we assume that attacks on clocks (e.g., pulse-delay (replay)) are detected by the extra delay they will introduce into the network as in [10], [17], [18].

The sink is the ultimate terminating point and decision maker. Nodes are statically deployed with the same communication ranges and more resources are available to the sink. Finally, the report (packet) size exchanged between the nodes is assumed to be fixed. The sink is perfectly synchronized with the outside world (e.g., via GPS). Regular sensors do not utilize GPS. Although, for some applications it may be necessary to utilize a GPS receiver on-board the sensor device, our rationale for not utilizing it is as follows: first, mounting a GPS receiver on a regular sensor requires the sensor to operate in two different frequencies: one is in the ISM band (i.e., 2.4 GHz) for the regular sensor communication, the other is in the L band (i.e., 1575.42 MHz (L1)) for the communication with MEO (Medium-Earth-Orbit) satellites [19]. Regular periodic transmissions to satellites will increase the cost of communication and, hence, the energy consumption [20]. Second, one of our design goals is to minimize the electronic emission footprint as much as possible to decrease the likelihood of detection by an adversary. Last, there may be environments (e.g., under water) where traditional radio-based GPS receivers would not work [21], [22].

In SOBAS, our main goal is to loosely synchronize and order events at the sink as energy-efficient, precise, and surreptitious as possible to reduce the likelihood of

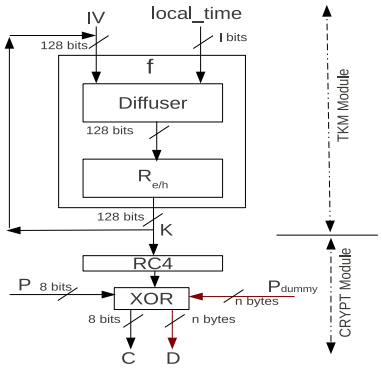


Fig. 2. Key generation in F function and encryption.

interception by an adversary. However, unlike [10], [11], [17], our goal is not to provide pairwise synchronization among the nodes.

2.2 Time-Based Key Management Module

In SOBAS, keys are generated dynamically using local time. This is addressed in the Time-Based Key Management module. When a source node has data to send to the sink due to either an external stimulation by the sink [23] or a self-initiated periodic report, it uses its local clock value as the key. However, the TKM module ensures that keys generated in the module are as random as possible as explained below. Specifically, the keys are a function of the current local time value (t_l) and either an initialization vector (IV) or previous key, K_{j-1} as

$$K_1^t = F(t_l, IV), \tag{1}$$

$$K_j^t = F(t_l, K_{j-1}), \tag{2}$$

$$F = G(Dif, R_{e/h}), \tag{3}$$

where G is a function of $R_{e/h}$ and Dif. $R_{e/h}$ is the operation of either encryption or hashing while Dif is the diffuser. The purpose of Dif is to diffuse the bits of XORed t_l and K_{j-1} with s-bit left circular shift operation before $R_{e/h}$ as follows:

$$Dif = (K_{j-1} \oplus t_l) \ll s. \tag{4}$$

The amount of shift, s, is determined by

$$s = l * n + n, \qquad n = 0, 1, \ldots, log_2 size(K) - 1, \tag{5}$$

where K and l denote the desired key size and the size of the timer/counter in bits for the microcontroller (e.g., ATmega 1281 [24]) employed by the sensor node. In essence, both Dif and $R_{e/h}$ in F are mainly utilized to increase the randomness of bits in the key, K, generated via (4) as subsequent readings from the timer may not change enough bits or may exhibit predictable patterns for malicious entities (e.g., differential cryptanalysis). The details of the generic F function is depicted in Fig. 2. It is possible to design $R_{e/h}$ using stream ciphers (e.g., RC4), cryptographic hash functions (e.g., MD5), or message digests using block ciphers [19], [25]. In SOBAS, the TKM module internally adopts 128-bit RC4 [19], [26]. Note that the purpose of F is to generate a key as random as possible with available cryptographic algorithms by considering their suggested precautions (e.g., at least 128 bits with RC4, no clear exchange of IVs, etc [19], [26]) as of this writing today. Notwithstanding, the effort of different class of adversaries in brute-forcing the key is analyzed in Section 4.

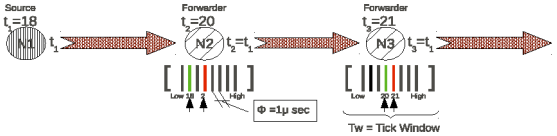


Fig. 3. An example illustration of packet delivery path with ticks.

Fig. 4. SOBAS packet structure.

In order to understand how the scheme works, consider a simple scenario in Fig. 3 and the key-derivation operation in Algorithm 1. In the figure, the source node is $N1$, and the forwarder nodes $N2$, and $N3$ are on the path to the sink that the report by $N1$ will traverse. Note that $N1$ inserts a copy of its ID and a local counter value inside the packet sent to the sink. The counter serves as a protection against replay attacks. It is increased each time a packet is sent from the source. The packet structure is illustrated in Fig. 4. The ID is used to verify the integrity of the packet by comparing the decrypted copy of it inside the packet (line 23 in Algorithm 1) with the clear value of it. As in Fig. 3, $N1$ uses its local clock value 18 to generate the key. This key is used by the CRYPT module to perform the desired cryptic operations depending on the security service (e.g., encryption, authentication, integrity) provided by the WSN application. When $N2$ receives the report from $N1$, it tries to find the value of the time at $N1$ using Algorithm 1. First, $N2$ subtracts the approximate packet flight time ($\Theta = \rho + \tau + \varphi + \varepsilon$) between itself and $N1$ from its local time (lines 4-6 in Algorithm 1) to be closer to the local time at $N1$, where ρ is the propagation time, τ is the packet transmission time, φ is the packet processing time, and ε is the approximation of errors for variability in transmissions due to fading, obstructions, and software errors, etc.[2] Also, in order for forwarder nodes to find the local clock value at the source node easily, all nodes are associated with a window of values, which we refer to as the *tick window*, (T_w) and a tick value (ϕ). Thus, $N2$ will try all values inside its tick window beginning from its local clock value. Once $N2$ finds the correct key value associated with the time at $N1$ (line 24 in Algorithm 1), using the found key, it will be able to compute the time offset from the sender. However, to combat against counterfeit values and to ensure a forwarder node does not futilely attempt to brute-force all time-based keys, a threshold value ($thresHold$ in Algorithm 1) is associated with each node's T_w. Note that proper choice for the size of the T_w depends on, among other parameters, ϕ and it is explained more in the next section. Moreover, two operational modes for the TKM module are available in SOBAS for incoming packets. The first mode, which we refer to as *Stateless Mode*, essentially governs the procedures explained above. Alternatively, the second mode, *Stateful Mode*, is able to provide further savings from the computation with a small increased

2. A realistic analysis of the uncertainty associated with errors is presented in the next section.

storage cost. Specifically, in the stateful mode, a receiver sensor can have a table for each sender sensor, where individual offset values for each sender is recorded. The next time the sensor receives a packet from the same sender, it will have a tick window (T_w) centered around the associated offset value for this sender. This makes the effort of the receiver easier when it tries to find the correct key for the sender. In the stateful mode, a sensor also remembers a previously seen malicious node.

Algorithm 1 Key Derivation at Nodes

```
1: DeriveKey(thresHold, NodeId)
2: begin
3: keyFound ← false; i ← 0; trialTime ← 0
4: Θ ← ρ + τ + φ + ε //packetFlightTime
5: startTime ← FetchLocalTime() − Θ
6: trialFwd ← TrialBack = startTime
7: Ctr ← FetchCounter(NodeId)
8: K_{j−1} ← FetchPreviousKey(NodeId) //Fetch IV if j = 0
9: while ((keyFound = 0)and(i <= thresHold)) do
10:    if ((i%2 = 1)and(i! = 0)) then
11:       trialFwd ← trialFwd + φ
12:       trialTime ← trialFwd
13:    end if
14:    if ((i%2 = 0)and(i! = 0)) then
15:       trialBack ← trialBack − φ
16:       trialTime ← trialBack
17:    end if
18:    if (i = 0) then
19:       trialTime ← startTime
20:    end if
21:    K ← ComputeTimeKey(trialTime, K_{j−1})//F in Figure 2
22:    ID_decrypted, Ctr_decrypted ← RC4(K, Pkt_dummy)
23:    if (ID_decrypted = ID_Clr)and(Ctr_decrypted = Ctr + 1) then
24:       keyFound ← true
25:    end if
26:    i ← i + 1
27: end while
28: return keyFound
29: end
```

2.3 Crypto Module

The CRYPT module addresses the security part of SOBAS. The CRYPT module obtains the dynamic key from the TKM module and performs the necessary security service. This is also the module where the key from the TKM is verified. If the key value received from the TKM module is not correct then a new key is obtained from the TKM module. This process continues until the correct key is found or the packet is marked as malicious to be discarded in the filtering-forwarding-synch module when all attempts to find the correct key are exhausted within the tick window, (T_w) (line 9 in Algorithm 1. The CRYPT module incorporates the RC4 algorithm into its body as the encryption mechanism. The rationale for choosing RC4 is due to its proven lightweight computational energy consumption on sensors [27], [28]. The risk of differential cryptanalysis is eliminated since a new key is used as input for each RC4 block [19]. Nonetheless, it may be still possible for an attacker to discover the key based on the first few bytes of the ciphertext (C in Fig. 2) [29]. Hence, to avoid that possibility, 768 bytes of the C, which is illustrated by P_{dummy} (known only by legitimate sensors)[3] and D in Fig. 2) are discarded as suggested by Mironov [30]. However, an adversary can always brute-force the keys and the effort of such an adversary is analyzed in Section 4. Note that since the key is generated in another module, any desired encryption (e.g., AES, 3DES), authentication, or integrity mechanism (e.g., HMAC, CMAC) can be implemented together or separately depending on the security service desired from the WSN application. After the correct value of the key used by the sender is determined by the current node, the offset value for the sender node is stored by the current node.

Three operational modes exist in the CRYPT module to determine how to forward the incoming packet. In the first mode, *No-reEnc* mode, the original incoming packet is forwarded to the upstream node without any re-encryption whereas in the second mode, *Full-reEnc* mode, the incoming packet is forwarded to the upstream node after re-encryption with the key associated with the local time at the receiver node. For *Full-reEnc* mode, the forwarder node uses its current local clock value and K_{j-1} value to create a new key when re-encrypting the incoming packet. Also in this mode, a forwarder node synchronizes itself loosely with the source as explained in the next module (FFS Module). Note that the advantage of the *No-reEnc* mode is one encryption computation, hence energy is saved by forwarding the original packet. However, if the current forwarding node is located too far away from the source node, the forwarding node may classify a healthy incoming packet as malicious. Specifically, this case occurs if the time difference between the local times of the source and the far-away node is bigger than the total time covered with $T_w * \phi$. Nonetheless, this is not an issue for *Full-reEnc* mode because the forwarder nodes refresh the key used to encrypt the forwarded packet. Finally, the third mode is referred to as *Selective-reEnc* (S-ReEnc) mode, where packets are selectively re-encrypted over some nodes along the data delivery path while these nodes are also loosely synchronized with the source as in *Full-reEnc*. Note that this mode is specifically introduced in SOBAS to solve the problem of classifying a healthy incoming packet as malicious (i.e., false-positive) that may occur in the *No-reEnc mode*. Moreover, in *Full-reEnc* and *Selective-reEnc* modes, re-encrypting the packets using new keys essentially refresh the keys. Eventually, in all the modes when the sink receives the report along the path, it also goes through the same intelligent key-finding procedure as forwarder nodes.

2.4 Filtering-Forwarding-Synch Module

The FFS module filters the incoming packet out of the network if it is classified as a bad packet by the CRYPT module or otherwise forwards it to the upstream nodes. In SOBAS, this module is also responsible for the synchronization process of the forwarder node with the source node along a data delivery path toward the sink with the *Full-reEnc* or *Selective-reEnc* modes of operation. At this module, the forwarder node gets the source's local clock value from the CRYPT module and updates its local clock value accordingly. For instance, in Fig. 5, all nodes along the path to the sink update their clock values. Therefore, a source-centric synchronization path is established up to the sink. The next time a packet from the same source travels over the same path, the nodes can put less effort in finding the proper time-based key. Note that in *Full-reEnc* all the nodes along the path do the aforementioned operations whereas in *Selective-reEnc* only a certain fraction of the nodes (e.g., every

3. In fact, any value in lieu of P_{dummy} that is common to nodes can be used as long as suggestions in [30] are considered.

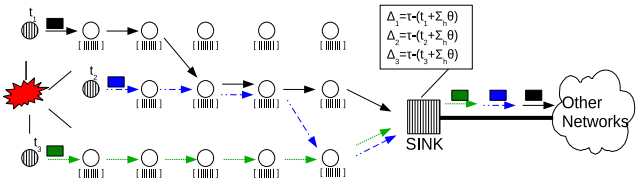


Fig. 5. Synchronization of nodes on more than one synch path in SOBAS.

third node) along the path do the operations. Hence, *Full-reEnc* provides "full path synchronization" while *Selective-reEnc* provides "partial path synchronization" and end-to-end synchronization along a data delivery path. Eventually, when the sink receives the report along the synchronization path, it will be able to see how much a particular source has diverged from the real clock value by extracting the key associated with the time at the source. Note that for this, the sink also goes through the same intelligent key-finding procedure as forwarder nodes. Hence, the sink calculates $\Delta_i = t_r - (t_i + \sum_h \Theta)$, with t_r being the real clock and h being the hop length while Δ_i and t_i denoting the time offset from source i and local time at source i, respectively. Thus, since nodes are synchronized prior to deployment, the sink can correct the timing of the reports coming from a particular source.

The synchronization described above synchronizes only one path according to a source (one synch path). However, in reality there are more synch paths because there may be more than one source in the deployed region and they may be reporting the same event to increase the accuracy of the reports at the sink [3]. Moreover, any node can be a forwarder and a source at the same time. We also note that there may be more than one sink collecting data from the sensors in the region. SOBAS handles the presence of multiple synch paths natively. It simply adheres to the logic of the synchronization of one path. In other words, when all the forwarder nodes receive a report from a source, or from another forwarder node, they simply forward the report directly or first synchronize themselves with the sender and then forward the report, depending on if they do the re-encryption operation or not. The advantage of this approach is that only one value is tracked. Since the sink(s) has more resources available, it can have a database of clock values for each sensor and their difference from the real clock. It can properly order any report from any source in the network using the Δ values associated with sources. The events received by the sink are placed in their proper order before leaving the sink for other networks. The case of multiple synch paths is illustrated in Fig. 5.

2.5 Summary of Operational Modes

Six different combinations of the operational modes exist in SOBAS depending on whether all (P-Synch) or some of the nodes along the data delivery path or only the end-to-end nodes (the source and sink) (E2E-Synch) are loosely synchronized and whether the nodes utilize memory (stateful) or not (stateless). These modes and tradeoffs associated with each mode are summarized briefly and illustrated in the kiviat diagram in Fig. 6.

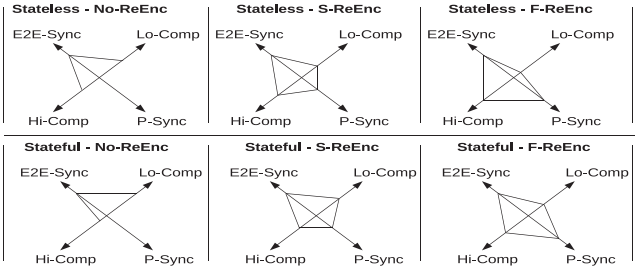


Fig. 6. Summary of operational modes in SOBAS: Hi-Comp: high computation; Lo-Comp: low computation; E2E-Sync: end-to-end loose synchronization; P-Sync: loose path synchronization.

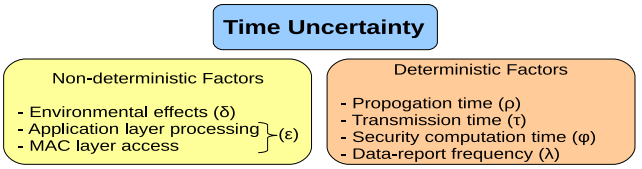


Fig. 7. SOBAS uncertainty parameters.

3 Time Uncertainty

Uncontrolled environmental conditions such as changes of temperature, humidity, pressure, and sudden vibrations in the deployment area cause internal clocks to gradually diverge from the real clock. Moreover, channel access time (at the medium access control layer) and send-time (including the time for preparing the packet at the application layer and passing it to the lower layers), can be considered as contributing to the unpredictable clocks [10]. In SOBAS, the environmental factors are captured with the parameter δ, which is the daily value of the drift per sensor given a deployment area, while the software-based factors are captured with ε. We adopt the values reported in [8], [10], [17], and [18] for ε and δ. Deterministic factors, on the other hand, depend on more predictable parameters. In SOBAS, as in [10] and [18], these include the transmission time of one packet (τ), the propagation delay (ρ), the packet processing time (φ) (e.g., due to cryptographic operations), and the average period of data from sensors (λ). The SOBAS uncertainty parameters used in coping with nondeterministic and deterministic factors are summarized in Fig. 7.

In SOBAS, the effect of all the factors are captured by the tick window, T_w, and it is the most significant parameter in dealing with the uncertainty in SOBAS. It provides a window of time values. However, even though the SOBAS protocol is designed with flexible T_w, a quantitative analysis is still needed. Therefore, in this section, first an analytical model is presented to investigate the relationship between the size of T_w and the tick value (ϕ). Then, a realistic T_w value is derived considering the capabilities of today's wireless sensor devices. With its current treatment of the uncertainty, the SOBAS protocol is conceived as a software-based solution consorting with other approaches and suggestions in the literature [17], [18], [31].

3.1 The Choice of Tick Window T_w

As briefly mentioned previously, the tick window T_w is available for the receiver node to choose from to decrypt the received packets. The window has upper and lower boundary values. In SOBAS, the T_w value is basically a

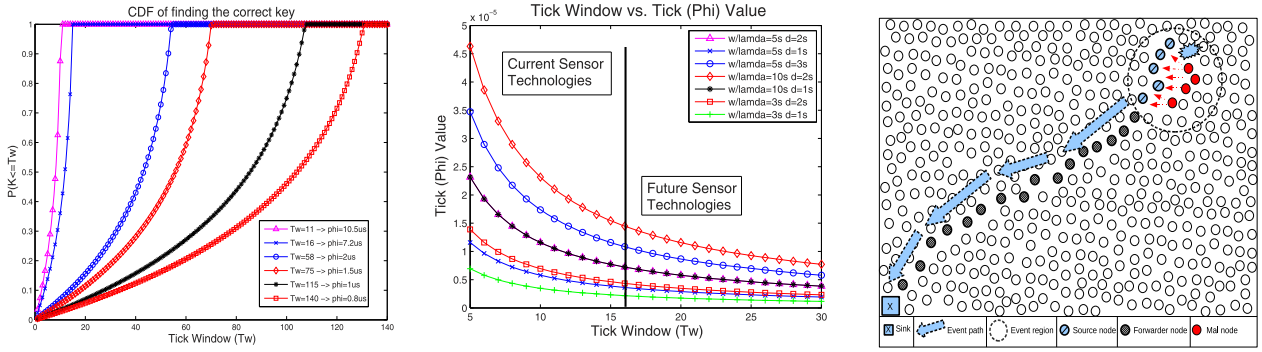


Fig. 8. (a) CDF of finding a correct key. (b) T_w (Tick window) versus ϕ (tick). (c) SOBAS simulation topology with GTSNetS.

function of the tick value (ϕ). Assuming that the sensor application sends its data periodically (or on the average) at certain time intervals to the sink [3], [23], T_w can be computed as

$$T_w = \frac{(\lambda + \rho + \tau + \varphi + \varepsilon) * \delta}{3600 * 24 * \phi}, \tag{6}$$

where τ is the transmission time of one packet, $\tau = \frac{l}{R}$ with l and R being packet length and rate of the WSN link, respectively; ρ is the propagation delay, $\rho = \frac{\chi}{c}$ with χ and c being distance between the sensors and the speed of light in the medium, respectively; λ is the average period of data in between sensed reports sent from a sensor; φ is the packet processing time, ε is the physical transmission error; δ is the daily value of the drift per sensor given a deployment area; and ϕ is the desired tick value. Note that (6) governs all the uncertainty factors into its body. Also, χ is taken based on the possible maximum distance to consider the worst case scenarios although nodes may be located closer than the maximum distance.

Moreover, the probability that kth trial out of T_w keys is the first success is geometrically distributed with parameter p, where p is $\frac{1}{T_w}$. Hence, the probability that T_w keys are tried in a tick window is

$$P\{K = T_w\} = (1-p)^{T_w - 1} * p, \qquad T_w = 1, 2, 3, \ldots. \tag{7}$$

Analytical results governing (6) and (7) with $l = 32$ bytes, $R = 250$ Kbps, $\lambda = 5$ s, $\chi = 100$ m, $\varphi = 558$ μs [27], $\varepsilon = 10$ μs [17], [18], and $\delta = 2$ s are shown in Fig. 8a for six different configurations of the tick window. For each T_w value, its corresponding ϕ value is also shown in the plot. As shown in Fig. 8a, the probability of success with a smaller value of T_w is greater, and therefore, less computational effort is required to guess the correct key of compared to when T_w is larger. Hence, the efficacy of the SOBAS protocol depends on the size of this window because the larger the size of T_w, the more time it takes for a receiver to find the key. Since the T_w value is basically a function of the tick value (ϕ), the smaller the value of the tick, the more keys could be tried by each sensor, hence T_w is larger and the accuracy of the scheme is increased. Several further observations are possible with a close examination of (6). When sensors send less frequently to the sink, hence λ is larger, the value of the T_w becomes larger. This obviously increases the computational effort of the sensor to find the correct key. A similar remark can be made for ε as well. On the other hand, when λ is smaller (i.e., more frequent data), the T_w is smaller. Hence, the scheme does not spend too much time trying to find the correct key; and the computational effort is smaller.

3.2 A Realistic Analysis of Tick Window (T_w) and Tick (ϕ) Value

A realistic value of the T_w considering the technical capabilities of today's wireless sensors is analyzed in this section. We see in (6) that more precision of ϕ comes with the cost of an increased number of keys that a sensor would try. SOBAS was designed to be energy efficient. If the computational effort of trying to find a key on a sensor is more than the communication cost of sending a separate keying message, then it may be better to send a separate keying message like other schemes in the literature. This depends on whether the benefit of a silent protocol is still desired at a cost of increased energy. Thus, in this part, we question what value of T_w is a plausible choice. In other words, we ask if we can derive a feasible T_w value given the capabilities of sensors today.

Assuming that in the worst case, a sensor will find the correct key at its last trial in the T_w window, the following inequality governs this case,

$$T_w * \zeta > \psi, \tag{8}$$

where ζ is the computational effort of finding a correct key, and ψ is the transmission cost of the separate keying message. Thus, if the left side of the inequality is bigger than the right side, then the energy advantage of the SOBAS scheme (not the advantages of "having dynamic keys" and "being surreptitious") would become obsolete and one can conclude that sending a separate keying message would be better than using SOBAS. The transmission cost of the keying message can be written as follows [32] (ignoring energy cost of sensing the event and staying-alive for simplicity):

$$\psi = (I_{tx} + I_{on}) * \tau * V, \tag{9}$$

$$\zeta = I_{on} * \mu * V, \tag{10}$$

where μ is the execution time required to process the desired encryption algorithm, τ is the packet transmission time, I_{tx} and I_{on} are the current consumptions in mA for

TABLE 1
SOBAS Simulation Parameters

# of Nodes	500	SensSize	32 bytes	E_{ini}	5000 mJ	E_{dec}	3.3μJ
Area	1000x1000 m	RecvInterval	5s	E_{rx}	66.7μJ	E_{enc}	3.3μJ
Link Rate	250Kbps	SimTime	3000s	E_{tx}	59.6μJ	E_{mac}	8.6μJ
Range	75 m	#of Mal Node	(0..10)	E_{sens}	9μJ	E_{sa}	11.4μJ
# of Healthy Nodes	10	T_w	16	Time Offset	U$[-3,+3]\mu$s	Voltage	3V

packet transmission and CPU processing, and V is the supply voltage of a given sensor node. Hence, an upper bound for T_w can be found as follows:

$$T_w \leq \frac{\psi}{\zeta}. \tag{11}$$

Fig. 8b plots T_w (Tick window) versus ϕ (precision) for several values of λ and δ. Assuming a sensor node with a microcontroller unit (MCU) of MSP430F16x [33] and a transceiver of CC2420 [2], [25], [32], and also assuming RC4 [27] as the encryption scheme, with $l = 32$ bytes, $R = 250$ Kbps, $\lambda = 5$ s, $\delta = 2$ s, T_w can be found to be 16; hence, the tick value, ϕ, of 7.24 μs. It should be noted that from the sender's perspective, as the system becomes more precise (i.e., the smaller the tick value), the chance of using a different key per packet transmitted increases. As long as the frequency of the events (packets) is larger than $\frac{1}{\phi}$, the system will use a different key per packet. Notwithstanding the technical capabilities of sensors today, the value of T_w computed in this section is instrumental in making SOBAS a realistic protocol as much as possible and will be used in the performance evaluation section.

4 Performance Evaluation

In this section, we evaluate the effectiveness of the SOBAS protocol both via simulations and analysis. First, using the realistic T_w value computed in the previous section, simulation results are presented to examine the energy efficiency of our scheme under normal operation and under attack. Second, the impact of the selective-reEnc operation in the network is studied. Third, different class of outsider adversaries are analyzed. Finally, a comparative study considering other similar synchronization and en-route filtering schemes is given in Section S1, which is available in the online supplemental material, which can be accessed via the IEEE's Computer Society Digital Library.

4.1 Security and Energy Consumption Analysis

In this section, we evaluate the performance of the SOBAS protocol via simulations. We focus on the energy consumption of the SOBAS protocol while under attack.

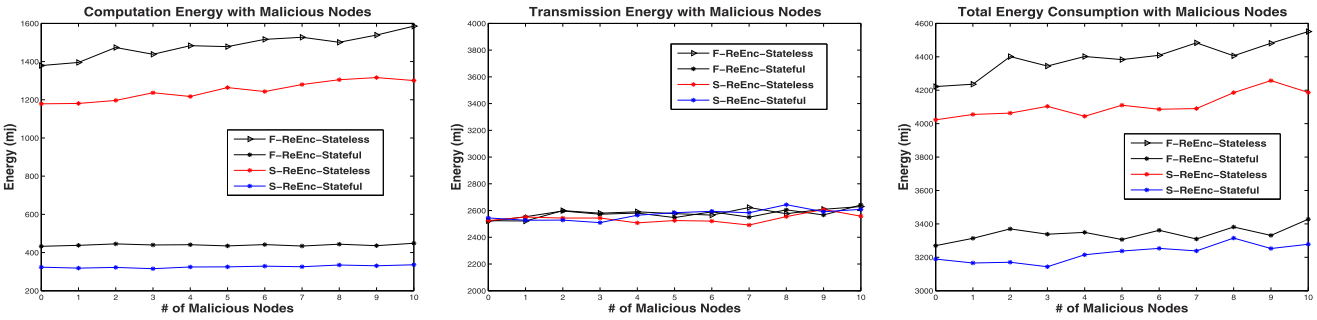
4.1.1 Assumptions, Threat Model, and Simulation Parameters

We use the Georgia Tech Sensor Network Simulator (GTSNetS) [34], which is an event-based sensor network simulator with C++, to perform the analysis of the SOBAS protocol. The topology and the parameters used are given in Fig. 8c and in Table 1. Nodes were located randomly in the deployment region and on average, source nodes were 25-35 hops away from the sink. The energy costs for different operations in the table are computed based on the values given in [32] and [33]. However, the costs for encryption and decryption operations are computed based on the reported values of the implementation of RC4 [27] on real sensor devices. E_{tx}, E_{rx}, and E_{sens} are the energy consumption of sending, receiving a packet and sensing an event, while E_{enc}, E_{dec}, and E_{mac} are the costs of encryption, decryption, and the message authentication code, respectively. We use 16 as the value of T_w as found in Section 3. Due to the broadcast nature of the wireless medium used in WSNs, attackers may try to eavesdrop, intercept, or inject false messages. In this paper, we mainly consider the false injection and eavesdropping of messages from an outside malicious node; hence similar to [10], [11], [12], [13], [14], [15], the insider attacks are outside the scope of this paper. In our attack scenario, the total number of healthy source nodes that collect the event information and send it toward the sink is assumed to be fixed, whereas the number of malicious nodes are increased over time. Letting i be the number of healthy source nodes and j be the number of malicious nodes, in our attack scenario, $j \leq i$, where $i = n$ and $n > 0$. The malicious sensors are randomly located inside the event collection region. We use en-route filtering to remove the malicious data as in [13], [14], [15]. Throughout this work, the following additional assumptions are made: each node has its local clock and its drift value from the real clock is generated using a uniform distribution between -3 and $+3$ μs similar to [17]. The Directed Diffusion routing protocol [23] is used, but others such as [35] can also be used. According to specifics of Directed Diffusion, after the sink asks for data via *interest* messages, a routing path is established from the sources in the event region to the sink. Thus, we assume that the path is fixed during the delivery of a particular sensed event report. Sensors are assumed to have the same communication range and may have different initial battery supplies. Finally, the simulation results presented in the figures are the average of 50 simulation runs for a specific analyzed parameter.

4.1.2 Simulation Results for Security and Energy Consumption

As mentioned in Section 2, there are three different modes for the CRYPT and two operational modes for the TKM modules. Modes for the CRYPT module are *No-reEnc*, *Selective-reEnc*, and *Full-reEnc* modes, whereas for the TKM module they are *Stateless* and *Stateful*. Figs. 9a, 9b, and 9c show the results for the attack scenario considering four different modes of operations except for the No-reEncode mode.[4] The x-axis represents the number of malicious nodes inside the region and y-axis represents the energy consumption in mJ. As seen from the figures, the computation cost

4. Note that this mode is specifically not included in the analysis because a healthy packet may be classified as malicious depending on the time difference between the local times of the source and the far-away node (see Section 2.3).



(a) Computation costs. (b) Transmissions costs. (c) Total energy.

Fig. 9. Computation, transmission, and total energy consumption under an attack scenario.

(i.e., E_{enc}, E_{dec}) for SOBAS-Selective-reEnc is less than that of SOBAS-Full-reEnc when Stateless mode is used. This is the direct result of the encrypting, decrypting, and re-encrypting of packets at every hop in the network for SOBAS-Full-reEnc. The cost of one encryption computation is saved by forwarding the original packet in SOBAS-Selective-reEnc while selectively re-encrypting packets over some nodes along the data delivery path. The Selective-reEnc mode of operation in the CRYPT module may be suitable for further limited networking deployments. Nonetheless, SOBAS-Full-reEnc is more suited for networks with more networking resources available. From the security stand-point, we see that as the number of malicious nodes increases inside the network, nodes spend more computation energy in the Stateless mode. This happens because the number of nodes who use all their key-trial attempts and ultimately classifies a packet as malicious, increases with the increased malicious traffic. Furthermore, as seen from Fig. 9a it is possible to save a significant amount of energy if stateful mode of operation is utilized. However, the stateful mode would depend on the availability of more storage on sensors.

As for the transmission costs (i.e., E_{tx}, E_{rx}), all modes are about the same because in all modes the same number of packets are transmitted or received. Moreover, analyzing the results for the total energy consumption, we see that the total energy consumption in the network exhibits a similar behavior as the computation costs. This is because the overall energy consumption is greatly dominated by the computation costs.

4.1.3 Simulation Results for Key-Trials

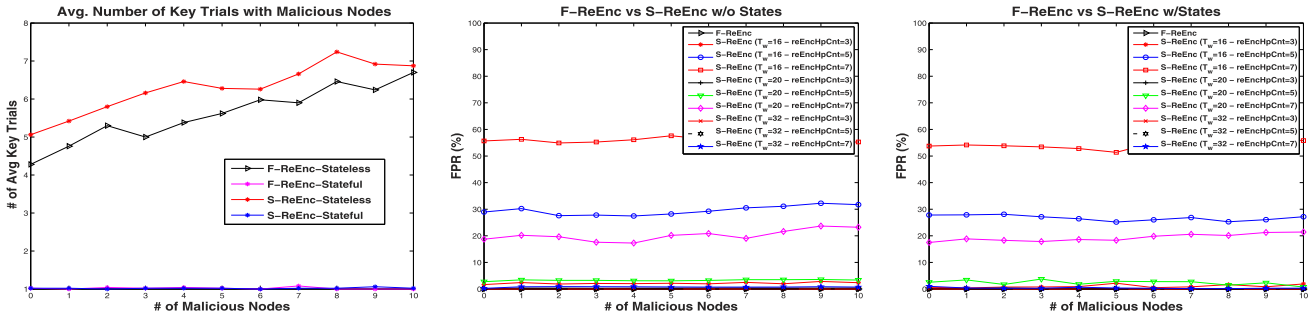
As explained earlier, when a sensor receives a packet from another sensor, it tries to find the time-based key value associated with this packet used by the sender when encrypting the packet before sending. However, the total trial attempts is limited by the value of synch window T_w, not to exhaust the resources on-board the sensor. For this, we have found and used in our simulations a feasible value for T_w (16) given for today's sensor technology (see Section 3). In this part of the simulation results, we discuss the average number of key-trial attempts by a sensor when attempting to decrypt the packet with both stateful and stateless operational modes. We observe that nodes do not use all the attempts with both modes; for the stateless mode of operation, the highest point for our attack scenario was around 6.9 (Fig. 10a). In general, we see that nodes with SOBAS-Selective-reEnc use more key-trials than SOBAS-Full-reEnc in the stateless mode. This is mainly because refreshing a packet with a new key based on the current local time at a node decreases the effort of the next hop node. On the other hand, comparing the stateful and stateless operational modes, one can easily observe that the number of key-trial attempts with stateful operation is significantly smaller than that of stateless operation. This is the direct result of sensor's ability to remember individual offset values for each sender sensor and malicious nodes. Keeping state information makes the effort of the receiver sensor easier when it tries to find the correct key for the sender. In fact, one implication of these results for both operational modes is that since nodes do not use all of their key trial attempts, the remaining effort can be utilized to increase the clock precision value. For instance, if on average half of the T_w is used with a certain ϕ, then $2 * \phi$ clock precision can be achieved by increasing the effective size of the T_w.

4.2 Impact of Selective Re-Enc Mode

In this section, we analyze the impact of the Selective-reEnc operation in the network by investigating the false-positive rate in the system.

The impact of Selective-reEnc was studied by considering the re-encrypting operation at different hops along the data delivery path to the sink. Specifically, three different cases were analyzed: reEncrypting at every third, fifth, and seventh nodes. Moreover, as discussed earlier (see Section 3), given the technical capabilities of sensors today, the value of T_w is 16. However, with near-term improvements in technology, further $T_w s$ that can provide better precision values are possible. Hence, we also include in our analysis different values for $T_w s$ as 16, 20, and 32. For all these different cases, further simulations were carried out and plotted in Figs. 10b and 10c. For each individual plot, the first number in parenthesis represents the T_w value while the second represents the reEnc hop count value. Also, in these figures, the x-axis represents the number of malicious nodes inside the event region and y-axis represents the false-positive-rate (FPR) in the system.

One general observation is that the more frequent the packets are re-encrypted, the smaller the rate of the FPR is in the network. This occurs because frequently refreshing a packet with a new key makes it easier to find the key at the next hop node. This situation also explains the superior performance of Full-reEnc over other configurations. Also, with near-term improvements in technology, it is possible to achieve small FPR values, which are even close to values in Full-reEnc mode of operation. Finally, overall, we



(a) Avg. key trials. (b) Stateless FPR analysis. (c) Stateful FPR analysis.

Fig. 10. Avg. key trials, stateless FPR analysis, and stateless FPR analysis under an attack scenario.

observe similar flat behaviors for both stateless and stateful operations because the system is able filter malicious data immediately from the network and the system is mostly influenced by distributed impact of reEncryption operations at different hops across the network. However, recalling energy consumption analysis (see Figs. 9a, 9b, and 9c) from the previous section, it should be noted that frequently refreshing a packet with a new key increases the overall energy cost of the system. Hence, SOBAS should be utilized depending on the needs of the network and application. For instance, if security is of utmost concern for the WSN application, then a SOBAS system that provides a smaller FPR rate could be preferred. On the other hand, if the lifetime of the network is more important, then a SOBAS scheme with a small value of reEnc hop count can be utilized. In this way, SOBAS is flexible and can be tailored to the needs of the WSN application.

4.3 Analysis of Outsider Attackers

As discussed earlier, SOBAS was designed as a dynamic en-route filtering system because it is an effective method for resource-constrained devices like sensors as the bad data is immediately filtered out from the network before propagating too much, helping save in energy. Hence, SOBAS mainly considers the false injection and eavesdropping of messages from an outside malicious node and the insider attacks are outside the scope of this work as in [10], [11], [12], [13], [14], [15]. In this section, we present the analysis of different class of outsiders with different capabilities.

In SOBAS, in order for an outside malicious node to be able to successfully inject a false packet, the malicious node must forge the packet. For the packet length, l,

$$l = ID \parallel TYPE \parallel E_{K_j}(ID \parallel DATA \parallel CTR), \tag{12}$$

where ID is the packet ID, TYPE is the packet type, the CTR is the counter, and E_{K_j} is the encryption of the fields with jth key (see Fig. 4), the complexity of the packet is 2^l. Hence, the probability of a malicious node correctly forging the packet is

$$P_{forge} = \frac{1}{2^{packetsize}} = \frac{1}{2^l}. \tag{13}$$

Then, the probability of an adversary incorrectly forging the packet and therefore the packet being dropped (P_{drop}) is

$$P_{drop} = 1 - P_{forge}. \tag{14}$$

Since SOBAS authenticates at every hop, forged packets will always be dropped at the first hop with a probability of P_{drop}. On the other hand, an adversary brute-forcing the keys will have to put an overall effort of

$$\Psi = \frac{\gamma}{\kappa}, \tag{15}$$

where γ is the average number of keys (e.g., 2^{l-1}) and is κ the speed of decryption in key/second. Given the 128-bit key size utilized in SOBAS, the overall effort, Ψ, is tabulated in Table 2 for different class of attackers with different capabilities. Three attackers are assumed. 1) Sensor-class attacker, which is able to perform as regular sensors; 2) middle-class attacker, which is equipped with more powerful resources than a regular sensor (e.g., a netbook); 3) powerful attacker with the most resources (e.g., a laptop with multicore CPUs) available. The value of the κ for the sensor-class is taken from [27], [28] while the values for the second and third class attackers were computed with *openssl* on Dell Latitude 2,120 Netbook and Sony Vaio i7 Laptop, respectively. Note that 128-bit keys are utilized without exchanging any IVs as suggested by Forouzan [19] and Barker and Roginsky [26]. Subsequent keys are the function of the previous key K_{j-1} and the local time. Hence, a new key is used for each packet refreshing the stale key. Moreover, two additional mechanisms exist in SOBAS against outsider attackers. One is the combined operations of the *Diffuser* and $R_{e/h}$ (e.g., encryption, hashing) in F (see Fig. 2), which are mainly utilized to increase the randomness of bits in the key, K, as subsequent readings from the local timer may not change enough bits or may exhibit predictable patterns for malicious entities. The second one is the dummy packet (P_{dummy}) concept, which is used to thwart the possibility that an attacker can discover the keys based on the first few bytes of the ciphertext, C, [29], by discarding (D in Fig. 2) the several hundred bytes of the C as suggested in [30]. Moreover, utilizing *Full-reEnc* and *Selective-reEnc* operational modes refreshes the keys by re-encrypting the packets with new keys in these modes.

TABLE 2
Adversary Models

Adversary	Class-1	Class-2	Class-3
Impact	Low	Middle	High
Platform	Sensor	Netbook	Laptop
κ (dec/sec)	1.79×10^{3}	9.36×10^{5}	5.07×10^{6}
Ψ (years)	3.01×10^{27}	5.75×10^{24}	1.06×10^{24}

5 Conclusion and Future Work

Current approaches that provide secure time synchronization send separate synchronization messages and utilize reference points, GPSs, or static pairwise key-based cryptographic mechanisms in order to provide perfect synchronization. Considering the event-based characteristics of WSN tasks and the resource-limited nature of sensors, and finally focusing on the fact that the transmission cost is significant in WSNs, in this work we relaxed the perfect synchronization condition for WSNs and we tackled the secure time synchronization problem from a completely different perspective. As opposed to current "chatty" perfect synchronization schemes, we focused on minimizing the communication in the network so that some of the energy savings from transmission cost could be utilized for the computation of local security operations. Thus, in SOBAS, instead of explicitly synchronizing sensors with synch messages, events/reports are synchronized accurately along the data delivery path as they occur. Moreover, In SOBAS, nodes use their local time values as a one-time dynamic key to encrypt each message. In this way, SOBAS provides an effective dynamic en-route filtering mechanism, where the malicious data is filtered from the network.

With SOBAS, we have been able to achieve our main goal of synchronizing events at the sink as quickly, accurately, and surreptitiously as possible. The protocol decreases the number of opportunities for malicious entities to eavesdrop, intercept packets, etc., by reducing the number of messages exchanged. Thus, energy savings from the reduced transmission is used for the local security computation. To the best of our knowledge, earlier secure synchronization studies for WSNs have not taken this approach. Both extensive analytical and simulation results verified the feasibility of the SOBAS scheme. SOBAS is able to provide $7.24\ \mu$s clock precision given today's sensor technology, which is much better than other comparable schemes (not including ones with GPS devices), and is at least twice better in energy efficiency than other schemes. Our novel approach to synchronization with dynamic en-route filtering is well-suited for the characteristics of many WSN applications, where utmost silence is necessary (like in military scenarios), as SOBAS is not "chatty." Our future work include studying further opportunities for increasing the clock precision by investigating unused key-trial attempts at a node, and addressing insider threats.

Acknowledgments

This work was partly supported by US National Science Foundation (NSF) Grant No. CAREER-CNS-0545667 844144 and DARPA Grant No. N10AP20022.

References

[1] G.J. Pottie and W.J. Kaiser, "Wireless Integrated Network Sensors," *Comm. ACM*, vol. 43, no. 5, pp. 51-58, 2000.

[2] R. Roman, C. Alcaraz, and J. Lopez, "A Survey of Cryptographic Primitives and Implementations for Hardware-Constrained Sensor Network Nodes," *Mobile Networks and Applications*, vol. 12, no. 4, pp. 231-244, Aug. 2007.

[3] O. Akan and I. Akyildiz, "Event-to-Sink Reliable Transport in Wireless Sensor Networks," *IEEE/ACM Trans. Networking*, vol. 13, no. 5, pp. 1003-1017, Oct. 2005.

[4] S. Uluagac, C. Lee, R. Beyah, and J. Copeland, "Designing Secure Protocols for Wireless Sensor Networks," *Wireless Algorithms, Systems, and Applications*, vol. 5258, pp. 503-514, 2008.

[5] D. of Defense, *Department of Defense Dictionary of Military and Assoc. Terms, Joint Publication 1-02*, 2001.

[6] B. Nguyen and R. Rom, "Communication Services under EMCON," *Proc. ACM SIGCOMM*, pp. 275-281. 1986,

[7] A.S. Uluagac, R.A. Beyah, and J.A. Copeland, "TIme-Based Dynamic Keying and En-Route Filtering (TICK) for Wireless Sensor Networks," *Proc. IEEE Global Comm. Conf. (Globecom)*, 2010.

[8] A. Boukerche and D. Turgut, "Secure Time Synchronization Protocols for Wireless Sensor Networks," *IEEE Wireless Comm.*, vol. 14, no. 5, pp. 64-69, Oct. 2007.

[9] Y. Xiao, V.K. Rayi, B. Sun, X. Du, F. Hu, and M. Galloway, "A Survey of Key Management Schemes in Wireless Sensor Networks," *Computer Comm.*, vol. 30, nos. 11/12, pp. 2314-2341, 2007.

[10] S. Ganeriwal, C. Pöpper, S. Čapkun, and M.B. Srivastava, "Secure time Synchronization in Sensor Networks," *ACM Trans. Information and System Security*, vol. 11, no. 4, pp. 1-35, 2008.

[11] S. Ganeriwal, S. Čapkun, C.-C. Han, and M.B. Srivastava, "Secure time Synchronization Service for Sensor Networks," *Proc. ACM Workshop Wireless Security (WiSe)*, pp. 97-106, 2005,

[12] A. Perrig, R. Szewczyk, J.D. Tygar, V. Wen, and D.E. Culler, "Spins: Security Protocols for Sensor Networks," *Wireles Networks*, vol. 8, no. 5, pp. 521-534, 2002.

[13] Z. Yu and Y. Guan, "A Dynamic En-Route Scheme for Filtering False Data Injection in Wireless Sensor Networks," *Proc. IEEE INFOCOM*, pp. 1-12, Apr. 2006.

[14] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical En-Route Filtering of Injected False Data in Sensor Networks," *IEEE J. Selected Areas Comm.*, vol. 23, no. 4, pp. 839-850, Apr. 2005.

[15] C. Kraub, M. Schneider, K. Bayarou, and C. Eckert, "Stef: A Secure Ticket-Based En-Route Filtering Scheme for Wireless Sensor Networks," *Proc. Second Int'l Conf. Availability, Reliability and Security (ARES)*, pp. 310-317, Apr. 2007.

[16] A. Uluagac, R. Beyah, Y. Li, and J. Copeland, "Vebek: Virtual Energy-Based Encryption and Keying for Wireless Sensor Networks," *IEEE Trans. Mobile Computing*, vol. 9, no. 7, pp. 994-1007, July 2010.

[17] K. Sun, P. Ning, and C. Wang, "Secure and Resilient Clock Synchronization in Wireless Sensor Networks," *IEEE J. Selected Areas Comm.*, vol. 24, no. 2, pp. 395-408, Feb. 2006.

[18] K. Sun, P. Ning, and C. Wang, "Tinysersync: Secure and Resilient time Synchronization in Wireless Sensor Networks," *Proc. 13th ACM Conf. Computer and Comm. Security (CCS '06)*, pp. 264-277, 2006.

[19] B.A. Forouzan, *Data Comm. and Networking*, fourth ed. McGraw-Hill, 2007.

[20] F. Ren, C. Lin, and F. Liu, "Self-Correcting Time Synchronization Using Reference Broadcast in Wireless Sensor Network," *IEEE Wireless Comm.*, vol. 15, no. 4, pp. 79-85, Aug. 2008.

[21] H. Yoshida and T. Mori, "Development of Precision Underwater Positioning System," *Proc. Underwater Technology and Workshop Scientific Use of Submarine Cables and Related Technologies Symp.*, pp. 217-222, Apr. 2007.

[22] J.W. Youngberg, *Method for Extending GPS to Underwater Applications*, US Patent 5119341, Washington, D.C.: Patent and Trademark Office, 1992.

[23] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," *Proc. ACM MOBICOM*, pp. 56-67, Aug. 2002.

[24] *ATmega128/L Datasheet*, Atmel Corp., www.atmel.com/atmel/acrobat/doc2467.pdf, June 2008.

[25] *CC2420DataSheet, 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver Rev. B*, Chipcon Products from Texas Instruments, focus.ti.com/lit/ds/symlink/cc2420.pdf, Nov. 2008.

[26] E. Barker and A. Roginsky, "Transitions: Recommendation for Transitioning the Use of Cryptographic, Algorithms and Key Lengths," *NIST Special Publication 800-131A*, Jan. 2011.

[27] R.V. Venugopalan, "Encryption Overhead in Embedded Systems and Sensor Network Nodes: Modeling and Analysis," *Proc. Int'l Conf. Compilers, Architecture and Synthesis for Embedded Systems (CASES '03)*, pp. 188-197. 2003.

[28] M. Passing and F. Dressler, "Experimental Performance Evaluation of Cryptographic Algorithms on Sensor Nodes," *Proc IEEE Int'l Conf. Mobile Adhoc and Sensor Systems (MASS),* pp. 882-887, Oct. 2006.
[29] S.R. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the Key Scheduling Algorithm of RC4," *Proc. Revised Papers from the Eighth Ann. Int'l Workshop Selected Areas in Cryptography (SAC),* pp. 1-24, 2001.
[30] I. Mironov, "(not so) Random Shuffles of RC4," Cryptology Eprint Archive, Report 2002/067, http://eprint.iacr.org/, 2002.
[31] S. Ganeriwal, R. Kumar, and M.B. Srivastava, "Timing-Sync Protocol for Sensor Networks," *Proc. ACM First Int'l Conf. Embedded Networked Sensor Systems (SenSys),* pp. 138-149, 2003.
[32] Xbow, "Crossbow Technology," http://www.xbow.com/, 2008.
[33] *MSP430x1xx Family User's Guide Rev. F,* Texas Instruments, http://www.ti.com/msp430, Nov. 2008.
[34] G.T.S.N. Simulator, "Gtsnets," 2007.
[35] K. Akkaya and M. Younis, "A Survey on Routing Protocols for Wireless Sensor Networks," *Ad Hoc Networks J.,* vol. 3, pp. 325-349, May 2005.

A. Selcuk Uluagac received the MS degree in information security from the School of Computer Science at Georgia Tech and the MS degree in electrical and computer engineering from Carnegie Mellon, in 2009 and 2002, respectively, and the PhD degree with a concentration in information security and networking from the School of Electrical and Computer Engineering at Georgia Tech in 2010. Prior to joining the faculty at Georgia Tech, he was a senior research engineer at Symantec. The focus of his research lies at the intersection of the networking and security fields. Of particular, his interests include designing secure and energy efficient communication protocols and architectures. He is a member of the research faculty in the School of Electrical and Computer Engineering at Georgia Tech and is affiliated with both the Communications Systems Center (CSC) and the Communications Assurance and Performance (CAP) Group. In 2007, he received the "Outstanding ECE Graduate Teaching Assistant Award" from the School of Electrical and Computer Engineering at Georgia Tech. He is a senior member of IEEE and a member of the ACM and ASEE.

Raheem A. Beyah received the bachelor of science degree in electrical engineering from North Carolina A&T State University in 1998, and the master's and PhD degrees in electrical and computer engineering from Georgia Tech, in 1999 and 2003, respectively. He is an associate professor in the School of Electrical and Computer Engineering at Georgia Tech where he leads the Georgia Tech Communications Assurance and Performance Group (CAP) and is a member of the Georgia Tech Communications Systems Center (CSC). Prior to returning to Georgia Tech, he was an assistant professor in the Department of Computer Science at Georgia State University, a research faculty member with the Georgia Tech CSC, and a consultant in Andersen Consulting's (now Accenture) Network Solutions Group. He served as a guest editor for MONET. He is an associate editor of several journals including the (Wiley) *Wireless Communications and Mobile Computing Journal*. His research interests include network security, wireless networks, network traffic characterization and performance, and security visualization. He received the National Science Foundation CAREER award in 2009 and was selected for DARPA's Computer Science Study Panel in 2010. He is a member of the NSBE, ASEE, and a senior member of ACM and IEEE.

John A. Copeland received the BS, MS, and PhD degrees in physics from the Ga.Tech. He holds the John H. Weitnauer, Jr., chair as a professor in the School of ECE at the Georgia Institute of Technology (Ga.Tech), and is a Georgia Research Alliance Eminent Scholar. He was VP Techn. at Hayes (1985-1993), and VP Eng.Techn. at Sangamo Weston, Inc. (1982-1985) and served at Bell Labs (1965-1982). He founded Lancope, Inc. (2000), and invented the StealthWatch network security monitoring system. He has been awarded 48 patents and has published more than 100 technical articles. In 1970, he received the IEEE's Morris N. Liebmann Award. He is a fellow of the IEEE.