DISH: Distributed Self-Healing in Unattended Sensor Networks

Di Ma and Gene Tsudik
University of California, Irvine
{dma1,gts}@ics.uci.edu

ABSTRACT: *Unattended wireless sensor networks (UWSNs) operating in hostile environments face the risk of compromise. Unable to off-load collected data to a sink or some other trusted external entity, sensors must protect themselves by attempting to mitigate potential compromise and safeguarding their data.*

In this paper, we focus on techniques that allow unattended sensors to recover from intrusions by soliciting help from peer sensors. We define a realistic adversarial model and show how certain simple defense methods can result in sensors re-gaining secrecy and authenticity of collected data, despite adversary's efforts to the contrary. We present an extensive analysis and a set of simulation results that support our observations and demonstrate the effectiveness of proposed techniques.

1 Introduction

Sensors and sensor networks are deployed and utilized for various applications in both civilian and military settings. One of the most attractive properties of sensors is their alleged ease of deployment. Because of the low cost of individual sensors and commensurately meager resources, security in sensor networks presents a number of formidable and unique challenges. A large body of research has been accumulated in recent years, dealing with various aspects of sensor network security, such as key management, data authentication/privacy, secure aggregation, secure routing as well as attack detection and mitigation.

Recently, unattended sensors and unattended sensor networks (UWSN) have become subject of attention in the security research community [1, 2]. In the unattended setting, a sensor is unable to communicate to a sink at will or in real time. Instead, it collects data and waits for an explicit signal (or for some pre-determined time) to upload accumulated data to a sink. In other words, there is no real-time reporting of sensed data. The inability to off-load it in real time exposes the potentially sensitive data accumulated on unattended sensors to certain risks. This is quite different from prior sensor security research where there is an assumption of an on-line sink collecting data in a more-or-less real-time fashion.

Unattended sensors deployed in a hostile environment represent an attractive attack target. Without external connectivity, sensors can be compromised with impunity and collected data can be altered, erased or substituted. Sensor compromise is a realistic threat since sensors are often mass-produced commodity devices with no secure hardware or tamper-resistant components. Prior security work typically assumed that some number of sensors can be compromised during the entire operation of the network and the main goal is to detect such compromise. This is a reasonable goal, since given a constantly present sink, attacks can be detected and isolated. The sink can then immediately take appropriate actions to prevent compromise of more sensors.

In our case, in contrast, the adversary can compromise a number of sensors within a particular interval. This interval can be much shorter than the time between successive visits of the sink. Thus, given enough intervals, the adversary can subvert the entire network as it moves between sets of compromised sensors, gradually undermining security. Generally speaking, this type of adversary is well-known in the cryptographic literature as the *mobile adversary* [3].[1]

Consequently, the main security challenge in the UWSN scenario is: *How can a disconnected sensor network protect itself from a mobile adversary?* Here, "protect", means: "maintain secrecy of collected information", i.e., can a sensor keep the adversary from learning sensed data even though the adversary might eventually break into that sensor and learn all of its secrets. We view this as an important problem because there are many scenarios where sensors are used to collect critical or high-value data.

Once a sensor is compromised and the adversary learns its secrets, collected data – even if encrypted – becomes exposed. This holds regardless of where encrypted data is stored: on the sensor that produced it or elsewhere. Some recent work [2] has analyzed and confirmed the futility of hiding data by moving it around the network.

We now zoom in further onto the problem of data secrecy. Considering that compromise of a given sensor has a certain duration, data collected by the said sensor can be partitioned into three categories, based on the time of compromise: (1) before compromise, (2) during compromise, and (3) after compromise. Obviously, nothing can be done about secrecy of data that falls into category (2) since the adversary is fully in control. The challenge thus becomes two-fold:

• **Forward** Secrecy: the term *forward* means that, category (1) data remains secret as time goes forward.

• **Backward** Secrecy: the term *backward* means that, category (3) data remains secret even though a compromise occurred before it was collected.

We are interested in the **confidentiality of data** col-

[1] The mobile adversary model is used to justify proactive cryptographic primitives, such as signatures and decryption [4, 5].

lected when sensors are not under direct control of the adversary. In the cryptographic literature, notions of *intrusion-resilience* [6] and *key insulation* [7][2] refer to techniques of providing both forward and backward security to mitigate the effect of exposure of decryption keys. However, these techniques are unsuitable for solving the problem at hand, as discussed in Section 3.2.

Data integrity is an equally important issue which is normally considered along with data secrecy. However, in this paper, we ignore data integrity, but, for a good reason. We distinguish between a **read-only** and a **read-write** adversary. The former is assumed to compromise sensors and leave no evidence behind: it merely reads all memory and storage. In contrast, a read-write adversary can delete or modify existing – and/or introduce its own fraudulent – data.[3] We argue that a read-only adversary is more realistic, since its goal is to learn data while remaining undetected. This kind of adversary wants to remain stealthy in order to repeatedly "visit" the network and "harvest" potentially valuable data. Whereas, a read-write (or active) adversary might be detected after the next sink visit and hence will no longer be able to achieve its goals once the sink takes appropriate measures. Thus, to protect the network against a read-only adversary, we focus – at least for now – on data secrecy alone.

Contributions: In this paper we propose DISH (**Di**stributed **S**elf-**H**ealing), a scheme where unattended sensors collectively attempt to recover from compromise and maintain secrecy of collected data. DISH does not absolutely guarantee data secrecy; instead, it offers probabilistic tunable degree of secrecy which depends on variables such as: adversarial capability (number of nodes it can compromise at a given time interval), amount of inter-node communication the UWSNs can support, and number of data collection intervals between successive sink visits. We believe that this work represents the first attempt to cope with the powerful mobile adversary in UWSNs. Consequently, it might open up a new line of research.

Organization: The organization of the paper is as follows: Section 2 states our assumptions about the network and the mobile adversary. We then propose a simple public key-based approach in Section 3. This approach, though less viable, is used as a security yeard-stick. We then present the symmetric key-based DISH scheme in Sections 4 and 5. Next, we analyze DISH with three types of adversaries in Section 6. Section 7 presents our simulation results. Section 8 reviews related work and Section 9 concludes the paper.

2 Assumptions

We now state our network assumptions and present our model of the adversary.

[2]Both extend the notion of *forward security* [8, 9].

[3]In the security literature, read-only is often referred to as a *passive* adversary. We do not use the term "passive" as it does not fit an adversary who is assumed capable of compromising sensors. Whereas, read-write is called an *active* adversary.

2.1 Sensor Network Assumptions

We envisage a homogeneous network consisting of peer sensors uniformly distributed over a certain region. The network operates as follows:

• Sensors are programmed to collect data periodically.[4]

• Each sensor obtains a single fixed-size data unit in each collection interval. T denotes the maximum number of collection intervals between successive sink visits.

• Sensors are unattended. Each sensor waits for either a signal or for some pre-determined time to upload accumulated data to the sink.

• The network is connected at all times. Any two sensors can communicate either directly or indirectly, via other sensors. We make no assumption about the communication media: it could, in fact, be wired or wireless.

• Sensors are capable of conducting certain cryptographic computations, such as one-way hashing, symmetric encryption and – optionally – public key encryption (but not decryption). However sensors can not run a sophisticated intrusion-detection system (IDS) on their own.

• Each sensor is equipped with either a Pseudo-Random Number Generator (PRNG) or a Physical/True Random Number Generator (TRNG). We elaborate on this later in the paper.

• Regardless of its type, encryption is always **randomized** [10]. Informally speaking, randomized encryption means that, given two encryptions under the same key, it is unfeasible to determine whether the corresponding plaintexts are the same.

• There is enough storage on a sensor to contain $O(T)$ sensed (encrypted) data items between successive sink visits.

• Each time a sink visits the network, the security "state" of all sensors is securely re-initialized. This includes all cryptographic keys as well as initial seeds for PRNGs.

• There are no power constraints. Although we try to minimize both computation and communication costs, we assume that security has a much higher priority than power conservation.

• All sensors maintain loosely synchronized clocks.

We make no assumptions about the **richness** of sensed data: the set of possible sensor readings might be very large or very small. It clearly depends on the specific sensor application. In some cases, sensed data can vary widely, e.g., for complex chemical sensors. Whereas, a simple light sensor might only collect 1-bit values (i.e., 0 or 1).

2.2 Adversarial Model

We now describe the anticipated adversary. We refer to it as ADV from here on. Our adversary model resembles that in [2], albeit with somewhat different operations and goals.

• Compromise power: ADV can compromise at most $k < n$ sensors during any single collection interval. We

[4]Event-driven sensing is also possible in the unattended setting; however, we do not consider it for the time being.

thus say that ADV is k-capable. The threshold k may be absolute, i.e., an integer, or relative, i.e., a fraction of n. When ADV compromises a node, and for as long as it remains in control of that node, it reads all of memory/storage contents and monitors all incoming and outgoing communication.

• Network knowledge: ADV knows the composition and topology of the network. It is capable of compromising any node it chooses.

• Key-centric: ADV is only interested in learning the secrets (keys) of sensors it compromises. (Since knowledge of keys allows it to decrypt data).

• No interference: ADV does not interfere with any communications of any sensor and does not modify any data sensed by, or stored on sensors it compromises. In other words, ADV is *read-only*, as discussed above.

• Stealthy operation: ADV's movements are unpredictable and untraceable. Specifically, it is infeasible to detect when and if the adversary ever compromised (or intends to compromise) a particular sensor.

• Atomic movement: ADV moves monolithically, i.e., at the end of each interval ADV selects at most k nodes to compromise in the next interval and migrates to them in a single action.

• Strictly local eavesdropping: ADV is unable to monitor and record **all** communication. It can only monitor incoming and outgoing traffic on currently compromised nodes.

ADV's main goal is to learn data collected by sensors. However, this does not imply that ADV can not *guess* that data. Since there might be only a few possible values a sensor could obtain, ADV might know well advance the entire range of all such possible values as discussed at the end of Section 3.1. Instead, ADV is interested in knowing exactly which value is being sensed. In the extreme case, this might correspond to a 1-bit flag.

Table 1 summarizes the notation used in the rest of the paper. Note that the terms *round* and *interval* are used interchangeably.

T	number of rounds between successive sink visits
n	number of sensor nodes in the network
i, j	sensor indices $0 < i, j \leq n$
r, r'	collection round (interval) indices, $0 < r, r' \leq T$
s_i	sensor i
d_i^r	data collected by s_i at round r
E_i^r	encrypted version of d_i^r
$\mathcal{H}()$	one-way, collision-resistant hash (e.g. SHA-2)
$Enc(X, Y)$	outputs randomized encryption of Y under key X
$Dec(X, Y)$	outputs decryption of Y under key X
C^r	set of *compromised* sensors at round r
k	maximum size of C^r; assumed to be constant
H^r	set of *healthy* sensors at round r
S^r	set of *sick* sensors at round r
$\|U\|$	number of elements in set U

Table 1: Notation summary (some defined later).

3 Public Key-based Schemes

Although, for usual performance reasons, we prefer a scheme based on symmetric cryptography, for the sake of completeness we start with a simple public key-based approach and examine its advantages and limitations.

3.1 A Simple Public Key Scheme

The main features of the simple public key-based scheme are as follows:

• The sink has a long-term public key, PK_{sink}, known to all sensors.

• As soon as a sensor collects data d_i^r at round r, s_i encrypts it to produce: $E_i^r = Enc(PK_{sink}, R_i^r, d_i^r, r, s_i, \cdots)$ where R_i^r refers to a one-time random number included in each randomized encryption operation, as specified in the OAEP+ quasi-standard [10].

• When the sink finally visits the UWSN and gathers encrypted data from all sensors, it can easily decrypt it with its private key SK_{sink}.

• Note that a sensor has no secret (private) key of its own – it merely uses the sink's public key to encrypt data.

Since ADV does not know the sinks's private key (SK_{sink}), the only way it can determine cleartext data is by guessing and trying to encrypt it with the sink's public key, PK_{sink}. In other words, given a ciphertext E_i^r (which conceals data d_i^r), ADV cycles through all possible data values d' and compares $Enc(PK_{sink}, d')$ to E_i^r. If they match, ADV learns that d' is the encrypted value. However, as discussed in Section 2.1, we use randomized encryption and each E_i^r is computed as: $Enc(PK_{sink}, R_i^r, d_i^r, ...)$ where R_i^r is a one-time random value produced by the sensor for each encryption operation. Assuming that bit-length of R_i^r is sufficient (e.g., 160 or more), the guessing attack becomes computationally infeasible.

There is, however, a crucial security distinction based on the source of random number R_i^r used in randomized encryption. If random numbers are obtained from a strong physical source of randomness, then we can trivially achieve both forward and backward secrecy. To argue this claim informally, we observe that a true random number generator (TRNG) generates statistically independent values. That is, given an arbitrarily long sequence of consecutive TRNG-generated numbers, removing any one number from the sequence makes any guess of the missing number equally likely. Let us suppose that ADV compromises a sensor s_i at round r' and releases it at round $r'' > r'$. Encrypted data from any round $r < r'$ remains secret, since it has the form: $Enc(PK_{sink}, R_i^r, d_i^r, ...)$ and all the random numbers that ADV learns while in control of s_i are statistically independent from R_i^r. Thus, we have forward secrecy. Similarly, any data encrypted after round r'' (after ADV releases s_i) also remains secret, because all random numbers ADV learns while in control of s_i are statistically independent from those generated later. Thus, we have backward secrecy.

On the other hand, if *random* numbers are obtained from a pseudo-random number generator (PRNG), the resulting security is much lower. This is because a typical

PRNG produces "random" numbers by starting with a (secret) seed value and repeatedly applying a suitably strong one-way function $\mathcal{H}()$ as: $R_i^{r+1} = \mathcal{H}(R_i^r)$. Therefore, again assuming that s_i is compromised at round r' and released at r'', data $E_i^r = Enc(PK_{sink}, R_i^r, d_i^r, ...)$ for $r < r'$ remains secret since computing R_i^r from $R_i^{r'}$ is computationally infeasible (even if $r' = r + 1$) due to the one-way property of function $\mathcal{H}()$. This implies that forward secrecy is preserved. However, for $r > r''$, encrypted data is easily guessed by ADV since it is efficient to compute R_i^r from $R_i^{r''}$ by repeatedly applying ($r - r''$ times) the function $\mathcal{H}()$. Therefore, backward secrecy is lost.

3.2 Key-Insulated and Intrusion-Resilient Schemes

We now consider more complex – and seemingly relevant – cryptographic techniques that provide both forward and backward secrecy. They include key-insulated [7] and intrusion-resilient [11, 12] encryption schemes. In both models, time is divided into fixed intervals. The public key remains fixed throughout the entire system lifetime, whereas, the private key is updated in each interval. When it is time to update the private key, the *user* contacts the *base*, a separate secure entity typically in the form of a remote trusted server or a local tamper-resistant hardware, for help in updating its key. This way, without simultaneously compromising both the *user* and the *base*, ADV is unable to learn future keys (thus backward security is achieved). The difference between a key-insulated encryption scheme and a intrusion-resilient encryption scheme is that when the *user* and *base* are compromised simultaneously all the security including forward security are lost in the key-insulated encryption scheme while forward security is still guaranteed in the intrusion resilient encryption scheme.

However, all such schemes are completely useless in our scenario since nodes (sensors) do not possess any decryption keys. They only use the sink's public key to encrypt data. Therefore, a key-insulated or an intrusion-resilient scheme can only help against sink's private key compromise – a problem irrelevant in our context.

Public Key Summary: To summarize our discussion thus far, simple public key encryption can help in achieving both backward and forward secrecy (our *"holy grail"* in this paper) only if each sensor is equipped with a physical source of randomness, i.e., a TRNG. Simple public key encryption with PRNG-equipped sensors achieves forward secrecy but fails with regard to backward secrecy. More exotic key-insulated and intrusion-resilient schemes are geared for digital signatures and decryption. They are unsuitable for the problem at hand.

4 A Simple Symmetric Key Scheme

We now construct a scheme based on symmetric cryptography and discuss its benefits and shortcomings.

We assume that, after each sink visit (at round 1), each s_i shares an initial and unique secret key K_i^1 with the sink. (This is in line with our assumptions in Section 2.1.) Then, at round $r \geq 1$, as it collects data, s_i produces $E_i^r = Enc(K_i^r, d_i^r, ...)$. If the encryption key does not change as rounds go by, all encrypted data can be trivially read by ADV. It only needs to compromise the sensor once, obtain its key and decrypt any encrypted data, whether generated before or after the compromise period. Instead, we require that, at the end of each round, each sensor evolve its key using a one-way hash function $\mathcal{H}()$, thus achieving forward secrecy. Specifically, round r (for $1 < r \leq T$) key is computed as: $K_i^r = \mathcal{H}(K_i^{r-1})$. If ADV breaks in at round r, it learns K_i^r but can not obtain K_i^{r-1} (which was used to encrypt d_i^{r-1}) due to the one-way property of $\mathcal{H}()$.

Unfortunately, backward secrecy is lacking. This is because ADV who breaks in at round r learns K_i^r. Then, by mimicking the key evolution process, it can obtain any future key $K_i^{r'}$ ($r' > r$) as:[5] $K_i^{r'} = \mathcal{H}^{r'-r}(K_i^r)$. Armed with $K_i^{r'}$, it can decrypt any data (that it might find later) encrypted with $K_i^{r'}$. Hence, there is no backward secrecy. Worse still, after $\frac{n}{k}$ rounds, ADV reaches a *steady state*, whereby all data collected and encrypted by all sensors is easily readable.

Based on our discussion in Section 3.1, it might seem that, if all sensors had TRNGs, both backward and forward secrecy are achievable. This intuition is wrong due to the following *paradox*: if s_i uses each random number R_i^r as an one-time symmetric encryption key to produce $E_i^r = Enc(R_i^r, d_i^r)$, there is no way for the sink to later decrypt it. This is because R_i^r, as a *true* random number, is unpredictable, unique to s_i and irreproducible by anyone, including the sink. So, there is no other way for s_i to communicate R_i^r to the sink. (We leave it as an exercise to the reader to check that all other "tricks" fail, i..e., without an out-of-band channel between each sensor and the sink, there is no way to benefit from true random numbers with purely symmetric cryptography.)

Symmetric Key Summary: Having reviewed simple public key and symmetric approaches, we observe that – except for the public key scheme used in conjunction with all sensors equipped with TRNGs – neither achieves the desired level of security: forward and backward secrecy of encrypted data. We believe that the combination of public key encryption and per-sensor TRNG is not realistic for many current and emerging sensor networks. Public key encryption requires more computation and consumes higher storage and bandwidth than symmetric encryption. Similarly, node-specific TRNGs are not always realistic, at least not on the scale in envisaged UWSNs. Therefore, below we focus on symmetric key techniques which do

[5] The notation $\mathcal{H}^p()$ means p repeated applications of $\mathcal{H}()$.

not assume any strong source of randomness on individual sensors.

5 Distributed Self-Healing

We now describe DISH: **Di**stributed **S**elf-**H**ealing scheme that provides probabilistic assurance of key-insulated data secrecy. DISH is based on symmetric cryptography, i.e., sensors are only required to perform hashing and symmetric encryption operations. We first describe the general idea and then present protocol details.

5.1 General Idea

Each sensor s_i shares an initial unique secret key K_i^1 with the sink, as in Section 2.1. At the start, none of these keys are known to ADV. As soon as the sink collects data and leaves the network unattended, ADV starts compromising sets of nodes, at most k per round.

We observe that, at round 1, when ADV first compromises k sensors in C^1, there are $n-k$ sensors that have not been compromised yet. We call such sensors *healthy* and the currently occupied sensors *compromised*. While ADV moves to the next compromised set C^2 in round 2, nodes in C^1 remain *sick*. The term *sick* refers to the ADV's ability to compute their secret keys for round 2 (and later), even though it no longer occupies them.

Our main idea is very simple: we let healthy sensors help *cure* sick sensors. Specifically, sick sensors ask contribution shares from healthy sensors and healthy sensors contribute secret values to sick sensors. A healthy sensor generates each contribution share - a random number - using its PRNG. This random number is secret to ADV since learning it requires knowledge of the healthy sensor's current PRNG state. A sick sensor uses contribution shares from healthy sensors – along with its current key – as input to a one-way function to generate its next round key. As long as there is at least one contribution from a healthy sensor, ADV is unable to learn the new key (unless it compromises the same sensor again in the future). Consequently, a previously sick sensor becomes healthy after a key update. We call a sensor a *sponsor* of another sensor if it furnishes the latter with a contribution in the latter's key update process. A sick sensor asks a set of t sponsors for their contribution shares at the end of every round.

Our approach can be characterized by the following axioms:

• Axiom 1: A healthy sensor in round r can not become sick in round $r+1$, unless ADV compromises it in round $r+1$. That is, a healthy sensor remains healthy until ADV compromises it.

• Axiom 2: A compromised sensor can not become healthy. (For it to have a chance of becoming healthy, ADV has to release it).

• Axiom 3: A sick sensor in round r can become healthy in round $r+1$ if and only if at least one healthy sensor contributes input to the computation of its (sick sensor's) key for round $r+1$.

To better illustrate the process, refer to Figure 1 which shows the sensor state diagram. The description so far is clearly too simplistic. First, a sensor has no idea whether it is sick or healthy, since we assume that ADV is stealthy: it moves unpredictably and leaves no trail. Thus, each sensor is potentially sick and potentially healthy. For this reason, we require all sensors (whether compromised, sick or healthy) randomly select a set of t sponsors at the end of every round and ask each sponsor for input. Because the cure comes from peer sensors, the network exhibits a self-healing property - something no individual node can provide– which emerges through collaboration of all nodes.

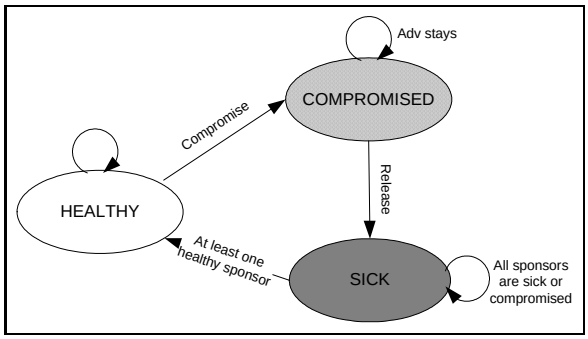

Figure 1: Sensor State Transition Diagram.

5.2 Details of the DISH Scheme

Within each round, a sensor runs two separate processes: main and sponsor. The main process is shown in Algorithm 1 and the sponsor process in Algorithm 2. As in Section 4, at every sink visit, each s_i is securely reinitialized with K_i^1 – a unique secret generated by the sink (details of this process are out of scope of the present work). All sensors are thus *healthy* at the initial stage.

The main process (loop at line 5) shows how s_i selects a set of t sponsors and obtains a random contribution $HELP[p]$ from each. All collected contributions, in addition to the current key, are then used to derive the next key K_i^{r+1}. The one-way property of $\mathcal{H}()$ ensures that it is infeasible for ADV to compute this key as long as at least one input out of: $\{K_i^r, HELP[1], ..., HELP[t]\}$ is unknown.

As shown in Algorithms 1 and 2, each node uses its PRNG for selecting sponsors as well as for generating contributions (as a sponsor). As mentioned earlier, a PRNG is typically realized as a one-way function, such as $\mathcal{H}()$. Suppose that ADV compromises s_i at round r, copies its PRNG state and releases s_i by round $r+1$. Then, for round $r+1$, ADV can still compute: (1) the set of s_i's sponsors as well as (2) the set of s_i's contributions in round $r+1$. In other words, even though a sensor like s_i is no longer compromised, it remains *sick*. Thus, for each

Algorithm 1: DISH: sensor s_i main process at round r

/* **start round r** */
collect new sensed data d_i^r
compute $E_i^r = Enc(K_i^r, d_i^r, r, \cdot\, \mathbf{3})$ store E_i^r on local storage
set $HELP[1..t] = \emptyset$
for $(p \leftarrow 1$ **to** $t)$ **do**
  set $j = PRNG()$ mod n
  send $REQUEST$ to s_j
  set $HELP[p] = REPLY(s_j)$
set $K_i^{r+1} = \mathcal{H}(K_i^r, HELP[1], ..., HELP[t])$
erase K_i^r and $\{HELP[1], ..., HELP[t]\}$
/* **end round r** */

Algorithm 2: DISH: sensor s_i sponsor process at round r

/* **start round r** */
while *(;;) DO* **do**
  receive $REQUEST$ from s_j
  set $HELP = PRNG()$
  compose $REPLY$ using $HELP$
  send $REPLY$ to s_j
  erase $HELP$
/* **end round r** */

sick sensor, ADV knows the entire set of its sponsors and all of its contributions.

Recall that the sink knows all initial secrets and can compute all intermediate states of all sensors. Therefore, it can also re-generate all sensor keys by mimicking the main and sponsoring processes in each round. That is, our key update process does not affect the sink's knowledge of sensors' round-specific keys and its ability to eventually decrypt data encrypted with these keys.

6 Analysis and Discussion

We analyze DISH with three classes of ADV, based on the compromise set (C^{r+1}) selection strategy:

Type A: A.ADV randomly selects k healthy nodes to compromise for the next round.

Type B: B.ADV selects k healthy sponsor nodes of sick sensors such that the largest number of nodes remain sick.

Type C: C.ADV selects k nodes such that most sick-healthy communication will pass through these nodes.

As shown in Figure 1, the game between the network and ADV runs as follows. The network tries to cure sick sensors through key update, while ADV moves around to compromise sensors. Therefore, key update and sensor compromise take place alternately round-by-round. We assume that key update is performed at the end of each round and ADV compromises the next set of victims immediately after key update. A sensor's status is its state during the interval after ADVś movement and before key update. Once ADV compromises s_i in round r, it records K_i^r for two reasons: (1) decrypting s_i's data generated in round r, (2) calculating contribution shares which s_i sponsors to others. ADV occupies s_i until its key update is finished, records the new key (K_i^{r+1}) and then (optionally) leaves s_i in the $r+1$-st round. We assume ADV maintains a network state map, so it knows current sensor states, i.e., S^r, C^r and H^r.

Key update affects sick sensors only. A sick sensor becomes healthy if at least one contribution comes from a healthy sponsor. A healthy sensor remains healthy since its own contribution (previous round key) is unknown to ADV. ADV's movement affects healthy and compromised sensors. A healthy sensor becomes compromised in the next round if ADV selects it in C^{r+1}. A compromised sensor becomes sick if ADV leaves it in the next round. We will show how the "game" between the network and different types of ADV affects sensor migration among groups.

We first consider two extreme cases. When $k \geq n/2$, no matter what t is, ADV can control the network in two rounds. Therefore in the following analysis, we assume $k < n/2$. If $k < n/2$ and each sensor selects $t = n - 1$ sponsors for every key update, all sick sensors become healthy after key update. As a consequence, there are always k compromised and k sick sensors in each round. ADV is never able to control the whole network. However this will incur a total communication overhead of $O(n^2)$. We want to find a balance between healing rate and communication overhead.

6.1 A.ADV

A.ADV randomly selects C^{r+1}. Selection happens at the end of round r and before round $r+1$. Using its up-to-date network state map, A.ADV simply picks k healthy nodes.

Note non-neighboring nodes in the network communicate to each other indirectly via other sensors. If a reply message from a sponsor is routed through any compromised node, ADV learns the contribution share. If ADV intercepts all the contribution shares from healthy sponsors of a sick node, ADV can calculate its next round key and consequently this sensor remains sick. So a sick sensor can become healthy only if at least one reply message from a healthy sensor is not routed through any of the compromised nodes.

We define the healing rate of a sick sensor in round r - $p^r(t)$ - as the probability at which the said sensor can become healthy in round $r+1$ after the r-th round key update with t sponsors. According to the diagram in Figure 1, the network composition in the $r+1$-st round with A.ADV is as follows:

$$|H^{r+1}| = |H^r| - k + |S^r| \cdot p^r(t) \quad (1)$$

$$|S^{r+1}| = |S^r| \cdot (1 - p^r(t)) + k \quad (2)$$

$$|C^{r+1}| = k \quad (3)$$

Now we show how to calculate $p^r(t)$. Let $p(i)$ denote the probability that i of t sponsors a sensor picks are healthy and $pp(i)$ the probability that at least one from i reply messages is not routed through any of the compromised nodes. $p^r(t)$ can be calculated as:

$$p^r(t) = \sum_{i=1}^{t} p(i) * pp(i) \quad (4)$$

$p(i)$ is calculated as $p(i) = \frac{\binom{|H^r|}{i} * \binom{n-|H^r|-1}{t-i}}{\binom{n-1}{t}}$. Let $l+1$ denote the average length of path between two nodes. That is, a message is routed through an average of l intermediate nodes. The probability that ONE message is NOT routed through any of the k compromised nodes is $p_1 = \frac{\binom{|H^r|-l_h}{k}}{\binom{|H^r|}{k}}$ where $l_h = \frac{|H^r|}{n} l$ is the average number of healthy sensors in a path. Then the probability that ALL i messages are routed through at least one of the k compromised nodes is calculated as $(1-p_1)^i$. So $pp(i)$ is calculated as $pp(i) = 1-(1-p_1)^i$.

To better understand this analytical model, we consider a mesh network of $n = 20 \times 20$. We can set $l = \sqrt{n}/2$. We calculate the number of sick and compromised nodes in the first 30 rounds with different k and t values. The results are shown in Figure 2. The analytical model demonstrates that:

[1] The system converges quickly, after 2 or 3 rounds, in a state with a stable number of sick and compromised sensors per round. For example, when $k = 8$ and $t = 1$, there are 19 sick and compromised sensors each round from the fourth round; when $k = 8$ and $t = 4$, there are 17 sick and compromised sensors from the third round. ADV is unable to make further progress.

[2] DISH can achieve the same level of security as the TRNG-based scheme if t is high enough, e.g., $t = k/2$, as shown in Figure 2.

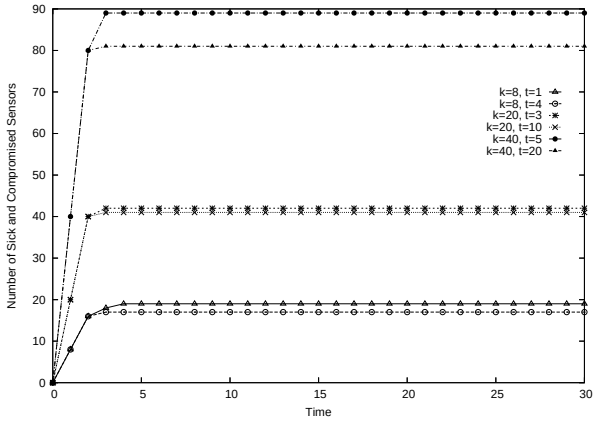

Figure 2: Analytical results with A.ADV.

6.2 B.ADV

B.ADV strategically selects k healthy sponsor nodes of sick sensors to C^r. Again, C^r selection occurs at the end of the $r-1$-st round when ADV gets to know S^r. ADV calculates sponsor nodes of sensors in S^r using Algorithm 2. Finally it selects k healthy ones from these sponsor nodes and move to them in the r-th round.

Suppose all the healthy sponsor nodes of s_i mentioned above are selected in C^r. ADV is able to compute K_i^{r+1} because it knows K_i^r and controls all healthy sponsors of s_i. Therefore s_i is not able to become healthy through the r-th round key update and remains sick in the $r+1$-st round. Let T^r denote the set of such nodes which are controlled by ADV indirectly through the compromise of C^r. Of $|S^r|$ sick sensors in round r, $|T^r|$ sensors remain sick in round $r+1$ and $|S^r| - |T^r|$ sensors either change or keep their status in round $r+1$ subject to the current healing rate of $p^r(t)$. $p^r(t)$ is also calculated with Equation 4. The network composition in $r+1$-th round is as follows:

$$|H^{r+1}| = |H^r| - k + (|S^r| - |T^r|) \cdot p^r(t) \quad (5)$$

$$|S^{r+1}| = (|S^r| - |T^r|) \cdot (1 - p^r(t)) + |T^r| + k \quad (6)$$

$$|C^{r+1}| = k \quad (7)$$

Compared with A.ADV, there are fewer healthy sensors, and, consequently, more sick sensors in each round. - both are contributed to the existence of sensors in T^r. B.ADV learns more sensor secrets, about $|T^r|$ more secrets, than A.ADV under the same set of network parameters.

To maximize its advantage, B.ADV must maximize the set of sensors in T^r, e.g., select k sponsor nodes so that the maximum number of sick sensors remain sick in the next round. It turns out that this maximum T^r problem is NP. As we consider a polynomial time ADV, the value of $|T^r|$ is determined by the actual C^r selection algorithm ADV uses to select k sponsor nodes. We use a greedy algorithm to simulate ADV's selection of k sponsor nodes in our simulation. Apparently, t and k affect T^r. If $t = 1$, $T^r \approx k$, whereas, if $t > k$, $T^r \approx 0$. Therefore, to defend against B.ADV, the network has to set t to a large(r) value. A large t helps in two ways: 1) increases the healing rate, and 2) decreases T^r since adv has to occupy all healthy sponsor nodes of s_i to keep it sick.

6.3 C.ADV

C.ADV takes advantage of communication passing through compromised nodes. Once C.ADV compromises s_i, it records K_i^r, stays on s_i until its key update is finished and records the new key (K_i^{r+1}). Meanwhile, C.ADV also records sick-healthy communication passing through s_i and other compromised nodes. Once C.ADV intercepts all healthy contributions for a sick sensor s_j, it can compute K_j^{r+1}. A sick sensor in round r becomes healthy in round $r+1$ only if at least one healthy contribution is **not** routed through any sensor in C^r. Therefore, C.ADV determines k nodes to compromise in the next round, such that the most sick-healthy communication will pass through these sensors.

To achieve its goal, C.ADV starts by compromising sensors from a corner. Then, it expands its invasion area by cordoning off all sick sensors with compromised sensors – this way, it can intercept all sick-healthy communication. If C.ADV can always physically separate sick and healthy sensors, it will eventually learn all sensors' keys and reach a steady state. Thus, the network topology directly influences security against C.ADV. Intuitively, narrow shapes do not fare well, while more balanced shapes are more resilient. For example, suppose

the network consists of 12 sensors arranged into a 2×6 mesh. Such a network fails against a 2-capable C.ADV. However, if the same 12 sensors are arranged into a 3×4 mesh, the network can defend itself against a 2-capable C.ADV. Therefore, to defend against C.ADV, the network should have a certain shape being large enough such that C.ADV is unable to separate sick and healthy sensors in every round. More generally, a $\sqrt{n} \times \sqrt{n}$ mesh network can defend itself against a k-capable C.ADV if $k < \sqrt{n}$. When $k \geq \sqrt{n}$, C.ADV can totally separate sick and healthy sensors and intercept all their communications, no sensor can be cured no matter how many sponsors it asks.

If the network is large enough, as C.ADV expands the "sick area", at some point it is no longer able to encircle all sick sensors, i.e., the cordon formed by compromised sensors will be broken and ADV will be unable to intercept all sick-healthy communication. Then, some sick sensors will become healthy. Thus far, we are unfortunately unable to construct an analytic model for the healing rate with C.ADV. This is because many factors (such as: C.ADV position, routing scheme and the values of t and k) affect the healing rate. This remains a major item for future work.

7 Simulation Results

To verify the above analysis, we developed a UWSN simulator and obtained some simulation results.

7.1 Network Parameters and Adversary Configuration

The UWSN is simulated as a $n = 20 \times 20$ ($n = 400$ sensors total) mesh with each point is occupied by a sensor. A shortest path scheme is used to route messages between two sensors. The network is static, so the communication path between any two nodes is fixed throughout the simulation. Each node follows the main and sponsor algorithms described in Section 5.

The simulation of A.ADV is straightforward. It maintains a network state map and uses it to randomly select k healthy nodes. We use a greedy C^r selection algorithm to simulate B.ADV. Let H_i be the healthy sponsor set of s_i. The greedy algorithm is shown in Algorithm 3.

Algorithm 3: B.ADV: C^r Selection

Input: S^r
Output: C^r
Procedure:
$S' \leftarrow S^r$
$C^r \leftarrow \emptyset$
while $|C^r| < k$ **do**
1 select s_i from S' such that:
1.1 $|C^r \bigcup H_i| \leq |C^r \bigcup H_j|$ for all $s_j \in S'$ where $j \neq i$
2 $C^r \leftarrow C^r \bigcup H_i$
3 remove s_i from S'
return C^r

C.ADV initially compromises the network from one corner and then expands the sick area outwards. If $k < \sqrt{n}$ ($k < 20$ in our case) the maximum number of sick sensors a k-capable C.ADV can totally cordon off is $k(k-1)/2$ and compromised sensors form a straight line, as shown in Figure 3. After C.ADV obtains $k(k-1)/2$ sick sensors, we simulate it in two variants. Variant 1 continues moving to compromise a new set of healthy sensors and the newly compromised sensors always form a straight line. Variant 2 comes back to re-compromise sensors among the initial $k(k-1)/2$ sick sensors (once they become healthy). Compromised sensors do not always form a straight line.

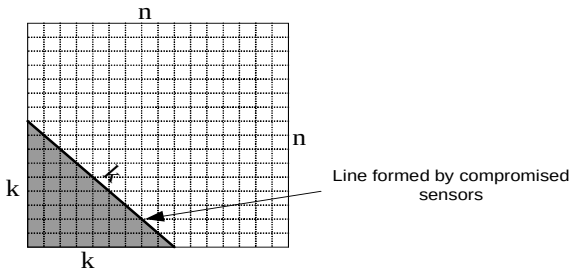

Figure 3: Area cordoned off by C.ADV.

7.2 Results

We set ADV's capabilities as: $k = 8,\ 20$ and 40 which corresponds to 2%, 5% and 10% of the network, respectively. For each k, we adjust t to different values. We first set $t = \frac{k}{8}$. We then set $t = \frac{k}{2}$ - half of the capability of ADV. Having fixed k and t, we let the network and ADV play the game for 30 rounds and record the number of sick and compromised nodes in each round.

Figure 4 shows our simulation results with an 8-capable ADV.

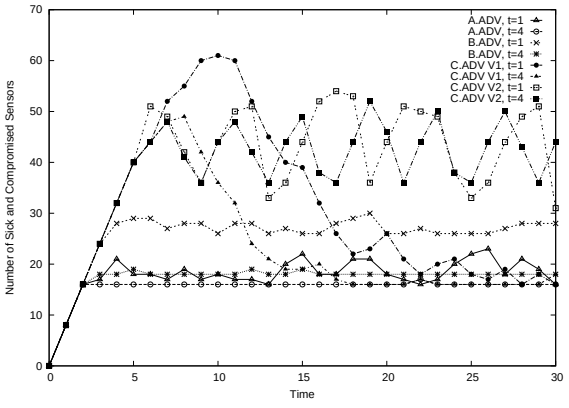

Figure 4: 8-capable ADV.

A.ADV**:** When $t = 1$, the number of sick and compromised sensors fluctuates around 19.64. When $t = 4$, the

system converges immediately after the first round with an average number of sick and compromised sensors of 16.03. Both numbers agree with our analytical prediction of 19 and 17, respectively.

B.ADV**:** With $t = 1$, the average number of sick and compromised sensors per round is 27.18 - about 8 more with B.ADV than with A.ADV- which also agrees with our prediction of $|T^r|$ extra sick sensors. As we increase t to 4, the average number of sick and compromised sensors per round is reduced to 18.11.

C.ADV**(Variant 1):** No sick sensor can become healthy in the first four rounds because C.ADV can intercept all sick-healthy communications. After the 4-th round, C.ADV no longer intercepts all sick-healthy communications if it continues moving and some sick sensors become healthy through key update. However, because the number of healed sensors in the first several rounds is less than k, the number of total sick and compromised sensors keeps rising. When $t = 1$, it reaches its peak of 61 in the 10-th round. After that, more sensors are healed. Because C.ADV keeps moving outward on a line, the system converges in the 28-th round into a state similar to that with A.ADV. As we increase t to 4, the number of sick and compromised sensors reaches its peak of 41 in the 8-th round and converges in the 19-th round.

C.ADV**(Variant 2):** C.ADV goes back to re-compromise sensors which are among the first $k(k-1)/2$ sick sensors that become healthy later. The system is does not converge and the number of sick and compromised sensors goes up and down but remains above the $k(k-1)/2$ line. The number of sick and compromised sensors when $t = 4$ has smaller amplitude than that when $t = 1$. Also, C.ADV goes back and forth more frequently when $t = 4$ than when $t = 1$. Apparently, the strategy of variant 2 is more effective than that of variant 1. From the simulation results, we see that larger t results in better protection against both B.ADV and C.ADV.

Figure 5 (top) shows simulation results with a 20-capable ADV. As ADV becomes more powerful, there are more sick and compromised sensors in each round. We can still achieve the same level of forward and backward security with both A.ADV and B.ADV by increasing t. However, since $k \geq \sqrt{n}$, C.ADV now can occupy all nodes in a column (or a row), it can intercept all sick-healthy communication. No matter what t is, ADV wins the game in the 20-th round (meaning that, thereafter, ADV knows all network secrets and all backward security is lost.) To complete our simulations, Figure 5 (bottom) shows the results with a 40-capable ADV. Its interpretation is similar to that of 20-capable ADV.

To get a clear idea of the level of security achievable by DISH, we compare it with the simple TRNG-based public key scheme discussed earlier in the paper. That scheme achieves the best level of security: ADV learns nothing about category (1) and (3) data.

We define *Data Secrecy Rate* (DSR) as the ratio – across all rounds – of healthy keys of all sensors (i.e., keys unknown to ADV) over the total number of keys of all sensors. We compare the two schemes based on DSR in a 30-round simulation in Table 7.2. With A.ADV and B.ADV, with sufficiently large t, DISH achieves DSR close to that of the TRNG-based public key scheme. With variant 1 of C.ADV, if k is small (e.g., $k = 8$), DISH also achieves DSR close to that of the TRNG-based scheme. However, DISH fares appreciably worse with the variant 2 C.ADV. If $k \geq \sqrt{n}$ ($k \geq 20$ since $n = 400$), DISH achieves only forward security, regardless of t. Clearly, its DSR is much lower than that of the the TRNG-based public key scheme.

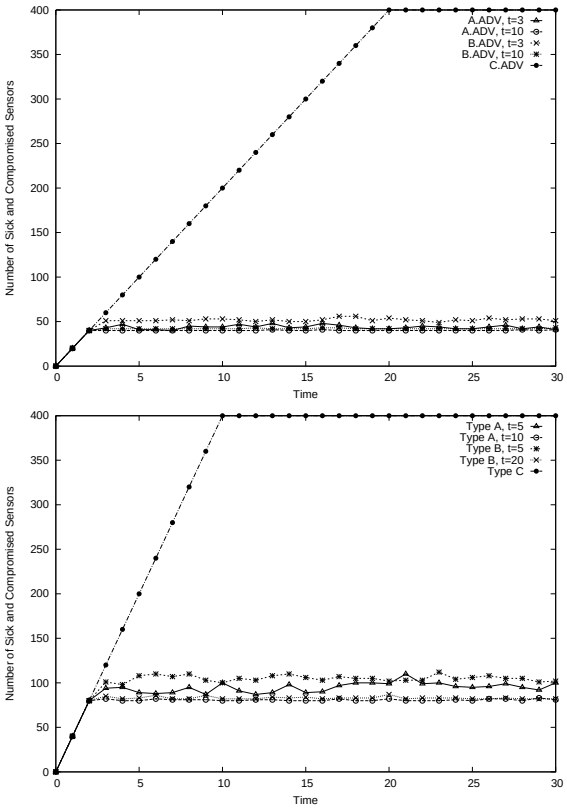

Figure 5: 20- and 40-capable ADV.

		k=8	k=20	k=40
TRNG-based public key scheme		96%	90%	80%
DISH	A.ADV	$95\%(t=1)$ $96\%(t=4)$	$89\%(t=3)$ $90\%(t=10)$	$77\%(t=5)$ $80\%(t=20)$
	B.ADV	$93\%(t=1)$ $96\%(t=4)$	$87\%(t=3)$ $90\%(t=10)$	$75\%(t=5)$ $80\%(t=20)$
	C.ADV v. 1	$92\%(t=1)$ $94\%(t=4)$	32%	15%
	C.ADV v. 2	$90\%(t=1)$ $90\%(t=4)$		

Table 2: Data Secrecy Rate (DSR) comparison after 30 rounds

8 Related Work

Data secrecy is a fundamental security issue in sensor networks and encryption is the standard way to achieve it [13, 14]. Much research effort has been invested in clever techniques for establishing pairwise keys used to secure sensor-to-sensor and sink-to-sensor communication, e.g., [15–18].

Sensor compromise is viable since sensors are built using low-cost commodity hardware components. Local keys are updated periodically to mitigate the effect of sensor compromise. Mauw, et al. [19] proposed some techniques to provide forward-secure data authentication and confidentiality for node-to-sink communication. Forward secure authentication has also been considered recently in the context of minimizing storage and bandwidth overhead due to data authentication in the presence of a powerful adversary [1]. The most related work to ours is Whisper [20], a protocol which provides both forward and backward security for communication between a pair of sensors. However, the scheme's security relies on a somewhat unrealistic assumption that the adversary is unable to compromise both sensors simultaneously. Also, every sensor must be equipped with a TRNG.

Recently, unattended sensors and sensor networks have become subject of attention in the security research community and various aspects of security have been explored [1, 2, 21, 22]. Parno, et al. proposed two distributed algorithms where sensors (without interference of sink) work collectively to detect node replication attack [21]. Security and privacy in data-centric sensor networks - typically running in unattended mode - have been recently studied in [22]. Di Pietro, et al. [2] have considered data survivability in UWSNs in the presence of a mobile adversary and proposed several simple network defense strategies. UWSNs have also been considered in the context of minimizing storage and bandwidth overhead due to data authentication [1]. The proposed forward-secure aggregate authentication techniques provide efficient *forward security*. Although this paper focuses on data secrecy, our results naturally extend to data authentication and to other peer group settings (e.g., P2P systems) where a set of nodes can be compromised by a powerful mobile adversary.

9 Conclusion

In this paper, we explored novel approaches to obtaining intrusion-resilient data secrecy in UWSNs. We proposed DISH, a symmetric key-based self-healing scheme that achieves both forward and backward secrecy. DISH successfully mitigates the effect of sensor compromise. Our simulation results clearly demonstrate the efficacy of DISH against A.ADV and B.ADV. However, it unfortunately does not appear as effective in the presence of a more powerful C.ADV. To this end, more advanced counter-strategies are subject to future work.

References

[1] D. Ma and G. Tsudik, "Forward-secure sequentical aggregate authentication," in *IEEE Symposium on Security and Privacy 2007*, May 2007.

[2] R. D. Pietro, L. Mancini, C. Soriente, A. Spognardi, and G. Tsudik, "Catch me (if you can): data survival in unattended sensor networks," in *IEEE PERCOM'08*, March 2008.

[3] R. Ostrovsky and M. Yung, "How to withstand mobile virus attacks," in *ACM PODC'91*, Omntreal, Quebec, Canada, August 19-21 1991.

[4] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive public key and signature systems," in *CCS '97: Proceedings of the 4th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 1997, pp. 100–110.

[5] Y. Frankel, P. Gemmel, P. MacKenzie, and M. Yung, "Proactive rsa," in *Crypto'97*, Santa Barbara, CA USA, August 17-21 1997, pp. 440–454.

[6] G. Itkis and L. Reyzin, "Sibir: signer-base intrusion-resilient signatures." in *Crypto'02*, 2003.

[7] Y. Dodis, J. Katz, S. Xu, and M. Yung, "Key-insulated public key cryptosystems," in *Eurocrypt 2002*, May 2002.

[8] R. Anderson, "Two remarks on public-key cryptology - invited lecture," in *Fourth ACM Conference on Computer and Communications Security (CCS)*, April 1997.

[9] M. Bellare and B. Yee, "Forward integrity for secure audit logs," in *Technical Report, Computer Science and Engineering Department, University of Siego*, November 1997.

[10] V. Shoup, "OAEP reconsidered," *Crypto 2001*, vol. 2139, 2001.

[11] Y. Dodis, M. Franklin, J. Katz, A. Miyaji, and M. Yung, "Intrusion-resilient public-key encryption," in *CT-RSA'03*, April 2003.

[12] ——, "A generic construction for intrusion-resilient public-key encryption," in *CT-RSA 2004*, February 2004.

[13] A. Perrig, J. Stankovic, and D. Wagner, "Security in wireless sensor networks," *ACM Commun.*, vol. 47, pp. 53–57, 2004.

[14] F. Hu and N. Sharma, "Security considerations in ad hoc sensor networks," *Ad Hoc Networks*, vol. 3, pp. 69–89, 2005.

[15] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks," in *Proceedings of the 9th ACM conference on Computer and Communications Security*, 2002.

[16] R. D. Pietro, L. V. Mancini, and A. Mei, "Random key assignment for secure wireless sensor networks," in *ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '03)*, October 2003.

[17] W. Du, J. Deng, Y. S. Han, S. Chen, and P. Varshney, "A key management scheme for wireless sensor networks using deployment knowledge," in *IEEE INFOCOM'04*, March 2004.

[18] H. Chan and A. Perrig, "Pike: peer intermediaries for key establishment in sensor networks," in *IEEE INFOCOM 2005*, March 2005, pp. 524–535.

[19] S. Mauw, I. van Vessem, and B. Bos, "Forward secure communication in wireless sensor networks," in *Third International Conference Security in Pervasive Computing (SPC'06)*, April 2006, pp. 32–42.

[20] V. Naik, A. Arora, S. Bapat, and M. Gouda, "Whisper: local secret maintenance in sensor networks," in *Principles of Dependable Systems (PoDSy)*, 2003.

[21] B. Parno, A. Perrig, and V. Gligor, "Distributed detection of node replication attacks in sensor networks," in *IEEE Symposium on Security and Privacy 2005*, May 2005.

[22] M. Shao, S. Zhu, W. Zhang, and G. Cao., "pdcs: Security and privacy support for data-centric sensor networks," in *IEEE INFOCOM'07*, 2007.