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-resistance 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]

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

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** collected 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 in tandem with data secrecy. However, in this paper, we ignore data integrity. This is because we distinguish between **read-only** and **read-write** adversaries. 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 consider a read-only adversary to be more realistic, especially since it aims to remain stealthy. A stealthy adversary has an incentive (and the ability) to visit the UWSN again and again, while a non-stealthy one might be unable to do so once an attack is detected and corresponding measurements are taken.

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. Section 6 shows our analytical and experiment results. Section 7 discusses drawbacks of DISH as well as possible ways to mitigate them. Section 9 concludes the paper. Related work is deferred to Appendix 8.

[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 Assumptions

We now state our network assumptions and present our model of the adversary. Table 1 summarizes the notation used in the rest of the paper. Note that the terms *round* and *interval* are used interchangeably.

v	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)
$Enc(X, Y)$	randomized encryption of Y under key X
$Dec(X, Y)$	decryption of Y under key X
O^r	set of *compromised* sensors at round r
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
k	maximum size of O^r; assumed to be constant

Table 1. Notation Summary

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. v 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 are not able to run 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 $O(v)$ 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. All sensors maintain loosely synchronized clocks.

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

- 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.

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 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.

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 decrypted by ADV since it is easy 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.

3.3 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 a 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.

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 not assume any strong source of randomness on individual sensors.

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

5 DISH: Distributed Self-Healing

We now describe DISH: **Di**stributed **S**elf-**H**ealing scheme providing probabilistic 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 O^1, there are still $n-k$ sensors that have not been compromised. We call such sensors *healthy* and the currently compromised sensors – *occupied*. While ADV moves to the next compromised set O^2 in round 2, nodes in O^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 *cure* sick sensors to become healthy. A healthy sensor is the one that has either never been compromised yet or regained its security through DISH. Specifically, sick sensors ask for contributions from healthy sensors and the latter 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 remains healthy until ADV compromises it.
- Axiom 2: An occupied sensor can not become healthy. (For it to have a chance of becoming healthy, ADV has to release it).
- Axiom 3: A sick sensor can become healthy in next round 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 transition 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 our 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 occupied, 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.

5.2 DISH Details

Within each round, each 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 re-initialized with K_i^1 – a unique secret generated by the sink (details

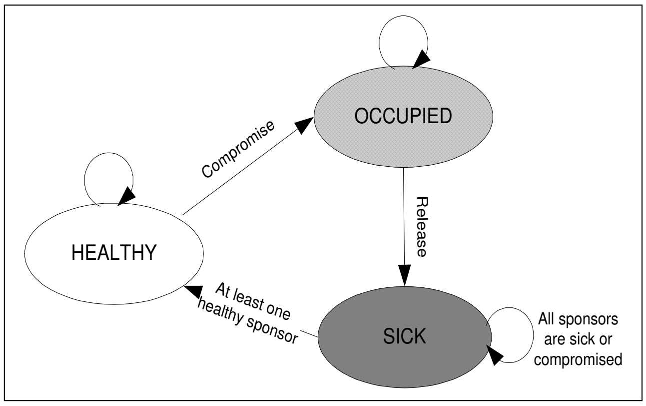


Fig. 1. DISH sensor state transition diagram.

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.

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

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

Algorithm 2: DISH: sensor s_i helper 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** */

As shown in Algorithm 1 and 2, each sensor node uses its local PRNG for both sponsor nodes selection and contribution share generation (as a sponsor). As mentioned earlier, a PRNG is often realized as a one-way function (such as our $\mathcal{H}()$). This allows ADV to compromise s_i at round r, copy the PRNG state, release s_i by round $r+1$, and still be able to compute the set of sponsors that s_i will ask for help and the set of contribution values that s_i will generate as a sponsor in round $r+1$. Thus, ADV knows the entire set of sponsors of each sick sensor and also all the contribution values the sick sensor will generate for each sponsoring request. Because 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, the proposed key update process does not affect the sink's knowledge of sensors' round keys and ability to eventually decrypt data encrypted with these keys.

Communication and Computation Overhead. In DISH, each sensor needs to contact t sponsors for help and also serve as a sponsor for t other sensors in every key update. This incurs a total of $2t$ messages traversing the UWSN in the end of each round. Each node needs to conduct $2t+1$ hash operations per round: t for sponsor selection, t for contribution generation and 1 for key generation.

6 Analysis and Simulations

In this section, we present some adversarial strategies, followed analysis and simulation results showing how DISH fares against these strategies.

6.1 ADV Migration Strategies

The goal of ADV is to minimize the set of healthy sensors - H^r (or maximize S^r). To achieve this goal, its best strategy is to always choose k healthy sensors to compromise in the next round.

We distinguish between two varieties of ADV, based on its O^r selection strategy: *Trivial Adversary* (T.ADV for short) and *Smart Adversary* (S.ADV for short).

T.ADV's strategy is to select and compromise k sensors from H^r randomly. T.ADV estimates current sensor states by maintaining a network state map which records IDs of sensors compromised and also the compromise time. Each round, T.ADV either chooses to compromise sensors that have not yet been compromised - these are absolutely healthy sensors - or those have ever been compromised a long time ago - there is higher probability that these sensors have regained their security through DISH.

S.ADV's strategy is to select k healthy sponsors of some sick sensors, such that the latter remain sick in the next round. S.ADV learns PRNG states of currently sick sensors; therefore, it can determine the entire set of sponsors for each sick sensor.

6.2 Analysis

We analyze the performance of DISH against T.ADV in terms of the number of healthy nodes at any round. Recall that a sick node becomes healthy if at least one of its healthy contributions is not intercepted by T.ADV. Let $p(i)$ denote the probability that i out of t sponsors for a given sensor are healthy and $pp(i)$ – the probability that at least one (out of i) replies is not routed through any occupied nodes. The probability that a sick sensor with t sponsors becomes healthy after the r-th round key update can be expressed as:

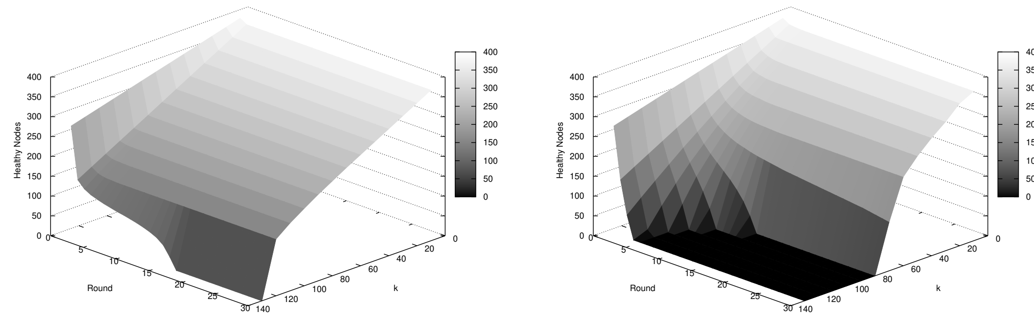
$$p^r(t) = \sum_{i=1}^{t} p(i) * pp(i) \tag{1}$$

where $p(i) = \frac{\binom{|H^r|}{i} * \binom{n-|H^r|-1}{t-i}}{\binom{n-1}{t}}$. Note that $pp(i)$ is influenced by the routing algorithm and UWSN topology. To make our analysis independent from these parameters, we define p as the probability of any contribution from a sponsor to a recipient being intercepted (eavesdropped on) by T.ADV. We then have: $pp(i) = 1 - p^i$. Therefore, expected number of healthy sensors at round $r + 1$ can be expressed as:

$$|H^{r+1}| = |H^r| + |S^r| * p^r(t) - k \tag{2}$$

¿From this we see that $|H^r|$ depends on k, p and t. More specifically, $|H^r|$ is proportional to t, and, inversely proportional to k and p. We now plot Equation 2 varying these three parameters. In this and all other plots in this paper, we fix UWSN size at $n = 400$.

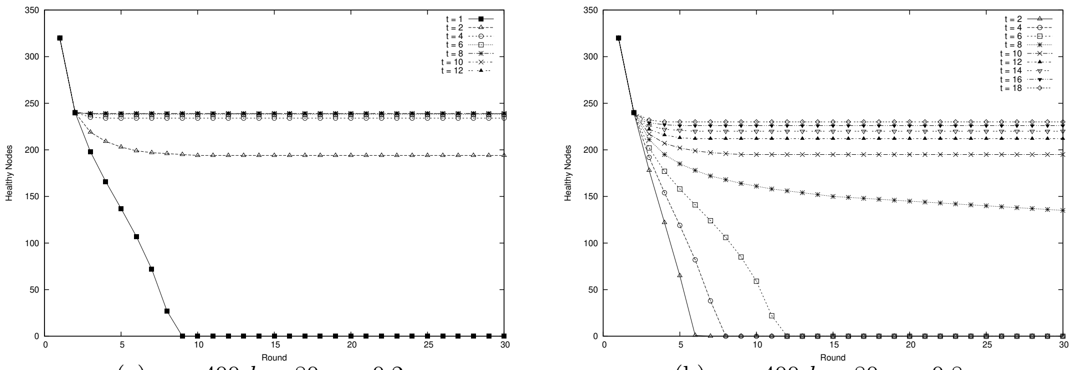
Figure 2 illustrates the influence of k and p on the number of healthy nodes with t fixed at 6. When T.ADV can intercept 20% of all traffic (e.g. $p = 0.2$ as shown in Figure 2(a)), for $k \leq 127$, the number of healthy nodes decreases in the first several rounds and then remains steady afterwards. That is, there are enough healthy nodes for the UWSN to successfully defend against T.ADV. However, when the compromise power of T.ADV increases above the threshold value of $k = 127$, healthy nodes eventually dwindle to none, as T.ADV controls all nodes' secrets from that round onwards. If T.ADV can intercept 80% of traffic (as shown in Figure 2(b)), the threshold k value decreases to 68.



(a) $n = 400$ $t = 6$ $p = 0.2$ (b) $n = 400$ $t = 6$ $p = 0.8$

Fig. 2. T.ADV analysis.

Figure 3 shows the effect of t on the number of healthy nodes when T.ADV compromises $k = 80$ nodes at each round. We identify two critical t values and denote them as t_r and t_o ($t_o > t_r$), respectively. t_r determines whether the network can successfully defend against T.ADV.If $t < t_r$, T.ADV eventually learns all secrets and wins. It is easy to see that a higher t brings better security with more healthy nodes when the network reaches stable state. However, it also incurs higher communication overhead. We note that there is a value t_o such that: when $t < t_o$, the number of healthy nodes (after the network reaches stable state) increases quickly with the increase in t. If $t > t_o$, increasing t brings little extra security. Since DISH is not designed to achieve guaranteed (deterministic) security, t_o represents a balance between security and performance. It also determines the communication overhead. As shown in Figure 3(a), if T.ADV intercepts 20% of all traffic, $t_r = 1$ and $t_o = 4$. Whereas, if T.ADV intercepts 80% of all traffic (Figure 3(b)), $t_r = 9$ and $t_o = 14$.



(a) $n = 400$ $k = 80$ $p = 0.2$ (b) $n = 400$ $k = 80$ $p = 0.8$

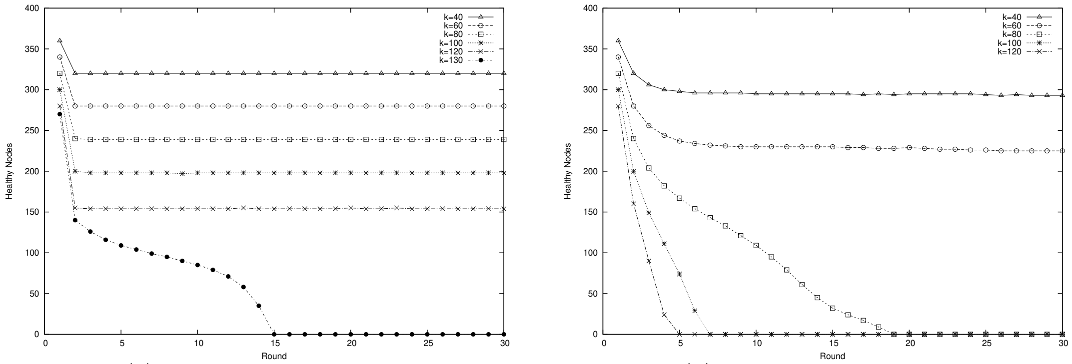
Fig. 3. Analysis of the effect of t.

In contrast with T.ADV, S.ADV strategically selects k healthy sponsor nodes from a subset of the sick sensor set S^r, such that sensors in this subset are unable to re-gain security through the key update process, since their sponsors are now controlled by S.ADV. To maximize its advantage, S.ADV must also maximize the number of sick sensors. It turns out, that this problem is reducible to the well-known *Subset Cover Problem* which is **NP**. Since we consider a polynomial-time S.ADV, the size of the covered subset is determined by the specific O^r selection algorithm used byu S.ADV. However, it is safe to say that there should be at least $\frac{k}{t}$ more sick sensors with S.ADV than with T.ADV, under the same set of parameters. We will validate this hypothesis by simulation in the next section.

6.3 Simulation Results

To re-confirm the above analysis, we developed a UWSN simulator and run numerous experiments. For each experiment, we defined network parameters (n, t) and ADV parameters (k, p). We run the simulator until either $|H^r|$ reached a steady state (UWSN won) or $|H^r| = 0$ (ADV won). Every node follows the main and helper algorithms described in Section 5.

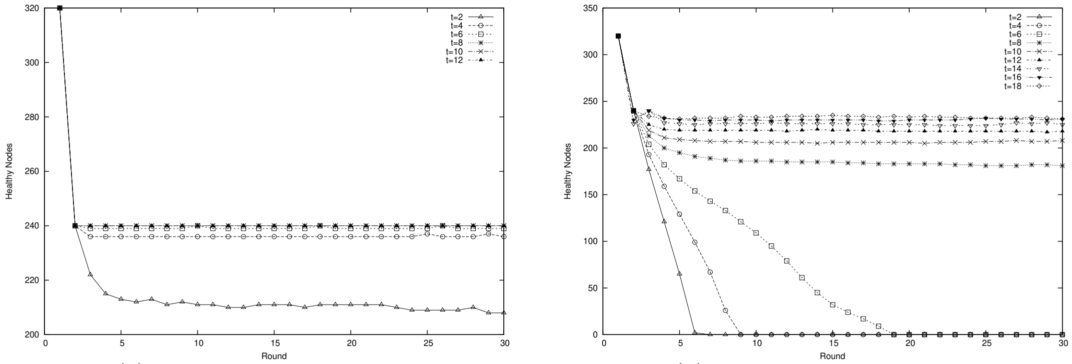
With $t = 6$ and $p = 0.2$, Figure 4(a) shows $\mathsf{T.ADV}$ wins when $k \geq 130$ but fails when $k \leq 120$. This matches our analytical prediction of the threshold value of $k = 127$. With $t = 6$ and $p = 0.8$, Figure 4(b) shows $\mathsf{T.ADV}$ wins when $k \geq 80$ but fails when $k \leq 60$. This also matches our analytical prediction of the threshold value as $k = 69$.



(a) $n = 400$ $t = 6$ $p = 0.2$ (b) $n = 400$ $t = 6$ $p = 0.8$

Fig. 4. Simulation of $\mathsf{T.ADV}$.

Simulation results with variable t are shown in Figure 5. They confirm that choice of t dramatically affects security and performance.



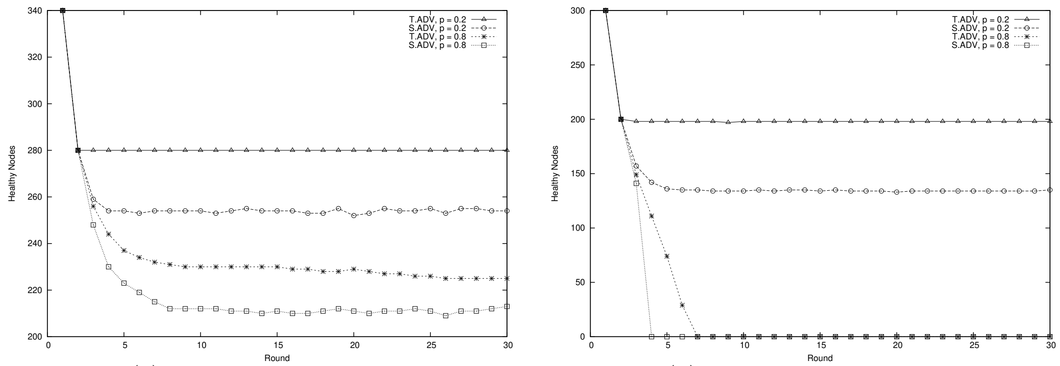
(a) $n = 400$ $k = 80$ $p = 0.2$ (b) $n = 400$ $k = 80$ $p = 0.8$

Fig. 5. Simulation with different values of t.

Figure 6 compares the performance of DISH under $\mathsf{T.ADV}$ and $\mathsf{S.ADV}$. We use a greedy O^r selection algorithm to simulate $\mathsf{S.ADV}$. Let H_i be the healthy sponsor set of s_i. The greedy algorithm is shown in Algorithm 3. Figure 6 shows when ADV is able to win the network, $\mathsf{S.ADV}$ achieves the goal faster than $\mathsf{T.ADV}$ does. Otherwise, $\mathsf{S.ADV}$ learns more sensor secretes than $\mathsf{T.ADV}$ learns.

Algorithm 3: O^r Selection

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



(a) $n = 400$ $k = 60$ $t = 6$ (b) $n = 400$ $k = 100$ $t = 6$

Fig. 6. Comparison between T.ADV and S.ADV.

7 Discussion

In this section, we discuss some limitations of the proposed technique and consider ways to mitigate them.

7.1 Attack Model Limitations

In this paper, we considered a relatively simple single-minded adversary who is only interested in learning secret keys of compromised sensors by reading all storage and eavesdropping on all traffic traversing these sensors. The proposed DISH scheme defends against such attacks, as discussed in Section 6.

However, it is not difficult to image other types of attacks that could be mounted by a more sophisticated ADV. For example, ADV can remain stealthy if it deletes existing measurements and replaces them with (the same number of) fraudulent measurements. Fraudulent data may change overall sensing statistics and affect sink's actions. Therefore, data integrity might be as important as data secrecy. We acknowledge that DISH cannot be applied directly to address data integrity since more issues (such as storage and bandwidth overheads) incurred by authentication need to be studied further.

Although it is in ADV's interest to be subtle, subtlety is not always possible. If ADV's goal is denial-of-service, by introducing fraudulent data, erasing existing measurements or interfering with legitimate communication, ADV cannot possibly avoid detection. In addition, nothing prevents ADV from physically destroying or damaging sensors, especially, since the network is unattended most of the time.

In summary, ADV can disrupt and attack the network in many other ways that are unaddressed by DISH. However, at least initially, we focused on the basic read-only type of adversarial behavior, since its successful mitigation will allow us to address more advanced (and perhaps more realistic) adversarial models in the future.

7.2 Communication and Sensor Model Limitations

We considered DISH in an *idealized* network model where no message is lost and no sensor fails. In this model, the sink can mimic the entire key evolution process for all sensors and re-generate all secret keys. However, message loss and/or sensor failures complicate this process.

Unreliable Communication & Reliable Sensors: If communication is unreliable but no sensor failures occur, a sensor might receive $< t$ contributions in a given round. It then considers the rest to be lost and records the ID-s of sensors whose it did not receive. This incurs additional storage and communication overhead of $O(p_l * t)$ per sensor per round, where p_l is message loss rate.

Unreliable Communication & Sensors: If both communication and sensors themselves are faulty, the sink cannot (later) mimic the correct key update process and is thus unable to decrypt all sensor data. It seems that no symmetric key-based approach (such as DISH) can fully address this problem, i.e., public key techniques are needed. There are two basic approaches to using public key cryptography in this context. In the first, each node encrypts its data with the sink's public key and uses K_i^r as input input to the randomized public key encryption function. In the second approach, each node encrypts sensed data with K_i^r and then uses the sink's public key to encrypt K_i^r. The security of the two approaches is the same. However, the latter is preferable if the size of sensed data exceeds the public key block size (e.g., 320 bits for Elliptic Curve ElGamal or 1024 bits for RSA).

7.3 Drawbacks of Reactive Sponsoring

The proposed DISH scheme is reactive in nature: a sensor selects its sponsors based on local pseudo-randomness and each sponsor generates a contribution to the next-round key. Reactive sponsoring has two drawbacks: First, it allows ADV to learn the sponsors of a sick sensor, thereby allowing more powerful S.ADV attacks. Second, it incurs the overhead of two messages for each contribution. An intuitive alternative is *proactive sponsoring*, whereby, in each round, every sensor unilaterally selects t sensors to sponsor. This simple change precludes ADV from learning the sponsor set of a sick sensor; thus, S.ADV attacks become ineffective. Also, without explicit sponsorship request messages, bandwidth overhead is reduced by half. We are currently conducting a detailed analysis and comparison of the two (proactive and reactive) approaches and hope to report on our findings in the near future.

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 [21, 1, 22, 2].

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 techniques for intrusion-resilient data secrecy in UWSNs. We proposed DISH, a symmetric key-based self-healing scheme that achieves both forward and (probabilistically) backward secrecy. DISH successfully mitigates the effect of sensor compromise. Our simulation results clearly demonstrate the efficacy of DISH against a stealthy mobile adversary.

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 *ACM CCS’97*, 1997.
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’02*, May 2002.
8. R. Anderson, “Two remarks on public-key cryptology - invited lecture,” in *ACM CCS’97*, April 1997.
9. M. Bellare and B. Yee, “Forward integrity for secure audit logs,” in *Technical Report, Computer Science and Engineering Department, University of San Diego*, 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’04*, 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 *ACM CCS’02*, 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’05*, 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.