GossiCrypt: Wireless Sensor Network Data Confidentiality Against Parasitic Adversaries

Jun Luo* Panos Papadimitratos† Jean-Pierre Hubaux†

***Abstract*— Resource and cost constraints remain a challenge for wireless sensor network security. In this paper, we propose a new approach to protect confidentiality against a *parasitic adversary*, which seeks to exploit sensor networks by obtaining measurements in an unauthorized way. Our low-complexity solution, GossiCrypt, leverages on the large scale of sensor networks to protect confidentiality efficiently and effectively. GossiCrypt protects data by symmetric key encryption at their source nodes and re-encryption at a randomly chosen subset of nodes en route to the sink. Furthermore, it employs key refreshing to mitigate the physical compromise of cryptographic keys. We validate GossiCrypt analytically and with simulations, showing it protects data confidentiality with probability almost one. Moreover, compared with a system that uses public-key data encryption, the energy consumption of GossiCrypt is from tens to thousands of times lower.**

I. Introduction

Wireless sensor networks (WSNs) have been an active field of research over the last few years, with a number of technical issues largely resolved. Onwards wider adoption, security becomes increasingly important and, eventually, security mechanisms a prerequisite [34]. Numerous significant efforts have been made along this line, including public-key cryptography (e.g., [42], [19]) as the means to digitally sign messages and establish symmetric keys, as well as symmetric-key based encryption and authentication for improved efficiency (e.g., [35], [24]). However, *sensor data confidentiality* has been largely overlooked to this date. Ensuring that sensor-collected data are accessed only by authorized entities has been viewed mostly as a secondary concern.

Encrypting data at their source sensor node, with a symmetric key shared with the sink, is a straightforward confidentiality mechanism. However, it does not fully address the problem at hand. An adversary can actively exploit the poor physical protection of nodes, as it would be too costly and thus unrealistic to make them tamper-resistant. It is relatively easy for an adversary to physically access the node memory contents [22], and extract the symmetric key used for data encryption. Such an attack is vastly simpler than a cryptanalytic one against the keys. In fact, the adversary could progressively compromise keys of numerous nodes, and eventually be able to decrypt a significant fraction of, if not all, data produced by the WSN.

*Jun Luo is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1. Email: j7luo@engmail.uwaterloo.ca.

†Panos Papadimitratos and Jean-Pierre Hubaux are with the School of Computer and Communication Sciences, EPFL (Swiss Federal Institute of Technology in Lausanne), CH-1015, Lausanne, Switzerland. Emails: {panos.papadimitratos, jean-pierre.hubaux}@epfl.ch.

We are concerned with sensor data confidentiality in such a setting, where cryptographic keys can be physically compromised. We focus on a novel type of adversary we term *parasitic*: it seeks to **exploit** a WSN, e.g., deployed for scientific measurements, industrial (mining, oil) field data, or even patients' health data collection, rather than disrupt, degrade, or prevent the WSN operation. A parasitic adversary, defined in detail in Sec. III, aims at obtaining measurements with the least expenditure of own resources, and the least disruption of the WSN it "attaches" itself to. Essentially, the longer the symbiotic relation of the adversary with a fully functioning WSN remains unnoticed, the more successful the parasitic adversary will be.

One naive solution against (symmetric) key compromise is to let sensors encrypt each outgoing measurement with the public key of the sink. As long as the sink is not compromised, it is the only one able to decrypt those message and the parasitic adversary is thwarted. However, software implementations of public-key operations, albeit computationally feasible, consume energy approximately **three orders of magnitude** higher than symmetric key encryption [36]. Hardware implementations of public key encryption (PKE), although significantly reducing energy consumption, remain accordingly costlier than symmetric-key encryption (SKE) hardware implementations (Sec. V-B).

Therefore, we are facing the challenge of protecting data confidentiality against parasitic adversaries in an energy efficient manner. To this end, we propose here **GossiCrypt**, whose mechanisms are tailored to and leverage on the salient features of WSNs. GossiCrypt comprises two building blocks: (i) a probabilistic *en-route re-encryption* scheme, with the source node always encrypting the data and with relaying nodes en route to the sink flipping a coin to "decide" whether to perform re-encryption, and (ii) a *key refreshing* mechanism that installs new sensor-sink shared symmetric keys to selected nodes.

Key refreshing is the immediate response to the compromise of a cryptographic key, but it can mitigate such an attack only to a certain extent: it is hard for the WSN operator to infer which keys were compromised. Also, running a network-wide key distribution protocol frequently can be very costly in an energy-constrained environment. More importantly, within two refreshing events, the adversary would still be fully capable to decrypt data from nodes whose keys were compromised. This is where the en-route re-encryption complements our (infrequent) key refreshing: data (or keys) can be decrypted by the adversary only if all the keys used for source and en-route encryption are compromised.

GossiCrypt has extremely simple key management requirements and very low complexity operation. Each sensor shares one data encryption symmetric key with the network sink. In addition, a single parameter drives probabilistically the participation of each node in en-route encryptions. This simplicity is inherent in gossiping protocols, with nodes flipping a coin to determine, e.g., if they should synchronize their databases or relay a message [14], [20]. This inspires the name of our scheme, as the decision is on (re-)encrypting rather than on relaying a packet. The simplicity in operation is also true for key refreshing performed with randomly chosen nodes. Overall, simplicity renders GossiCrypt broadly applicable.

Our main contribution is an efficient and highly effective, as our evaluation shows, scheme to ensure sensor data confidentiality. The objectives of GossiCrypt are specified in Sec. IV. We validate the effectiveness of our scheme analytically and experimentally. Attacked by a parasitic adversary that continuously compromises new nodes to obtain their encryption keys, GossiCrypt protects the confidentiality of data with probability almost one. At the same time, the comparison with a public-key encryption (PKE) solution shows that the GossiCrypt energy expenditure is significantly lower than that of PKE. Another contribution is the introduction of the parasitic adversary, a realistic type of attacker for a wide range of commodity and tactical WSNs. To the best of our knowledge, this is a novel yet realistic and highly effective, unless thwarted, type of adversary.

In the rest of the paper, we first provide the system and adversary models. Then, we present an overview of our scheme and present in detail its constituent protocols. In Sec. V and VI, we analyze our scheme and provide an experimental validation. Finally, we discuss related work in Sec. VII, before our future work and conclusion.

II. System Model

The WSN comprises N *sensor nodes*, each with a unique identity S_i, and a *network sink* Θ performing data collection and key refreshing. It is straightforward to consider multiple sinks, even with distinct roles, yet we omit this for simplicity in presentation. Each node S_i shares a symmetric key, $K_{i,\Theta}$, with the sink, and knows the public key, PuK_{Θ}, of the sink. The sink is equipped with all $K_{i,\Theta}$.

Beyond these *end-to-end*, sensor-to-sink, associations, nodes may share symmetric keys with their neighbors, to enable link-layer security primitives (e.g., TinySec [24]) or for other security purposes (e.g., revocation [12] or secure routing [25]). However, such security mechanisms are beyond the scope of this work and they can clearly coexist with our scheme.

We describe the data of interest with the help of two parameters, T and δ; the user seeks to collect data:

- From a fraction $0 < \delta \leq 1$ of the WSN nodes,
- Over a period of T seconds, for each node S_j, for $j = 1, \ldots, \lceil \delta N \rceil$.

The actual values of T and δ can vary. T can range from a short period, t_0, for a single sensor measurement, to a sufficiently long period for a comprehensive measurement collection. In general, $T = kt_0$, with $k > 0$ an integer. Similarly, $\delta = 1/N$, i.e., targeting at a certain node, may be meaningful, but in practice δ will be a significant fraction of N.[1] We do not dwell on the exact measurement extraction method, which can be performed in many ways orthogonal to our scheme.

We assume that N ranges from hundreds to thousands, as, for example, in WSNs for commercial inventory, habitat monitoring, industrial and mining field data, and geological measurements. Experience from prior deployments, with nodes' placement sparser than the monitored physical system and a relatively long history of measurements necessary to capture the studied phenomena, teaches that data sensed by each and every node is significant. This implies that in-network data aggregation is not an option in such deployments; we assume this is so in this work. We also assume WSNs enabling applications that do not undergo development. Thus, the entire operating system (apart from certain tunable parameters) is stored in *read-only memory* (ROM). Finally, WSN nodes are not tamper-resistant or store cryptographic keys in tamper-resistant components, due to cost considerations.

III. Adversary Model

We identify a new type of adversary we denote as *parasitic*. Its objective is to exploit deployed wireless sensor networks, by accessing in an unauthorized manner data collected by those WSNs. More specifically, a parasitic adversary:

1. Seeks to obtain the WSN data collected according to the parameters δ and T.
2. Can be physically present, at each point in time, **only** at a much smaller fraction of the area covered by $\lceil \delta N \rceil$ sensor nodes.
3. Can physically access data stored at sensor nodes and retrieve their cryptographic keys.
4. Can be mobile [31], i.e., compromise different sets of nodes over different time intervals. "Mobile" traditionally refers to virtual moves (in terms of compromising system entities); here, it also represents physical moves of the adversary.
5. Can compromise in the above-described manner at most one sensor per τ seconds. We assume $\tau \ll T$.

The characteristics of the parasitic adversary reflect its realism. Constrained presence (assumption 2) is meaningful, because, otherwise, the adversary could deploy its own WSN and trivially obtain the data the WSN user collects (assumption 1). It exploits obvious weaknesses of WSNs (assumption 3): poor physical protection makes it relatively easy to obtain data encryption keys [22]. The parasitic adversary is *unobtrusive*, that is, cannot modify the implemented protocols stored in ROM (Sec.II). Furthermore, it can utilize its resources intelligently. Mobility (assumption 4), illustrated in Fig. 1, shows that the adversary can be in the proximity of different nodes for periods of time during which it either compromises the node or obtains snapshots of the measurement histories.

The strength of the adversary is evident from assumption 5: the time needed to physically compromise a single node, albeit

[1] WSNs deployed for (often one-time) event detection (e.g., forest fire or bridge structural faults) would correspond to $\delta = 1$, and T equal to the period from the WSN deployment to the event/alarm occurrence.

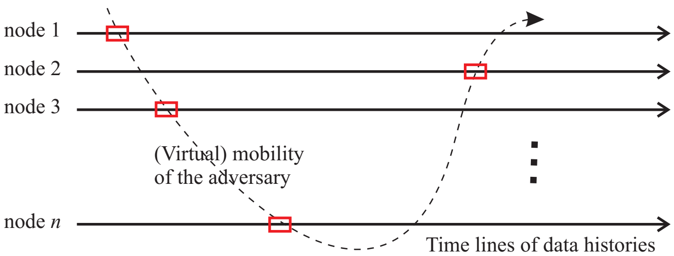


Fig. 1. Mobility of the parasitic adversary.

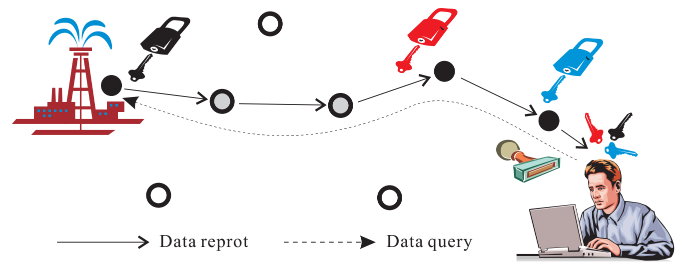


Fig. 2. Securing data collection with GossiCrypt and query authentication (μTESLA [32] for example).

significant if nodes are carefully designed, is much shorter than T, the period over which data are to be collected. In other words, the benefit of the adversary from compromising sensor nodes is far reaching. Compromising a key does not imply the adversary automatically obtains the node's measurements. The adversary could remain within range of the compromised node and achieve that. But such an attack would be self-defeating: from assumption 2, the adversary would certainly capture much less than $\lceil \delta N \rceil$ measurements. From a different point of view, assumption 2 captures the difficulty to deploy a network of eavesdroppers within one hop of all previously compromised nodes. The eavesdroppers' transceivers would need to be highly sensitive (and thus more expensive than that of a sensor node) to cover a meaningful fraction of the targeted WSN. Overall, leaving "sentry" nodes behind would be comparable to the deployment of a WSN by the adversary.

We assume that the protocol design and implementation are such that remote node compromise is prevented. For example, the adversary cannot exploit arbitrary software weaknesses and make a sensor node disclose its cryptographic keys. Such robustness should be possible given the relatively simple functionality of WSN node software, compared to that of more complex systems (e.g., desktop or portable computers). We also assume that the sink *cannot* be compromised by the adversary. Readers are referred to [44] for the investigations on compromise of low-end mobile sinks. Moreover, denial-of-service (DoS) attacks, including jamming in various protocol layers [43], [28], Sybil/Node replication attacks [30], [33], or "wormhole" formation [8], [37] are beyond the scope of this work: countermeasures to those attacks can coexist with our protocols. Neither do we consider physical destruction of WSN nodes, which would not benefit the adversary.

IV. GossiCrypt

GossiCrypt aims at ensuring confidentiality, that is, preventing any unauthorized access to data collected by a WSN. It does not seek to protect data coming from every single sensor, but rather intends to fulfill the following property, for some protocol-specific constant $0 < \Delta < 1$:

Δ_T-**Confidentiality:** *Data collected from a WSN comprising N nodes are Δ_T-confidential if the adversary cannot obtain all measurements performed by more than $\lceil N\Delta \rceil$ sensor nodes over a given time interval T.*

This is a *safety* property, i.e., a property related to a system-specific unwanted situation: obtaining measurements from a given fraction of sensor nodes over a period of time, meaningful with respect to the system and application, is prevented. In Sec. V-A we will show that GossiCrypt satisfies this property against parasitic adversaries with probability almost one.

We emphasize that GossiCrypt does not seek to provide sensor data authenticity and integrity. The reason is that if a key is compromised, an adversary (not necessarily a parasitic one) can impersonate the corresponding sensor and inject fabricated messages. Nonetheless, data that originate from non-compromised nodes have their authenticity and integrity protected. We also clarify that GossiCrypt does not seek to hide the identities of sensor nodes, achieve data source untraceability, or satisfy any notion of anonymity, unlinkability, or privacy. Clearly, confidentiality relates to privacy, but, again, all GossiCrypt seeks to provide is the *confidentiality* of the data provided by sensor nodes.

A. Data Encryption

We distinguish sensor nodes into two categories, *data sources* and *relaying nodes*, with each node assuming either role at different points in time. We denote the data encryption operation of GossiCrypt as $GossiCrypt_E$ and illustrate in Fig. 2: it is executed by nodes on the path from a data source to a sink (inclusive), with the outcome (i.e., re-encrypting or not) at each relaying node being random (with probability $\boldsymbol{q}$).

The path may be one hop, if the sink is within the transmission range of the sensor node, but this is not cost-effective; in general, the sink is at a distance of multiple hops from data source(s). The path discovery is orthogonal to $GossiCrypt_E$. It can be determined by a (secure) routing protocol, for example, forming an authenticated tree rooted at the sink [35], possibly on-the-fly, as a result of the query sent out from a sink. $GossiCrypt_E$ can be employed on top of any path discovery protocol and does **not** impose extra requirements. For the rest of the discussion, we assume that, minimally, each S_i knows the next node towards Θ on a $path_{S_i,\Theta}$ without the transmitted packet carrying the routing information.

For a sensor measurement m, a symmetric key $K_{i,\Theta}$ shared by Θ and S_i, a message authentication code $MAC(K_{i,\Theta},\ldots)$, and $q \in (0,1)$ the protocol-specific parameter governing the re-encryption, $GossiCrypt_E(K_{i,\Theta}, path_{S_i,\Theta}, q, m)$ is invoked by S_i acting as a source:

1. **Source node, S_i:**
   1.a. Generate a nonce n for the communication with sink Θ.
   1.b. Calculate $H = MAC(K_{i,\Theta}, m, n, S_i)$.
   1.c. Encrypt m, n, H with $K_{i,\Theta}$ to obtain ciphertext $\sigma_i = \{m, n, H\}_{K_{i,\Theta}}$.
   1.d. Transmit packet $p_i = \sigma_i, S_i$ to the first relaying node S_j on $path_{S_i,\Theta}$.
2. **Relaying node, S_j:**
   2.a. Upon receipt of a packet p_i, generate a random number $x \in [0, 1]$. If $x > q$, relay p_i to the next relaying node S_k on $path_{S_j,\Theta}$, or to Θ. Otherwise,
   2.b. Generate ciphertext $\sigma_j = \{p_i\}_{K_{j,\Theta}}$.
   2.c. Append own identity S_j to σ_j.
   2.d. Relay packet $p_j = \sigma_j, S_j$ to the next relaying node S_k along $path_{S_j,\Theta}$, or to Θ.
3. **Sink Θ:**
   3.a. Upon receipt of a packet p_k, retrieve $K_{k,\Theta}$, the key shared with S_k, and decrypt σ_k. If the source clear-text, m, n, H, is obtained, go to (c). Otherwise,
   3.b. Obtain ciphertext σ_l and S_l. Decrypt σ_l with $K_{l,\Theta}$. Repeat successively for all S_l that re-encrypted the packet, till obtaining the source clear-text m, n, H.
   3.c. Determine if n was previously seen. If so, discard the packet. Otherwise,
   3.d. Compute $H' = MAC(K_{i,\Theta}, m, n, S_i)$. Discard the packet if $H' \neq H$. Otherwise, deliver m to the WSN user.

B. Key Refreshing

To defend against the progressive compromise of an increasing number of nodes, $K_{i,\Theta}$ keys should be refreshed, i.e., replaced with new $K'_{i,\Theta}$ keys. The sink is typically unaware of which nodes are already compromised. Thus, it selects randomly an S_i node to refresh, among a set of $N' \leq N$ nodes. This selection is, in general, made among the data source nodes of interest (the δ fraction of N as defined in Sec. II), and all the intermediate nodes that connect those sources to the sink. In other words, the refreshing effort focuses on the same part of the network that is meaningful for the adversary to target.

Given a particular system design for the nodes, it is not very difficult to have an arguably pessimistic estimation of the rate of physical node compromise, as per Sec. III. Then, based on this estimate of τ^{-1}, the *key refreshing rate* λ_r can be selected accordingly by the sink, and conveyed to all nodes via an authenticated control message. Confidentiality of λ_r is not needed, as the adversary would, at best, compromise nodes at its maximum possible rate τ^{-1}. Authenticity, however, is clearly required, to ensure that an active adversary does not "slow down" the key refreshing.

Symmetric key based key transport techniques, similar to those in [1], are effective only if the adversary, having previously compromised $K_{i,\Theta}$, cannot intercept the refresh protocol messages. Moreover, an interactive key establishment protocol, for example, initiated by the sink, would reveal the identity of the node whose key is being refreshed. The adversary could easily eavesdrop all messages sent and received from the sink, and hence gain a significant advantage: to know which nodes were refreshed and then re-compromise them.

To thwart these two vulnerabilities, we propose a key refreshing protocol with two variants. This is essentially a key transport protocol; but it leverages on (i) the $GossiCrypt_E$ operation, with optional public key encryption at the source sensor node, and (ii) the integration of the key refreshing with the data collection. As a result, the key refreshing protocol is similar to the data encryption protocol, presented in Sec. IV-A. There are two main differences: a random point process generator [10] $RGen(\lambda_r)$ used to generate (key refreshing) events with intensity λ_r, and a *flag* set to indicate to the sink that a new key $K'_{i,\Theta}$ is included in the message (which, otherwise, externally appears identical to any measurement/data reporting message). The protocol operates as follows:

1. **Source node, S_i:**
   1.a. Upon an event of $RGen(\lambda_r)$, generate a new key $K'_{i,\Theta}$; wait for the time till the next data report.
   1.b. Upon a data report to be returned, delay the report to be combined with the next one and generate a nonce n for the communication with sink Θ.
   1.c. Calculate $H = MAC(K_{i,\Theta}, \mathit{flag}, K'_{i,\Theta}, n, S_i)$.
   1.d. Encrypt $\mathit{flag}, K'_{i,\Theta}, n, H$ with $K_{i,\Theta}$, to obtain ciphertext $\sigma_i = \{\mathit{flag}, K'_{i,\Theta}, n, H\}_{K_{i,\Theta}}$.
   1.e. Transmit packet $p_i = \sigma_i, S_i$ to the first relaying node S_j on $path_{S_i,\Theta}$.
2. **Relaying node, S_j:**
   Identical to the operation for $GossiCrypt_E$ (Sec. IV-A).
3. **Sink Θ:**
   3.a. Perform the steps (3).(a)-(b) as specified in Sec. IV-A, to obtain the clear-text $\mathit{flag}, K'_{i,\Theta}, n, H$.
   3.b. Determine if n was previously seen. If so, discard the packet. Otherwise,
   3.c. Calculate $H' = MAC(K_{i,\Theta}, \mathit{flag}, K'_{i,\Theta}, n, S_i)$. If $H' \neq H$, discard the packet. Otherwise, replace $K_{i,\Theta}$ with $K'_{i,\Theta}$.

The protocol installs a new key even if the adversary intercepts the message en route to the sink, unless the adversary is physically within one hop from the previously compromised and now to-be-refreshed S_i. In the later case (which is extremely rare due to the restrained physical present of an adversary), the adversary can decrypt the message and obtain the key. To prevent this, we propose the following variant of the above protocol:

1. **Source node, S_i:** Identical to the above key refreshing operation, with the additional step between (b) and (c), and replacing $K'_{i,\Theta}$ with $\sigma\kappa_i$ afterward:
   1.b$^+$. Encrypt $K'_{i,\Theta}$ with PuK_Θ, the public key of the sink, and obtain the ciphertext $\sigma\kappa_i = \{S_i, K'_{i,\Theta}\}_{PuK_\Theta}$.
2. **Relaying node, S_j:**
   Identical to the operation for $GossiCrypt_E$ (Sec. IV-A).
3. **Sink Θ:** Identical to the above key refreshing operation, with the additional step:
   3.d. Decrypt $\sigma\kappa_i$ with PrK_Θ and check if the obtained node identity is S_i. If so, replace $K_{i,\Theta}$ with $K'_{i,\Theta}$.

This second variant's use of PKE resembles mechanism 1 of the ISS/IEC 11770-3 standard [2]. It ensures that even in the unlikely event the adversary is within one hop of the refreshed node, still, it cannot obtain the new $K'_{i,\Theta}$. The only option for the adversary would be to re-compromise S_i.

V. Protocol Analysis

We analyze the security level of GossiCrypt and also compare its energy expenditure with a possible alternative in this section. Our security analysis focuses only on the parasitic adversary; further discussion on general adversaries are given in Sec. VII. The security analysis applies to both data encryption and key refreshing (with or without PKE) protocols, as they essentially follow the same principle.

A. Security Analysis

In this section, we describe a model of GossiCrypt and evaluate it against the Δ_T-Confidentiality property (Sec. IV) and the parasitic adversary (Sec. III). Our analysis, accompanied by simulation results in Sec. VI, shows that even with a significant fraction of sensor nodes compromised, GossiCrypt safeguards confidentiality **with probability almost one**.

Fundamental for the analysis is the fraction of *correct*, i.e., not compromised, nodes; this is determined by the behaviors of the sink refreshing and the adversary compromising keys. Therefore, we model the *state* of the system, the number of correct nodes, as a stochastic process. Our security analysis on GossiCrypt is based on the stationary regime of this process.

Since the sink cannot in general know which keys are already compromised, a randomized strategy on selecting which node to refresh is a reasonable choice. We assume that the sink does so with an effective[2] refresh rate λ. Recall that the sink governs the selection procedure through setting the parameter λ_r.

The adversary, compromising nodes at rate τ^{-1}, is also modeled as selecting the next node to compromise (or to test if the key was refreshed)[3] arbitrarily. This is so, because the adversary would not be more effective by choosing a deterministic attack pattern. To illustrate this, consider a static sink network, where the adversary might gradually, over a long period of eavesdropping, infer (part of) the sensors-sink communication paths. Such an adversary could first compromise the sink neighboring sensor nodes, and then move outwards, compromising their upstream nodes. This might allow the adversary to fight back only against symmetric-key based refreshing: this would be possible only if it compromised the entire path connecting the refreshed node to the sink. In contrast, this attack would be completely ineffective against a public key based refreshing (as described in Sec. IV-B). Therefore, the deterministic, targeted compromise pattern would be no more effective than a random one. [4] Although an adversary physically close to a source node S_i, may detect a key-refreshing, its physical presence is limited to a negligible fraction of the network. Note that re-encryption deprives the adversary from this ability elsewhere.

The system size depends on the behavior of the sink. If the sink is static and the data collection paths change slowly, if at all, over time, both the sink and the adversary could have a clear view on which nodes they need to target: the source sensor nodes of interest and the relaying nodes en-route to the sink. Or better even, from the adversary's point of view, the slightly smaller subset of sources and relaying nodes en-route to the point it intercepts the measurement packets. As a result, the system is this known subset of nodes with size $N' < N$. On the other hand, if a mobile sink is used [23], [41], [27], the adversary cannot predict the data collection paths. This results in a larger system size, which essentially can be all nodes, offering higher robustness against the adversary at the expense of complexity in operating the mobile sink. We emphasize however that our analysis is applicable to both cases. All one needs to do is to view N below as the effective system size.

We assume that the times of performing refreshing and compromising can be modeled as two independent Poisson processes with intensities λ and τ^{-1} respectively. We also assume that, at each time point in the processes, either the sink approaches a node and refreshes it or the adversary captures a node and compromises it, no matter whether the node has been compromised or not. The Poissonian and independence assumptions are not essential. The easily drawn analogies between our model and the teletraffic models [7] imply that the stationary distribution is insensitive to all other characteristics beyond the intensities.

Based on these assumptions, we describe the system states as a continuous Markov chain $\{X(t)\}_{t\geq 0}$ driven by the Poisson processes. Since such a chain is characterized by its subordinated chain $\{\hat{X}_n\}_{n\geq 0}$ [10], we focus on this discrete Markov chain. A direct observation on the system is that the more numerous the compromised nodes, the less the efficiency of the adversary (thus the higher the efficiency of the sink) is and vice versa. The reason is clear: when many nodes are compromised, the probability of fruitlessly re-compromising becomes high. This reminds us of the celebrated model described by Paul and Tatiana Ehrenfest (sometimes referred to as The Urn of Ehrenfest) [16] for understanding the diffusion through a porous membrane.[5] The system we

[2] The model covers the two options (with or without PKE) of the key refreshing protocol described Sec. IV-B. Although the key refreshing without PKE might allow the adversary to obtain the new key, it is still highly possible that new keys are not exposed to the adversary, as the adversary cannot be ubiquitously present (also pointed out in [5]). Thus, the model still applies but with the refreshing rate λ_r discounted by a factor.

[3] A model that assumes the rate of testing differing from that of compromising does not fundamentally change the stationary distribution.

[4] This is also true when one considers that public key encryption (PKE) refreshing is robust against an adversary in range of a refreshed node. We also note that it is also possible that the sink counters deterministic attack patterns with similarly structured refresh patterns. However, investigation of those albeit interesting is not provided here due to space limitations. For example, the efficiency of the scheme could greatly enhanced if the public key refreshing protocol is run with nodes near the sink, to "break" chains of fully compromised paths and make symmetric-key refreshing effective even against this deterministic attack.

[5] The model can be briefly described as follows [10]: there are N particles that can be either in compartment A and B. Suppose at time t, there are i particles in A. The diffusion process behaves as if someone chooses a particle at random and moves it to another compartment at time $t+1$. Therefore, the transition probability is $p_{ij} = \frac{i}{N}$ $(j = i-1)$, or $\frac{N-i}{N}$ $(j = i+1)$, or 0 (otherwise).

consider differs from the Urn of Ehrenfest in that the "self" transition probability is non-zero (i.e., $p_{ii} > 0$) and also that the transition probability depends on the rates λ and τ^{-1}.

Therefore, the transition matrix of the subordinated chain $\{\hat{X}_n\}_{n\geq 0}$ is as follows:

$$\mathbf{P} = \begin{bmatrix} s_0 & \nu_0 & & & & & \\ \mu_1 & s_1 & \nu_1 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \mu_i & s_i & \nu_i & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \mu_{N-1} & s_{N-1} & \nu_{N-1} \\ & & & & & \mu_N & s_N \end{bmatrix}$$

where i is the number of correct nodes in the system, $\mu_i = \frac{i}{N\tau(\lambda+\tau^{-1})}$ and $\nu_i = \frac{(N-i)\lambda}{N(\lambda+\tau^{-1})}$ represent the transitions resulting from a compromising and a refreshing, respectively, and $s_i = \frac{N-i}{N\tau(\lambda+\tau^{-1})} + \frac{i\lambda}{N(\lambda+\tau^{-1})}$ expresses those fruitless operations. One can easily see that this is a birth-and-death process in continuous time with reflecting barriers at 0 and N [10]. The chain $\{\hat{X}_n\}_{n\geq 0}$ is *irreducible* (i.e., every state is reachable from all other states) and *positive recurrent* (i.e., the system does not freeze at some states). It has the following stationary distribution (the detailed computation is omitted):

$$\pi_0 = \left\{1 + \frac{\nu_0}{\mu_1} + \frac{\nu_0\nu_1}{\mu_1\mu_2} + \cdots + \frac{\nu_0\nu_1\cdots\nu_{N-1}}{\mu_1\mu_2\cdots\mu_N}\right\}^{-1} \quad (1)$$

$$\pi_i = \pi_0 \frac{\nu_0\nu_1\cdots\nu_{i-1}}{\mu_1\mu_2\cdots\mu_i} \quad (2)$$

Note that this is also the stationary distribution of $\{X(t)\}_{t\geq 0}$. It has the following properties:

- The system can rarely be free either of correct nodes ($X(t) = 0$) or of compromised nodes ($X(t) = N$), because both π_0 and π_N vanish with increasing N.
- The most likely state (i.e., $\arg\max_i \pi_i$) lies between 0 and N; it depends on the magnitude of λ and τ^{-1}. The larger the value of $\lambda\tau$ (the ratio between the rate of refreshing and that of compromising) is, the closer is this state to N.

These two properties can be easily observed in Fig. 3. It shows that even if the sink is more efficient than the adversary ($\lambda\tau = 1.5$, the red curve), there are still approximately 40% compromised nodes.

Now, we evaluate the probability of having at least one correct node re-encrypting the data on a routing path of length L from a source to the adversary. Let a random variable Y be the number of correct nodes re-encrypting the data and hence

$$Y = \sum_{m=1}^{M} \Omega_m \qquad M \leq L \quad (3)$$

where M is the random variable representing the number of nodes that re-encrypt the data and $\{\Omega_m\}$ are i.i.d. Bernoulli variables indicating the state of each of the M nodes ($\Omega_m = 1$ if correct and 0 otherwise). We want to calculate $\mathrm{P}\{Y > 0\} = 1 - \mathrm{P}\{Y = 0\}$, the *success probability* (in the sense that GossiCrypt successfully provides confidentiality). To this end, we make use of the generating function $g_Y(z)$ of Y, because

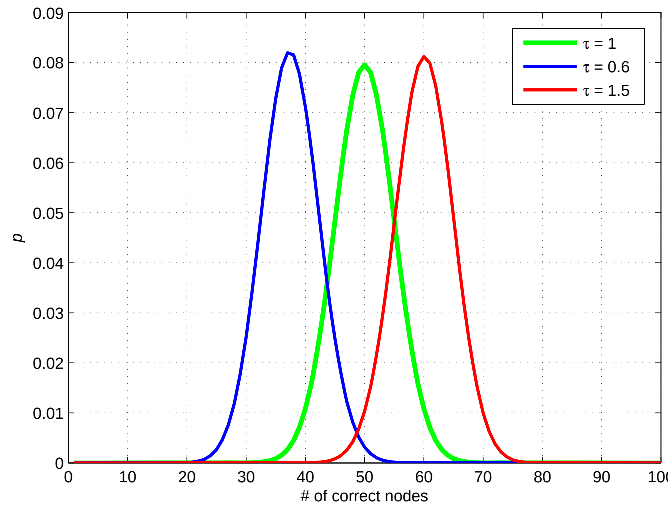


Fig. 3. Stationary distribution π with $N = 100$, $\lambda = 1$, and $\tau = 0.6, 1, 1.5$. The y-axis is the probability density corresponding to a certain number of correct nodes. Since only the product $\lambda\tau$ matters, we choose the values of λ and τ arbitrarily without a dimension.

$\mathrm{P}\{Y = 0\} = g_Y(0)$ and, by the rule of *random sum of i.i.d. variables* [10], $g_Y(z) = g_M(g_\Omega(z))$. Therefore,

$$\begin{aligned}\mathrm{P}\{Y = 0\} &= g_M(g_\Omega(0)) \\ &= \mathrm{E}_M[\mathrm{P}\{\Omega_0 = 0\}^m] \\ &= \sum_{m=1}^{L} \mathrm{P}\{\Omega_0 = 0\}^m \binom{L}{m} q^m (1-q)^{L-m} \quad (4)\end{aligned}$$

Given the stationary distribution π of $\{X(t)\}_{t\geq 0}$,

$$\begin{aligned}\mathrm{P}\{\Omega_0 = 0\} &= \sum_{i=0}^{N} \mathrm{P}\{\Pi, X(t) = i\} \\ &= \sum_{i=0}^{N} \frac{N-i}{N}\pi_i = \frac{N - \mathrm{E}_\pi(X)}{N} \quad (5)\end{aligned}$$

where Π is the event of picking a node within $N - i$ compromised ones. We illustrate the success probability $\mathrm{P}\{Y > 0\}$ under different values of L and q in Table I, assuming $N = 100$, $\lambda = 1$, and $\tau = 1.5$. One might think the case where $\mathrm{P}\{Y > 0\} = 0.8258$ (for $L = 5$ and $q = 0.5$) is an unfavorable bet for the legitimate user (because the adversary is able to decrypt the data with probability 0.1742); the adversary, however, gains nothing from this. To understand this point, we refer again to Fig. 1. Since what the adversary might decrypt (with probability 0.1742) is just a snapshot, the probability of observing the whole data history goes to zero (the probability of obtaining three snapshots is already very low: $0.1742^3 = 0.0053$). Note that we take for granted that the events of decrypting two different snapshots are independent; this is guaranteed by the coin flipping procedure even if two snapshots are transmitted through the same routing path.

We analyzed to this point the system state process and the per-message protection due to GossiCrypt given the path length L. In general, L is a random variable. If we knew its probability distribution $\mathrm{P}(L)$, the probability of breaking the

L \ q	0.5	0.6	0.7	0.8	0.9
5	0.8258	0.8875	0.9303	0.9590	0.9773
6	0.8772	0.9273	0.9591	0.9783	0.9894
7	0.9134	0.9531	0.9760	0.9886	0.9950
8	0.9390	0.9697	0.9859	0.9940	0.9977
9	0.9570	0.9804	0.9917	0.9968	0.9989
10	0.9697	0.9873	0.9951	0.9983	0.9995
11	0.9786	0.9918	0.9871	0.9991	0.9998
12	0.9849	0.9947	0.9983	0.9995	0.9999

TABLE I

SUCCESS PROBABILITY $\mathrm{P}\{Y > 0\}$ UNDER DIFFERENT VALUES OF L (PATH LENGTH) AND q (COIN FLIP PROBABILITY).

confidentiality of a single measurement ($T = t_0$) from a given node ($\Delta = 1/N$) would be

$$\mathcal{F}_{t_0, \frac{1}{N}} = \mathrm{E}_L[1 - \mathrm{P}\{Y > 0\}] \quad (6)$$

What we are interested though, as per our specification, is the confidentiality with respect to any $\Delta \geq 1/N$, and $T = kt_0$ for integer $k \geq 1$. Clearly, it depends on $\mathrm{P}(L)$ that is a complicated consequence of the relative placement of the sink and sources, as well as the patterns by which the adversary compromises nodes and the sink refreshes them. As a result, we proceed without making an assumption on $P(L)$ and describe the property of GossiCrypt in an asymptotical sense.

Claim: *GossiCrypt guarantees the* Δ_T-**Confidentiality** *property for* $\Delta \geq 1/N$ *with probability* $\mathcal{P}$ *(with* N *being the system size), and* $\mathcal{P} \to 1$ *when* $T \gg t_0$.

Proof: As it is at least as hard to breach the confidentiality of two or more measurements as that of a single one, it is clear that $\mathcal{F}_{t_0,\Delta} \leq \mathcal{F}_{t_0,\frac{1}{N}}$ for any $\Delta > \frac{1}{N}$. The strict inequality holds if the events of compromising two or more measurements are independent. Furthermore, we have that $\mathcal{F}_{T,\Delta} = (\mathcal{F}_{t_0,\Delta})^k$ for $T = kt_0, k > 0$. Therefore, $\mathcal{P} = 1 - \mathcal{F}_{T,\Delta} \geq 1 - (\mathcal{F}_{t_0,\frac{1}{N}})^k \to 1$ if $k \to \infty$. In other words, as k grows, the probability of safeguarding the confidentiality of Δ measurements over a period T goes to one. Literally, if the data history to be captured is sufficiently long, there is virtually no opportunity for the adversary to succeed in breaking its confidentiality. ∎

As shown in Fig. 3, it is always preferable to have $\lambda\tau > 1$ (although $\lambda\tau < 1$ can be compensated by aggressively setting q). This is not hard to achieve because, whereas the adversary obtains keys via its physical presence, the key refreshing is performed automatically and remotely. A conservative way to achieve this is to estimate $\tau_{\min}$ (the lower bound of τ) and to set $\lambda > \tau_{\min}^{-1}$. Estimating τ online can be preferable. We also note the the convergence of $\mathcal{P}$ persists even if $\lambda\tau < 1$ but, of course, with a lower speed.

B. Energy Expenditure

As we mentioned in Sec. I, applying PKE is an alternative solution to thwart a parasitic adversary. We will show in this section that, a sound in theory PKE-based solution is inferior to GossiCrypt due to the much higher energy expenditure it incurs.

For a quantitative comparison between PKE and GossiCrypt, we make the following assumptions:

1. The network size $N < 2^{16}$, so node identity S_i needs at most 16 bits.
2. Each message has a length of 20 bytes.
3. GossiCrypt makes use of AES-128 encryption.
4. The PKE can either be RSA-1024 or ECC-160.[6]
5. The energy expenditure for transmission is 0.21 $\mu J/\mathrm{bit}$.

The transmission cost refers to MICA2 nodes, and so are the computation delays for cryptographic operations, and the related power dissipation, based on available experimental results. Note that the fourth assumption strongly favors PKE, with its 80-bit security compared with the AES 128-bit security level. The energy costs are taken from [36]. Although hardware implementations could significantly reduce energy consumption for all primitives [21], [6], [17], the order of difference is maintained.

Table II compares GossiCrypt with two variants of PKE in terms of computation[7] and communication complexity.

	GossiCrypt	PKE-RSA	PKE-ECC
Comp.	32.4 $\mu J/\mathrm{msg}$	14.1 mJ/msg	53.4 mJ/msg
Comm.	An increase of $16q$ bits per message per hop	1024 bits per message	320 bits per message

TABLE II

COMPARISON BETWEEN GOSSICRYPT AND PKES.

We have the following observation on Table II: First, the energy expenditure in computation of GossiCrypt at a source node is 2 to 3 orders of magnitude lower than the those of PKEs. Second, the energy expenditure in communication of GossiCrypt for each node en-route remains lower than those of PKEs up to $10q^{-1}$ (for PKE-ECC) and $54q^{-1}$ (for PKE-RSA) hops (note that $q < 1$).

It is clear that the communication cost of GossiCrypt is lower than that of PKE-ECC below $10q^{-1}$ hops and that of PKE-RSA below $54q^{-1}$ hops. We assume the scale of the WSN meets these criteria and we only compare the computation cost below. Note that assuming 20 bytes message actually favors PKE-ECC, whose cost would be doubled if, for example, the message were one byte longer.

The additional computation cost for GossiCrypt compared with PKE stems from key refreshing; we denote it as c_{refresh}. Based on the analysis in Sec. V-A, let us assume refresh rate equal to the adversary compromise rate (i.e., $\lambda\tau = 1$). For $T = kt_0$, let $\tau = T/k$ as per the definition of the parasitic adversary, or in other words, the adversary compromises one node per measurement period t_0. Then, for a (sub-)network of N nodes among which the sink picks randomly, each node will be refreshed on the average once every N measurement

[6]Rabin PKE, in theory, is more efficient than RSA (though the difference can be as low as one modular multiplication for low RSA exponent operations) [29]. However, we are not aware of sensor network software implementations for Rabin PKE. Moreover, Rabin appears to be costlier than RSA certain implementations in other platforms [13].

[7]The computational complexity is measured in different units for symmetric-key and public-key encryption in [36]. So we need to fix the message size in order to compare them.

periods. The advantage for GossiCrypt per source node is approximately the ratio of $\frac{N\times c_{\mathrm{GC}}+c_{\mathrm{refresh}}}{N\times c_{\mathrm{PKE}}} \approx \frac{N+1}{N}\frac{c_{\mathrm{GC}}}{c_{\mathrm{PKE}}}$ without public-key encryption (as $c_{\mathrm{GC}} \approx c_{\mathrm{refresh}}$) or $\approx \frac{1}{N}\frac{c_{\mathrm{refresh}}}{c_{\mathrm{PKE}}}$ with public-key encryption (as $\frac{c_{\mathrm{GC}}}{c_{\mathrm{PKE}}} \ll 1$), where c_{GC} and c_{PKE} are the computation costs for GossiCrypt and PKEs, respectively, given in Table II.

As the advantage of GossiCrypt over PKEs is **tremendous** without public-key encryption, we only consider the key refreshing with ECC-based public-key encryption. In this case, the cost of refreshing is dominated by one ECC encryption, thus $\frac{c_{\mathrm{refresh}}}{c_{\mathrm{PKE}}} \approx 1$. Therefore, the ratio $\frac{1}{N}\frac{c_{\mathrm{refresh}}}{c_{\mathrm{PKE}}}$ decreases as N grows, thus making GossiCrypt increasingly advantageous. For example, if $N = 100$, GossiCrypt can be **100 times** less costly then PKE-ECC. For PKE-RSA, $c_{\mathrm{refresh}} \approx 3c_{\mathrm{PKE}}$ and GossiCrypt is still 33 times less costly. However, the very high communication cost of PKE-RSA is a significant disadvantage that makes PKE-RSA infeasible.

VI. Experiment Results

We perform simulations in Matlab. We only simulate the operations of GossiCrypt without taking the MAC/PHY effects into account. We assume a grid network where nodes appear on a $\sqrt{N} \times \sqrt{N}$ square lattice. The movements[8] of both sink and adversary follow a 2D random walk: they take identical probability $1/4$ in choosing one direction out of four possibilities. The intervals between two successive events of moving follow exponential distributions with mean λ^{-1} and τ for the sink and the adversary, respectively. To remove the boundary effect, we project the lattice on a torus, i.e., moving out of the one side of the lattice leads to an entering on the opposite side. We illustrate these settings in Fig. 5.

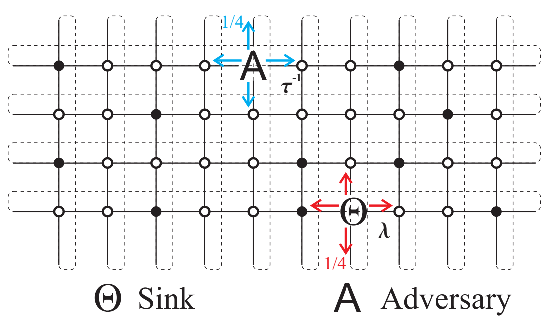


Fig. 5. Simulation settings.

Since the stochastic process described above can be proved to be aperiodic and positive recurrent, all the states are ergodic [10]. Therefore, we can use statistics over time to characterize the stationary distribution. We run each simulation for 11000 transitions and truncate the first 1000 points (which are in transient phase), such that the results are measured in steady state. Fig. 4 (a) shows the comparison between four empirical stationary distributions resulting from four simulation runs and the analytical one obtained in Sec. V-A, with $N = 100$, $\lambda = 1$, and $\tau = 1.5$. It is clear that the analytical results describe the stationary regime of the system very well.

Based on these statistics, we can again verify the success probability $\mathrm{P}\{Y > 0\}$ by randomly choosing routing paths between nodes and the adversary. For briefness, we only illustrate the case with $L = 6$ in Fig. 4 (b) (showing the medians and 95% quantiles) and compare the results with the analytical ones shown in Table I. The comparison shows that the analytical results are a bit overoptimistic, but the differences with the experiment results are negligible.

Finally, we verify our claim that GossiCrypt guarantees the Δ_T-**Confidentiality** property with probability almost one when $T = kt_0$ is sufficiently long. To this end, we randomly pick two nodes on the grid and considering one as the source and the other as the data collector. By applying GossiCrypt to the shortest path between the two nodes, we can evaluate the quantity $\mathcal{F}_{kt_0,\frac{1}{N}}$ for different value of k. As shown in Fig. 6, this probability converges very fast to zero with an increasing k, according to both simulation and analytical results. This corroborates our claim that $\mathcal{P} = 1 - \mathcal{F}_{T,\Delta} \to 1$.

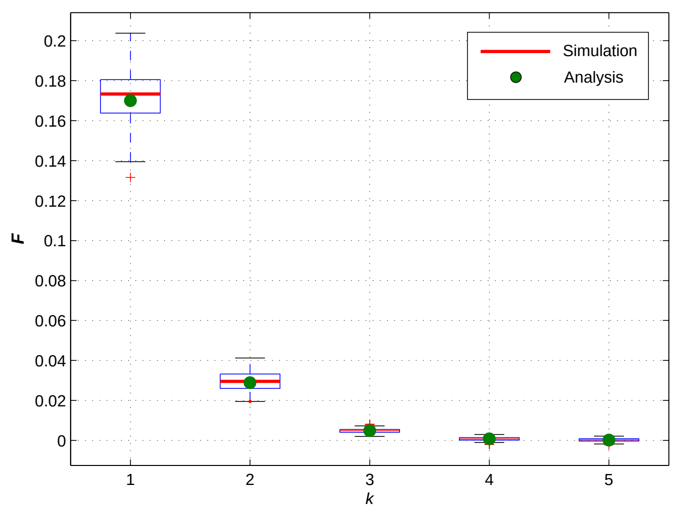


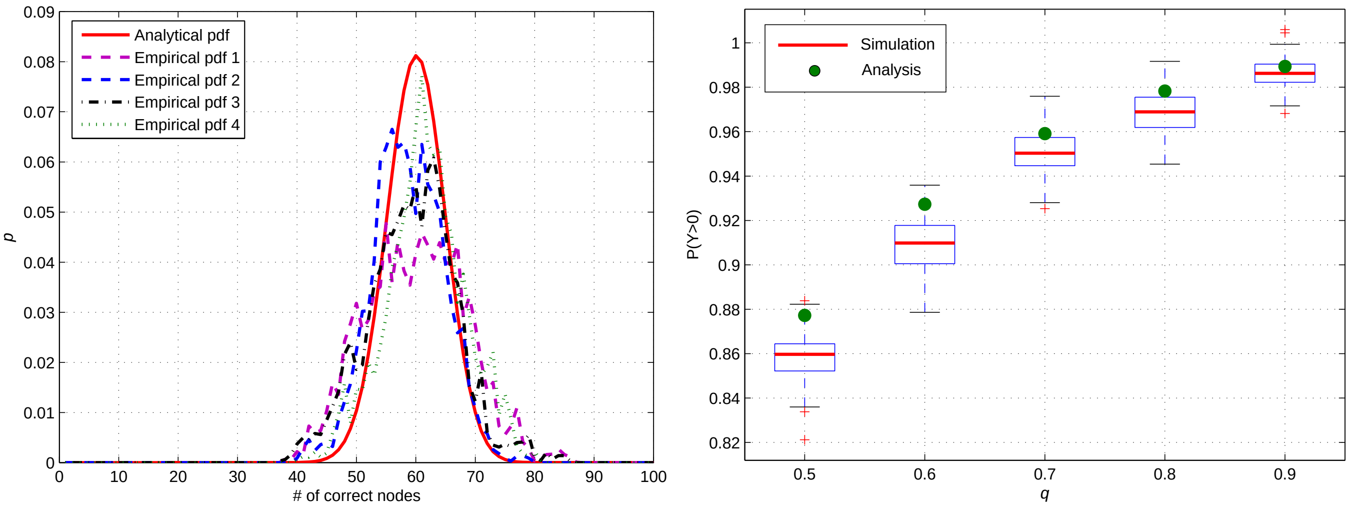
Fig. 6. Analytical and empirical results on $\mathcal{F}_{kt_0,\frac{1}{N}}$ with $N = 100$, $\lambda = 1$, and $\tau = 1.5$.

To summarize our results in the analysis of Sec. V-A and the experiments in this section: we showed that, for any protocol- or application-specific objective $\Delta \geq 1/N$, the confidentiality of the sensed data can be safeguarded with probability almost equal to one. Although this seems to require that a sufficiently high number of measurements (or equivalently long period T) are of interest, analytic and experimental values show that even very short sequences (e.g., $T = 5t_0$) of measurements originating from a single source node can be protected with probability fast approaching one. This is achieved thanks to the GossiCrypt en-route encryption, resulting in particularly robust operation even when approximately 40% of the nodes are compromised by the adversary.

VII. Related Work and Discussions

Among many works on WSN security, including those referenced above, confidentiality has received little attention.

[8] We note that the sink may make a virtual movement by simply changing the target of the key refreshing protocol, but the adversary has to always physically move to a node to launch its attack.



(a) Stationary distributions

(b) Successful probability $\mathrm{P}\{Y > 0\}$

Fig. 4. Comparison between analytical and empirical with $N = 100$, $\lambda = 1$, and $\tau = 1.5$.

Symmetric-key mechanisms for sensor-sink encryption were proposed [35], and homomorphic encryption was devised to protect data aggregation [11], but none considered the compromise of sensor keying material. The vulnerability to eavesdropping was modeled in [4], without a solution for confidentiality but discussion of non-cryptographic approaches. Confidentiality recurs in [3], with the need for protection against node compromise attacks pointed out. Our work, developed in parallel and independently, addresses this challenge posed by [3].

Our scheme is somewhat related to onion routing [18], [15], in the sense that one could view a deterministic version of GossiCrypt ($q = 1$) as the dual of onion routing. In the latter, a successive re-encryption of the message directs it across the network as dedicated routers successively decapsulate the message. In our scheme, layers of encryption are accumulated as the message traverses the network. Of course, onion routing was developed for a context completely different from WSNs, and an objective, to protect the sender/receiver anonymity, also differs from what GossiCrypt is after.

Our overall approach bears some resemblance, beyond mechanisms, to the overall approach of other security schemes, albeit with unrelated objectives. In key establishment, [5] takes a probabilistic approach that leverages on the scale of the network and the assumption of a realistic, not omnipotent, adversary, to propose lightweight solution. From a different point of view, the use of redundancy as the means to enhance security has been used in different ways, for distributed certification authority instantiation [45] or secure communication [32].

Our assumption on unobtrusive adversarial behavior (see Sec. III) stems from storing the operating system in ROM (Sec. II). Re-programmability would enable an adversary to behave obtrusively, and induce arbitrary deviations [26] by planting rogue code. A meaningful deviation could be to manipulate sensor-sink communication paths, so that more compromised nodes become part of those paths, and data and key refreshing messages are re-directed towards the adversary. However, to avoid detection, this deviant behavior should coexist with the routine node actions. But such sophisticated rogue code might not fit in the limited sensor memory space, while the resultant additional traffic may be relatively easy to detect, given the low channel capacity. Finally, one possibility to thwart such an attacker is to require that each node performs a remote software-based attestation (SBA) [38], [39]. Research in this direction is promising yet recent, while the WSN environment is challenging (e.g., network delays); experimental results that verify the effectiveness of SBA would be welcome.

The impact of active adversaries is discussed next. After compromising a key, they can impersonate S_i, and invoke a fake key refreshing.[9] The adversary could then establish a new shared key with the sink, while preventing the reception of messages from the actual S_i. However, once the actual refreshing occurs, S_i operates with a different key from its impostor. The unobtrusive adversary cannot prevent S_i from launching a refresh and cannot upload its own "new" key to S_i. Consequently, Θ will detect the inconsistency at a later interaction with S_i.

An active adversary may launch a DoS attack (e.g., jamming or packet dropping) to prevent key refreshing. Note that the adversary cannot predict the randomly triggered refresh event, and key refresh messages are disguised as data measurements. As a result, the adversary would have to attack randomly or consistently, for all packets. If randomly with low probability, the attack would be ineffective. Otherwise, the sink, knowing both the data report rate and the key refreshing rate, would be able to perform rate-based Bayesian estimations [9] and detect the attack.

As a follow-up work, we intend to consider specific instantiations of WSNs, e.g., network sizes and topologies, data

[9]Public key cryptography (e.g., digital signatures generated by a source node S_i) is not advantageous: the private key of S_i can be compromised as well.

extraction[10] and key refreshing methods, and value ranges for other system characteristics such as δ, T, and Δ, and τ and λ. Extending our work in this way, through analytical and experimental means, would allow us to investigate a number of interesting questions. For example, postulate fine-grained claims conditional on specific networks, revealing design trade-offs due to the relative roles of Δ and T. Or, identify the right "mix" of symmetric- and public-key based key refreshing techniques, as a function of the adversary presence, to evaluate the trade-off of effectiveness for cost.

VIII. Conclusion

As security becomes an important requirement for WSN, the salient characteristics of WSNs clue the more relevant threats and types of exploit to thwart with practical defense mechanisms. With this consideration in mind, we identify here a novel threat, a parasitic adversary, targeting exactly the most valuable asset of a WSN, its measurements. The parasitic adversary is a practical and realistic threat because of (i) its well-aimed exploit, unauthorized access to WSN data, (ii) its well-chosen methods, targeting at the weakest system point, the low physical sensor node protection, and (iii) its resource constraints and "low-profile" operation.

The second and main contribution of this paper is GossiCrypt, a scheme to ensure WSN data confidentiality. GossiCrypt's two building blocks are a probabilistic en route encryption of the data towards the sink and a key refreshing mechanism, both leveraging on the scale of WSNs. The former relies on very simple key management assumptions, it is simple in operation. The latter reverses the impact of the physical compromise of sensor nodes.

Our evaluation shows that GossiCrypt can prevent the breach of WSN confidentiality in a wide range of settings. Even though the adversary could obtain solitary or sparse measurements, our analysis and simulations show that GossiCrypt prevents the compromise of a meaningful set of measurements over a period of time with probability going to one. The most intriguing feature of GossiCrypt lies in its ability of defending the WSN data confidentiality with simple and low-cost mechanisms. We believe that such approaches that leverage on the WSN characteristics, rather than imitating iron-clad approaches from other distributed computing paradigms, can be effective in addressing security challenges for wireless sensor networks.

References

[1] ISO, Information Technology - Security Techniques - Key Management - Part 2: Mechanisms Using Symmetric Techniques. In *ISO/IEC 11770-2, International Standard*, 1996.

[2] ISO, Information Technology - Security Techniques - Key Management - Part 3: Mechanisms Using Asymmetric Techniques. In *ISO/IEC 11770-3, International Standard*, 1999.

[3] M. Anand, E. Cronin, M. Sherr, M. Blaze, Z. Ives, and I. Lee. Sensor Network Security: More Interesting Than You Think. In *Proc. of the 1st USENIX HotSec*, 2006.

[10] For example, in data centric sensor networks where nodes are used as in-network storage [40], we can apply $GossiCrypt_E$ to paths between sources and storages.

[4] M. Anand, Z. Ives, and I. Lee. Quantifying Eavesdropping Vulnerability in Sensor Networks. In *Proc. of the 2nd International VLDB Workshop on Data Management for Sensor Networks (DMSN)*, 2005.

[5] R. Anderson, H. Chan, and A. Perrig. Key infection: Smart trust for smart dust. In *Proc. of the 12th IEEE ICNP*, 2004.

[6] G. Bertoni, L. Breveglieri, and M. Venturi. ECC Hardware Coprocessors for 8-bit Systems and Power Consumption Considerations. In *Proc. of the 3rd IEEE ITNG*, 2006.

[7] T. Bonald. The Erlang Model with Non-Poisson Call Arrivals. *ACM SIGMETRICS Perform. Eval. Rev.*, 34(1), 2006.

[8] S. Brands and D. Chaum. Distance-Bounding Protocols. *Springer LNCS 839*, pages 344–359, 1994.

[9] P. Bremaud. *Point Processes and Queues: Martingale Dynamics*. Springer-Verlag, New York, 1981.

[10] P. Bremaud. *Markov Chains, Gibbs Fields, Monte Carlo Simulation, and Queues*. Springer, New York, 1999.

[11] C. Castelluccia, E. Mykletun, and G. Tsudik. Efficient Aggregation of Encrypted Data in Wireless Sensor Networks . In *Proc. of the 2nd ACM/IEEE MobiQuitous*, 2005.

[12] H. Chan, V. Gligor, A. Perrig, and G. Muralidharan. On the Distribution and Revocation of Cryptographic Keys in Sensor Networks. *IEEE Trans. on Dependable and Secure Computing*, 2(3):233–247, 2005.

[13] Crypto++ library benchmarks, http://gd.tuwien.ac.at/privacy/crypto/libs/cryptlib/benchmarks.html.

[14] A. Demers, D. Greene, C. Hauser, W. Irish, and J. Larson. Epidemic Algorithms for Replicated Database Maintenance. In *Proc. of the 6th ACM PODC*, 1987.

[15] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proc. of the 13th USENIX Security*, 2004.

[16] P. Ehrenfest and T. Ehrenfest. *The Conceptual Foundations of the Statistical Approach in Mechanics*. Dover Publications, New York, reprint edition, 1990.

[17] G. Gaubatz, J.-P. Kaps, and B. Sunar. Public key cryptography in sensor networks – Revisited. In *Proc. of the 1st ESAS*, 2004.

[18] D. Goldschlag, M. Reedy, and P. Syversony. Onion Routing for Anonymous and Private Internet Connections. *Commun. ACM*, 42(2):39–41, 1999.

[19] V. Gupta, M. Wurm, Y. Zhu, M. Millard, S. Fung, N. Gura, H. Eberle, and S.C. Shantz. Sizzle: A Standards-Based End-to-End Security Architecture for the Embedded Internet. *Elsevier Pervasive and Mobile Computing*, 1(4):425–445, 2005.

[20] Z.J. Haas, J.Y. Halpern, and L. Li. Gossip-based Ad Hoc Routing. In *Proc. of the 21st IEEE INFOCOM*, 2002.

[21] P. Hamalainen, T. Alho, M. Hamalainen, and T. Hamalainen. Design and Implementation of Low-area and Low-power AES Encryption Hardware Core. In *Proc. of the 9th EUROMICRO DSD*, 2006.

[22] C. Hartung, J. Balasalle, and R. Han. Node Compromise in Sensor Networks: The Need for Secure Systems. Technical Report CU-CS-990-05, University of Colorado at Boulder, 2005.

[23] A. Kansal, A. Somasundara, D. Jea, M. Srivastava, and D. Estrin. Intelligent fluid infrastructure for embedded networks. In *Proc. of the ACM MobiSys'04*, 2004.

[24] C. Karlof, N. Sastry, and D. Wagner. TinySec: A Link Layer Security Architecture for Wireless Sensor Networks. In *Proc. of the 2nd ACM SenSys*, 2004.

[25] C. Karlof and D. Wagner. Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures. *Elsevier Ad Hoc Networks*, 1(2-3):293–315, 2003.

[26] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Trans. Programming Languages and Systems*, 4(3):382–401, 1982.

[27] J. Luo, J. Panchard, M. Piorkowski, M. Grossglauser, and J.-P. Hubaux. MobiRoute: Routing towards a Mobile Sink for Improving Lifetime in Sensor Networks. In *Proc. of the 2nd IEEE/ACM DCOSS*, 2006.

[28] J.M. McCune, E. Shi, A. Perrig, and M.K. Reiter. Detection of Denial-Of-Message Attacks on Sensor Network Broadcasts. In *Proc. of IEEE Symposium on Security and Privacy*, 2005.

[29] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[30] J. Newsome, E. Shi, D. Song, and A. Perrig. The Sybil Attack in Sensor Networks: Analysis and Defenses. In *Proc. of the 3rd IPSN*, 2004.

[31] R. Ostrovsky and M. Yung. How to Withstand Mobile Virus Attacks. In *Proc. of the 10th ACM PODC*, 1991.

[32] P. Papadimitratos and Z.J. Haas. Secure Data Communication in Mobile Ad Hoc Networks. *IEEE JSAC, Special Issue on Security in Wireless Ad Hoc Networks*, 24(2):343–356, 2006.

[33] B. Parno, A. Perrig, and V. Gligor. Distributed Detection of Node Replication Attacks in Sensor Networks. In *Proc. of IEEE Symposium on Security and Privacy*, 2005.

[34] A. Perrig, J. Stankovic, and D. Wagner. Security in Wireless Sensor Networks. *Commun. ACM*, 47(6):53–57, 2004.

[35] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. Culler. SPINS: Security Protocols for Sensor Networks. *ACM/Kluwer Wireless Networks*, 8(5):521–534, 2002.

[36] K. Piotrowski, P. Langendoerfer, and S. Peter. How Public Key Cryptography Influences Wireless Sensor Node Lifetime. In *Proc. of the 4th ACM SASN*, 2006.

[37] R. Poovendran and L. Lazos. A Graph Theoretic Framework for Preventing the Wormhole Attack in Wireless Ad Hoc Networks. *ACM/Kluwer Wireless Networks*, 13(1):27–59, 2005.

[38] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. Distributed Detection of Node Replication Attacks in Sensor Networks. In *Proc. of IEEE Symposium on Security and Privacy*, 2004.

[39] M. Shaneck, K. Mahadevan, V. Kher, and Y. Kim. Remote Software-based Attestation for Wireless Sensors. In *Proc. of the 2nd ESAS*, 2005.

[40] M. Shao, S. Zhu, W. Zhang, and G. Cao. pDCS: Security and Privacy Support for Data-Centric Sensor Networks. In *Proc. of the the 26th IEEE INFOCOM*, 2007.

[41] Y. Tirta, Z. Li, Y. Lu, and S. Bagchi. Efficient Collection of Sensor Data in Remote Fields Using Mobile Collectors. In *Proc. of the 13th IEEE ICCCN*, 2004.

[42] R. Watro, D. Kong. S. Cuti, C. Gardiner, C. Lynn1, and P. Kruus. TinyPK: Securing Sensor Networks with Public Key Technology. In *Proc. of the 2nd ACM SASN*, 2004.

[43] A. Wood and J. Stankovic. Denial of Service in Sensor Networks. *IEEE Computer*, 35(10):54–62, 2003.

[44] W. Zhang, H. Song, S. Zhu, and G. Cao. Least Privilege and Privilege Deprivation: Towards Tolerating Mobile Sink Compromises in Wireless Sensor Networks. In *Proc. of the 6th ACM MobiHoc*, 2005.

[45] L. Zhou, F. B. Schneider, and R. van Renesse. COCA: A Secure Distributed On-line Certification Authority. *ACM Trans. on Computer Systems*, 20(4):329–368, 2002.