A Randomized Countermeasure Against Parasitic Adversaries in Wireless Sensor Networks

Panagiotis Papadimitratos[2⋆], Jun Luo[1⋆], Jean-Pierre Hubaux[2]

***Abstract*—Due to their limited capabilities, wireless sensor nodes are subject to physical attacks that are hard to defend against. In this paper, we first identify a typical attacker called *parasitic adversary*, who seeks to exploit sensor networks by obtaining measurements in an unauthorized way. As a countermeasure, we first employ a randomized key refreshing: with low communication cost, it aims at confining (but not eliminating) the effects of the adversary. Moreover, our low-complexity solution, GossiCrypt, leverages on the large scale of sensor networks to protect data confidentiality, efficiently and effectively. GossiCrypt applies symmetric key encryption to data at their source nodes and re-encryption at a randomly chosen subset of nodes *en route* to the sink. The combination of randomized key refreshing and GossiCrypt protects data confidentiality with a probability of almost 1; we show this analytically and with simulations. In addition, the energy consumption of GossiCrypt is lower than a public-key based solution by several orders of magnitude.**

***Index Terms*—Confidentiality, Security, Probabilistic Key Refreshing and En-route Encryption, GossiCrypt**

I. Introduction

Although wireless sensor networks (WSNs) have been actively investigated for a quite long time, their wider adoption still faces certain obstacles, among which the security issues become increasingly prominent [19]. Significant efforts have been made to address these issues (see [15] and the references therein). However, the issue of ensuring only authorized access to the sensor-collected data are not well addressed, especially considering that sensor nodes can be physically compromised. This is the main focus of our paper.

A straightforward way to prevent unauthorized access is end-to-end data encryption, with a unique symmetric key shared between the sink and every node. However, as making a sensor node tamper-resistant is too costly, it is fairly easy for an adversary to actively exploit the poor physical protection of nodes, for example, to physically access the node memory and extract the symmetric key used for data encryption. Such an attack is vastly simpler than a cryptanalytic one against the key, and the adversary's ability to progressively compromise different nodes makes the situation even worse.

In this paper, 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 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 parasitic relation of the adversary with a fully functioning WSN remains unnoticed, the more successful the parasitic adversary will be.

One immediate solution against (symmetric) key compromise is to let sensors encrypt each outgoing measurement with the public key of the sink, a solution we term throughout the paper as the Public Key Encryption (PKE) scheme. Unfortunately, public-key operations (both software and hardware implementations), albeit computationally feasible, consume energy approximately **three orders of magnitude** higher than symmetric key encryption [20]. An alternative countermeasure is key refreshing. However, as the WSN operator can hardly infer which keys were compromised, a costly network-wide key refreshing protocol would have to be invoked frequently to eliminate the effects of the adversary.

Consequently, the challenge lies in protecting data confidentiality against parasitic adversaries in an energy-efficient manner. To address this, we first propose a *randomized key refreshing* (**RKR**) mechanism: it infrequently installs new sensor-sink shared symmetric keys to randomly selected nodes. As a complement, we then propose **GossiCrypt** to leverage on the multitude of a WSNs. GossiCrypt employs a probabilistic *en route re-encryption* scheme; the source node always encrypts the data and the relaying nodes *en route* to the sink "flip a coin" to "decide" whether to perform re-encryption. This combination, **RKR-GossiCrypt**, provides a strong and efficient defence against a parasitic adversary: It is highly improbable that the adversary would be able to decrypt the data, as it would need to compromised all the keys used at the source and for en-route encryptions.

Our main contribution is a simple, low-cost, yet effective scheme to ensure sensor data confidentiality. The effectiveness and the efficiency of our scheme are shown analytically and through experimental validation. We find that RKR-GossiCrypt, which relies primarily on symmetric key operations and infrequent public key operations, achieves confidentiality with a probability almost 1. Our analytical and experimental validation also shows that RKR-GossiCrypt achieves very significantly (two to three orders of magnitude) lower network overhead and hence lower delays than the public-key encryption of the sensor readings. Another contribution we make is the introduction of the parasitic adversary, which is realistic for a wide range of commercial and tactical WSNs. To the best of our knowledge, this type of adversary is novel, yet realistic and highly effective, unless thwarted.

[1]Jun Luo is with the School of computer Engineering, Nanyang Technological University, Singapore.

[2]Panagiotis Papadimitratos and Jean-Pierre Hubaux are with School of Computer and Communication Sciences, EPFL (Swiss Federal Institute of Technology in Lausanne), CH-1015, Lausanne, Switzerland.

⋆Panagiotis Papadimitratos and Jun Luo are equally contributing authors.

In this 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 conclude in Sec. VII. Due to space limitations, we omit the literature survey and discussion sections, available in the accompanying technical report [15] and the conference paper [14].

II. System Model

Each *sensor node* has a unique identifier S_i, and the *network sink* is denoted as Θ. Each node S_i shares a symmetric key, $K_{i,\Theta}$, with the sink, and knows the sink public key, PuK_Θ. The data of interest are described 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 $j = 1, \ldots, \lceil \delta N \rceil$.

In general, $T = kt_0$, with $k > 0$ an integer and t_0 for a single sensor measurement, and δ will be a significant fraction of N.

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 node placement sparser than the monitored physical system and relatively long history of measurements 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, due to cost considerations, WSN nodes are neither tamper-resistant nor do they store cryptographic keys in tamper-resistant components.

III. Adversary Model

The *parasitic* adversary consists of two entities: a *pEavesdropper* that positions itself at a strategic location to overhear data traffic, and a *pCompromiser* that physically compromises nodes and provides the obtained keys to the pEavesdropper.

More specifically, a parasitic adversary:

1. Seeks to obtain the WSN data collected according to the parameters δ and T, in an unauthorized manner;
2. Can be physically present, at each point in time, **only** 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. Is mobile and can compromise different nodes over time, with a rate of 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 it wants. The adversary exploits obvious weaknesses of WSNs (assumption 3): poor physical protection makes it relatively easy to obtain data encryption keys [9]. Furthermore, it can utilize its resources intelligently. Assumption 4 shows that the adversary (the pCompromiser in particular) can be in the proximity of different nodes for periods of time during which it compromises the node and retrieves the symmetric key. We illustrate the parasitic relation of the adversary with a WSN in Fig. 1.

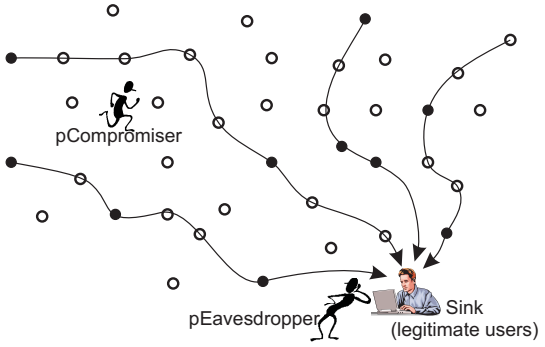

Fig. 1. The parasitic relation of the adversary with a WSN. Nodes whose keys have been compromised are highlighted in black.

The parasitic adversary is *unobtrusive*, i.e., it cannot modify the implemented protocols stored in ROM (Sec. II). The adversary could remain within range of the compromised node and trivially intercept all its transmissions. But such an attack would be self-defeating: Assumption 2 implies that the adversary would then capture much less than $\lceil \delta N \rceil$ measurements. In other words, assumption 2 embodies the hardness to deploy a network of eavesdroppers: The overall cost for leaving pEavesdropper nodes behind such that the adversary always remain physically attached to a significant fraction of the sensor nodes would be comparable to deploying a WSN by the adversary.

We assume that the protocol design and implementation are such that remote node compromise is prevented. We also assume that the sink *cannot* be compromised by the adversary. Moreover, denial-of-service (DoS) attacks, including jamming in various protocol layers [21], Sybil/Node replication attacks [18], or "wormhole" formation [17] are beyond the scope of this work: countermeasures to those attacks can coexist with our protocols. Nor do we consider physical destruction of WSN nodes, which would not benefit the adversary.

IV. RKR-GossiCrypt

We describe briefly our solution here, referring the reader to [15], [14] for a precise description of the protocol. We first describe our *random key refreshing* (RKR) mechanism, which thwarts the parasitic adversary in a light-weight fashion. However, as RKR may be unable to fully mitigate the key compromise, we propose GossiCrypt to strengthen *confidentiality*, i.e., to prevent any unauthorized access to the data collected by the WSN. More formally, we do not seek to protect every piece of data coming from every single sensor. Rather, our goal is the following (Δ, T)-Confidentiality 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 to the system and application (as well as the adversary), is prevented. In Sec. V-A we will show that RKR-GossiCrypt satisfies this property against parasitic adversaries with probability almost one under realistic scenarios.

We emphasize that we do **not** seek to provide sensor data authenticity and integrity; our scheme can complement any other scheme that provides end-to-end (e.g., sensor-to-sink) or hop-by-hop (e.g., sensor-to-sensor) or other security mechanism [15], [14]. But we do not mandate authenticity and integrity because the adversary that compromises the key(s) of a node can then impersonate the node and inject fabricated messages on its behalf. Any such action does not benefit the parasitic adversary that seeks to breach data confidentiality.

A. Random Key Refreshing (RKR)

To defend against the progressive compromising of WSN nodes, the $\{K_{i,\Theta}\}$ keys are refreshed, i.e., they are replaced with new $\{K'_{i,\Theta}\}$ keys. The sink is typically unaware of which nodes are already compromised. At the same time, the adversary is also unaware of which keys are being refreshed: As it will be explained shortly, the key refreshment is randomly performed and initiated by any sensor node, and the adversary can be certain that a key is refreshed only if it (re)compromises (physically accesses) the node.

Due to the space constraints, we omit the detailed descriptions of the key refreshing protocols and refer the interested readers to [15], [14]. Basically, the protocols (both symmetric-key and public-key based) are variants of the ISO/IEC standards [1], [2]. However, there are two major differences:

1. The key refreshing protocol is initiated by individual sensor nodes and it is **not** interactive. Basically, a node generates a new key and it transports it to the sink. If an interactive key establishment protocol were used, the pEavesdropper could eavesdrop all messages sent from the sink, and hence gain a significant advantage: it would get to know which nodes were refreshed and would then promptly re-compromise them.
2. The key refreshing protocol message is **encapsulated** as the payload of the data encryption protocol, presented in Sec. IV-B. This hides the relatively infrequent key refreshing traffic among data traffic, hence it prevents the adversary from detecting the key refreshing events by intercepting the traffic.

More precisely, a node uses a random point process generator [5], $RGen(\lambda)$, to generate key refreshing events with intensity λ. Upon each event, the node generates a new key $K'_{i,\Theta}$ and includes it in a newly created message, along with a *flag* to indicate to the sink that a new key, rather than a piece of data, is in the message payload. The whole message, after encryption, externally appears identical to any measurement/data reporting message.

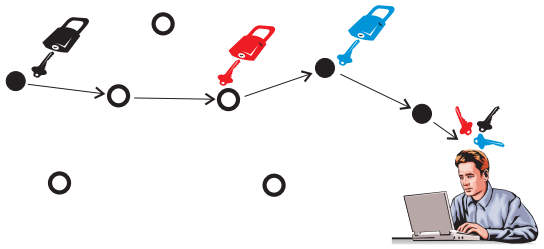

Fig. 2. Securing data collection with GossiCrypt. Even though the data source and another two nodes en-route already have their keys compromised, the encryption applied by one correct node is sufficient to thwart the adversary.

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 can be selected accordingly by the sink, and conveyed to all nodes via a (broadcast) authenticated control message (possibly, long after the deployment if also needed). Confidentiality of λ is not needed, as the adversary would, at best, compromise nodes at its maximum possible rate τ^{-1}.

To prevent an otherwise easy disruption and/or even Denial of Service (DoS), RKR messages include a symmetric authenticator using $K_{i,\Theta}$. This is not to ensure authentication in the presence of the parasitic adversary (which can compromise node key(s) and hence forge RKR messages), but to allow key refreshing only with knowledge of a prior key. A parasitic adversary is capable of launching a DoS attack, but this is not the adversary's objective. More important, the adversary would not gain anything by preventing data decryption at the sink (e.g., by a forged RKR message): S_i would later refresh its key to some $K''_{i,\Theta}$ (at the next random event) and then encrypt its data with $K''_{i,\Theta}$ (which the adversary would obtain only if it re-compromised S_i).

Our scheme always operates with the latest key established by RKR and it does not revert under any circumstances (e.g., inability to decrypt messages or other disruption) to an older $K_{i,\Theta}$. A disruptive adversary could at most, after physically compromising a node, cause a transient DoS for that node. But this could signal the attack to the WSN user, contrary to the objectives of a parasitic adversary.

B. GossiCrypt

We distinguish sensors as *data sources* and *relaying nodes*, each assuming either role at different times. As illustrated in Fig. 2, GossiCrypt is executed by nodes on the path from a source to a sink, with the source, S_i, always encrypting its data with $K_{i,\Theta}$ and each relaying node, S_j, randomly (with probability $\boldsymbol{q}$) re-encrypting the data with $K_{j,\Theta}$.

The sink can correctly decrypt a message if it knows the sequence of encryptions applied to it. In its basic form [15], [14], GossiCrypt appends the identifiers of the re-encrypting nodes to a message. However, the increasing message size with the number or re-encryptions might offset the benefit of

GossiCrypt in large networks. Here, we propose to compress the identifier string in case of long paths, to reduce the GossiCrypt overhead at the expense of (i) a mild increase of processing at the sink (or the server of the WSN owner) that is not resource-limited, and (ii) a very low probability of having messages across long paths discarded.

Initially, the identifiers of the re-encrypting nodes are appended. If the message size grows beyond a certain **threshold**, compression is invoked at a re-encrypting node. As shown in Sec. V-B, the advantage of GossiCrypt over PKE-ECC disappears after the tenth re-encryption due to the cumulated identifiers. Therefore, the threshold can be set such that the compression takes place before the tenth re-encryption. The compression is based on a *Bloom filter* [4]. Assume that an m-bit string is consumed by the identifiers when the compression is invoked. This identifier string is then replaced by an 8-bit counter C and an empty Bloom filter of $m-8$ bits. The identifiers are added to the filter following the standard k-hash mapping,[1] and the cardinality of the identifier set is stored in the counter. Here the k-hash mapping takes $s_i \| S_i$ (where s_i is the sequence number of S_i in the encryption sequence), instead of S_i, as an input. From then on, any re-encrypting node will add its identifier into the filter and update C.

The sink queries all possible values deduced from $s_i \| S_i$ (where $s_i \leq C$ and S_i can be any node identifier) to recover the encryption sequence. The length of the Bloom filter (hence the compression threshold) should be large enough to guarantee a negligible false positive probability. If we assume a 16-bit node identifier and a threshold that invokes the compression at the tenth re-encryption, we get a 152-bit Bloom filter, whose false positive probability is less than 0.77% if 15 nodes conduct re-encryption. The sink may also utilize the network topology information to speed up the query (limiting the scope of S_i) and also to discern false positive cases.

V. Protocol Analysis

A. Security Analysis

We first describe a model for the joint effect of key compromise and refreshing. Then we evaluate the performance of RKR-GossiCrypt against the (Δ, T)-Confidentiality property (Sec. IV). The model and analysis presented here are new and enhanced compared to those in [15], [14], capturing more precisely the adversarial behavior by considering location (in addition to the refinement of the adversary in Sec. II). Our analysis, accompanied by simulation results in Sec. VI, shows that even with a significant fraction of sensor nodes compromised, RKR-GossiCrypt safeguards confidentiality **with probability almost one**.

Fundamental for the analysis is the fraction of *correct*, i.e., not compromised, nodes. Our eventual goal is to model the *state of the system*, the number of correct nodes, as a stochastic process. The security analysis on RKR-GossiCrypt is then based on the stationary regime of this process. To achieve this, we need to identify three components: *system size*, *key refreshing process*, and *key compromising process*.

[1]There is no need for k different hash functions; we may apply hashing once and slice the output into k sections.

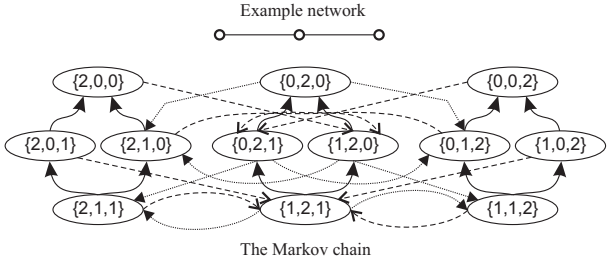

Fig. 3. A 3-node line network and its corresponding Markov chain. A link between two nodes indicates a neighborhood relation. The solid, dashed, dotted arrows represent different transitions.

The *system size* depends on the behavior of the adversary. In general, we assume that the sink and the adversary share the same interest. This implies that, as we briefly mentioned in Sec. IV-A, the system is a known subset of nodes with size $N' \leq N$, which includes the data source nodes of interest (the δ fraction of N as defined in Sec. II), and all the intermediate nodes between the sources and the sink. We hereafter slightly abuse the terminology by viewing N as the system size.

Given the behavior of the key refreshing protocol (in particular the random point process generator $RGen(\lambda)$), the *key refreshing process* can be modeled as a Poisson processes with intensity λ. The *key compromising process*, on the contrary, cannot be modeled as a homogeneous random process (in terms of the system state), as the transition rate is location dependent: the adversary can only compromise nodes that are geographically close. Therefore, we have to first model the system in a microscopic way by looking at the *state of nodes*, and then extrapolate a macroscopic model of the system state.

We define the state space of nodes as $\mathcal{E} = \{0, 1, 2\}$, where 0 means *correct*, 1 means *compromised*, and 2 means *infectious*. To clarify the terminology, a compromised node with an adversarial node (pCompromiser) in its vicinity is labeled **infectious** in the sense that the "infection", i.e., the compromise of possibly another node in the network, appears to continue spreading from the latest-in-time location of the pCompromiser (which coincides with that of the infectious node).

Under this microscopic model, the system state is represented by an N-dimensional vector in $\mathcal{E}^N$, with only one element taking the value of 2. Note that the transition from one state to another is potentially affected by the location of the infectious node (or the pCompromiser). In other words, a node with state 0 or 1 may transit to 2 only if it has a neighbor in state 2, and such a transition takes place with intensity τ^{-1}. Given the aforementioned, it is easy to see that a Markov chain can be used to describe transitions among (microscopic) system states. Fig. 3 illustrates such a chain for a simple network. The adversary co-located with an infectious node k will compromise every neighboring node at an identical rate $\frac{\tau^{-1}}{N_k}$ (where N_k is the *neighborhood size* of k), even if some of them might have already been compromised. This is so because the adversary is unable to tell which keys were refreshed by the sink. Directly computing the stationary distribution of this microscopic chain is not easy. Fortunately,

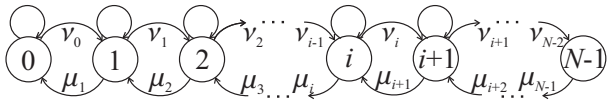

Fig. 4. Aggregated Markov chain representing the system state (the number of correct nodes). Note that the state $N-1$ has no self-transition.

what we are really interested in is the macroscopic system state: the number of correct nodes. The macroscopic chain can be obtained by aggregating the states of the microscopic chain: the i-th macro-state is the aggregation of all the micro-states with i 0s. Using the aggregation, we can now describe the system states as a continuous Markov chain $\{X(t)\}_{t\geq 0}$ driven by Poisson processes, as shown in Fig. 4. For example, we have $N = 3$ (the number of nodes in the network) after aggregating the chain shown in Fig. 3. We compute the aggregated transition rates ν_i and μ_i as:

$$\nu_i = \frac{(N-1-i)\lambda}{N} \quad (1)$$

$$\mu_i = \sum_{k=1}^{N} \left[\frac{\sum_{j=\max(1,N_k+i+1-N)}^{\min(N_k,i)} j \binom{N_k}{j} \binom{N-1-N_k}{i-j}}{\tau \binom{N-1}{i} \sum_{k=1}^{N} N_k} \right] \quad (2)$$

The chain $\{X(t)\}_{t\geq 0}$ is an instance of the birth-death process with a finite number of states. It has the following stationary distribution (the detailed computation is omitted here; we refer to [15]):

$$\pi_0 = \left\{ 1 + \frac{\nu_0}{\mu_1} + \frac{\nu_0\nu_1}{\mu_1\mu_2} + \cdots + \frac{\nu_0\nu_1\cdots\nu_{N-1}}{\mu_1\mu_2\cdots\mu_N} \right\}^{-1} \quad (3)$$

$$\pi_i = \pi_0 \frac{\nu_0\nu_1\cdots\nu_{i-1}}{\mu_1\mu_2\cdots\mu_i} \quad (4)$$

The stationary distribution has the following properties:

- The system can rarely be free either of correct nodes ($X(t) = 0$) or of compromised nodes ($X(t) = N-1$), because both π_0 and π_{N-1} vanish with increasing N.
- The most likely state (i.e., $\arg\max_i \pi_i$) 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 $N-1$.

These properties are reflected on Fig. 5: Even if the system is more efficient in refreshing keys than the adversary is in compromising them ($\lambda\tau = 1.5$), there are still approximately 40% compromised nodes.

Next, 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 $Y = \sum_{m=1}^{M} \Omega_m$, $M \leq L$, 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 $\mathrm{P}\{Y = 0\} = g_Y(0)$ and,

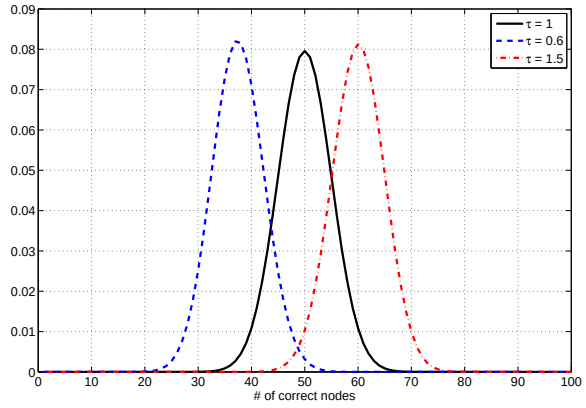

Fig. 5. Stationary distribution π with $N = 100$, $\lambda = 1$, and $\tau = 0.6, 1, 1.5$. The y-axis is the probability density corresponding to a certain number of correct nodes. Since only the product $\lambda\tau$ matters, we choose the values of λ and τ arbitrarily without a dimension.

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).

by the rule of *random sum of i.i.d. variables* [5], $g_Y(z) = g_M(g_\Omega(z))$. Therefore,

$$\begin{aligned} \mathrm{P}\{Y=0\} &= g_M(g_\Omega(0)) = \mathrm{E}_M[\mathrm{P}\{\Omega_0 = 0\}^m] \\ &= \sum_{m=1}^{L} \mathrm{P}\{\Omega_0 = 0\}^m \binom{L}{m} q^m (1-q)^{L-m} \\ &= \sum_{m=1}^{L} \left(\frac{N - \mathrm{E}_\pi(X)}{N} \right)^m \binom{L}{m} q^m (1-q)^{L-m} \end{aligned} \quad (5)$$

given the stationary distribution π of $\{X(t)\}_{t\geq 0}$. 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 $\tau = 1.5$. We might think the case where $\mathrm{P}\{Y > 0\} = 0.8258$ (for $L = 5$ and $q = 0.5$) is unfavorable (because the adversary can decrypt the data with probability 0.1742). However, the data the adversary might decrypt (with probability 0.1742) is just one measurement of a given node; the probability of observing a meaningful data history goes to zero (e.g., the probability of obtaining three measurements by the same node is already very low: $0.1742^3 = 0.0053$). We assume events of decrypting two measurements by a node are independent, based on the randomized operation of GossiCrypt even if data are transmitted across the same routing path.

We have 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 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 in 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, this depends on $\mathrm{P}(L)$, which in turn depends on a complex manner on the relative placement of the sink and source nodes, but also on the pattern of the adversary compromising nodes. As a result, we proceed without making an assumption on $P(L)$, rather we reason on whether our scheme satisfies the (Δ, T)-Confidentiality property with an asymptotic argument.

Claim: *RKR-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* $T \gg t_0$.

The proof is given in [15], [14] and it is omitted here. As shown in Fig. 5, it is always preferable to have $\lambda\tau > 1$ (although $\lambda\tau < 1$ can be compensated by aggressively setting q (i.e., a high value)). Maintaining a relatively high refresh rate is not difficult: Whereas the adversary compromises keys via its physical presence, keys are refreshed through a simple protocol invoked by each node separately. 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 that the convergence of $\mathcal{P}$ persists even if $\lambda\tau < 1$ but with a lower speed.

B. Energy Expenditure

As we mention in Sec. I, PKE is an alternative solution to thwart a parasitic adversary. We show here that PKE is inferior to RKR-GossiCrypt because of its much higher energy expenditure. For a quantitative comparison between PKE and RKR-GossiCrypt, we make the following assumptions:

1. Each node identifier consumes 16 bits, and for GossiCrypt, the identifier compression starts at the tenth re-encryption.
2. Each message (sensor reading) has a length of 20 bytes.[2]
3. GossiCrypt makes use of AES-128 encryption.
4. The PKE can either be RSA-1024 or ECC-160.[3]
5. The energy expenditure for transmission is 0.21 $\mu J/\mathrm{bit}$.

Most parameters refer to MICA2 nodes, 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 [20]. Although hardware implementations could significantly reduce energy consumption for all primitives [8], [3], [7], the order of difference is maintained. Table II compares GossiCrypt with two variants of PKE in terms of computation[4] and communication complexity.

	GossiCrypt	PKE-RSA	PKE-ECC
Comp.	$32.4\mu J/\mathrm{msg}$	14.1 mJ/msg	53.4 mJ/msg
Comm.	160 bits + increase of $16q$ bits 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 the computation of GossiCrypt at a source node is 2 to 3 orders of magnitude lower than those of PKEs. Second, the energy expenditure in the communication of GossiCrypt for each node en-route is again lower than those of PKEs. Considering the basic GossiCrypt [14], or its counterpart with the compressed identifiers (Sec. IV-B): for 16 bit node identifiers, the advantage in terms of transmission is valid for path lengths of up to $10q^{-1}$ (for PKE-ECC) and $54q^{-1}$ (for PKE-RSA) hops (note that $q < 1$). If the compression is used, the advantage of GossiCrypt remains for a longer path length, whose specific value depends on the tolerable false positive probability and also on the sink's ability to further reduce this probability using topology information. The additional computation cost for RKR-GossiCrypt compared with PKE stems from key refreshing and en-route re-encryption; we denote the former by c_{refresh}. Based on the analysis in Sec. V-A, let us assume a 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, each node will be refreshed on the average once every N measurement periods. Given an average number C (Sec. IV-B) of re-encryptions for each message, the advantage for RKR-GossiCrypt during this period is approximately the ratio of $\frac{(N+1)\times C\times c_{\mathrm{GC}}+c_{\mathrm{refresh}}}{N\times c_{\mathrm{PKE}}} \approx \frac{N+1}{N}\frac{C\times c_{\mathrm{GC}}}{c_{\mathrm{PKE}}}$ with symmetric-key based key refreshing (as $c_{\mathrm{GC}} \approx c_{\mathrm{refresh}}$) or $\approx \frac{1}{N}\frac{c_{\mathrm{refresh}}}{c_{\mathrm{PKE}}}$ with public-key based key-refreshing (if $\frac{C\times c_{\mathrm{GC}}}{c_{\mathrm{PKE}}} \ll 1$), where c_{GC} and c_{PKE} are the computation costs for GossiCrypt and PKEs, respectively, given in Table II.

As the advantage of RKR-GossiCrypt over PKEs is **tremendous** without public-key encryption, we consider here RKR only 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 RKR-GossiCrypt increasingly advantageous. For example, if $N = 100$, RKR-GossiCrypt can be **100 times** less costly than PKE-ECC. For PKE-RSA, $c_{\mathrm{refresh}} \approx 3c_{\mathrm{PKE}}$ and RKR-GossiCrypt is still 33 times less costly. However, the very high communication cost of PKE-RSA is a significant disadvantage that renders that scheme impractical.

The comparison above might seem unfair, as it could be argued that using PKE on a per-message basis is not

[2]The size is designed to transport the new keys. For sensor readings that are of a smaller size, padding is used to fill up the message. As shown in our analysis, RKR-GossiCrypt is still superior to PKE in terms of energy consumption, even with the padding overhead.

[3]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) [16]. However, we are not aware of sensor network software implementations for Rabin PKE. Moreover, Rabin appears to be costlier than RSA in certain implementations in other platforms [6].

[4]The computational complexity is measured in different units for symmetric-key and public-key encryption in [20], thus we fix the message size in order to compare them.

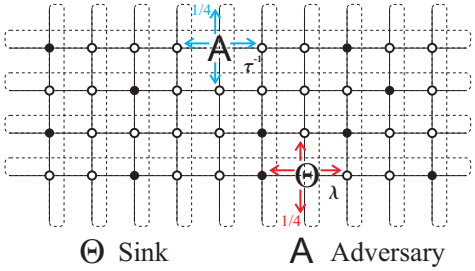

Fig. 6. Simulation: We approximate the key refreshing process (Sec. IV-A) as a 2D random walk in this regular grid. For illustration purposes, Θ, the symbol used for the sink, is used to indicate this random walk. We also approximate the node compromise by the adversary, shown above by A, as a 2D random walk. Black and white dots illustrate compromised and correct nodes respectively.

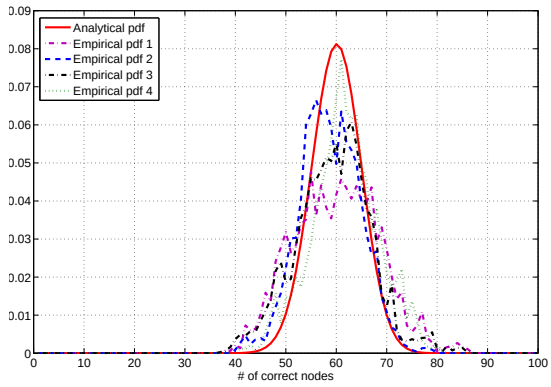

Fig. 7. Stationary distribution of the number of correct nodes.

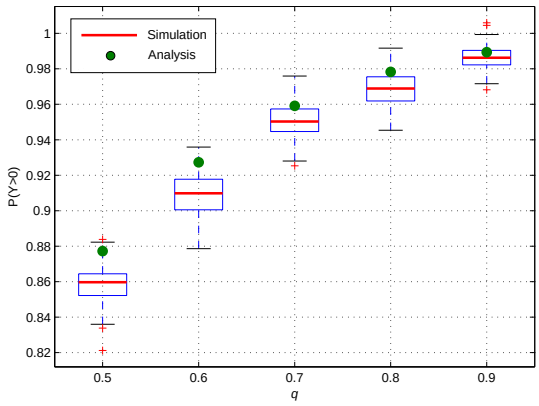

Fig. 8. Success probability $\mathrm{P}\{Y > 0\}$ as function of the GossiCrypt parameter q.

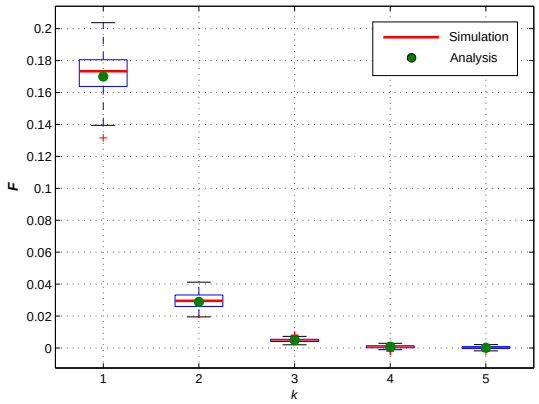

Fig. 9. The probability of breaching the confidentiality of k measurements from a given node $\mathcal{F}_{kt_0,\frac{1}{N}}$ as function of k.

necessary; for example, PKE could be used only to "transport" a symmetric key from each source sensor node to the sink. Then, such end-to-end symmetric keys could be used for data encryption. However, as we emphasized in Sec. I, in order for such conventional key refreshing to reach the security level achieved by RKR-GossiCrypt, conventional key refreshing has to be performed frequently for almost all nodes. Given our assumption that the adversary compromises one node per measurement period t_0, without GossiCrypt all N (symmetric) keys would have to be refreshed every t_0. As $c_{\text{refresh}} \geq c_{\text{PKE}}$ in general, it would be more efficient to just use PKE on a per-message basis.

VI. Experiment Results

We implement RKR-GossiCrypt and the benchmark PKE scheme in TinyOS [10] and we perform simulation experiments with TOSSIM [12] and in Matlab. The detailed TinyOS/TOSSIM implementations and experiments are concerned with performance aspects, and are presented in Sec. VI-B. The effectiveness of RKR-GossiCrypt, in terms of data confidentiality, is evaluated through the simpler Matlab simulations in Sec. VI-A. [5]

[5]Considering implementation aspects, including the MAC/PHY layers, would make simulations cumbersome for large networks, and they would not add to the validation of the analytical derivation.

A. Protocol Effectiveness

We assume a grid network where nodes appear on a $\sqrt{N} \times \sqrt{N}$ square lattice. The movements of the adversary follow a 2D random walk: it takes an identical probability $1/4$ in choosing one direction out of four possibilities. The intervals between two successive events follow exponential distributions with mean λ^{-1} and τ respectively. We assume $N = 100$, $\lambda = 1$, and $\tau = 1.5$. To remove the boundary effect, we project the lattice on a torus: moving out of the one side of the lattice leads to entering on the opposite side. We illustrate these settings in Fig. 6.

As the stochastic process described above is aperiodic and positive recurrent, all its states are ergodic [5]. 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 (warm-up period, so that results are measured in a steady state). Fig. 7 shows the comparison among four empirical stationary distributions resulting from four simulation runs and the analytical one obtained in Sec. V-A.

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 brevity, we only illustrate the case with $L = 6$ in Fig. 8 (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 over-optimistic, but the differences with the experiment results are negligible.

Finally, we verify our claim that RKR-GossiCrypt guarantees the (Δ, T)-Confidentiality property with probability almost one when $T = kt_0$ is sufficiently long. To this end, we randomly choose two nodes on the grid and consider 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 values of k. As shown in Fig. 9, this probability converges very fast to zero with an increasing k, according to both simulation and analytical results. This corroborates our claim that $\mathcal{P} = 1 - \mathcal{F}_{T,\Delta} \to 1$.

To summarize our analysis (Sec. V-A) and experimental results in this section: We show that, for any protocol- or application-specific objective $\Delta \geq 1/N$, the confidentiality of the sensed data can be safeguarded with a 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 (as shown by Fig. 7).

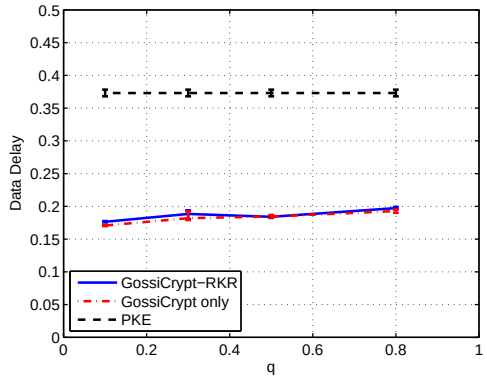

Fig. 10. Data (sensor reading) delay, $\mathcal{D}$, in seconds, as a function of q.

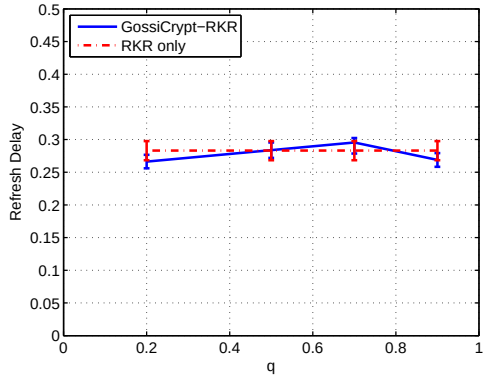

Fig. 11. Key refreshing delay, $\mathcal{R}$, in seconds, as a function of q.

B. Protocol Efficiency

We implement RKR-GossiCrypt in TinyOS 1.x and nesC, using the corresponding libraries for data collection, applications, sensor-PC communication, etc. We implemented the protocol with all the cryptographic functionality, leveraging on existing cryptographic libraries, TinySec [11] and TinyECC [13], for symmetric and public key operations respectively. The implementation was integrated in the TOSSIM simulation framework, which also provided simple data collection applications. We utilize CrossBow Mica2 motes and the corresponding models in TOSSIM [12]. In the simulations, we consider three relatively simple settings: (i) a fork topology of 13 nodes, with three source sensor nodes at the ends of three branches converging into a single path, and the sink at the end of this, (ii) a line topology of 10 nodes with a source sensor node and the sink at the two ends of the line, and (iii) randomly generated topologies of 30 nodes, with a randomly placed sink and all nodes reporting readings, as well as variants of the experiments for settings (i) and (ii) with all nodes sending readings to the sink. Here, due to space limitations, we present the results from setting (i).

We measure (I) the data (sensor readings) acquisition delay in seconds, $\mathcal{D}$, that is, the period from the data reading transmission until the successful reception at the sink, (II) the key refreshing delay in seconds, $\mathcal{R}$, that is, the period from the transmission of the key refreshing message until its successfully reception at the sink, and (III) the total network (transmission) overhead, $\mathcal{T}$, measured as the total number of bytes sent over the wireless network by all nodes, divided by the total number of useful bytes delivered at the sink.

The simulated time is 2000 seconds, sensor readings are obtained every 6 seconds, the average key refreshing period takes values from 60, 300, 900 seconds, and the q parameter takes values from 0.1, 0.3, 0.5, 0.8. The cryptographic parameters are as those in Sec. V-B: 80-bit symmetric key for SKIPJACK, and 160-bit elliptic curve encryption both for the key refreshing messages and the benchmark PKE. For simplicity, only results with a key refreshing period of 60 seconds are shown.

Fig. 10 shows that $\mathcal{D}$, the sensor reading delay, for the combined GossiCrypt and RKR is consistently less than half of that of PKE. The reason is clearly the combination of the higher transmission delays over the same path for the public key encrypted readings. For GossiCrypt-RKR, the effect of the infrequent refreshing operations (and the corresponding transmissions) on the data acquisition is amortized. Even with the one key refreshing per minute, more than 90% of the transmissions of a source node are symmetric-key encrypted (Recall: one reading per 6 seconds).

We do not factor into $\mathcal{D}$ and $\mathcal{R}$ the public key encryption

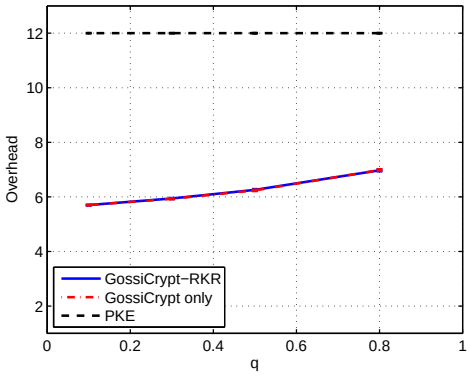

Fig. 12. Total network (transmission) overhead, $\mathcal{T}$, as a function of q.

processing delay, for three reasons. First, TOSSIM allows for the fine-grained timing of network and radio operations, but it does not model code execution time. [6] Moreover, public key processing delays dwarf the network delays, thus yielding results even farther in favor of the combined GossiCrypt+RKR. In addition, the periods of data readings and key refreshing operations are significantly higher than the delay for a public key encryption, allowing key refreshing operations for RKR to be processed when the node performs no other task.

In Fig. 10, the increase of q causes a mild increase in $\mathcal{D}$, but this is less than 11% from $q = 0.1$ to $q = 0.8$, thanks to the efficient symmetric key operations and the smaller size packets (compared to PKE). For comparison, Fig. 11 shows the delay for a key refreshing $\mathcal{R}$, and the larger size packets result in $\mathcal{R}$ on the average up to 45% higher than the data reading delay. But, again, refreshing operations are relatively infrequent. Moreover, we confirm that the en-route re-encryption of key refreshing messages, as expected, does not affect delays (Recall: key refresh messages are not distinguished and are handled as data by intermediate nodes).

Finally, Fig. 12 shows the total network overhead $\mathcal{T}$, accounting for all cryptographic and other headers sent over the wireless medium, considering typical sensor readings of five 5 bytes (e.g., the Surge application provided in TOSSIM). GossiCrypt+RKR result in a total overhead that is lower than 41% to 53% of that of PKE. Note that the PKE overhead (and delay earlier in Fig. 10) is not dependent on q. Moreover, note that the GossiCrypt implemented here is the basic one, as in [14]. This means that the overhead increasing mildly with the increase of q and path length. Thus, the shown performance is an upper bound of the overhead, which does not increase for high path lengths.

VII. Conclusion

As security becomes an important requirement for WSNs, 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, the 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 RKR-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 RKR-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 RKR-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 RKR-GossiCrypt lies in its ability to ensure 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.

[6] Cryptographic processing delays are investigated in detail in the literature, as discussed in Sec. V-B. Our implementation serves the purpose of a proof of concept, and we do not attempt to replicate those investigations here.

Acknowledgements

The authors would like to thank Sibel Demirkol for her help with the implementation of GossiCrypt.

This work was funded in parts by the NCCR/MICS.

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] 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.

[4] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

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

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

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

[8] 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.

[9] 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.

[10] http://www.tinyos.net/.

[11] 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.

[12] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *Proc. of the 1st ACM SenSys*, 2003.

[13] A. Liu and P. Ning. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In *Proc. of the 7th IPSN*, 2008.

[14] J. Luo, P. Papadimitratos, and J.-P. Hubaux. GossiCrypt: Wireless Sensor Network Data Confidentiality Against Parasitic Adversaries. In *Proc. of the 5th IEEE SECON*, 2008.

[15] J. Luo, P. Papadimitratos, and J.-P. Hubaux. GossiCrypt: Wireless Sensor Network Data Confidentiality Against Parasitic Adversaries. Technical Report LCA-REPORT-2007-002, EPFL, 2010.

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

[17] P. Papadimitratos, M. Poturalski, P. Schaller, P. Lafourcade, D. Basin, S. Capkun, and J.-P. Hubaux. Secure Neighborhood Discovery: A Fundamental Element for Mobile Ad Hoc Networking. *IEEE Communications Magazine*, 46(2):132–139, 2008.

[18] 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.

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

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

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