Efficient and Provably Secure Aggregation of Encrypted Data in Wireless Sensor Networks

20

CLAUDE CASTELLUCCIA
INRIA
ALDAR C-F. CHAN
National University of Singapore
EINAR MYKLETUN
QUEST Software
and
GENE TSUDIK
University of California, Irvine

Wireless sensor networks (WSNs) are composed of tiny devices with limited computation and battery capacities. For such resource-constrained devices, data transmission is a very energy-consuming operation. To maximize WSN lifetime, it is essential to minimize the number of bits sent and received by each device. One natural approach is to aggregate sensor data along the path from sensors to the sink. Aggregation is especially challenging if end-to-end privacy between sensors and the sink (or aggregate integrity) is required. In this article, we propose a simple and provably secure encryption scheme that allows efficient additive aggregation of encrypted data. Only one modular addition is necessary for ciphertext aggregation. The security of the scheme is based on the indistinguishability property of a pseudorandom function (PRF), a standard cryptographic primitive. We show that aggregation based on this scheme can be used to efficiently compute statistical values, such as mean, variance, and standard deviation of sensed data, while achieving significant bandwidth savings. To protect the integrity of the aggregated data, we construct an end-to-end

Some preliminary results were originally published in Castellucia et al. [2005]. The present article is a reworked and extended version. Major new components include the security analysis and the technique for authentication of encrypted aggregated data.
C. Castelluccia's work was performed in part while visiting the University of California, Irvine. A. C-F. Chan's work was performed in part while at INRIA. E. Mykletun's work was perfomed while at the University of California, Irvine. A. C-F. Chan would like to thank the Ministry of Education, Singapore, for providing financial support through research grant R-252-000-331-112.
Authors' addresses: C. Castelluccia, INRIA, Zirst-avenue de l'Europe, 38334 Saint Ismier Cedex, France; email: claude.castelluccia@inria.fr; A. C-F. Chan, Department of Computer Science, School of Computing, National University of Singapore, Singapore; email: aldar@comp.nus.edu.sg; E. Mykletun, Quest Software, Aliso Viejo, CA; email: einar.mykletun@quest.com; G. Tsudik, Computer Department, University of California, Irvine; email: gts@ics.uci.edu.

DOI 10.1145/1525856.1525858 http://doi.acm.org/10.1145/1525856.1525858

aggregate authentication scheme that is secure against outsider-only attacks, also based on the indistinguishability property of PRFs.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*

General Terms: Design, Security

Additional Key Words and Phrases: Authentication, privacy, pseudorandom functions, secure data aggregation, stream ciphers, wireless sensor networks, cryptography

ACM Reference Format:

Castelluccia, C., Chan, A. C-F., Mykletun, E., and Tsudik, G. 2009. Efficient and provably secure aggregation of encrypted data in wireless sensor networks, ACM Trans. Sensor Netw., 5, 3, Article 20 (May 2009), 36 pages. DOI = 10.1145/1525856.1525858
http://doi.acm.org/1525856.1525858

1. INTRODUCTION

Wireless sensor networks (WSNs) are becoming increasingly popular in many spheres of life. Application domains include monitoring of the environment (e.g. temperature, humidity, and seismic activity) as well as numerous other ecological, law enforcement, and military settings.

Regardless of the application, most WSNs have two notable properties in common: (1) the network's overall goal is typically to reach a collective conclusion regarding the outside environment, which requires detection and coordination at the sensor level, and (2) WSNs act under severe technological constraints: individual sensors have severely limited computation, communication and power (battery) resources while operating in settings with great spatial and temporal variability.

At the same time, WSNs are often deployed in public or otherwise untrusted and even hostile environments, which prompts a number of security issues. These include the usual topics, for example, key management, privacy, access control, authentication, and denial of service (DoS)-resistance. What exacerbates and distinguishes security issues in WSNs is the need to miniaturize all security services in order to minimize security-induced overhead. In other words, if security is a necessary hindrance in other (e.g., wired or MANET) types of networks, it is much more so in WSNs. For example, public key cryptography is typically avoided[1] as are relatively heavyweight conventional encryption methods.

WSN security is a popular research topic and many advances have been made and reported on in recent years. Most prior work focused on ultra-efficient key management, authentication, routing, and DoS resistance [Eschenauer and Gligor 2000; Zhu et al. 2004; Karlof and Wagner 2003; Wood and Stankovic 2002]. An overview of security-related issues and services in the WSN context can be found in Perrig et al. [2004].

Also, much attention has been devoted to sensor communication efficiency. Since data transmission is very energy-consuming, in order to maximize sensor

[1]While some sensor devices have sufficient computation power to perform public key operations, transmitting the resulting large ciphertexts is expensive; the smallest public key ciphertext is around 160 bits.

lifetime, it is essential to minimize the sheer number of bits sent by each sensor device. One natural approach involves aggregating sensor data as it propagates along the path from the sensors to the so-called sink—a special node that collects sensed data. Of course, aggregating data is not quite equivalent to collecting individual sensor readings. In some applications, for example, perimeter control, aggregation is not applicable, since only individual sensor readings are of interest. However, many WSN scenarios that monitor an entire microenvironment (e.g., temperature or seismic activity) do not require information from individual sensors; instead, they put more emphasis on statistical quantities, such as mean, median and variance.

End-to-end privacy and aggregate integrity/authenticity are the two major security goals in a secure WSN. Regardless of information leakage due to the correlation among sensor measurements, end-to-end privacy ensures that nobody other than the sink can learn the final aggregate, even if it controls a subset of sensor nodes. Informally speaking, aggregate authentication assures the sink that the aggregate value is a function of authentic components (individual sensor inputs) and the data has not been tampered with en route.

Although simple and well-understood, aggregation becomes problematic if end-to-end privacy between sensors and the sink is required. If we assume that all sensors are trusted, they could encrypt data on a hop-by-hop basis. For an intermediate sensor (one that receives and forwards data), this would entail: (1) sharing a key with each neighboring sensor, (2) decrypting encrypted messages received from each child, (3) aggregating all received values, and (4) encrypting the result for transmission to its parent. Though viable, this approach is fairly expensive since each value has to be decrypted before aggregation. It also complicates key management since each node must share a key with each of its neighbors. Furthermore, hop-by-hop encryption assumes that all sensors are trusted with the authenticity and privacy of other sensors' data. This assumption may be altogether unrealistic in some settings, whereas, in others, trust can be partial, for example, intermediate nodes are trusted with only authenticity or only privacy.

Alternatively, if a single global key was used by all sensors, by subverting a single sensor node, the adversary could learn measured values of any and all nodes in the network. Since only the sink should gain an overview of WSN measurements, this approach is not attractive. Nevertheless, we do not rule out using a single global key for message authentication of the aggregate, which is another challenging security goal in WSNs. In fact, aggregate authentication against outsider-only attacks might be the best one can achieve for end-to-end integrity in the WSN scenario. In other words, additive aggregate authentication secure against malicious insiders might not be achievable without using heuristic tools, such as outlier detection or range checking [Buttyán et al. 2006; Yang et al. 2006]. These techniques have to be based on the stronger assumption that the statistical distribution of measurements is known (partially or completely) beforehand; such techniques are essentially data/aggregate plausibility checks [Wagner 2004]. If an attacker can inject a contribution (an arbitrary value) into an additive aggregate through compromised insiders, it can actually offset—without detection—the final aggregate by any desired amount.

1.1 Contributions

In this article, we focus on efficient, bandwidth-conserving privacy in WSNs. We blend inexpensive encryption techniques with simple aggregation methods to achieve very efficient aggregation of encrypted data. To assess the practicality of the proposed techniques, we evaluate them and present encouraging results which clearly demonstrate appreciable bandwidth conservation and small overhead stemming from both encryption and aggregation operations. We also provide a security argument for the proposed encryption scheme. We prove that the proposed scheme achieves semantic security if encryption keys are generated by a good pseudorandom function family. We also extend the proposed scheme to provide end-to-end aggregate authentication, which is provably secure against outsider-only attacks.

1.2 Organization

In the next section we discuss some background and the assumptions about our system model. Section 3 describes the problem statement along with the security model, Section 4 describes our homomorphic encryption scheme, and Section 5 describes how to utilize this encryption scheme in a WSN. We give a security proof of the proposed scheme in Section 6. Performance is analyzed and results are discussed in Section 7. The aggregate authentication scheme and its security analysis are given in Section 8. Related work is summarized in Section 9, and Section 10 concludes this article.

2. BACKGROUND

In this section we describe the key features of, and assumptions about, the network and provide an overview of aggregation techniques.

2.1 Wireless Sensor Networks (WSNs)

A WSN is a multi-hop network composed of a multitude of tiny autonomous devices with limited computation, communication, and battery facilities. One commonly cited WSN application is monitoring the environment. This may include sensing motion, measuring temperature, humidity, and so on. Data monitored by the sensors is sent to a sink (usually a more powerful device), that is responsible for collecting the information.

The multi-hop nature of a WSN implies that sensors are also used in the network infrastructure; not just sending their own data and receiving direct instructions, but also forwarding data for other sensors. When sensors are deployed, a delivery tree is often built from the sink to all sensors. Packets sent by a sensor are forwarded to the sink by the sensors along the delivery tree.

Although sensor nodes come in various shapes and forms, they are generally assumed to be resource-limited with respect to computation power, storage, memory and, especially, battery life. A popular example is the Berkeley mote [Madden et al. 2002]. One common sensor feature is the disproportionately high cost of transmitting information, as compared to performing local computation. For example, a Berkeley mote spends approximately the same amount of energy to compute 800 instructions as it does in sending a single bit of data [Madden

et al. 2002]. It thus becomes essential to reduce the number of bits forwarded by intermediate nodes, in order to extend the entire network's lifetime. The sink node acts as a bridge between the WSN and the outside world. It is typically a relatively powerful device, such as a laptop computer.

2.2 Aggregation in WSN

Aggregation techniques are used to reduce the amount of data communicated within a WSN, thus conserving battery power. Periodically, as measurements are recorded by individual sensors, they need to be collected and processed to produce data representative of the entire WSN, such as average and/or variance of the temperature or humidity within an area. One natural approach is for sensors to send their values to certain special nodes: aggregating nodes. Each aggregating node then condenses the data prior to sending it on. In terms of bandwidth and energy consumption, aggregation is beneficial as long as the aggregation process is not too CPU-intensive.

The aggregating nodes can either be special, more powerful nodes, or regular sensor nodes. In this article, we assume that all nodes are potential aggregating nodes and that data gets aggregated as they propagate towards the sink. In this setting, since sensors have very limited capabilities, aggregation must be simple and should not involve any expensive and/or complex computations. Ideally, it should require only a few simple arithmetic operations, such as addition or multiplication.[2]

We note that aggregation requires all sensors to send their data to the sink within the same sampling period. This either requires the sensors to have—at least loosely—synchronized clocks or the ability to respond to explicit queries issued by the sink.

One natural and common way to aggregate data is to simply add up values as they are forwarded towards the sink. This type of aggregation is useful when the sink is only interested in certain statistical measurements, for example, the mean or variance of all measured data. As noted in Section 1, some WSN applications require all sensor data and, therefore cannot benefit from aggregation techniques. Similarly, applications requiring boundary values, for example, min and/or max, are obviously not a good match for additive aggregation.

With additive aggregation, each sensor sums all values, x_i, it receives from its k children (in the sink-rooted spanning tree) and forwards the sum to its parent. Eventually, the sink obtains the sum of all values sent by all n sensors. By dividing the sum by n, the total number of sensors, it computes the average of all measured data.

This simple aggregation is very efficient since each aggregating node only performs k arithmetic additions.[3] It is also robust, since there is no requirement for all sensors to participate, as long as the sink gets the total number of sensors that actually provided a measurement.

[2]This is indeed what we achieve in this work.

[3]We assume that an aggregating node has its own measurement to contribute; thus k additions are needed.

Additive aggregation can be also used to compute the variance, standard deviation and any other moments on the measured data. For example, in case of variance, each aggregating node not only computes the sum, $S = \sum_{i=1}^{k} x_i$, of the individual values sent by its k children, but also the sum of their squares: $V = \sum_{i=1}^{k} x_i^2$. Eventually, the sink obtains two values: the sum of the actual samples, which it can use to compute the mean and the sum of the squares, which it can use to compute the variance:

$$Var = E(x^2) - E(x)^2; \text{ where}$$
$$E(x^2) = \left(\sum_{i=1}^{n} x_i^2\right)/n \text{ and } E(x) = \left(\sum_{i=1}^{n} x_i\right)/n.$$

3. GOALS AND SECURITY MODEL

To provide data privacy, our goal is to prevent an attacker from gaining any information about sensor data, aside from what it can infer by direct measurements. We define our privacy goal by the standard notion of semantic security [Goldwasser and Micali 1984].

The attacker is assumed to be global, that is, able to monitor any area (even the entire coverage) of the WSN. Furthermore, we assume the attacker is able to read the internal state of some sensors. The attacker is also assumed capable of corrupting a subset of sensors. However, we assume that it can only perform chosen-plaintext attacks. That is, the attacker can obtain the ciphertext of any plaintext it chooses. In a real-world situation, this means that the attacker can manipulate the sensing environment and obtain the desired ciphertext by eavesdropping.

In light of our requirement for end-to-end privacy between sensors and the sink, additive aggregation, although otherwise simple, becomes problematic. This is because popular block and stream ciphers, such as AES [NIST 2001] or RC5 [Rivest 1995], are not additively homomorphic. In other words, operating on encrypted values does not allow for the retrieval of the sum of the plaintext values.

To minimize trust assumptions, we assume that each of the n sensors shares a distinct long-term encryption key with the sink. This key is originally derived, using a pseudo-random function (PRF), from the master secret known only to the sink. We denote this master secret as K and the long-term sensor-sink shared key as ek_i, where the subscript $0 < i \leq n$ uniquely identifies a particular sensor. This way, the sink only needs to store a single master secret and all long-term keys can be recomputed as needed.

As opposed to encryption, authentication schemes that allow for aggregation seem to be very difficult, and perhaps impossible, to design. It should be noted that the problem of aggregate authentication considered in this article is different from the problem considered in aggregate signatures [Boneh et al. 2003].[4] In aggregate authentication, the messages themselves are aggregated and hence, the original individual messages are not available for verification.

[4]More precisely, the latter should be called *aggregatable* signatures.

Whereas, in aggregate signatures, the signatures on different messages are aggregated and all individual signed messages must be available to the verification algorithm in order to validate the aggregate signature. Consequently, we observe that there does not exist a secure end-to-end aggregate authentication scheme (which provides existential unforgeability against chosen-message attacks). As described in Wagner [2004], other external techniques are needed to verify the plausibility of the resulting aggregate and to increase the aggregation resiliency.

Generally, providing end-to-end aggregate authentication in a WSN is difficult since messages lose entropy through aggregation, making it hard to verify an aggregate. However, it is still possible to prevent unauthorized (external) nodes from injecting fake data into the aggregate. That is, group-wise aggregate authentication can be achieved, wherein only nodes that possess a common group key can contribute to an aggregate and produce valid authentication tags that would subsequently be verified by the sink. Of course, such a scheme would be vulnerable to compromised or malicious insiders. Later in the article, we construct an example of an end-to-end aggregate authentication scheme secure against outsider-only attacks.

4. ADDITIVELY AGGREGATE ENCRYPTION

Encrypted data aggregation or aggregate encryption is sometimes called concealed data aggregation (CDA), a term coined by Westhoff et al. [2006]. (Appendix A gives an abstract description of CDA showing the desired functionalities.)

In this section we describe the notion of *homomorphic encryption* and provide an example. Our notion is a generalized version of the one widely used for homomorphic encryption—we allow the homomorphism to be under different keys, while the homomorphism in common notions is usually under the same key. We then proceed to present our additively homomorphic encryption scheme, whose security analysis is given in Section 6 and Appendix B. The encryption technique is very well-suited for privacy-preserving additive aggregation.

For the sake of clarity, in Section 4.2, we will first describe a basic scheme assuming the encryption keys are randomly picked in each session (which is the same scheme as given in our earlier work [Castelluccia et al. 2005]); the header part is also excluded in the discussion. Then we will give a concrete construction in which the session keys and the encryption keys are derived using a pseudorandom function family. The concrete construction can be proved to be semantically secure in the CDA model [Chan and Castelluccia 2007], the details of which are given in Appendix A. Compared to our earlier work [Castelluccia et al. 2005], this article provides the details of a concrete construction using a pseudorandom function in Section 4.3, with the security requirements on the used components specified.

Our scheme can be considered as a practical, tailored modification of the Vernam cipher [Vernam 1926], the well-known one-time pad, to allow plaintext addition to be done in the ciphertext domain. Basically, there are two modifications. First, the exclusive-OR operation is replaced by an addition operation. By

choosing a proper modulus, multiplicative aggregation is also possible.[5] Second, instead of uniformly picking a key at random from the key space, the key is generated by a certain deterministic algorithm (with an unknown seed) such as a pseudorandom function [Goldreich et al. 1986]; this modification is actually the same as that in a stream cipher. As a result, the information-theoretic security (which requires the key be at least as long as the plaintext) in the Vernam cipher is replaced with a security guarantee in the computational-complexity theoretic setting in our construction.

4.1 Homomorphic Encryption

A homomorphic encryption scheme allows arithmetic operations on ciphertexts. One example is a multiplicatively homomorphic scheme, where the decryption of the efficient manipulation of two ciphertexts yields the multiplication of the two corresponding plaintexts. Homomorphic encryption schemes are especially useful whenever some party not having the decryption key(s) needs to perform arithmetic operations on a set of ciphertexts. A more formal description of homomorphic encryptions schemes is as follows.

Let $Enc()$ denote a probabilistic encryption scheme and let M and C be its plaintext and ciphertext spaces, respectively. If M is a group under operation $\oplus$, we say that $Enc()$ is a $\oplus$-homomorphic encryption scheme, if, for any instance $Enc()$ of the encryption scheme, given $c_1 = Enc_{k_1}(m_1)$ and $c_2 = Enc_{k_2}(m_2)$ for some $m_1, m_2 \in M$, there exists an efficient algorithm that can generate—from c_1 and c_2—a valid ciphertext $c_3 \in C$ for some key k_3 such that:

$$c_3 = Enc_{k_3}(m_1 \oplus m_2).$$

In other words, decrypting c_3 with k_3 yields $m_1 \oplus m_2$. In this article, we mainly consider additive homomorphisms: $\oplus$ is the $+$ operation. We do not require k_1, k_2, k_3, to be the same, although they need to be equal in most homomorphic encryption schemes. Since k_3 can be distinct from k_1 and k_2, some identifying information, (denoted as hdr) needs to be attached to the aggregated ciphertext to identify the keys required for decryption.

One good example is the RSA cryptosystem [Rivest et al. 1978], which is *multiplicatively homomorphic* under a single key. The RSA encryption function is $Enc(m) = m^e = c \pmod{n}$ and the corresponding decryption function is $Dec(c) = c^d = m \pmod{n}$, where n is a product of two suitably large primes (p and q), e and d are encryption and decryption exponents, respectively, such that $e * d = 1 \pmod{(p-1)(q-1)}$. Given two RSA ciphertexts, c_1 and c_2, corresponding to respective plaintexts, m_1 and m_2, it is easy to see that $c_3 = c_1 c_2 \equiv m_1^e m_2^e \equiv (m_1 m_2)^e \pmod{n}$. Hence, it is easy to obtain a ciphertext, c_3, corresponding to $m_3 = m_1 m_2 \pmod{n}$. Note that, since c_1, c_2, and c_3 are all encrypted using the same encryption key (e, n), no hdr is needed.

[5]Our construction can achieve either additive or multiplicative aggregation but not both at the same time. Besides, multiplication aggregation seems to have no advantage as the size of a multiplicative aggregate is the same as the sum of the size of its inputs.

4.2 Basic Encryption Scheme using Random Keys

We now introduce a simple *additively homomorphic* encryption technique. The main idea is to replace the *xor* (exclusive-OR) operation typically found in stream ciphers with modular addition.[6] The basic scheme is as follows.

Basic Additively Homomorphic Encryption Scheme

Encryption:

(1) Represent message m an integer $m \in [0, M-1]$ where M is the modulus.
(2) Let k be randomly generated keystream, where $k \in [0, M-1]$.
(3) Compute $c = Enc_k(m) = m + k \bmod M$.

Decryption:

(1) $Dec_k(c) = c - k \bmod M$.

Addition of Ciphertexts:

(1) Let $c_1 = Enc_{k_1}(m_1)$ and $c_2 = Enc_{k_2}(m_2)$.
(2) Aggregated ciphertext: $c_l = c_1 + c_2 \bmod M = Enc_k(m_1 + m_2)$ where $k = k_1 + k_2 \bmod M$.

The correctness of aggregation is assured if M is sufficiently large. The reason is as follows: $c_1 = m_1 + k_1 \bmod M$ and $c_2 = m_2 + k_2 \bmod M$, then $c_l = c_1 + c_2 \bmod M = (m_1 + m_2) + (k_1 + k_2) \bmod M = Enc_{k_1+k_2}(m_1 + m_2)$. For $k = k_1 + k_2$, $Dec_k(c_l) = c_l - k \bmod M = (m_1 + m_2) + (k_1 + k_2) - (k_1 + k_2) \bmod M = m_1 + m_2 \bmod M$.

We assume that $0 \leq m < M$. Note that, if n different ciphers c_i are added, M must be larger than $\sum_{i=1}^{n} m_i$. Otherwise, correctness does not hold. In fact, if $\sum_{i=1}^{n} m_i > M$, decryption produces $m' < M$. In practice, if $t = max_i\{m_i\}$, M must be chosen as $M = 2^{\lceil log_2(t*n) \rceil}$.

Note that this basic scheme is provided for illustration purposes only and does not represent the actual construction. Since the encryption key, k, is assumed to be randomly chosen by each sensor node in every reporting session, a secure channel has to be maintained at all times between each sensor node and the sink. In the actual construction (in Section 4.3), such a secure channel is not required.

4.3 A Scheme with PRF-Generated Keys

The main difference between the actual construction and the basic encryption scheme is that encryption keys in each session are now generated by a pseudorandom function (PRF) instead of being truly random. Two components are used in the construction: a PRF f and a length-matching hash function h. Their details are as follows.

4.3.1 *Pseudorandom Functions (PRFs).* For an in-depth treatment of PRFs [Goldreich et al. 1986], we refer to Goldreich [2001]. In our context, a PRF is

[6]For clarity's sake, the discussion of hdr and pseudorandom functions is deferred to Section 4.3.

needed to derive encryption keys. Let $F = \{F_\lambda\}_{\lambda \in \mathbb{N}}$ be a PRF family where $F_\lambda = \{f_s : \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda\}_{s \in \{0,1\}^\lambda}$ is a collection of functions indexed by key $s \in \{0,1\}^\lambda$. Informally, given a function f_s, from a PRF family with an unknown key s, any PPT distinguishing procedure allowed to get the values of $f_s(\cdot)$ at (polynomially many) arguments of its choice should be unable to distinguish (with non-negligible advantage in λ) whether the answer of a new query is supplied by f_s or randomly chosen from $\{0,1\}^\lambda$.

Most provably secure PRFs such as Naor et al. [2002] are based on the hardness of certain number-theoretic problems. However, such constructions are usually computationally expensive, especially, for sensors. Instead, key derivation in practice is often based on functions with conjectured or assumed pseudorandomness: it is inherently assumed in the construction rather than proven to follow from the hardness of a computational problem. One common example is the use of cryptographic hash functions for key derivation such as Perrig et al. [2001]. Some well-known primitives, such as HMAC [Bellare et al. 1996] and OMAC [Iwata and Kurosawa 2003] (conjectured PRFs), are based on assumed pseudorandomness. (HMAC assumes that the underlying compression function of the hash function in use is a PRF, while OMAC assumes the underlying block cipher is a pseudorandom permutation.)

The additive aggregate encryption scheme in this article does not impose a restriction on the type of underlying PRFs. The security guarantee provided by the proposed construction holds as long as the underlying PRF has the property of pseudorandomness or indistinguishability. We note that the aforementioned pseudorandomness property is also a basic requirement for the hash function used for key derivation purposes [Perrig et al. 2001; Bellare et al. 1996], for example, in the well-known IPSec standard.

4.3.2 *Length-Matching Hash Function.* The length-matching hash function $h : \{0,1\}^\lambda \rightarrow \{0,1\}^l$, matches the length of the output of the PRF f, to the modulus size of M, where $|M| = l$ bits. The purpose of h is to shorten a long bit-string, rather than to produce a fingerprint of a message. Hence, unlike cryptographic hash functions, h is not required to be collision-resistant. The only requirement on h is that: $\{t \leftarrow \{0,1\}^\lambda : h(t)\}$ must be uniformly distributed over $\{0,1\}^l$. By uniformly picking an input from the domain of h, the output is uniformly distributed over the range of h.

This requirement is pretty loose and many compression maps from $\{0,1\}^\lambda$ to $\{0,1\}^l$ satisfy it. For instance, h can be implemented by truncating the output of a PRF and taking l least significant bits as output. The sufficiency of this requirement is based on the assumption that an ideal PRF is used. For such a function, without knowledge of the seed key, it is unpredictable whether an output bit is 0 or 1, for all inputs. In practice, key derivation is usually based on conjectured PRFs with unproven pseudorandomness. For example, a collision-resistant hash function is commonly used for deriving secret keys from a seed [Perrig et al. 2001; Bellare et al. 1996]. Hence, it might be that, for some inputs to these conjectured PRFs, there is a higher chance (greater than $\frac{1}{2}$) of correctly predicting some output bits. To tolerate the imperfect nature of conjectured PRFs, if $l < \lambda$, a better construction could be as follows: truncate the output of

the PRF into smaller strings of length l, and then take exclusive-OR on all these strings and use it as the output of h. The output of h should look random to any computationally bounded procedures if at least one of the l-bit strings looks sufficiently random. This can be explained by a hybrid argument as follows.

Without loss of generality, assume $\lambda = 2l$. Consider the worst-case scenario wherein the first l output bits of the PRF are fixed for all inputs (even if the key is kept secret); that is, there is zero entropy for the first l bits of the PRF output. Denote the output of the PRF by $x_0||x_1$, where both x_0 and x_1 are l bits long. Suppose x_0 is the bad l bits but x_1 remains indistinguishable from an l-bit random string, y. That is, the distribution of x_1 (denoted by $\{x_1\}$) is indistinguishable from a uniform distribution U_l over $\{0, 1\}^l$; denote this indistinguishability relation by $\{x_1\} \stackrel{c}{\equiv} U_l$. The output of h would be $x_0 \oplus x_1$ (denoted by z). The fact $\{x_1\} \stackrel{c}{\equiv} U_l$ implies the following two distributions are indistinguishable: $\{x_0 \oplus x_1\}$, $\{x_0 \oplus y\}$, where $y \leftarrow U_l$. The former distribution is $\{z\}$ (the output distribution of h). Note that (since x_0 is fixed) the latter distribution $\{x_0 \oplus y\}$ is actually the uniform distribution U_l itself. Hence, $\{z\} \stackrel{c}{\equiv} U_l$. In other words, even though the first half of the PRF output is completely predictable (with zero entropy), the output of h would still look random provided the second half of the PRF output looks random.

Assume there is a sink and n nodes in the system. In the following description, f is a PRF for key stream generation and h is a length-matching hash function. The details of the proposed aggregate encryption scheme are as follows.

Additively Homomorphic Encryption Scheme using a PRF Family

Assume the modulus is M.

Key Generation:

(1) Randomly pick $K \in \{0, 1\}^\lambda$ and set it as the decryption key for the sink.
(2) For each $i \in [1, n]$, set encryption key for node i as $ek_i = f_K(i)$.

Encryption:

(1) Given encryption key ek_i, plaintext data m_i and nonce r, output $c_i = Enc_{ek_i}(m_i) = m_i + h(f_{ek_i}(r)) \bmod M$.
(2) Set header $hdr_i = \{i\}$.
(3) Output (hdr_i, c_i) as ciphertext.

Decryption:

(1) Given ciphertext (hdr, c) and nonce r used in encryption, generate $ek_i = f_K(i), \forall i \in hdr$.
(2) Compute $x = Dec_K(c) = (c - \sum_{i \in hdr} h(f_{ek_i}(r))) \bmod M$ (where $K = \sum_{i \in hdr} h(f_{ek_i}(r))$), and output plaintext aggregate x.

Addition of Ciphertexts:

(1) Given two CDA ciphertexts (hdr_i, c_i) and (hdr_j, c_j), compute $c_l = (c_i + c_j) \bmod M$
(2) Set $hdr_l = hdr_i \cup hdr_j$.
(3) Output (hdr_l, c_l).

The keystream for a node is now generated from its secret key ek_i, and a unique message ID, or nonce r. No randomness in the nonce is needed. This secret key is precomputed and shared between each node and the sink, while the nonce can be either included in the query from the sink or derived from the current (and unique) time period identifier.

In the proposed scheme, a PRF is used: (1) by the sink to generate the encryption keys ek_i's from the root key K, and (2) by sensor node i to generate the key stream $f_{ek_i}(r)$ from ek_i and the nonce r. It is not necessary to use two different PRF schemes for the instantiations. The same PRF scheme can be used for these purposes, and security analysis in Section 6 and Appendix B shows that the proposed scheme is semantically secure (Appendix A) as long as the used PRF is secure.

For the sake of clarity, hdr is used to represent the set of IDs of all reporting nodes in our discussion. Nevertheless, there is no restriction on how the actual headers (which could be different from hdr) should be constructed in implementation. The only criteria is that the sink can determine the set of reporting nodes (hdr) unambiguously from the received header. For instance, the actual header may contain the set of nonreporting nodes (with each node reporting the IDs of its children that are not responding), the sink (assumed to have knowledge of all the node IDs) can determine hdr simply from the actual header by taking complement. When the nonreporting nodes only form a small percentage of all the nodes in the network, this header scheme could considerably reduce the overhead. In fact, this scheme is used in the calculations in Section 7.

5. AGGREGATION OF ENCRYPTED DATA

As previously noted, efficient aggregation in WSNs becomes challenging when end-to-end privacy of data is required. One solution is to disregard aggregation altogether in favor of privacy, for each sensor to encrypt and forward upstream its individual measurement. The sink, upon receiving as many packets as there are responding sensors, proceeds to decrypt all ciphertexts and sum them in order to compute the desired statistical measurements. We denote this approach as *No-Agg*. A variant of this scheme consists of having the intermediate nodes concatenate the packets they receive from their children into a smaller number of packets in order to avoid the overhead due to the headers. We denote this variant as *CON*. These two approaches have two obvious disadvantages. First, because all packets are forwarded towards the sink, much bandwidth (and, hence, power) is consumed. Second, as illustrated in Section 7.2, there is an extreme imbalance among sensors in terms of the amount of data communicated. Sensors closer to the sink send and receive up to several orders of magnitude more bits than those on the periphery of the spanning tree. The CON scheme performs better than the *No-Agg* scheme but it remains quite costly.

A second approach, that does not achieve end-to-end privacy but does aggregate data, is the *hop-by-hop (HBH)* encryption method, which is also used for comparison between aggregation methods in Girao et al. [2004]. In HBH all nodes create pair-wise keys with their parents and children at bootstrapping phase. When answering a query, a node decrypts any packets received

from downstream, aggregates the plaintext data with its own, encrypts the aggregated result and forwards to its parent. This approach is obviously more bandwidth-efficient than No-Agg since no packet is sent twice. However, there is certain cost involved with the decryption and encryption performed at every non-leaf node. This increases energy consumption; see [Girao et al. 2004]. More importantly, from a privacy perspective, HBH leaves nodes vulnerable to attacks since aggregated data appears in plaintext in each non-leaf node. In particular, nodes close to the sink become attractive attack targets since the aggregated values they handle represent large portions of the overall data in the WSN.

We propose an end-to-end privacy preserving aggregation approach (denoted as *AGG*) where each sensor encrypts its data using the encryption scheme presented in Section 4.3. Since this scheme is additively homomorphic, values can be added (aggregated) as they are forwarded towards the sink. The sink can easily retrieve—from the aggregates it receives—the sum of the samples, and derive statistical information. AGG retains the attractive properties of both the No-Agg (end-to-end privacy) and HBH (energy efficient) schemes.

5.1 Robustness

An important feature of the proposed scheme is the requirement for the sink to know the encrypting sensors' IDs so that it can regenerate the correct keystream for decryption purposes.

Since communication in WSNs is not always reliable and node failures are possible, we do not assume that all sensors reply to all requests. Therefore, a mechanism is needed for communicating to the sink the IDs of non-responding sensors (or their complements). The simplest approach, and the one used in our evaluation, is for each sensor to append the IDs of its non-responding children to each message.[7]

5.2 Computing Statistical Data

In this section, we show how the proposed additively homomorphic encryption scheme aggregates encrypted data, while allowing the sink to compute the average and moments. Since multiple moduli can be used in different instances of the aggregate encryption scheme, in the following discussion, the modulus is explicitly reflected in the notation, for example, $Enc_k(x; M)$ stands for: encryption of plaintext x under key k with (public) modulus M.

5.2.1 *Average.* Each sensor encrypts its plaintext data, x_i, to obtain $c_{x_i} = Enc_{k_i}(x_i; M)$. Recall that M is large enough to prevent any overflow; it is thus set to: $M = n * t$, where t is the range of possible measurement values and n is the number sensors. The size of each ciphertext c_{x_i} is therefore $log(M) = log(t) + log(n)$ bits.

The sensor forwards c_{x_i} along with key identifying information hdr_{x_i} to its parent, who aggregates all c_{x_j}'s of its k children by simply adding them up

[7]Depending on the number of nodes responding to a query, it could be more efficient to communicate the IDs of nodes that successfully reported values.

(modulo M). The resulting value is then forwarded upstream. Assuming, for simplicity, that it is directly connected to only one sensor, the sink ends up with $C_x = \sum_{i=1}^{n} c_{x_i} \text{mod} M$ and the associated hdr, which identifies the key-set $\{k_1, ..., k_i, ..., k_n\}$. It then computes $S_x = Dec_K(C_x; M) = C_x - K \bmod M$, where $K = \sum_{i=1}^{n} k_i$, and derives the average as: $\text{Avg} = \frac{S_x}{n}$.

5.2.2 *Variance.* Our scheme can be also used to derive the variance of measured data. For this, two moduli are necessary: M for the sum and M' for the sum of squares. Each sensor, i, computes $y_i = x_i^2$ and encrypts it as: $c_{y_i} = Enc_{k_i'}(y_i; M')$. It then also encrypts x_i as in the previous section. As expected, M' needs to be large enough to prevent any overflow; it is set to: $M' = n * t^2$. The size of each ciphertext c_{y_i} is therefore $log(M') = 2 * log(t) + log(n)$ bits. The sensor forwards c_{y_i} and c_{x_i} to its parent. The combined size of the resulting data is $3 * log(t) + 2 * log(n)$. The parent aggregates all of its children's c_{y_j} values via addition. It also separately aggregates c_{x_j} values, as previously described. The two results are then forwarded upstream. The sink ends up with: C_x and $C_y = \sum_{i=1}^{n} c_{y_i} \text{mod } M'$. It computes $V_x = Dec_{K'}(C_y; M') = C_y - K' \text{mod } M'$, where $K' = \sum_{i=1}^{n} k_i'$. The variance is then obtained as: $V_x/n - Av^2$.

6. SECURITY ANALYSIS

We use the CDA security model [Chan and Castelluccia 2007] to analyze the security of the construction presented in Section 4. For completeness, the security model is included in Appendix A. As usual, the adversary is assumed to be a probabilistic polynomial time (PPT) Turing machine. In the model, the adversary can choose to compromise a subset of nodes and obtain all secrets of these nodes. With oracle access, it can also obtain—from any of the uncompromised nodes—the ciphertext for any chosen plaintext. The security goal is the adversary's inability to extract, in polynomial time, any information about the plaintext from a given ciphertext. This is the well-known notion of semantic security [Goldwasser and Micali 1984]. (This is described more formally in Appendix A.)

The concrete construction in Section 4.3 can be shown to achieve semantic security or indistinguishability against chosen-plaintext attacks (IND-CPA), an equivalent notion to semantic security [Goldwasser and Micali 1984], if the underlying key generation function is selected from a PRF family. The security can be summarized by the following theorem.

THEOREM 1. *For a network with n nodes, the concrete construction is semantically secure against any collusion with at most* $(n - 1)$ *compromised nodes, assuming* $F_\lambda = \{f_s : \{0,1\}^\lambda \to \{0,1\}^\lambda\}_{s \in \{0,1\}^\lambda}$ *is a PRF and* $h : \{0,1\}^\lambda \to \{0,1\}^l$ *satisfies the requirement that* $\{t \leftarrow \{0,1\}^\lambda : h(t)\}$ *is uniformly distributed over* $\{0,1\}^l$.

PROOF SKETCH. The detailed proof is in Appendix B. The basic idea is that we assume the existence of a PPT adversary that can break the semantic security of the proposed encryption scheme. We then show how this adversary can be used to break the indistinguishability property of the underlying PRF. By a

contrapositive argument, we say that, if the PRF possesses the indistinguishability property described in Section 4.3, then the proposed encryption scheme is semantically secure.[8] □

Note the standard security goal for encryption is indistinguishability against chosen-ciphertext [Naor and Yung 1990; Katz and Yung 2006]. If a stateful decryption mechanism is assumed: the decryption algorithm keeps track of all nonces previously used, our scheme can also be proven to be secure against chosen-ciphertext attacks. However, the resulting scheme would be inefficient. Nevertheless, it could still be of interest, since in our setting, only the sink decrypts. Because the aggregation allows ciphertexts to be modified in some way without invalidating them, achieving chosen-ciphertext security (more specifically, indistinguishability against adaptive chosen-ciphertext attacks (IND-CCA2)) with a stateless decryption mechanism is likely impossible in our scenario.

7. OVERHEAD ANALYSIS

We now compare the bandwidth consumption of the proposed AGG protocol with the No-Agg (forwarding individual data packets), CON (concatenating and forwarding data packet), and HBH (hop-by-hop encryption and aggregation) approaches, as described in Section 5. The overall bandwidth in the WSN and the number of bits sent by individual nodes are considered for different WSN tree-like topologies. We next describe the network model used in the measurements. The comparison is for two cases: (1) average value only, and (2) both average and variance values.

7.1 Network Model

We assume a multilevel network tree with a multitude of sensors and one sink. To simplify our discussion, we assume a balanced k-ary tree, as shown in Figure 1. Let t denote the range of possible measurement values (e.g., if a sensor measures temperatures between 0 and 120 Fahrenheit, then $t = 121$). We also assume, for simplicity, that only the leaves of the tree are sensors and that the intermediate nodes are just forwarding nodes.

We analyze bandwidth in this WSN model from two perspectives: (1) number of bits sent per node at different levels in a 3-ary tree, and (2) total number of bits transmitted in the WSN for 3-ary trees of various heights. These measurements are performed for the four models: No-Agg, CON, HBH, and AGG.

Next, we show how to compute the number of bits (header and payload) sent per node. We choose the packet format used in TinyOS [Karlof et al. 2004], which is the OS of choice for popular Berkeley motes. The packet header is 56 and maximum supported data payload is 232 bits, respectively. If a data payload is larger than 232 bits, it is sent over several packets. For example, the transmission of a data payload of 300 bits results in the transmission of 2 packets: one of size 288 bits ($232 + 56$) and another of size 124 bits ($68 + 56$). The total cost is then equal to 312 bits.

[8]See Appendix B.

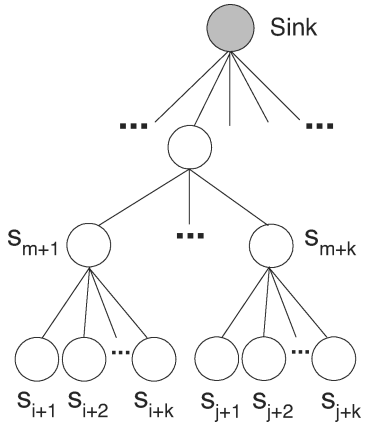

Fig. 1. Multi-level WSN model with nodes of degree k.

For No-Agg, a node only needs $log(t)$ bits to encode its data. Also, all internal nodes forward packets sent to them by their children, and the number of packets received grows exponentially in k as we move higher in the tree—closer to the sink. The CON scheme reduces the required bandwidth by reducing the overhead due the headers, but still has an exponential growth. Note that with the CON scheme, each intermediate node needs to append—to the concatenate packet—the IDs of its children that did not reply to the query. These IDs must be propagated to the sink along with the aggregate.

In HBH, the number of sent bits depends on the node's level in the WSN tree. Leaf nodes only send $log(t)$ bits (as in No-Agg), while higher-up nodes receive aggregated data and therefore send more bits. Additionally, when the variance is also requested, the aggregating nodes need to keep track of this value separately, and use approximately $log(n't)$ bits to encode it (where n' is the number of node-values aggregated so far).

Finally, in AGG, the number of bits sent by a node depends on the size of the modulus M. Its size corresponds to the maximum possible aggregate value, which is $M = n * t$, that is, all sensors report the largest possible reading. Therefore, in encoding the average, each node uses $log(M) = log(t) + log(n)$ bits. If variance is desired, a node sends an additional ciphertext corresponding to x^2. This requires extra $log(n * t^2) = 2*log(t)+log(n)$ bits. Also, each aggregator needs to append to the aggregate, the IDs of its children that did not reply to the query. These IDs must be propagated to the sink along with the aggregate.

7.2 Numerical Results

We now compare the performance of the four schemes.

Forwarding Cost per node (fairness). Tables I, II, and III show the number of bits sent per node at each level in a 3-degree tree of height[9] 7 when $t = 128$ for the different schemes.

[9]The sink is at level 0.

Table I. Number of Bits Sent per Node for Each Level with the No-Agg and CON Schemes

Levels	Num Nodes	No-A (0%)	No-A (10%)	No-A (30%)	CON (0%)	CON (10%)	CON (30%)
1	3	45927	41334	32149	6335	6811	7708
2	9	15309	13778	10716	2149	2270	2564
3	27	5103	4593	3572	735	775	856
4	81	1701	1531	1191	245	258	285
5	243	567	510	397	119	123	132
6	729	189	170	132	77	78	81
7	2187	63	57	44	63	63	64

Table II. Number of Bits Sent per Node for Each Level with the HBH Scheme

Levels	Nodes	HBH-A (0%)	HBH-A (10%)	HBH-A (30%)	HBH-AV (0%)	HBH-AV (10%)	HBH-AV (30%)
1	3	73	72	72	96	96	95
2	9	71	71	70	93	92	92
3	27	69	69	69	90	89	89
4	81	68	68	67	86	86	85
5	243	66	66	66	83	83	82
6	729	64	64	64	80	80	79
7	2187	63	62	61	63	62	61

Table III. Number of Bits Sent per Node for Each Level with the AGG Scheme

Levels	Num Nodes	Agg-A (0%)	Agg-A (10%)	Agg-A (30%)	Agg-AV (0%)	Agg-AV (10%)	Agg-AV (30%)
1	3	75	1117	3315	100	1142	3340
2	9	75	422	1117	100	447	1142
3	27	75	172	422	100	197	448
4	81	75	107	172	100	132	197
5	243	75	85	108	100	111	132
6	729	75	78	85	100	103	110
7	2187	75	67	52	100	91	71

We considered three scenarios: (1) all nodes reply,[10] (2) 90% of the nodes reply,[11] and (3) 70% of the nodes reply.[12] We assumed for simplicity that the distribution of non responding nodes is uniform among all nodes. We believe that this assumption is reasonable since only leave nodes are sensors in our simulation setup. Therefore, the number of nonresponsive nodes is the parameter that primarily affects the results.

For *No-Agg*, it is obvious from the data that the communication load fluctuates widely among sensors at different levels, for example, nodes at level 7 send 3 orders of magnitude less data than those at level 1. Because nodes closer to the sink send significantly larger amounts of data than their descendants, they

[10]Referred to in the tables as $Scheme - A(0\%)$ when only the average is computed and as $Scheme - AV(0\%)$ when the average and variance are computed.
[11]Referred in the table as $Scheme - A(10\%)$ when only the average is computed and as $Scheme - AV(10\%)$ when the average and variance are computed.
[12]Referred in the table as $Scheme - A(30\%)$ when only the average is computed and as $Scheme - AV(30\%)$ when the average and variance are computed.

deplete their batteries and die sooner. As soon as one level of nodes in the tree stops functioning, the entire WSN stops functioning as well. To achieve communication load balancing, either sensors must be moved around periodically, or the sink has to move (thus changing the root and the topology of the tree). We consider both approaches to be quite impractical. The *CON* scheme reduces the required bandwidth by a factor of 4, but it is still unfair. The nodes that are close to the sink need to forward many more bits that the other nodes. Note that with the *No-Agg* scheme, the required bandwidth decreases as the number of nonresponding sensors increases. This is the result of the network having fewer messages to forward. In contrast, with the *CON* scheme, the bandwidth increases because the IDs of the nonresponding nodes must be appended to the concatenated messages.

Table II shows a steady increase in bits-per-node for HBH, for both the average only (HBH-A), and average-plus-variance (HBH-AV) cases. Note a relatively dramatic increase in bits transmitted between nodes at level 7 and 6 for HBH-AV. This is because leaf nodes only send a ciphertext of x. The ciphertext representing x^2 (needed for the computation of the variance) can be computed by their parents from x and therefore does not need to be transmitted. Since in HBH, packets are not forwarded (as in No-Agg), we observe a significant reduction in bits per node at all non-leaf levels.

With *AGG*, when all the sensors are replying, a constant number of bits is sent by each node at every level in the tree. However, this number is larger than the maximum in any HBH approach, due to the size of the modulus M. As previously discussed, the number of bits sent by leaves is larger with the aggregation methods (AGG-A: $56 + log(t) + log(n) = 75$ bits, AGG-AV: $56 + 3 * log(t) + 2 * log(n) = 100$ bits) than when no aggregation is used ($56 + log(t) = 63$ bits). However, aggregation distributes the load evenly over all nodes, regardless of their distance to the sink. We believe this to be a major advantage in WSNs. In the second and third scenarios (90% and 70% of sensors, respectively, reply), the number of bits processed by each node gets larger the closer it gets is to the sink. This is the result of appending IDs of nonresponding children to the aggregate. As we move up the tree, the list of nonresponding nodes increases. If we assume that $Z\%$ of the nodes do not reply, an intermediate node must append to the aggregated message, IDs of $Z\%$ of its k children. For example, for $Z = 30\%$, a node at level 3 has $3^4 = 81$ children and has to append $81 * 0.3 = 25$ IDs. The total size of the aggregated message payload is thus $9 + 25 * 12 = 310$ bits. This results in the transmission of 2 packets, at a total cost of 422 bits ($2 * 56 + 310$), as shown in Table III.

Bandwidth Gain. Tables IVand V show the bandwidth transmission gains of HBH, CON, and AGG over No-Agg, assuming 3-degree WSNs of various heights. We consider gains for two cases: (1) only the average, and (2) both the average and variance.[13] These gains are obtained from the respective total bandwidth costs: C_{HBH}, C_{AGG}, C_{No-Agg} and C_{CON}, by adding, for each scheme, the total number of bits forwarded by each node. The bandwidth gain

[13]Recall that, in No-Agg and CON, no extra values need to be sent if the variance is needed.

Table IV. WSN Bandwidth Performance Gain of the AGG and HBH Schemes when Aggregating the Average for a 3-Tree and $t = 128$

Levels	Num Nodes	Agg (0%)	Agg (30%)	HBH (0%)	HBH (30%)	CON (0%)	CON (30%)
3	40	1.75	1.48	2.05	1.48	1.85	1.31
4	121	2.27	1.86	2.7	1.92	2.27	1.59
5	364	2.8	2.18	3.31	2.37	3.62	1.8
7	3280	3.92	2.71	4.61	3.3	3.2	2.08
8	9841	4.48	2.86	5.27	3.78	3.45	2.10

Table V. WSN Bandwidth Performance Gain of the AGG and HBH Schemes when Aggregating the Average-and-Variance for a 3-Tree and $t = 128$

Levels	Num Nodes	Agg (0%)	Agg (30%)	HBH (0%)	HBH (30%)	CON (0%)	CON (30%)
3	40	1.30	1.11	1.91	1.37	1.85	1.31
4	121	1.69	1.4	2.47	1.77	2.27	1.59
5	364	2.1	1.67	3.05	2.19	3.62	1.8
7	3280	2.92	2.14	4.25	3.1	3.2	2.08
8	9841	3.34	2.3	4.85	3.49	3.45	2.10

of HBH, AGG, and CON are defined as C_{No-Agg}/C_{HBH}, C_{No-Agg}/C_{AGG}, and C_{No-Agg}/C_{CON}, respectively.

For example, in a 3-tree of height 5, there are 364 nodes, and computing the average only, AGG-A achieves a factor of 2.8 speedup over No-Agg. As expected, HBH-A and HBH-AV offer better performance than both AGG-A and AGG-AV, respectively, although both outperform No-Agg. The main reason for using AGG over HBH is end-to-end privacy. With HBH, it is enough for an attacker to compromise one node close to the sink to gain a lot of knowledge about aggregated data in the WSN. This is because each node in HBH stores the secret key needed for decryption and encryption. In contrast, in AGG, nodes do not store sensitive key material and the only data an attacker can learn is a single sensor's individual reading. The gains achieved by the CON scheme are quite comparable to the gains obtained with AGG. However, as seen in the previous section, AGG provides a better balance of the load among the nodes and therefore extends the overall lifetime of the network.

7.3 Computation Costs

We now discuss computation costs for the proposed scheme and issues related to implementation. Let t_{add} and t_{multi} denote the respective costs of addition and multiplication operation mod M. Let t_{prf} and t_h denote the costs of a PRF and a length-matching hash function, respectively. Let t_{ce} and t_{cd} denote the costs of one encryption and one decryption with a cipher used in the HBH scheme. Overall computation costs for proposed protocols are shown in Table VI, assuming L reporting nodes ($|hdr| = L$). For the aggregation operation, our calculations assume that each aggregation involves only two inputs.

AGG places all decryption tasks at the sink, while HBH distributes the decryption cost over all non-leaf nodes in the network. Thus in HBH, a sensor may need to perform more computation than the sink. Since the sink is usually a more powerful device, AGG is clearly preferable.

Table VI. Computation Cost Comparison

	Hop-by-hop Encryption (HBH)	Aggregate Encryption (AGG)
Encryption	t_{ce}	$t_{prf} + t_h + t_{add}$
Decryption	t_{cd}	$2L \cdot t_{prf} + L \cdot t_h + L \cdot t_{add}$
Aggregation (per 2 inputs)	$2 \cdot t_{cd} + t_{ce} + t_{add}$	t_{add}

In AGG, to encrypt its value, a node performs one PRF invocation, one length-matching hash, and one mod M addition. It also performs one extra addition for aggregation. If the hash is implemented by bit truncation, its computation cost is negligible compared to that of addition. Else, if the hash is implemented by truncation combined with exclusive-OR, the computation cost is roughly equal to the cost of addition. We thus consider the cost of evaluating h to be negligible in the calculation of overall computation cost for encryption. As a result, the cost of encryption is dominated by a single PRF invocation.

As mentioned in Section 4.3, a collision-resistant hash can be used in key derivation if its assumed pseudorandomness is acceptable. For example, Perrig et al. [2001] illustrate such usage in the WSN context and demonstrate feasibility in terms of computation complexity. Hence, the computation cost of the proposed scheme is reasonable for most WSN applications.

8. AGGREGATE AUTHENTICATION AGAINST OUTSIDER-ONLY ATTACKS

Although the proposed aggregate encryption scheme provides end-to-end privacy, (like others, e.g., Madden et al. [2002]) it is vulnerable to false data injection attacks. In its current form, even an external attacker can add an arbitrary value to an aggregate ciphertext.

The AGG scheme is complementary to most authentication techniques in the literature, including Chan et al. [2006]; Yang et al. [2006]; Przydatek et al. [2003]; and Hu and Evans [2003]. Any of these techniques can be used in conjunction with the proposed scheme. It should be noted that these techniques are not end-to-end; in particular, some form of call-backs to the aggregating nodes (after the sink receives the aggregate) are necessary. This section provides an end-to-end alternative to aggregate authentication. However, it is only secure against external attackers who do not know the secret group key. The existence of any malicious or compromised nodes would imply a total breach of security.

In Chan and Castelluccia [2008], the notion of aggregate message authentication codes (AMAC) is proposed as a natural extension of one-to-one message authentication codes (MAC) and it is shown that no scheme may be designed to achieve such a notion. Since the proposed notion is not a contrived one, we could conclude that no scheme can be constructed to provide end-to-end aggregate integrity against chosen-message attacks in the presence of compromised nodes without a relaxation to the unforgeability notion against chosen message attacks.

Even with call-backs, as in Chan et al. [2006], the only guarantee is that an adversary cannot manipulate an aggregation result by an amount beyond what can be achieved through data injection at the compromised nodes, unless prior knowledge of the statistical distribution of the data is utilized for outlier

detection at the sink. In the context of additive aggregation, without asking each sensor to provide a range proof for its contribution, the impact of a compromised node in Chan et al. [2006] (regarding manipulation of an aggregation result) is essentially the same as its counterpart in our proposed aggregate authentication scheme. Indeed, a range proof requires prior knowledge of the statistical distribution of data.

If there are no compromised nodes, our scheme assures that no data can be injected into an aggregate without being detected. The basic idea of our scheme is to add—to each node's data—a keyed, aggregatable checksum/authenticator computed with the aid of a shared group key. Without knowledge of this group key, it is infeasible for an external attacker to compute a valid checksum for any modified data. We re-emphasize that, compromise of any node (and hence the group key) would cause a complete security breach of the authentication scheme. Nevertheless, this is the best we can achieve for end-to-end aggregate authentication.

8.1 Details of the Protocol

The aggregate authentication scheme is as follows.

Combined Encryption and Aggregate Authentication Scheme

Key Distribution:
Each sensor i has 3 secret keys (k_i, k_i', k). They can be generated from three independent master keys using a PRF, as in the basic scheme. The sink keeps all three master keys. k_i and k_i' correspond to the encryption key ek_i in the basic scheme. Each node receives a distinct pair, (k_i, k_i') and also gets a common group key, k.

Encryption + Checksum Computation:
Let M be the modulus. For a reporting epoch r.

(1) Each node i generates session keys $(k_i^{(r)}, k_i^{(r)'}, k^{(r)})$ from its secret keys (k_i, k_i', k) using a PRF and the length-matching hash function, as in the basic scheme. $(k_i^{(r)} = h(f_{k_i}(N_r))$, $k_i^{(r)'} = h(f_{k_i'}(N_r))$, and $k^{(r)} = h(f_k(N_r))$ where $f(\cdot)$ is the PRF, $h(\cdot)$ is the length-matching hash and N_r is the nonce used for epoch r.)
(2) For plaintext message $m_i \in [0, M-1]$, encrypt m_i using $k_i^{(r)}$ and obtain ciphertext $x_i = m_i + k_i^{(r)} \bmod M$.
(3) Compute checksum: $y_i = m_i \cdot k^{(r)} + k_i^{(r)'} \bmod M$.
(4) Ciphertext and checksum are: (hdr, x_i, y_i), where $hdr = \{i\}$.

Decryption + Verification:

(1) Given ciphertext (hdr, x, y), generate session keys $(k_i^{(r)}, k_i^{(r)'}, k^{(r)})$ for each $i \in hdr$.
(2) Compute $m = x - \sum_{i \in hdr} k_i^{(r)} \bmod M$. m is resulting plaintext.
(3) Check $y \stackrel{?}{=} \sum_{i \in hdr} k_i^{(r)'} + k^{(r)} \cdot m \bmod M$. If yes, set $b = 1$, otherwise, set $b = 0$.
(4) Return (m, b). Note that $b = 0$ indicates verification failure.

Addition of Ciphertexts: Given two ciphertexts (hdr_i, x_i, y_i) and (hdr_j, x_j, y_j),

(1) Compute $hdr_l = hdr_i \cup hdr_j$.
(2) Compute $x_l = x_i + x_j \bmod M$.
(3) Compute $y_l = y_i + y_j \bmod M$.
(4) Aggregated ciphertext is: (hdr_l, x_l, y_l).

The final aggregated ciphertext (hdr, x, y) received at the sink can be expressed as two equations:

$$\begin{cases} x = K_1^{(r)} + m \\ y = K_2^{(r)} + K^{(r)} \cdot m \end{cases}, \tag{1}$$

where m is the final aggregate of the plaintext data and $K_1^{(r)}, K_2^{(r)}, K^{(r)}$ are two sums of node keys and the common group key (for epoch r) given by the following expressions:

$$K_1^{(r)} = \sum_{i \in hdr} k_i^{(r)}, \; K_2^{(r)} = \sum_{i \in hdr} k_i^{(r)'}, \text{ and } K^{(r)} = k^{(r)}.$$

Equation (1) can be viewed as a set of constraint equations for a particular hdr that a correct pair (x, y) should satisfy. For each epoch, hdr is part of the input to the verification process to define the coefficients $K_1^{(r)}, K_2^{(r)}, K^{(r)}$ of the constraint equations in (1); hdr uniquely specifies a subset of nodes whose data are supposed to have been incorporated in (x, y).

If (x, y) has not been tampered with, the plaintext aggregate, m, extracted from the first constraint equation in (1) should satisfy the second constraint equation in (1); m is a correct aggregate of the data contributed by the nodes in hdr when they all act honestly. The goal of an external adversary is thus to find a different valid pair (x', y') for the same hdr such that

$$\begin{cases} x' = K_1^{(r)} + m' \\ y' = K_2^{(r)} + K^{(r)} \cdot m' \end{cases}$$

for some $m' \neq m$ and m' is not necessarily known by the adversary. Note that the coefficients $K_1^{(r)}, K_2^{(r)}, K^{(r)}$ have to be the same as that in the equations for (x, y) for a successful forgery. Without knowledge of $K^{(r)}$, the probability for any PPT adversary to find such a valid pair (x', y') for the given hdr should be negligibly small. The proposed protocol guarantees with high probability that, for an epoch r, any pair (x, y) that passes the verification test for a given hdr has to allow the recovery of a correct aggregate whose contributions can only come from nodes in hdr with knowledge of $K^{(r)}$ (with exactly one contribution from each node in hdr).

In any epoch, by passively observing transmissions from honest nodes in a network, an adversary without knowledge of $K^{(r)}$ can still create multiple tuples of the form (hdr, x, y), each with a distinct hdr, to pass the verification test of Equation (1). This can be achieved by simply aggregating valid ciphertext-checksum pairs eavesdropped in the transmissions of the honest nodes. However, it should be noted that, for each hdr, there is at most one such tuple and

the corresponding pair of (x, y) is indeed a correct ciphertext-checksum pair for hdr in the sense that this pair of (x, y), upon verification, can recover an aggregate m, the contributions of which only originate from the honest nodes specified in hdr, that is, $m = \sum_{i \in hdr} m_i$, where m_i is the measurement of node i. In other words, in the set $\mathcal{C}$ of ciphertext-checksum pairs obtained by combining eavesdropped pairs through the aggregation functionality, if a pair $(x, y) \in \mathcal{C}$ passes the verification equations in (1) for hdr, any pair $(x', y') \in \mathcal{C}$ that can satisfy the same set of equations (with the same set of coefficients) has to be equal to (x, y). Hence, any external attacker without knowledge of $K^{(r)}$ still cannot inject its data into an aggregate ciphertext pair (x, y) that satisfies the constraint equations in (1) even though he may be able to create such a pair from the ciphertext-checksum pairs obtained from eavesdropping the transmissions of honest nodes; neither can the attacker modify an existing valid pair of (x, y) to pass the verification test for the same hdr, but produce a different aggregate output except with a negligibly small probability.[14]

It is thus fair to say the best that an external adversary without knowledge of $K^{(r)}$ can achieve in breaking the proposed scheme, is essentially limited to excluding the contributions of some honest nodes from being incorporated into an aggregate. Such exclusion would usually have slight impact in the calculation of mean and variance unless the exclusion makes up a pretty large fraction of nodes, in which case it would make the sink suspect the occurrence of a possible attack. It should be emphasized that to achieve so with impact, the adversary must be able to intercept and modify a considerable portion of the transmissions in the entire network, which is normally hard for an attacker to achieve. To defend against this node exclusion attack, we present, in Section 8.4, an add-on mechanism to protect the integrity of hdr.

8.2 Security Analysis

Recall that the goal of the proposed extension of aggregate authentication is to guard against any external attackers (without knowledge of the keys) from injecting data into an aggregate. The security of the proposed scheme is summarized by the following theorem.

THEOREM 2. *Given a genuine ciphertext-checksum pair (x, y), corresponding to an aggregate m, which incorporates data from a group of nodes specified by hdr and all other communication transcripts between nodes, the probability of successfully forging a valid pair $(x', y') \neq (x, y)$ for some $m' \neq m$ to pass the verification test of the aggregate authentication scheme for the same hdr is negligible for any external PPT (Probabilistic Poly-Time) adversary without knowing K, assuming the encryption keys and the group key are generated by a PRF based on different seed keys.*

PROOF. Assume the PRF has some indistinguishability property as usual. We prove by contradiction, showing that a PPT adversary that can forge a valid pair (x', y') can also break the indistinguishability property of the underlying

[14]An adversary may be able to obtain another valid (x, y) pair but it is valid only for a different hdr.

PRF. We show the reduction[15] in two steps: first, we show that a forging algorithm to find (x', y') can be used as a subroutine to solve a newly defined problem called "Under-determined Equation Set with Pseudorandom Unknowns (UESPU)"; then we show that the UESPU problem is computationally hard if the underlying PRF has the usual indistinguishability property. The UESPU problem is defined as follows:

Under-determined Equation Set with Pseudorandom Unknowns (UESPU) Problem—Suppose K_1, K_2, K are independent random seeds. Let $K_1^{(r)}$, $K_2^{(r)}$ and $K^{(r)}$ denote the hashed outputs of a PRF f, at input r, corresponding to seed keys K_1, K_2 and K.[16] Given a 3-tuple (m, x, y), where $x = K_1^{(r)} + m$ and $y = K_2^{(r)} + K^{(r)} \cdot m$, find $(K_1^{(r)}, K_2^{(r)}, K^{(r)})$ while allowed to evaluate the PRF at any input $r' \neq r$.[17]

Without loss of generality, in the UESPU problem, each of $K_1^{(r)}$, $K_2^{(r)}$ and $K^{(r)}$ is treated as a single hashed output of f. In the proposed aggregate authentication, they are the sums of hashed outputs of f. If they are represented as the sums of hashed output of f instead, the modified problem would remain hard if f is a PRF.

Solving the UESPU problem using a forger of (x', y'). Suppose there exists a PPT adversary $\mathcal{A}$ that can forge a valid pair (x', y') at an epoch with nonce r with non-negligible probability p_f. Using $\mathcal{A}$ as a subroutine, we can construct another algorithm, $\mathcal{A}'$, to find $(K_1^{(r)}, K_2^{(r)}, K^{(r)})$ from (m, x, y) with probability p_f, in any instance of the UESPU problem. Note that $\mathcal{A}'$ should be able to answer queries from $\mathcal{A}$ for any $r' \neq r$ by passing the queries to its challenger.

The construction of $\mathcal{A}'$ is as follows: Give $\mathcal{A}$ the pair (x, y). When $\mathcal{A}$ returns a pair $(x', y') \neq (x, y)$, we can determine $K_1^{(r)}$, $K_2^{(r)}$, $K^{(r)}$ from the resulting set of equations. The explanation is as follows:
Note that

$$\begin{aligned} x &= K_1^{(r)} + m \\ y &= K_2^{(r)} + K^{(r)} \cdot m. \end{aligned}$$

So we have two equations and three unknowns. If (x', y') is a valid forgery, then it must satisfy the following two equations (with the same $K_1^{(r)}$, $K_2^{(r)}$ and

[15]The reduction of the problem of breaking the indistinguishability of the PRF to the problem of forging a valid (x', y') pair.

[16]That is, $K_1^{(r)} = h(f_{K_1}(r))$, $K_2^{(r)} = h(f_{K_2}(r))$, and $K^{(r)} = h(f_K(r))$ where h is the length-matching hash function.

[17]The UESPU problem is typically hard if f is a PRF. More formally defined, given that l is the key length of the PRF f, and h is a length-matching hash function, the following probability is negligible in l for any PPT algorithm, $\mathcal{A}$.

$$Pr\left[\begin{array}{l} K_1 \leftarrow \{0,1\}^l; K_2 \leftarrow \{0,1\}^l; K \leftarrow \{0,1\}^l; r \leftarrow \{0,1\}^l; \\ K_1^{(r)} = h(f_{K_1}(r)); K_2^{(r)} = h(f_{K_2}(r)); K^{(r)} = h(f_K(r)); \\ m \leftarrow \mathbb{Z}_M; x = K_1^{(r)} + m; y = K_2^{(r)} + K^{(r)} \cdot m \end{array} : \mathcal{A}^f(m, x, y) = (K_1^{(r)}, K_2^{(r)}, K^{(r)})\right]$$

$K^{(r)}$) in order to pass the verification test:

$$\begin{aligned} x' &= K_1^{(r)} + m' \\ y' &= K_2^{(r)} + K^{(r)} \cdot m' \end{aligned}$$

for some unknown value $m' \neq m$.

The pair (x', y') adds in two new equations and one unknown, m'. Since $(x', y') \neq (x, y)$ and $m' \neq m$, it can be assured that the four equations are independent. Hence, there are four independent equations and four unknowns in total and it should be easy to solve for $K_1^{(r)}$, $K_2^{(r)}$, $K^{(r)}$ (a contradiction to the UESPU assumption). The probability of solving the problem in the UESPU assumption is hence p_f.

Suppose there are n reporting nodes. The communication transcripts can be easily simulated by randomly picking $(n-1)$ random pairs of ciphertext-checksum (x_i, y_i) and subtracting them from (x, y) to obtain the n-th pair. Since $\mathcal{A}$ does not have any knowledge about the node keys, real pairs of (x_i, y_i) should look random to $\mathcal{A}$. Hence, $\mathcal{A}$ could not distinguish its view in the simulation and that in the real attack. On the other hand, it could be concluded that knowing (x_i, y_i) without knowing the node keys would not help in creating a valid forgery. In the preceding discussion, we treat $K_1^{(r)}$, $K_2^{(r)}$, $K^{(r)}$ as a single output of a PRF for the sake of clarity and easy comprehension; more precisely, in the aggregate authentication scheme, each one of them is the sum of outputs of a PRF seeded with distinct keys (one from each sensor node). Nonetheless, the arguments and conclusion apply to both cases.

A distinguisher for the PRF using an algorithm that solves the UESPU problem. The UESPU problem is hard if $K_1^{(r)}$, $K_2^{(r)}$, $K^{(r)}$ are generated by a PRF. Obviously, m and x can uniquely determine $K_1^{(r)}$. But the equation $y = K_2^{(r)} + K^{(r)} \cdot m$ has two unknowns, which cannot be uniquely determined. It could be shown that if there exists an algorithm $\mathcal{A}'$ solving in poly-time $K_2^{(r)}$ and $K^{(r)}$ from m and y, then the indistinguishability property of the underlying PRF is broken.

The idea is as follows: assume the seed key for generating $K^{(r)}$ is unknown, but the seed key for generating $K_2^{(r)}$ is known. That is, we can generate $K_2^{(r')}$ for any r'. When a challenge, $K^{(r)}$, is received, we have to determine whether it is randomly picked from a uniform distribution or generated by the PRF with an unknown seed key. We generate $K_2^{(r)}$ from the known seed key. Then we pass $y = K_2^{(r)} + K^{(r)} \cdot m$ to $\mathcal{A}'$. If the solution from $\mathcal{A}'$ does not match the generated $K_2^{(r)}$, we reply that $K^{(r)}$ is randomly picked, otherwise it is generated from the PRF. If $\mathcal{A}'$ has non-negligible probability of breaking the UESPU assumption, the preceding construction would also have the non-negligible advantage of breaking the indistinguishability property of the underlying PRF. Note that all queries from $\mathcal{A}'$ could be answered by sending queries to the challenger and running the PRF with the known key. □

8.3 Additional Overheads

The aggregate authentication extension leads to additional costs in both communication and computation. For the communication cost, the length of each

Table VII. Additional Computation Costs of the Extension of Aggregate Authentication (Assuming L is the Number of Nodes Contributing to an Aggregate)

	Additional Computation Costs
Checksum Generation	$2 \cdot t_{prf} + 2 \cdot t_h + t_{add} + t_{multi}$
Checksum Verification	$(2L+1) \cdot t_{prf} + (L+1) \cdot t_h + (L+1) \cdot t_{add} + t_{multi}$

ciphertext is now increased by $|M|$ (where M is the modulus of the arithmetics in use). This is the size of the added checksum. For the computation cost, the notations of Section 7.3 are used. The additional computation costs needed for checksum generation and verification are summarized as follows. In the calculation of verification cost, the cost of a comparison operation in mod M is assumed similar to the cost of an addition operation in mod M.

8.4 Defense Against the Node Exclusion Attack

The proposed aggregate authentication scheme is vulnerable to a node exclusion attack wherein an adversary can eliminate the contributions of selected nodes from being incorporated into the final aggregate. To defeat this node exclusion attack, we need to protect the integrity of hdr. We briefly describe a modification on hdr generation to assure hdr integrity. The modification is only on the hdr part and makes use of a standard one-to-one message authentication code (MAC). Each sensor node now needs to store an extra key, q_i, shared with the sink. As in the original scheme, q_i is used to generate a session key, $q_i^{(r)}$, for epoch r. $q_i^{(r)}$ is used as an input key to the MAC algorithm for tag generation.

In the original scheme, hdr is merely a list of reporting nodes' IDs. In the new scheme, the header generation for a leaf node remains the same (that is, node i generates $hdr_i = \{i\}$) and the difference is at the interior nodes. For an interior node, j, with child nodes i_1, i_2, i_3, the new header format is: $hdr_j = \{j||i_1||i_2||i_3\}$ (where $||$ denotes concatenation), that is, each interior node appends a list of its child nodes in its header. For a binary aggregation tree with node IDs numbered in a top-down and left-to-right manner (that is, the root is 1, the next level nodes are 2 and 3, and so on), the sink would receive a header of this form: $hdr = \{1||2||3, 2||4||5, 3||6||7, \ldots, (n-1), n\}$. Note that the information in hdr would allow the sink to reconstruct the aggregation tree topology.

Another modification is each node now generates a MAC tag for its header. For example, node i would use its session key $q_i^{(r)}$ to generate the tag $t_i = MAC_{q_i^{(r)}}(hdr_i)$. For aggregation, the part on hdr remains the same and MAC tags are aggregated by taking exclusive-OR on the tags. In details, given two headers and the corresponding MAC tags (hdr_i, t_i) and (hdr_j, t_j), the aggregation result is (hdr_l, t_l), where $hdr_l = hdr_i \cup hdr_j$ and $t_l = t_i \oplus t_j$. For verification, upon receipt of (hdr, t), the sink regenerates the tag t_i for each $hdr_i \in hdr$ using the session key $q_i^{(r)}$ and takes exclusive-OR on the regenerated tags and checks whether the result matches t. The sink accepts hdr if and only if the result matches t.

If hdr is the original aggregate header generated by honest nodes, this scheme would reject all other $hdr' \neq hdr$ created by a malicious outsider. If the MAC scheme is unforgeable against chosen message attacks [Bellare et al. 1996], it would be impossible for an external adversary to remove any node from the original aggregation tree while still able to pass the verification test at the sink. Like the proposed aggregate authentication scheme, this header protection scheme is vulnerable to insider attacks but is good enough as an add-on to the proposed aggregate authentication scheme.

8.4.1 *Security Analysis.* Denote the correct header and its tag received at the sink by (hdr, t). To exclude a particular node, say node i, from contributing to the final aggregate, an adversary needs to remove a sub-tree rooted at node i and add back other nodes in this sub-tree. Suppose we consider removing a sub-tree rooted at node v whose parent and sibling are node u and node w respectively. Using the MAC tags eavesdropped, an adversary can remove from t all the tags generated by nodes in the concerned sub-tree; taking exclusive OR of the eavesdropped tags, and t would work. In order to pass the verification test, an adversary needs to replace tag t_u (contributed by node u) in the final tag t. Originally, $t_u = MAC_{q_u^{(r)}}(u||v||w)$. Now, the adversary needs to replace it with a new tag $t'_u = MAC_{q_u^{(r)}}(u||w)$ in order to pass the verification at the sink. If the adversary can do so without knowing $q_u^{(r)}$, t'_u is a valid forgery, thus breaking the security of the underlying MAC scheme.

Note that forging t is not any easier than forging t_u alone. The reason is if we have an adversary $\mathcal{A}$ that can forge t without knowing all the node keys, this adversary can be used by an algorithm $\mathcal{A}'$ (knowing all keys except $q_u^{(r)}$) to forge t_u. Since $\mathcal{A}'$ has the knowledge of all the node keys except that of node u, it can answer all tag generation queries from $\mathcal{A}$ for any node except u. For queries on node u, it can pass the queries to its challenger. When $\mathcal{A}$ outputs a forger t, $\mathcal{A}'$ can generate a forged t_u by taking exclusive-OR of t with $t_i, \forall i \neq u$. Generating all these t_i's is possible for $\mathcal{A}'$ since it knows all these keys. If t is a valid forgery, so is t_u.

8.4.2 *Additional Overheads.* The communication overhead would include a MAC tag plus a longer header that is bounded by a constant factor of two. The header overhead would be at most doubled. The worst case would be hdr and its size is $(2n - 1)$ for an aggregation tree with n nodes.

9. RELATED WORK

The problem of aggregating encrypted data in WSNs was partially explored in Girao et al. [2004]. In this paper, the authors propose using an additive and multiplicative homomorphic encryption scheme to allow aggregation of encrypted data. While this work is very interesting, it has several important limitations. First, it is not clear how secure the encryption scheme really is. Second, as acknowledged by the authors, the encryption and aggregation operations are very expensive and therefore require quite powerful sensors. Finally, in the proposed scheme, the encryption expands the packet size significantly. Given all these drawbacks, it is questionable whether aggregation is still beneficial.

In contrast, our encryption scheme is proven to be secure and is very efficient. Encryption and aggregation only require a small number of single-precision additions. Furthermore, our encryption scheme only expands packet sizes by a small number of bits. As a result, it is well adapted to WSNs consisting of very resource constrained sensors.

Hu and Evans [2003] propose a protocol to securely aggregate data. The article presents a way to aggregate MACs (message authentication code) of individual packets such that the sink can eventually detect nonauthorized inputs. This problem is actually complementary to the problem of aggregating encrypted data, we are considering in this article. The proposed solution introduces significant bandwidth overhead per packet. Furthermore, it requires the sink to broadcast n keys, where n is the number of nodes in the network, at each sampling period. This makes the proposed scheme nonpractical.

Although not related to data privacy, Przydatek et al. [2003] present an efficient mechanism for detecting forged aggregation values (min, max, median, average and count). In their setting, a trusted outside user can query the WSN. The authors then look into how to reduce the trust placed in the sink node (base station) while ensuring correctness of the query response. A work by Wagner [2004] examines security of aggregation in WSNs, describing attacks against existing aggregation schemes before providing a framework in which to evaluate such a scheme's security.

10. CONCLUSION

This article proposes a new homomorphic encryption scheme that allows intermediate sensors to aggregate encrypted data of their children without having to decrypt. As a result, even if an aggregator is compromised, it cannot learn the data of its children, resulting in much stronger privacy than a simple aggregation scheme using hop-by-hop encryption.

We show that, if key streams are derived from a good PRF, our scheme can achieve semantic security against any node collusion of size less than the total number of nodes.

We evaluate the performance of our scheme and show, as expected, that it is slightly less bandwidth-efficient than the naíve hop-by-hop scheme. However it provides a much stronger level of privacy—comparable to that provided by end-to-end encryption with no aggregation. We also show that our scheme distributes the communication load quite evenly among all nodes, resulting in better network longevity.

Finally, we augmented our scheme to provide end-to-end aggregate authentication. Without knowledge of a group key, an external attacker cannot tamper with any aggregate, without being detected.

In conclusion, we offer efficient and provably secure techniques for end-to-end privacy and authenticity, with reasonably good security assurances, in WSNs. The proposed scheme only supports mean and variance computation. However, as shown in Castelluccia and Soriente [2008], the same construction could be used as a building block for other aggregation schemes that support more advanced functions, such as median, mode, and range.

APPENDIXES

APPENDIX A: SEMANTIC SECURITY OF CONCEALED DATA AGGREGATION (CDA) [CHAN AND CASTELLUCCIA 2007]

Notation. We follow the notations for algorithms and probabilistic experiments that originate in Goldwasser et al. [1988]. A detailed exposition can be found there. We denote by $z \leftarrow A(x, y, \ldots)$, the experiment of running probabilistic algorithm A on inputs $x, y \ldots$, generating output z. We denote by $\{A(x, y, \ldots)\}$, the probability distribution induced by the output of A. The notations $x \leftarrow \mathcal{D}$ and $x \in_R \mathcal{D}$ are equivalent and mean randomly picking a sample x from the probability distribution $\mathcal{D}$; if no probability function is specified for $\mathcal{D}$, we assume x is uniformly picked from the sample space. We denote by $\mathbb{N}$ the set of non-negative integers. As usual, PPT denotes probabilistic polynomial time. An empty set is always denoted by ϕ.

CDA Syntax. A typical CDA scheme includes a sink R and a set U, of n source nodes, (which are usually sensor nodes), where $U = \{s_i : 1 \leq i \leq n\}$. Denote the set of source identities by ID; in the simplest case, $ID = [1, n]$. In the following discussion, $hdr \subseteq ID$ is a header indicating the source nodes contributing to an encrypted aggregate. A source node, i, has the encryption key ek_i, while the sink keeps the decryption key dk from which all ek_i's can be computed. Given a security parameter λ, a CDA scheme consists of the following polynomial time algorithms.

Key Generation (KG). Let $\mathsf{KG}(1^\lambda, n) \rightarrow (dk, ek_1, ek_2, \ldots, ek_n)$ be a probabilistic algorithm. Then, ek_i (with $1 \leq i \leq n$) is the encryption key assigned to source node s_i and dk is the corresponding decryption key given to the sink R.

Encryption (E). $\mathsf{E}_{ek_i}(m_i) \rightarrow (hdr_i, c_i)$ is a probabilistic encryption algorithm taking a plaintext m_i and an encryption key ek_i as input to generate a ciphertext c_i and a header $hdr_i \subset ID$. Here hdr_i indicates the identity of the source node performing the encryption; if the identity is i, then $hdr_i = \{i\}$. *Sometimes the encryption function is denoted by* $\mathsf{E}_{ek_i}(m_i; r)$ *to explicitly show by a string* r*, the random coins used in the encryption process.*

Decryption (D). Given an encrypted aggregate c and its header, $hdr \subseteq ID$ (which indicates the source nodes included in the aggregation), $\mathsf{D}_{dk}(hdr, c) \rightarrow m/\perp$ is a deterministic algorithm that takes the decryption key dk, hdr, and c as inputs and returns the plaintext aggregate m or possibly $\perp$ if c is an invalid ciphertext.

Aggregation (Agg). With a specified aggregation function f such as additive aggregation considered in this article, $\mathsf{Agg}_f(hdr_i, hdr_j, c_i, c_j) \rightarrow (hdr_l, c_l)$ aggregates two encrypted aggregates, c_i and c_j, with headers hdr_i and hdr_j respectively (where $hdr_i \cap hdr_j = \phi$), to create a combined aggregate c_l, and a new header $hdr_l = hdr_i \cup hdr_j$. Suppose c_i and c_j are the ciphertexts for plaintext aggregates m_i and m_j respectively. The output c_l is the ciphertext for the aggregate $f(m_i, m_j)$, namely, $\mathsf{D}_{dk}(hdr_l, c_l) \rightarrow f(m_i, m_j)$. This article considers $f(m_i + m_j) = m_i + m_j \bmod M$. *Note that the aggregation algorithm*

does not need the decryption key dk or any of the encryption keys ek_i as input; it is a public algorithm.

It is intentional to include the description of the header hdr in the security model to make it as general as possible (to cover schemes requiring headers in their operations). hdr is needed in some schemes to identify the set of decryption keys required to decrypt a certain ciphertext. Nonetheless, generating headers or including headers as input to algorithms should not be treated as a requirement in the actual construction or implementation of CDA algorithms. For constructions that do not need headers, all hdr's can simply be treated as the empty set ϕ in the security model.

The Notion of Semantic Security. Only one type of oracle query (adversary interaction with the system) is allowed in the security model, namely the encryption oracle $\mathcal{O}_E$. The details are as follows:

Encryption Oracle $\mathcal{O}_E(i, m, r)$. For fixed encryption and decryption keys, on input of an encryption query $\langle i, m, r\rangle$, the encryption oracle retrieves s_i's encryption key ek_i, runs the encryption algorithm on m, and replies with the ciphertext $\mathsf{E}_{ek_i}(m)$ and its header, hdr. The random coins, or nonce r, are part of the query input to $\mathcal{O}_E$.

The encryption oracle is needed in the security model since the encryption algorithm uses private keys.

To define security (more specifically, indistinguishability) against chosen plaintext attacks (IND-CPA), we use the following game played between a challenger and an adversary, assuming there is a set U of n source nodes. If no PPT adversary, even in collusion with at most t compromised nodes, can win the game with non-negligible advantage (as defined in the following), we say the CDA scheme is t-secure. The adversary is allowed to freely choose parameters n and t.

Definition 3. A CDA scheme is t-secure (indistinguishable) against adaptive chosen plaintext attacks if the advantage of winning the following game is negligible in the security parameter λ for all PPT adversaries.

Collusion Choice. The adversary chooses to corrupt t source nodes. Denote the set of these t corrupted nodes and the set of their identities by S' and I' respectively.

Setup. The challenger runs the key generation algorithm KG to generate a decryption key dk and n encryption keys $\{ek_i : 1 \leq i \leq n\}$, and gives the subset of t encryption keys $\{ek_j : s_j \in S'\}$ to the adversary but keeps the decryption key dk and the other $(n - t)$ encryption keys $\{ek_j : s_j \in U \backslash S'\}$.

Query 1. The adversary can issue to the challenger one type of query:

Encryption Query $\langle i_j, m_j, r_j\rangle$. The challenger responds with $\mathsf{E}_{e_{i_j}}(m_j)$ using random coins r_j. The adversary is allowed to choose and submit his choices of random coins for encryption queries.

Challenge. Once the adversary decides that the first query phase is over, it selects a subset S, of d source nodes (whose identities are in the set I), such that $|S \backslash S'| > 0$, and outputs two different sets of plaintexts $M_0 = \{m_{0k} : k \in I\}$ and $M_1 = \{m_{1k} : k \in I\}$ to be challenged. The only constraint is that the two resulting plaintext aggregates, x_0 and x_1, are not equal, where $x_0 = f(\ldots, m_{0k}, \ldots)$ and $x_1 = f(\ldots, m_{1k}, \ldots)$.

The challenger flips a coin $b \in \{0, 1\}$, to select between x_0 and x_1. The challenger then encrypts each $m_{bk} \in M_b$ with ek_k, and aggregates the resulting ciphertexts in the set $\{\mathsf{E}_{ek_k}(m_{bk}) : k \in I\}$ to form the ciphertext C, of the aggregate, that is, $C = \mathsf{E}_{\{ek_k : k \in I\}}(x_b)$, and gives C to the adversary. The challenger chooses and passes the nonce to the adversary. The global random coins should be chosen to be different from those used in the Query 1 phase, and no query on them should be allowed in the Query 2 phase.

Query 2. The adversary is allowed to make more queries as previously done in the Query 1 phase.

Guess. Finally, the adversary outputs a guess, $b' \in \{0, 1\}$, for b.

Result. The adversary wins the game if $b' = b$. The advantage of the adversary is defined as: $Adv_{\mathcal{A}} = |Pr[b' = b] - \frac{1}{2}|$.

Note that in CDA, what the adversary is interested in is the information about the final aggregate. Consequently, in the preceding game, the adversary is asked to distinguish between the ciphertexts of two different aggregates, x_0 and x_1, as the challenge, rather than to distinguish the two sets of plaintexts, M_0 and M_1. Allowing the adversary to choose the two sets, M_0, M_1, is to give him more flexibility in launching attacks.

APPENDIX B: PROOF OF THEOREM 1

PROOF. For the sake of clarity, we first prove the security of a version without using the hash function, h. Then we show why the proof also works for the hashed version. The reduction is based on the indistinguishability property of a PRF which is stated as follows:

Indistinguishability Property of a PRF. Assume f is taken from a PRF. Then for a fixed input argument, x, and an unknown, randomly picked key, K, the following two distributions are computationally indistinguishable provided that polynomially many evaluations of $f_K(\cdot)$ have been queried:

$$\{y = f_K(x) : y\}, \{y \leftarrow \{0, 1\}^{\lambda} : y\}.$$

That is, the output $f_K(x)$ is computationally indistinguishable from a randomly picked number from $\{0, 1\}^{\lambda}$ to any PPT distinguisher who has knowledge of the input argument x and a set of polynomially many 2-tuples $(x_i, f_K(x_i))$ where $x_i \neq x$. More formally, for any PPT distinguisher $\mathcal{D}$,

$$|Pr[y = f_K(x) : \mathcal{D}(x, y) = 1] - Pr[y \leftarrow \{0, 1\}^{\lambda} : \mathcal{D}(x, y) = 1]| < \varepsilon(\lambda),$$

where $\varepsilon(\lambda)$ is a negligible function in λ.

PROOF FOR THE NON-HASHED SCHEME. Without loss of generality, we prove the security of a modified version of the construction in which each encryption key is uniformly picked from $\{0, 1\}^\lambda$, compared with keys generated by a PRF in the actual scheme. We then provide a justification for why the inference applies to the actual implementation.

Suppose there exists a PPT adversary, D, which can break the semantic security of the scheme with non-negligible advantage, Adv_D^{CMT}. We show in the following how D can be used to construct an algorithm, D', which can distinguish the above distributions with non-negligible advantage. Assume the key K in question is unknown to D'.

Algorithm D'

Setup. Allow the adversary D to choose any $n-1$ sources to corrupt. Randomly pick $n-1$ encryption keys $ek_i \in_R \{0, 1\}^\lambda$ and pass them to the adversary. Assume node n is uncorrupted. The encryption key for node n is taken to be K, the key of the PRF with which D' is being challenged.

Query. Upon receiving an encryption query $\langle i_j, m_j, r_j \rangle$ with nonce r_j, return $c_j = (f_{ek_{i_j}}(r_j) + m_j) \bmod M$ if $i_j \neq n$. Otherwise, pass r_j to query the PRF to get back $f_K(r_j)$ and reply with $c_j = (f_K(r_j) + m_j) \bmod M$.

Challenge. In the challenge phase, receive from D, two sets of plaintext messages $M_0 = \{m_{01}, m_{02}, \ldots, m_{0n}\}$ and $M_1 = \{m_{11}, m_{12}, \ldots, m_{1n}\}$.

Randomly pick a number w and output it to the PRF challenger to ask for a challenge. Note w is the nonce used for CDA encryption in the challenge for D. The PRF challenger flips a coin $b \in \{0, 1\}$ and returns t_b, which is $f_K(w)$ when $b = 0$ and randomly picked from $\{0, 1\}^\lambda$ when $b = 1$. These two cases correspond to the two distributions previously discussed.

Randomly flip a coin, $d \in \{0, 1\}$, and return the challenge ciphertext c_d to D, where $c_d = \sum_{i=1}^{n} m_{di} + \sum_{i=1}^{n-1} f_{ek_i}(w) + t_b$.

Guess. D returns its guess, b'. Return b'', which is 0 when $b' = d$ and 1 otherwise.

Obviously, if D is PPT, then D' is also PPT. Denoting the expression $\sum_{i=1}^{n} m_{di} + \sum_{i=1}^{n-1} f_{ek_i}(w)$ by X_d, the challenge passed to D can be expressed as $c_d = X_d + t_b$. When $b = 0$, $t_b = f_K(w)$; when $b = 1$, t_b is a randomly picked number from $\{0, 1\}^\lambda$. In the following discussion, we denote the output of D on input c_d by $D(c_d)$. The probability of success for D' to distinguish between $f_K(w)$ and a random number is:

$$\begin{aligned} Pr_{D'}^{PRF}[Success] &= Pr[b'' = b] \\ &= \frac{1}{2}\{Pr[b'' = 0|b = 0] + Pr[b'' = 1|b = 1]\} \\ &= \frac{1}{4}\{Pr[b'' = 0|b = 0, d = 0] + Pr[b'' = 0|b = 0, d = 1] \\ &\quad + Pr[b'' = 1|b = 1, d = 0] + Pr[b'' = 1|b = 1, d = 1]\} \end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{4}\{Pr[D(t_0 + X_0) = 0] + Pr[D(t_0 + X_1) = 1] \\
&\quad + Pr[D(t_1 + X_0) = 0] + Pr[D(t_1 + X_1) = 1]\} \\
&= \frac{1}{4}\{Pr[D(t_0 + X_0) = 0] + Pr[D(t_0 + X_1) = 1] \\
&\quad + 1 - Pr[D(t_1 + X_0) = 1] + Pr[D(t_1 + X_1) = 1]\} \\
&= \frac{1}{4}\{2Pr_D^{CMT}[Success] + 1 - (Pr[D(t_1 + X_0) = 1] \\
&\quad - Pr[D(t_1 + X_1) = 1])\}.
\end{aligned}
$$

Note that $t_0 + X_0$ and $t_0 + X_1$ are valid ciphertexts for the two challenge plaintext sets M_0 and M_1 respectively. In the last step, we make use of the fact that the probability of success for D to break the semantic security of the scheme is given by:

$$Pr_D^{CMT}[Success] = \frac{1}{2}Pr[D(t_0 + X_0) = 0] + \frac{1}{2}Pr[D(t_0 + X_1) = 1].$$

Rearranging terms, we have

$$
\begin{aligned}
&4Pr_{D'}^{PRF}[Success] + Pr[D(t_1 + X_0) = 1] - Pr[D(t_1 + X_1) = 1] \\
&\quad = 2Pr_D^{CMT}[Success] + 1
\end{aligned}
$$

$$
\begin{aligned}
&4\left(Pr_{D'}^{PRF}[Success] - \frac{1}{2}\right) + Pr[D(t_1 + X_0) = 1] - Pr[D(t_1 + X_1) = 1] \\
&\quad = 2\left(Pr_D^{CMT}[Success] - \frac{1}{2}\right).
\end{aligned}
$$

Taking the absolute value on both sides and substituting $Adv_{D'}^{PRF} = |Pr_{D'}^{PRF}[Success] - \frac{1}{2}|$ and $Adv_D^{CMT} = |Pr_D^{CMT}[Success] - \frac{1}{2}|$, we have

$$2Adv_{D'}^{PRF} + \frac{1}{2}\left|Pr[D(t_1 + X_0) = 1] - Pr[D(t_1 + X_1) = 1]\right| \geq Adv_D^{CMT}.$$

Since t_1 is a randomly picked number, $\{t_1 + X_0\}$ and $\{t_1 + X_1\}$ are identically distributed. That is, for any PPT algorithm D, $Pr[D(t_1 + X_0) = 1] = Pr[D(t_1 + X_1) = 1]$. Hence,

$$2Adv_{D'}^{PRF}(\lambda) \geq Adv_D^{CMT}(\lambda).$$

Note also that:

$$
\begin{aligned}
|Pr[x \leftarrow \{0,1\}^\lambda; y = f_K(x) : D'(y) = 1] - Pr[y \leftarrow \{0,1\}^\lambda : D'(y) = 1]| \\
> 2Adv_{D'}^{PRF}(\lambda).
\end{aligned}
$$

If Adv_D^{CMT} is non-negligible in λ, then so is $Adv_{D'}^{PRF}$. As a result, if D can break the semantic security of the scheme with non-negligible advantage, D' could distinguish between the output of PRF f and a random number. Equivalently, $|Pr[x \leftarrow \{0,1\}^\lambda; y = f_K(x) : D'(y) = 1] - Pr[y \leftarrow \{0,1\}^\lambda : D'(y) = 1]|$ is non-negligible (a contradiction to the indistinguishability property of a PRF).

This security argument applies to the actual implementation since the view of the adversary, D, in the simulation is in essence the same as that in the

actual scheme. For each one of the $n-1$ corrupted nodes, the encryption key is $f_{K'}(i)(1 \leq i \leq n-1)$ for some randomly picked master key K'. By the property of the PRF, $f_{K'}(i)$ is indistinguishable from a randomly picked key (as used in this simulation game) for all PPT distinguisher algorithms. For the uncorrupted node, its output for encryption is now $f_{f_{K'}(n)}(x)$ instead of $f_K(x)$ (with randomly picked K) as used in the above simulation game. It can be shown by a contrapositive argument that, for fixed n, the two distributions are computationally indistinguishable, that is,

$$\left\{K' \leftarrow \{0,1\}^{\lambda}; x \leftarrow \{0,1\}^{\lambda} : f_{f_{K'}(n)}(x)\right\} \stackrel{c}{\equiv} \left\{K \leftarrow \{0,1\}^{\lambda}; x \leftarrow \{0,1\}^{\lambda} : f_K(x)\right\}.$$

The argument is as follows: Assume f is a PRF. That is, $A = \{K' \leftarrow \{0,1\}^{\lambda} : f_{K'}(n)\}$ is indistinguishable from $B = \{K \leftarrow \{0,1\}^{\lambda} : K\}$ for all PPT distinguishers. If there exists a PPT distinguisher, D, which can distinguish between $X = \{K' \leftarrow \{0,1\}^{\lambda}; x \leftarrow \{0,1\}^{\lambda} : f_{f_{K'}(n)}(x)\}$ and $Y = \{K \leftarrow \{0,1\}^{\lambda}; x \leftarrow \{0,1\}^{\lambda} : f_K(x)\}$, we can use D to distinguish between A and B. The idea is that when we receive a challenge, s, which could be from A or B, we send $f_s(x)$ as a challenge for D. If s belongs to A, $f_s(x)$ belongs to X, and if s belongs to B, $f_s(x)$ belongs to Y. We could thus distinguish X from Y (a contradiction).

Security of the Hashed Version. Only a few modifications to this security proof are needed in order to prove the security of the hashed variant.

First, in the algorithm D', all ciphertexts are now generated using the hashed values of the PRF outputs or replies from the challenger of D'. With such changes, we now denote the expression $\sum_{i=1}^{n} m_{di} + \sum_{i=1}^{n-1} h(f_{ek_i}(w))$ by X_d. Of course, the modulus size would be l instead of λ.

Second, the challenge passed to D would be: $c_d = X_d + h(t_b)$. Then the derivation for the advantage expressions is essentially the same as that for the non-hashed scheme.

Third, the security proof of the non-hashed scheme relies on the fact that $\{t_1 \leftarrow \{0,1\}^{\lambda} : t_1 + X_0\}$ and $\{t_1 \leftarrow \{0,1\}^{\lambda} : t_1 + X_1\}$ are identical distribution. On the contrary, to prove the security of the hashed scheme, we need the following distributions to be identical:

$$\{t_1 \leftarrow \{0,1\}^{\lambda} : h(t_1) + X_0\}, \{t_1 \leftarrow \{0,1\}^{\lambda} : h(t_1) + X_1\}.$$

If h fulfills the requirement, then $\{t_1 \leftarrow \{0,1\}^{\lambda} : h(t_1)\}$ is the uniform distribution over $\{0,1\}^{l}$. Consequently, the two distributions are identical. This thus concludes the proof that the hashed scheme is semantically secure. □

REFERENCES

BELLARE, M., CANETTI, R., AND KRAWCZYK, H. 1996. Keying hash functions for message authentication. In *Proceedings of Advances in Cryptology (CRYPTO'96)*. Lecture Notes in Computer Science, vol. 1109, Springer. 1–15.

BONEH, D., GENTRY, C., LYNN, B., AND SHACHAM, H. 2003. Aggregate and verifiably encrypted signatures from bilinear maps. In *Proceedings of Advances in Cryptology (EUROCRYPT'03)*. Lecture Notes in Computer Science, vol. 2656, 416–432.

BUTTYÁN, L., SCHAFFER, P., AND VAJDA, I. 2006. RANBAR: RANSAC-based resilient aggregation in sensor networks. In *Proceedings of the 4th ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'06)*. 83–90.

CASTELLUCCIA, C., MYKLETUN, E., AND TSUDIK, G. 2005. Efficient aggregation of encrypted data in wireless sensor networks. In *Proceedings of MobiQuitous*. 1–9.

CASTELLUCCIA, C. AND SORIENTE, C. 2008. ABBA: Secure aggregation in WSNS - a bins and balls approach. In *Proceedings of the 6th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*.

CHAN, A. C.-F. AND CASTELLUCCIA, C. 2007. On the privacy of concealed data aggregation. In *Proceedings of ESORICS*. Lecture Notes in Computer Science, vol. 4734, 390–405.

CHAN, A. C.-F. AND CASTELLUCCIA, C. 2008. On the (im)possibility of aggregate message authentication codes. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*.

CHAN, H., PERRIG, A., AND SONG, D. 2006. Secure hierarchical in-network aggregation in sensor networks. In *Proceedings of the ACM Conference on Computer and Communication Security (CCS'06)*. 278–287.

ESCHENAUER, L. AND GLIGOR, V. D. 2000. A key management scheme for distributed sensor networks. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*. 41–47.

GIRAO, J., WESTHOFF, D., AND SCHNEIDER, M. 2004. CDA: Concealed data aggregation in wireless sensor networks. In *Proceedings of the ACM Conference on Web Information Systems (WiSe)*.

GOLDREICH, O. 2001. *Foundations of Cryptography: Part 1*. Cambridge University Press.

GOLDREICH, O., GOLDWASSER, S., AND MICALI, S. 1986. How to construct random functions. *J. ACM 33*, 4, 792–807.

GOLDWASSER, S. AND MICALI, S. 1984. Probabilistic encryption. *J. Comput. Syst. Sci. 28*, 2, 270–299.

GOLDWASSER, S., MICALI, S., AND RIVEST, R. 1988. A secure signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput. 17*, 2, 281–308.

HU, L. AND EVANS, D. 2003. Secure aggregation for wireless networks. *Workshop on Security and Assurance in Ad hoc Networks*. http://www.cs.virginia.edu/evans/pubs/wsaan.ps

IWATA, T. AND KUROSAWA, K. 2003. OMAC: One-key CBC MAC. In *Proceedings of Fast Software Encryption (FSE'03)*. Lecture Notes in Computer Science, vol. 2887, 129–153.

KARLOF, C., SASTRY, N., AND WAGNER, D. 2004. Tinysec: a link layer security architecture for wireless sensor networks. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys)*. 162–175.

KARLOF, C. AND WAGNER, D. 2003. Secure routing in wireless sensor networks: Attacks and countermeasures. In *Proceedings of the IEEE Workshop on Sensor Network Protocols and Applications*.

KATZ, J. AND YUNG, M. 2006. Characterization of security notions for probabilistic private-key encryption. *J. Cryptology 19*, 1, 67–95.

MADDEN, S. R., FRANKLIN, M. J., HELLERSTEIN, J. M., AND HONG, W. 2002. TAG: a Tiny AGgregation service for ad-hoc sensor networks. In *Proceedings of the 5th Annual Symposium on Operating Systems Design and Implementation*. 131–146.

NAOR, M., REINGOLD, O., AND ROSEN, A. 2002. Pseudorandom functions and factoring. *SIAM J. Comput. 31*, 5, 1383–1404.

NAOR, M. AND YUNG, M. 1990. Public-key cryptosystems provably secure against chosen-ciphertext attacks. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*. 427–437.

NIST. 2001. Advanced encryption standard. National Institute of Standards and Technology. FIPS PUB 197.

PERRIG, A., STANKOVIC, J., AND WAGNER, D. 2004. Security in wireless sensor networks. *Commun. ACM 47*, 53–57.

PERRIG, A., SZEWCZYK, R., WEN, V., CULLER, D., AND TYGAR, D. 2001. SPINS: Security protocols for sensor networks. In *Proceedings of the ACM Conference on Mobile Computing and Networking (MOBICOM)*. 189–199.

PRZYDATEK, B., SONG, D., AND PERRIG, A. 2003. SIA: Secure information aggregation in sensor networks. In *Proceedings of the ACM Conference on Embedded Networks in Sensor Systems (SENSYS)*. 255–265.

RIVEST, R. L. 1995. The RC5 encryption algorithm. *Dr. Dobb's J. 1008*.

RIVEST, R. L., SHAMIR, A., AND ADLEMAN, L. M. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM 21*, 120–126.

VERNAM, G. S. 1926. Cipher printing telegraph systems for secret wire and radio telegraphic communications. *J. Amer. Inst. Elect. Eng. 45*, 105–115.

WAGNER, D. 2004. Resilient aggregation in sensor networks. In *Proceedings of the ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN)*.

WESTHOFF, D., GIRAO, J., AND ACHARYA, M. 2006. Concealed data aggregation for reverse multicast traffic in sensor networks: Encryption, key distribution, and routing adaption. *IEEE Trans. Mobile Comput. 5*, 10, 1417–1431.

WOOD, A. D. AND STANKOVIC, J. A. 2002. Denial of service in sensor networks. *IEEE Comput. 35*, 54–62.

YANG, Y., WANG, X., ZHU, S., AND CAO, G. 2006. SDAP: A secure hop-by-hop data aggregation protocol for sensor networks. In *Proceedings of the ACM Internation Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*.

ZHU, S., SETIA, S., JAJODIA, S., AND NING, P. 2004. An interleaved hop-by-hop authentication scheme for filtering false data in sensor networks. In *Proceedings of the IEEE Symposium on Security and Privacy*.

Received June 2007; revised February 2008, July 2008; accepted September 2008