A Reliable and Secure Distributed In-Network Data Storage Scheme in Wireless Sensor Networks

Muhammad Bashir Abdullahi, Guojun Wang*, and Felix Musau
School of Information Science and Engineering
Central South University
Changsha, Hunan Province, P. R. China, 410083
*Correspondence to: csgjwang@mail.csu.edu.cn

***Abstract*—In a battlefield surveillance scenario, data readings and events emerging from a wireless sensor network deployed that may not be used immediately by or simply impossible to transmit to an authorized user (a Soldier) in real time are stored in the network. Without proper protection for the sensitive data generated in this setting, a compromised storage node (by an enemy soldier) may divulge its stored sensitive data about the monitored environment, and even worse, it may alter the data. In this paper, we integrate an elliptic curve cryptography scheme and an erasure coding scheme to provide reliable and secure distributed in-network data storage for sensor networks. The main idea is to distribute each erasure coded fragment appended with a fingerprint to different storage nodes. The fingerprint is to allow each coded data fragment to be independently verified as a valid and correct subset of a specific data item. So, the scheme achieves localization of data error. The proposed scheme is resilient to collusion attack, pollution attack, and data dropping attack, and guarantees forward and backward data secrecy as well. The security of the proposed scheme relies on the intractability of the elliptic curve discrete logarithm problem. Different from the existing solutions, the uniqueness of our method comes from the use of lightweight encryption scheme, which is well suited for resource constrained wireless sensors.**

Keywords-*Wireless sensor network, distributed in-network storage, resiliency, reliability, security, energy consumption*.

I. INTRODUCTION

A. Background and Motivation

Wireless sensor networks (WSNs) could be widely deployed for military and civilian applications, such as battlefield surveillance, environment and habitat monitoring, healthcare applications, home automation, and traffic control [1]. A WSN consists of a large number of spatially distributed, autonomous and resource-constrained sensing devices that cooperatively provide a useful interface to the physical world with data acquisition and processing capabilities. Data generated from a sensor network that may not be used immediately by or simply impossible to transmit to an authorized user in real time is stored in the network, either centrally or in a distributed manner for later analysis and querying purposes. The distributed in-network data storage and query scheme has been identified as an effective architecture for sensor data management [2], [3]. Although it is motivated by situations where the data generated is not accessed in real time, it facilitates not only energy savings, but also long-term data storage. We consider an in-network storage architecture where, beside regular sensor nodes, some set of nodes are selected as storage nodes. These storage nodes are responsible for storing the data generated by regular sensor nodes and at regular time interval off-load to an authorized external sink node or a trusted external entity when it is available (i.e., connected to the network). We adopt the dispersal protocol proposed in ref.[7]. However, some application areas (e.g. military battlefield surveillance and health care monitoring scenarios) impose severe privacy and security requirements on data transmissions, data processing and data management. Therefore, the role played by storage nodes in such sensor networks make them vulnerable to various security attacks; especially that they off-load their stored data at a pre-determined interval of time. The adversary can easily compromise some storage nodes within a particular time interval. A compromised storage node may divulge its stored sensitive data to the adversary, return spurious or incomplete data readings for a query, exhibit Byzantine failures, and maliciously drop important data items. These attacks are detrimental to the integrity, confidentiality, and availability of data and consequently prevent the authorized users from retrieving the original data correctly.

Therefore, in order to provide the reliability and security of the sensitive data throughout its lifetime, we employ a distributed in-network storage scheme in which a sensor node splits its collected data into slices (also known as data shadows or data fragments) and stores each slice at different storage nodes in a sensor network. This makes the data more secure against Byzantine failures and unauthorized access. In other words, this means that no single storage node should be allowed to store a complete data set or observe all query results. Consequently, any forged or incomplete data item should be detected by the authorized data collector as soon as possible. Also, any compromised malicious storage node that exhibits Byzantine behaviors should be detected and isolated.

Intuitively, data reliability and integrity are salient, indispensable ingredients of dependable and secure data storage architecture, but realizing them in a lightweight manner for resource poor sensor network requires further optimization. The previous solutions to this problem for distributed in-network data storage in sensor networks were proposed in ref.[4] and ref.[5]. The solutions have some drawbacks: the amount of redundancy is high and subsequent increase in energy consumption and memory, algebraic signatures used

978-0-7695-4600-1/11 $26.00 © 2011 IEEE
DOI 10.1109/TrustCom.2011.71

for data integrity checking are not cryptographically secure because it is easy to construct two strings that have the same algebraic signature, and their schemes also require two symmetric encryptions and decryptions and two sets of polynomial evaluations, which consumes more energy. The solution in ref.[6] did achieve lower energy consumption but only a read-only adversary scenario is considered. Thus, the scheme in ref.[6] did not address integrity checking scheme. In our case, we consider both read-only and read-write adversary scenarios.

B. Our Contributions

In this paper, we present a reliable and secure distributed in-network data storage scheme for resource-constrained sensor networks based on the combination of an elliptic curve cryptography scheme and an erasure coding scheme. The goal is to ensure energy-efficient data reliability and security. The efficiency of the scheme is evaluated in terms of computation cost, communication cost, and memory overhead.

We integrate an elliptic curve based stateful public-key encryption (PKE) scheme [8] and an authenticated encryption mode - offset codebook (OCB) [9] to provide not only the confidentiality, integrity and authentication, but also the forward and backward secrecy of data confidentiality at lower energy consumption and memory overhead. By forward and backward secrecy, we mean, even if the secret key of the source node is disclosed, the confidentiality of the data before and after the disclosure are not affected. Reducing the computation cost of the PKE scheme for energy constrained sensor networks is of particular importance. Thus, our choice in combining the two schemes was based on their energy-efficiency gain in maintaining state and one pass of the block cipher. The elliptic curve based stateful PKE scheme is used only to generate key for encryption and decryption. It allows a sender to maintain a "state" that is re-used across different encryptions. Thus,it reduces the number of computations required for discrete logarithm based public-key encryptions. The scheme needs just one point multiplication each for encryption and decryption of the key [8], thereby reducing the computational overhead compared to "stateless" schemes. On the other hand, the OCB is a block cipher mode of operation that provides both data confidentiality and authentication in only one pass over the plaintext, and it avoids ciphertext expansion [9]. In contrast, previous solutions use an encryption mode that requires two passes over the plaintext (one for authentication and one for encryption), which as a result, increases the energy consumption.

To achieve data link reliability, bandwidth saving and memory space per storage node, we use an erasure coding scheme (e.g., Reed-Solomon code - RS code [10]), which involves encoding a data into a set of redundant fragments that guarantee the original data recovery against pollution attack. To provide probabilistic high reliability that a storage node maintains and responds with a valid coded data fragment that corresponds to the same data item during the reconstruction, we propose a simple and efficient data fragment integrity and consistency verification scheme. This allows each erasure coded data fragment to be verified independently as a complete, unaltered and valid copy that corresponds to a specific data item. For each erasure coded data fragment, a fingerprint (or verification metadata) is generated before distribution. An erasure coded data fragment and its corresponding verification metadata are stored on the same storage node. The security of the scheme relies on the one-way property of a hash function.

In brief, the contributions of this paper are threefold: first, we use OCB mode of encryption instead of CBC-encryption and CBC-MAC for a symmetric encryption scheme, not only for the cost of OCB that is about 54% of the cost of CBC encryption combined with CBC MAC, but also for the key used for OCB that is a single block-cipher key, and all block-cipher invocations are keyed by this one key, saving memory space and key-setup time [11]. Second, the amount of traffic (or redundancy) is minimal and consequently lower memory overhead per storage node. We use only (n, l)RS instead of both (n, l) secret sharing (SS) scheme and (n, l)RS as used in the existing approaches. Third, a simple and lightweight data fragment integrity and consistency checking scheme is presented to ensure partial data integrity and consistency verifications independently during the reconstruction phase.

II. Related Work

A hybrid encryption is a scheme that uses a public-key algorithm to establish and encrypt a key that is then used to encrypt an actual message using a symmetric-key algorithm [12]. The public-key algorithm used is called a key encapsulation mechanism (KEM) and the symmetric-key algorithm used is called a data encapsulation mechanism (DEM) [12]. Cramer and Shoup have shown that this hybrid cryptosystem is secure against chosen ciphertext attack (CCA-secure) if both KEM and DEM are CCA-secure [12].

One of the public-key cryptosystems that are based on the hybrid encryption is the Diffie-Hellman integrated encryption scheme (DHIES) proposed by Abdalla et al [13]. The elliptic curve version is ECIES. ECIES has been proven to be semantically secure against adaptive chosen-ciphertext attacks under the assumptions that (i) the encryption scheme used is secure, (ii) the message authentication code algorithm is secure, and (iii) certain non-standard variants of the Diffie-Hellman problem are intractable [13].

The advantage of ECIES over the Massey-Omura and El-Gamal public-key methods is that it provides the capability of encrypting arbitrary bit strings without increasing the number of group operations for encryption and decryption, and without increasing the key sizes, while the other methods require a message to be a group element. Moreover, they are not secure against chosen-ciphertext attack [13],[14].

Bellare *et al*.[8] presented two schemes on how to significantly speedup the PKE by simply allowing a sender to maintain a "state" (i.e. maintaining the random ephemeral secret key value and its corresponding public key value as state, so that both need not to be computed each time.) that is re-used across different encryptions. The schemes reduce the number of computations required for discrete logarithm

based public-key encryptions. One of the schemes is a stateful version of DHIES (or StDH). Baek *et al*.[15] further reduced the communication overhead in their implementation of the stateful PKE in WSNs by introducing a technique known as "indexing". The indexing technique replaces the repeated part of the ciphertext, which is usually long, by a short string.

Wang *et al*.[4] and Ren *et al*.[5] independently proposed a secure and dependable distributed in-network storage scheme for WSNs. The schemes are based on the principles of perfect secret sharing (SS) and Reed-Solomon (RS) coding schemes to achieve reliability and fault-tolerant data storage in WSNs. The schemes employ SS and RS to encode the generated random session key and the sensed data into the required shares, respectively. To ensure distributed data integrity checking, the schemes use algebraic signatures technique [19], whereby the source node generates a distinct parity based on all data shares and appends the parity to each data share. Once a shareholder initiates an integrity verification protocol, other shareholders can also act as a verifier to validate the integrity of the stored data shares as long as they have the corresponding parities. The validity of integrity of a data share can be ascertained only if its signature is combined with a sufficient number of signatures of other corresponding data shares. The schemes assumed that every node in the network generates data and at the same time stores its neighbors' data.

Similarly, Ren *et al*.[6] considers the same system, in which an optimized data distribution scheme is proposed. The authors assumed that each node has a probability vector that shows the security level of its neighbors set. Thus, neighbors with lower probability of being compromised than the threshold value are selected to store its generated data. The scheme considers only a read-only adversary scenario. However, the scheme achieves both forward and backward secrecy of data confidentiality. In contrast, we consider a system where the data generated at a sensor node are hashed to different locations in the network for storage. This is to ensure efficient query by directing it to the location of the storage nodes rather than flooding the entire network. We consider scenarios where an adversary is either read-only, read-write or both. And we achieve both forward and backward secrecy.

III. Models and Assumptions

A. System Model

We consider a wireless sensor network consisting of a mobile sink (*MS*) and a densely deployed sensor nodes. For ease of description, we denote data generating sensor nodes by *r-nodes* and data storage nodes by *s-nodes*. Each sensor node has a unique identity and knows its geographic coordinates. We assume that each sensor node generates data readings; each associated to a data type, and at the end of a certain time interval sends the generated data to the temporary repository *s-nodes*. The generated data stored at the *s-nodes* are collected by the *MS* when it is available (i.e., connected to the network). We assume that each data generated can be represented as a matrix of equal sized data vectors, whose symbols are elements from a Galois field. We assume that all sensor nodes are loosely synchronized with *MS*. With loosely synchronization, we mean the entire network lifetime is divided into fixed time intervals known as *epochs*. Thus, every sensor node is aware of the beginning and the end of an epoch.

Each data generated in an epoch by a *r-node* is managed by a set of *s-nodes* called a storage domain (i.e. *s-domain*), which may support any number of *r-nodes*. The set of *s-nodes* that comprise this s-domain are selected by means of a hash operation applied on the corresponding data type to a particular sensed data, which returns a pair of geographic coordinate in the sensor field [16]. We adopt the dispersal protocol in ref.[7], which first identifies a *home node* (i.e., the sensor node that is closer to the coordinate $(x, y) = h(\textit{data type})$), and then stores the replicas of the data in a ball of sensors centered in the *home node* (i.e. in the set of sensors within a given distance from the home node) rather than in the home perimeter as proposed in ref.[16]. We assume that each *r-node* encrypts its generated data readings using a *MS*'s public key (or alternatively, by the group public key of authorized users). To store and retrieve an encrypted data we employ **put**(*data type*, *D*, *Q*) and **get**(*data type*) primitives, respectively [7],[17]; where *D* denotes the data and *Q* the required number of *s-node*.

B. Trust Requirements

Since *MS* serves as an interface between a sensor network and outside of the network, compromising it can render the entire network useless. For this reason, we assume that *MS* is always trustworthy and sufficiently powerful to defend itself against any security threats.

C. Attack Model

We assume that the adversary roams around the network and randomly compromises one or more different sensor nodes within a certain time interval; especially between successive visits of the *MS*. We envisaged two categories of security attacks:

- *Passive attack*: this is by eavesdropping incoming and outgoing messages on currently compromised nodes in an attempt to learn about the sensed data. This is a read-only attack.
- *Active attack*: If an *s-node* is compromised and under the full control of an adversary, the adversary can access and even modify the data stored on that *s-node* and also instruct the *s-node* to drop data, return forge or incomplete data in response to queries from *MS*. Also, some compromised *s-nodes* may collude and pull their shares together in order to reconstruct the origiinal data. This is a read-write attack.

Therefore, our goals are to ensure data reliability and security for in-network sensor data storage under the above mentioned attacks, and to achieve localization of data error and dependability in a lightweight manner.

D. Preliminary

Erasure Coding Scheme: An erasure code takes l data symbols and makes n encoded symbols, out of which any

l encoded symbols are enough to recover the original l data symbols, where n and l are integers. An example of such a code is Reed-Solomon (RS) code [10]. RS code is a code defined over GF(2^q). The codes may be obtained by letting a message $M = (m_1, m_2, .., m_n)$ denotes the coefficient of a polynomial $P(x) = \sum_{j=0}^{l-1} m_j x^j$ and letting the encoding of M be $C(M) = (c_1,...,c_n)$, where $c_i = P(\alpha_i)$ and $\alpha_i,...,\alpha_n$ are n distinct elements of GF(2^q). A RS code is specified as (n, l)RS with s-bits symbols. This means that, the *encoder* is a probabilistic algorithm that takes l data symbols of s-bits each and adds parity symbols to make an n symbols codeword. Thus, there are $n - l$ parity symbols of s-bits each. While a *decoder* is a deterministic algorithm that takes any l symbols codeword to reconstruct the original l data symbols. This means that, it can correct up to e symbols that contain errors in a codeword, where $2e = n - l$. If a *r-node* wants to send an l-digit plaintext data D, using RS code, it will send $n = l + 2s$ digits, and be well guaranteed that the correct data can be reconstructed at the other end, if there are fewer than s corrupted digits.

IV. Reliable and Secure In-network Storage

The proposed scheme integrates ECIES and the erasure coding scheme to provide data reliability and security for distributed in-network sensor data storage. It is divided into basic and enhanced schemes. To simplify the description of our scheme, we describe it in terms of data items between a *r-node*, a set of *s-nodes* of an *s-domain* and a *MS*.

A. Basic Scheme Description

The basic scheme contains the following phases:

Setup phase: The *MS* selects an elliptic curve E defined over a $GF(p)$ of size p, where p is a large prime number and a system base point G ($G \neq O$, where O is a point at infinity) of order q, where q is also a large prime number on that curve. The *MS* then chooses a message authentication code function, *MAC*, a CCA-secure symmetric block cipher encryption scheme, *Encrypt* and *Decrypt*, a key derivation function, H, of length η and E domain parameters T = (p, a, b, G, q, h) [18]. Then, the *MS* generates at random $d \in_r$GF(2^q) and computes $Q = dG$. The *MS* then makes all the parameters (E, G, q, Q) public and conceals d as its private key.

Data Encryption phase: Each *r-node u*, firstly downloads *MS*'s public key (E, G, q, Q), then chooses a random integer $w \in_r$GF(2^q) to compute $X = wG$ as its ephemeral public key and $Y = wQ$ as an implied shared secret value with the *MS*. Then it computes key $K = H(X\|Q\|Y)$and splits it into two by applying hash function H repeatedly on K to obtain keys of length λ_j as $\lfloor k_1 \| k_2 \rfloor$ (i.e. k_1 is followed by k_2, where k_1 is a key for *Encrypt* and k_2 is a key for *MAC*); where $k_i = H(K\|i\|Y)$ for $i = 1, .., n$ and $n = \lceil \lambda/\eta \rceil$. The plaintext data D_u is encrypted as $M = Encrypt(k_1, D_u)$ and its tag $\tau = MAC(k_2, M)$ and returned $C_u = \{X\|M\|\tau\}$ as ciphertext [18].

Data Fragment Generation phase: Each *r-node u*, uses the (n, l)RS code to encode a ciphertext C_u into n data fragments denoted as $C_u = \{c_{u,1}c_{u,2}, ..., c_{u,n}\}$ For each coded data fragment, it also generates verification metadata to be used during the reconstruction phase for integrity and consistency check. Detailed description is in section V.

Data Distribution phase: Each *r-node u* randomly selects n *s-nodes* without replacement from an *s-domain* using the dispersal protocol in ref.[7]. For each *s-node v* (where v = 1,..,n), u employs **put** primitive in ref.[7] to distribute for storage the fragment $F_{u,v} = Encrypt(k_{uv}, (u\|f_{id}\|c_{u,i}\|V_i))$, Where is a pairwise secret key shared with an *s-node v*. Then it erases the original C_u and the verification metadata from its memory.

Data Reconstruction phase: To reconstruct data, *MS* collects any l shares from l honest *s-nodes* in an *s-domain*, using **get** primitive as described in ref.[7] and then reconstructs C_u based on (n, l)RS code. Assuming $l = \lfloor n \times \delta \rfloor$, where δ is a security parameter $(e.g.\ 0.5 \sim 0.8)$ [5]. The larger the value of δ, the higher the percentage of shares required for reconstruction.

Observation: It is interesting to note that an *s-node v* might be malicious and unreliable due to the hostile environment. It might deliberately alter or respond with a spurious or different (that corresponds to another data item) data fragment in its storage so as to hamper data reconstruction process. However, tolerating random Byzantine failures also requires coping with data consistency. Therefore, it is insufficient to detect that a Byzantine behavior has occurred after reconstructing to a wrong value. Secondly, it will be difficult at that time, to actually detect the *s-node*(s) that misbehaved. Thus, it is a matter of importance to guard against this ill behavior prior to reconstruction. The benefits are enormous: (i) it saves time for re-reconstruction; as it is only the first l valid shares that will be collected and validated, (ii) it helps to detect unreliable *s-node*(s), i.e., ensures localization of data error(s). We present a simple and lightweight scheme that will enable a *MS* to perform integrity and consistency check for each received coded fragment; independently, before reconstruction.

Data Decryption phase: The *MS* first computes the secret value $Y = dX$ using its secret key d. Then it computes key $K = H(X\|Q\|Y)$ and splits it into $\lfloor k_1 \| k_2 \rfloor$ as explained earlier. It then computes tag $\tau = MAC(k_2, M)$, if it is not equal to τ, *MS* stops and rejects the ciphertext. Otherwise, it continues and then decrypts $D_u = Decrypt(k_1, M)$ as a plaintext.

Discussion: The basic scheme is indistinguishably secure against chosen-ciphertext attacks (IND-CCA) as it offers both integrity/authentication and confidentiality services [12]. However, it is relatively inefficient in our setting; especially, where a *r-node* encrypts, splits and distributes all the data it generated in an epoch. A *r-node* is a constrained device that lacks sufficient amount of energy to carry out all the operations for a significantly long period of network lifetime. The encrypt-then-MAC construction using CBC-encryption and CBC-MAC employed in the basic scheme require a *r-node* to process a long message and a long ciphertext separately thereby doubles the amount of computation and consequently the energy consumption. In addition, there is a tendency of message expansion in the basic scheme. Therefore, we present in the next section an efficient scheme that achieves the same

level of security at lower energy consumption.

B. Enhanced Scheme

To achieve lower energy consumption, we employ two techniques in KEM-part and DEM-part of the hybrid encryption scheme, respectively. We employ a modified Diffie-Hellman based stateful public key encryption (PKE) scheme in WSNs[15] and OCB, a block cipher mode of operation that offers both authentication and confidentiality in only one pass of a block cipher [9]. Combining these two schemes will reduce the cost of encryption and decryption to half [8].

Recall that, the original stateful PKE [8] is a scheme that allows a sender to maintain a state, which is re-used across different encryptions thereby reducing the computation overhead, optimizes only the KEM-part of the hybrid cryptosystem. This scheme is further improved by Baek et al. [15] to save the communication overhead. The long ephemeral public key, which must be transmitted for a number of different sessions, is replaced with a much shorter string using an indexing method. The index is a combination of a unique identity of a sender and a sequence number for the current epoch. Ideally, in the beginning (i.e., indexing phase), both the sender's ephemeral public key and the index will be sent to the receiver for identity correlation and caching (for further details see [15]). At least that adds little communication overhead. Subsequently, only the index will be sent (i.e., normal phase) to the receiver, as described below.

In our enhanced scheme, we will describe only the phases that have been optimized.

Data Encryption phase: Each *r-node u*, firstly downloads *MS*'s public key (*E, G, q, Q*), then chooses a random integer $w \in_r \mathrm{GF}(2^q)$ to compute $X = wG$ as its ephemeral public key and $Y = wQ$ as an implied shared secret value with the *MS*. Then it chooses an index $I_X = (id_X \| t)$ for its ephemeral public key, X, in such a way that uniquely identifies it [15], and computes key $K = H(I_X \| X \| Q \| Y)$. The plaintext data D_u is encrypted now, using OCB mode of encryption, by firstly choosing a non-repeating nonce $N_u \in \{0,1\}^*$, which can be a monotonically increasing counter and then computes $[M, \tau]$ = *ocbEncrypt*(K, N_u, A, D_u). The OCB [9] takes as input the key, the nonce, the associated data and the plaintext data to generate the ciphertext core and then, simultaneously, generates a tag using the plaintext, the generated ciphertext and the associated data. It returns, deterministically, either a ciphertext $C_u = \{I_X \| M \| \tau\}$ or the distinguished value *Invalid*, $\perp$ [9].

Data Decryption phase: The *MS* firstly searches its database for X that corresponds to I_X; if it does not exist, *MS* stops and rejects the ciphertext; otherwise, it continues to compute the secret value $Y = dX$ using its secret key d. Then it computes key $K = H(I_X \| X \| Q \| Y)$. Finally, it then computes D_u = *ocbDecrypt*(K, N_u, A, C_u) if it returns a different tag τ, *MS* stops and rejects the ciphertext. Otherwise, it accepts D_u as a plaintext.

Discussion: In the proposed schemes the forward and backward secrecy of a compromised *s-node* are guaranteed. The adversary gets no secret knowledge that will enable it to decrypt data, even if it compromises one or more *s-nodes*. Besides, each time a *r-node u* wants to send data generated D_u to *s-node v* for storage, it uses the *MS*'s public key to control the encryption procedure, and the *MS* uses its key pair to control the decryption operation [18]. Thus, it is hard for an adversary who does not possess *MS*'s secret key to recover plaintext D_u from its ciphertext C_u.

V. Lightweight Partial Data Integrity Checking

Indeed, *s-nodes* are often not trusted in terms of reliability and security for data-sensitive sensor network applications, due to hostile and unattended nature of the environment in which they are deployed. Intuitively, tolerating Byzantine failures also requires coping with consistency of the partial data. It is insufficient to detect that a Byzantine behavior has taken place after reconstructing to a wrong value without detecting the misbehaving *s-node*. Thus, there is need to provide a probabilistic high reliability for the integrity and consistency of the stored data fragment before and during data reconstruction, and at the same time detects unreliable *s-node*(*s*).

A. Fingerprint Generation and Distribution

In this section, we describe the process of generating and distributing a verification metadata in our scheme.

To generate a verification metadata (or fingerprint) for each data fragment in a lightweight manner, we reuse the effort expended in generating the key for the encryption. Thus, a *r-node u* first splits K into n keys as explained in section IV-A (i.e. $k_1, k_2, .., k_n$). Let the n erasure coded fragment be $c_{u,1}, c_{u,2}, ..., c_{u,n}$. We use $f_{id} = H_K(c_{u,1} \| c_{u,2} \| ... \| c_{u,n})$ as the identifier for the set of coded fragment belonging to the same data item, where $H_K(\bullet)$ is a keyed cryptographic hash function. The *r-node u* then generates the metadata $V_i = z_i \oplus c_{u,i} \bmod p$, where $z_i = H_K(t \| f_{id} \| k_i \| i)$, t is the epoch round and $i = 1...n$ is the key index that corresponds to the number of coded fragments. At the end of the data distribution phase a *r-node u* stores each erasure coded fragment with its corresponding verification metadata on a distinct randomly selected *s-nodes*. The *r-node u* then deletes from its memory the original data, verification metadata and the coded fragments. This is shown in detail in Algorithm 1.

Algorithm 1: Fingerprint Generation and Distribution

1: **Input:** $t \leftarrow epoch$, $C_u \leftarrow c_{u,1}, c_{u,2}, ..., c_{u,n}$, $K = H(X \| Q \| Y)$
2: **Output:** V_i and $F_{u,v} = Encrypt(k_{uv}, (u \| f_{id} \| c_{u,i} \| V_i))$
3: Compute $k_i = H(K \| i \| Y)$ for $i = 1, .., n$ and $n = \lceil \lambda / \eta \rceil$
4: Compute $f_{id} = H_K(c_{u,1} \| c_{u,2} \| ... \| c_{u,n})$
5: **for** ($i \leftarrow 1$ **to** n) **do**
6: Compute $z_i = H_K(t \| f_{id} \| k_i \| i)$
7: Compute $V_i = z_i \oplus c_{u,i} \bmod p$
8: $u \rightarrow v : F_{u,v} = Encrypt(k_{uv}, (u \| f_{id} \| c_{u,i} \| V_i))$
9: **end for**
10:**end**

B. Verification of Partial Data Integrity and Consistency

In Prior to reconstruction of C_u based on (n, l) RS code, an *MS* performs integrity and consistency check through voting on each coded fragment received to ascertain that it is a

Algorithm 2: Verification of Partial Data Integrity and Consistency

1: **Input:** $t \leftarrow epoch$, $v \rightarrow MS : (f_{id} \| c_{u,i} \| V_i)$
2: **Output:** "Valid"or "Invalid"
3: *MS* generate its key pair *K*= $H(X\|Q\|Y)$;
4: Compute $k_i = H(K\|i\|Y)$ for $i = 1, .., n$ and $n = \lceil \lambda/\eta \rceil$
5: **for** $(i \leftarrow 1$ **to** $m)$ **do** // where $m \leq n$
6: Compute $z'_i = H_K(t\|f_{id}\|k_i\|i)$
7: Compute $V'_i = z_i \oplus c_{u,i} \bmod p$
8: **if** $V'_i \neq V_i$ **then**
9 **return** "Invalid"and *reject* $c_{u,i}$ for reconstruction
10: **else**
11: **return** "Valid"and *accept* $c_{u,i}$ for reconstruction
12: **end if**
13:**end for**
14:**end**

valid copy and at the same time corresponds to the data item requested. It first generates its key pair for decryption as explained earlier. Then it splits *K* into *n* keys (i.e. k_1, k_2,..., k_n). For each coded fragment received it computes $z'_i = H_K(t\|f_{id}\|k_i\|i)$ and generates $V'_i = z_i \oplus c_{u,i} \bmod p$ and compares if $V'_i \neq V_i$ then it is invalid and *rejects* it. Otherwise it *accepts* the fragment as valid and belongs to the requested data item. This shows that the integrity and consistency check is correct and secure. The *s-node v* should not pass verification check unless it is in possession of a *complete* and unaltered version of its data share. If the coded fragment received from an *s-node v* return invalid after verification, then it is evident that *s-node v* is misbehaving. Consequently, necessary steps can be taken to isolate it from the network. This is shown in detail in Algorithm 2.

The attractiveness of this verification scheme relies on the fact that: (i) no *s-node v* can generate a valid metadata and pass verification without being detected; (ii) the amount of communication required between an *s-node* and the *MS* is minimal and (iii) it is efficient in terms of computation - hash and exclusive-or operations. The security of the scheme relies on the existence of a collision-resistant hash function and the difficulty in the derivation of the key that controls both the encryption and decryption processes due to elliptic curve discrete logarithm problem.

VI. Security and Performance Analysis

In this section, we evaluate the security of our scheme in terms of collusion attack, eavesdropping attack, data dropping attack, and pollution attack and then its performance analysis.

A. Security Analysis

Collusion attack: In this attack, some compromised *s-nodes* illegally conspire by pulling together their shares in order to reconstruct or extract the original data. This is possible if they have access to the decryption key. However, it is impossible to reconstruct the original data even though more than $(n - l)$ compromised *s-nodes*, storing coded data fragments of the same data item, D_u, collude together or even at worst with a *r-node u* that initially encrypted the data. It is hard to derive the key for decryption due to the elliptic curve discrete logarithm problem. Because the original data was encrypted using *MS*'s public key, it can be decrypted by only *MS*'s corresponding private key. Moreover, a *r-node u* has erased the ciphertext from its memory. Thus, colluding attacks by the compromised *s-nodes* cannot succeed. Besides, their collusion cannot afford them any secret knowledge that will enable them to decrypt stored data at any time. This assures a perfect forward and backward secrecy of data confidentiality.

Eavesdropping attack: Here the adversary snoops on the transmitted messages as they travel to the storage areas to learn the contents. A *r-node u* encrypts and splits all data into fragments, which individually has no useful information before multicast. Therefore, the adversary learns nothing from the fragments.

Pollution attack: Here some compromised *s-nodes* maliciously modified their data shares so as to hamper reconstruction. They will succeed if they passed verification undetected by constructing a valid fingerprint. However, it is hard to derive the key for encryption and decryption due to the elliptic curve logarithm problem. Besides, *MS* can reconstruct the whole data from any *l* or more data fragments of the *l* honest *s-nodes* of an *s-domain* due to properties of the (*n*, *l*)RS coding scheme that can correct up to *e* symbols that contain errors in a codeword, where $2e = n - l$. Thus, even though $(n - l)$ or less dishonest *s-nodes* alter their data fragments or misbehave in a Byzantine way.

Data dropping attack: Here some compromised *s-nodes* dropped their data shares so as to hamper reconstruction. The attack will succeed if $l + 1$ *s-nodes* holding data shares corresponding to the same data item dropped their shares. Otherwise, *MS* can reconstruct the whole data from any *l* or more data fragments of the *l* honest *s-nodes* of an *s-domain* due to properties of the (*n*, *l*)RS coding scheme.

B. Performance Analysis

Here we analyze the computation cost, the communication cost and the storage cost during data encryption, data encoding, data distribution, fingerprint generation and verification. Then we detailed performance evaluation results obtained using MATLAB and also compare the efficiency with the existing schemes [4], [5] and [6].

Computation Cost: At each epoch, a *r-node u* uses *StDH* scheme to only generate key $K = H(I_X\|X\|Q\|Y)$ used for encryption. To achieve this, it performs one point multiplication $Y = wQ$ and one hash operation. Then it uses *OCB mode* to encrypt the data. It performs two Elliptic Curve based encryptions: $F_{u,v} = Encrypt(k_{uv}, (\bullet))$ and $[M, \tau] = ocbEncrypt(\bullet)$. A *r-node* uses (*n*, *l*)RS code to encode $C_u = \{I_X\|M\|\tau\}$ into *n* data fragments. To generate a verification metadata, $V_i = z_i \oplus c_{u,i} \bmod p$, it performs $k_i = H(\bullet)$, $f_{id} = H_K(\bullet)$ and $z_i = H_K(\bullet)$ hash operations, *n* XOR and *n* modulo reduction operations. Following the example of analysis in [4] and [6], let α, β, γ and μ denote the size of $(I_X\|X\|Q\|Y)$, $(c_{u,1}\|c_{u,2}\|...\|c_{u,n})$, $(K\|i\|Y)$ and $(t\|f_{id}\|k_i\|i)$, respectively. The total computation cost at the *r-node u* is $PtMul^1$ + $Hash^1_\alpha$ + $ECCEncrypt^2$ + $RSCoding^1$ + $Hash^1_\beta$ + $Hash^n_\gamma$ + $Hash^n_\mu$ + Xor^n + Mod^n, where $PtMul^1$ denotes one point multiplication, $Hash^1_\alpha$ denotes one hash operation with input size of α,

TABLE I
SUMMARY OF COMPUTATION COST

Entity	Total Computation Cost
r-node	$PtMul^1 + Hash^1_\alpha + ECCEncrypt^2 + RSCoding^1 + Hash^1_\beta + Hash^n_\gamma + Hash^n_\mu + Xor^n + Mod^n$
s-node	$ECCDecrypt^1$

TABLE II
SUMMARY OF COMMUNICATION AND STORAGE COSTS

Entity	Communication Cost	Storage Cost
r-node	$n * \left(\frac{\pi}{n} + 4\right) * q$	$2 * q$
s-node	$\left(\frac{\pi}{n} + 2\right) * q$	$\left(\frac{\pi}{n} + 4\right) * q$

TABLE III
COMPARISON OF COMMUNICATION AND STORAGE COSTS

Scheme	Communication Cost	Storage Cost
Ours	$n * \left(\frac{\pi}{n} + 4\right) * q$	$\left(\frac{\pi}{n} + 6\right) * q$
Ref.[4], [5]	$n * \left(\frac{\pi}{n} + \omega + 5\right) * q$	$\left(\frac{\pi}{n} + \omega + 5\right) * q$
Ref. [6]	$n * \left(\frac{\pi}{n} + 2\right) * q$	$\left(\frac{\pi}{n} + nb_i + 2\right) * q$

$ECCEncrypt^2$ denotes two Elliptic Curve based encryptions, $RSCoding^1$ denotes one (n, l)RS code operation, Xor^n denotes n XOR operations, and Mod^n denotes n modulo reductions. However, when a *r-node u* keeps (w, X) as its internal state information that it re-uses across different encryptions; it need not to compute $Y = wQ$ each time.

Thus, it has reduced the cost of point multiplication. The computation cost at an *s-node v* is only one Elliptic Curve based decryption operation. The total computation cost at an *s-node v* is $ECCDecrypt^1$. The summary is shown in Table I.

Communication Cost: Let's assume that all symbols used are elements of a Galois Field GF(2^q), where q = 8 or 16. After a *r-node u* uses (n, l)RS code to encode C_u of size π, into n fragments; each data fragment will be $\frac{\pi}{n}$ symbols. It then distributes $Encrypt(k_{uv}, (u \| f_{id} \| c_{u,i} \| V_i))$ to n *s-nodes* in an *s-domain* as data shares. Thus, the communication cost at the *r-node u* is approximately $n * \left(\frac{\pi}{n} + 4\right) * q$ bits. During integrity verification, at least $m \leq n$ *s-nodes* (m denotes the first l honest *s-nodes*) send $(f_{id} \| c_{u,i} \| V_i)$ to *MS*. Therefore, the communication cost at an *s-node v* is approximately $\left(\frac{\pi}{n} + 2\right) * q$ bits. The summary is shown in Table II.

Storage Cost: Each *r-node u* requires $2 * q$ bits storage cost to keep (w, X) as internal state information. Each *s-node v* requires $\left(\frac{\pi}{n} + 4\right) * q$ bits storage cost to keep the data share. The summary is shown in Table II.

we compare the efficiency of our proposed enhanced scheme with the existing schemes [4], [5] and [6]. The existing schemes [4] and [5] suffer from high energy consumption and high redundancy bits. In our scheme, we employ OCB, as it provides secrecy and authentication in only one pass of block cipher and with only one key. Furthermore, the ciphertext has the same length of the plaintext. Also as it only requires a nonce, not an Initialization Vector (IV), and there is a distinct difference. A nonce does not drop security if an adversary can guess the next one, but an IV can cause this problem.

TABLE IV
COMPARISON OF SECURITY PROPERTIES

Property	Ref.[4], [5]	Ref.[6]	Ours
Public Key Cryptography	No	No	Yes
Symmetric Key Cryptography	Yes	Yes	No
Integrity Check	Yes	No	Yes
Forward Secrecy	Partial	Yes	Yes
Backward Secrecy	Probabilistic	Enhanced Probabilistic	Perfect

The existing schemes [4], [5], [6] and our basic scheme provide the same security guarantee, but use an encryption scheme that requires two passes of the block cipher: one pass achieves secrecy with CBC-encryption, and another pass achieves authenticity with CBC-MAC. Thus, it doubles the amount of computation, and consequently doubles the amount of energy consumption. Besides, the cost of OCB is about 54% of the cost of CBC encryption combined with CBC MAC.

During data distribution, the existing schemes [4] and [5] use two sets of encoding techniques to generate data fragments: one set uses the (n, l)RS code to partition original data of l-symbols into n-symbols data fragments with $(n - l)$ redundant symbols and another set uses the (n, m)SS scheme to obtain n shares of the random session key (RSK). The (n, m)SS scheme requires that the size of any share be at least the same size as the original secret. Hence, the amount of redundancy bits used doubles and consequently consumes more energy in terms of computation and communication and subsequently more storage space at the storage nodes. Table III summarizes the cost of communication and storage during data distribution among the existing schemes and the proposed scheme. Let ω be the size of the RSK and nb_i be the number of neighbors to node i. Thus, from the analysis it shows that the proposed scheme is more efficient compare to the existing schemes.

Fig.1 shows the comparison of communication overhead during data distribution to the storage nodes. The following parameters are used: n = 5, 10, 15, 20, 25, 30, 35, 40; π = 1024, q = 8; $|RSK| = \omega$ = 64bits and $|nbi|$ = 20. As we can see, the cost is proportional to the size of the packet distributed. The existing schemes [4] and [5] consumed more energy as huge amount of data with multiple redundancy bits are transmitted. The scheme in ref. [6] consumed relatively less energy than our scheme as no information on data integrity check is transmitted.

Fig. 2 shows the comparison of storage cost during data distribution. The trends are the same; meaning that the more the number of storage nodes the less the size of share per storage node. The existing schemes [4] and [5] consumed more storage space as huge amount of data with multiple redundancy bits are stored. The increase in the cost of storage in ref. [6] is as a result of keeping probability vectors of its neighbors. As we can see our scheme consumed less storage space.

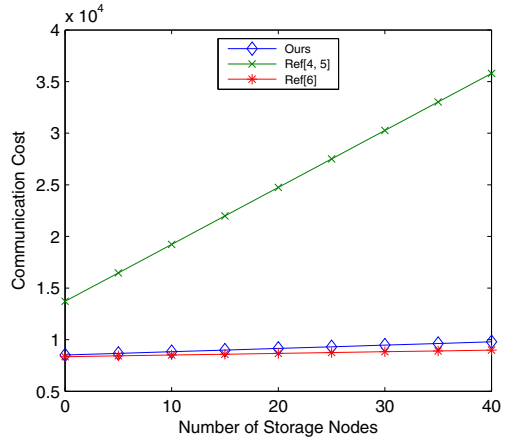

Fig. 1. Comparison of Communication Cost During Data Distribution

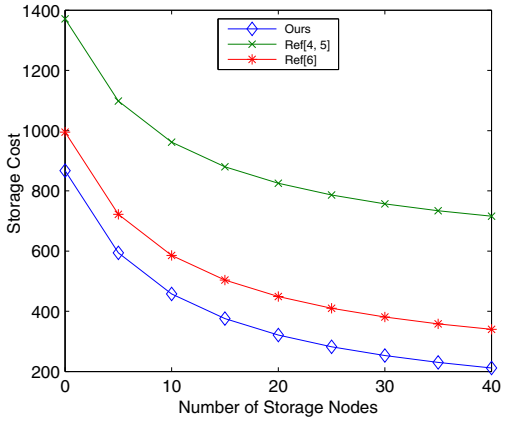

Fig. 2. Comparison of Storage Cost During Data Distribution

VII. Conclusion

In this paper, we presented a scheme for reliable and secure distributed in-network data storage in wireless sensor networks. We integrate an elliptic curve based stateful PKE scheme and an authenticated encryption mode, offset codebook (OCB) and the Reed-Solomon codes to achieve data reliability and security with low energy consumption, and low memory overhead. To provide probabilistic high reliability of partial data integrity and consistency during data reconstruction, we further proposed a simple and lightweight partial data integrity and consistency checking scheme. Each data share is verified independently thereby help to localize data error efficiently. The security of the proposed integrity checking scheme relies on the one-way property of a hash function and the difficulty in the derivation of the key that controls both the encryption and decryption processes due to the elliptic curve discrete logarithm problem. Our schemes achieved perfect forward and backward secrecy. Extensive security and performance analysis showed that the proposed scheme is secure and energy efficient compared to the existing schemes.

In the future, we plan to work on a trust-based data distribution scheme and an efficient integrity-preserving scheme that will address how to repair a corrupted or deleted data share from the existing uncorrupted ones without involving the source node.

Acknowledgments

This work is supported by National Natural Science Foundation of China under grant number 61073037 and 61103035, and Hunan Provincial Science and Technology Program under grant numbers 2010GK2003 and 2010GK3005.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey", *Computer Networks: Elsevier Science* 38(4), Mar. 2002, pp. 393-422.

[2] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin, "Data-centric storage in sensornets", *ACM SIGCOMM Computer Communications Review* 33(1), Jan. 2003, pp. 137-142.

[3] Y. Diao, D. Ganesan, G. Mathur, and P. Shenoy, "Rethinking data management for storage-centric sensor networks", *Proc. of the Third Biennial CIDR 2007*, Jan. 2007, pp. 22-32.

[4] Q. Wang, K. Ren, W. Lou, and Y. Zhang, "Dependable and secure sensor data storage with dynamic integrity assurance", in *Proc. of the IEEE INFOCOM 2009*, Rio de Janeiro, Brazil, April 2009, pp. 954-962.

[5] W. Ren, Y. Ren, and H. Zhang, "Secure, dependable and publicly verifiable distributed data storage in unattended wireless sensor networks", *Science China Information Sciences* 53(5), April 2010, pp. 964-979.

[6] Y. Ren, V. Oleshchuk, and F. Y. Li, "A scheme for secure and reliable distributed data storage in unattended WSNs", In *Proc. GLOBECOM* 2010, 1-6.

[7] M. Albano, S. Chessa, F. Nidito, and S. Pelagatti, "Q-NIGHT: adding qos to data centric storage in non-uniform sensor networks", In *Proc. MDM* 2007. pp. 166-173.

[8] M. Bellare, T. Kohno, and V. Shoup, "Stateful public-key cryptosystems: how to encrypt with one 160-bit exponentiation", *Proc. of ACM CCS* 2006, pp. 380-389, 2006.

[9] Krovetz T. and Rogaway, P. "The software performance of authenticated-encryption mode", *Fast Software Encryption - FSE 2011*, LNCS, Springer (2011)

[10] S. Reed, and G. Solomon, "Polynomial codes over certain finite fields", *Journal of the SIAM* 8(2), (1960), pp. 300-304.

[11] P. Rogaway, M. Bellare, and J. Black, "OCB: a block-Cipher mode of operation for efficient authenticated encryption", *Proc. of ACM TISSEC*, 6 (3), pp. 365-403, Aug. 2003.

[12] R. Cramer, and V. Shoup, "Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack", *SIAM Journal of Computing*, 33(1), pp. 167-226, 2003.

[13] M. Abdalla, M. Bellare and P. Rogaway, "The oracle Diffie-Hellman assumptions and an analysis of DHIES", *Topics in Cryptography - CT-RSA 2001, LNCS* vol. 2020, pp. 143-158, Springer-Verlag, 2001.

[14] L. C. Washington, *Elliptic Curves: Number Theory and Cryptography, 2nd edition*. Chapman and Hall/CRC, Taylor and Francis Group, LLC, 2008.

[15] J. Baek, H. C. Tan, J. Zhou, and J. W. Wong, "Realizing stateful public key encryption in wireless sensor network", *Proc. of the IFIP TC 11 23rd IFIP-SEC 2008*, pp. 95-108, Springer-Verlag, 2008.

[16] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu, "Data-centric storage in sensornets with GHT a geographic hash table", *Mobile Networks and Applications*, 8(4),(2003), pp. 427-442.

[17] M. Albano, and S. Chessa, "Distributed erasure coding in data centric storage for wireless sensor networks", in *Proc. ISCC 2009*, pp. 22-27.

[18] D. R. L. Brown, *Standards for Efficient Cryptography 1(SEC-1)*, Certicom Research. http://www.secg.org/secg_docs.htm.

[19] W. Litwin and T. Schwarz, "Algebraic signature for scalable distributed data structure", in *Proc. of the 20th ICDE 2004*, pp. 412-423.