A Lightweight Secure Scheme for Detecting Provenance Forgery and Packet Drop Attacks in Wireless Sensor Networks

Salmin Sultana,Gabriel Ghinita, *Member, IEEE,* Elisa Bertino, *Fellow, IEEE,* and Mohamed Shehab, *Member, IEEE*

Abstract—Large-scale sensor networks are deployed in numerous application domains, and the data they collect are used in decision-making for critical infrastructures. Data are streamed from multiple sources through intermediate processing nodes that aggregate information. A malicious adversary may introduce additional nodes in the network or compromise existing ones. Therefore, assuring high data trustworthiness is crucial for correct decision-making. Data provenance represents a key factor in evaluating the trustworthiness of sensor data. Provenance management for sensor networks introduces several challenging requirements, such as low energy and bandwidth consumption, efficient storage and secure transmission. In this paper, we propose a novel lightweight scheme to securely transmit provenance for sensor data. The proposed technique relies on *in-packet Bloom filters* to encode provenance. We introduce efficient mechanisms for provenance verification and reconstruction at the base station. In addition, we extend the secure provenance scheme with functionality to detect *packet drop attacks* staged by malicious data forwarding nodes. We evaluate the proposed technique both analytically and empirically, and the results prove the effectiveness and efficiency of the lightweight secure provenance scheme in detecting packet forgery and loss attacks.

Index Terms—Provenance, Security, Sensor Networks

1 Introduction

Sensor networks are used in numerous application domains, such as cyberphysical infrastructure systems, environmental monitoring, power grids, etc. Data are produced at a large number of sensor node sources and processed in-network at intermediate hops on their way to a Base Station (BS) that performs decision-making. The diversity of data sources creates the need to assure the trustworthiness of data, such that only trustworthy information is considered in the decision process. Data provenance is an effective method to assess data trustworthiness, since it summarizes the history of ownership and the actions performed on the data. Recent research [1] highlighted the key contribution of provenance in systems where the use of untrustworthy data may lead to catastrophic failures (e.g., SCADA systems). Although provenance modeling, collection, and querying have been studied extensively for workflows and curated databases [2], [3], provenance in sensor networks has not been properly addressed. We investigate the problem of secure and efficient provenance transmission and processing for sensor networks, and we use provenance to detect packet loss attacks staged by malicious sensor nodes.

- *S. Sultana is with the Dept. of Electrical and Computer Engineering, Purdue University. E-mail: ssultana@purdue.edu*
- *G. Ghinita is with the Dept. of Computer Science, University of Massachusetts, Boston. Email: Gabriel.Ghinita@umb.edu.*
- *E. Bertino is with the Dept. of Computer Sciences, Purdue University. Email: bertino@purdue.edu.*
- *M. Shehab is with the Dept. of Software and Information Systems, University of North Carolina at Charlotte. E-mail: mshehab@uncc.edu.*

In a multi-hop sensor network, *data provenance* allows the BS to trace the source and forwarding path of an individual data packet. Provenance must be recorded for each packet, but important challenges arise due to the tight storage, energy and bandwidth constraints of sensor nodes. Therefore, it is necessary to devise a light-weight provenance solution with low overhead. Furthermore, sensors often operate in an untrusted environment, where they may be subject to attacks. Hence, it is necessary to address security requirements such as *confidentiality*, *integrity* and *freshness* of provenance. Our goal is to design a *provenance encoding and decoding mechanism* that satisfies such security and performance needs. We propose a provenance encoding strategy whereby each node on the path of a data packet securely embeds provenance information within a *Bloom filter* that is transmitted along with the data. Upon receiving the packet, the BS extracts and verifies the provenance information. We also devise an extension of the provenance encoding scheme that allows the BS to detect if a *packet drop attack* was staged by a malicious node.

As opposed to existing research that employs separate transmission channels for data and provenance [4], we only require a single channel for both. Furthermore, traditional provenance security solutions use intensively cryptography and digital signatures [5], and they employ append-based data structures to store provenance, leading to prohibitive costs. In contrast, we use only fast Message Authentication Code (MAC) schemes and *Bloom filters (BF)*, which are fixed-size data structures that compactly represent provenance. Bloom filters make efficient usage of bandwidth, and they yield low error rates in practice. Our specific contributions are:

- We formulate the problem of secure provenance transmission in sensor networks, and identify the challenges specific to this context;
- We propose an in-packet Bloom filter provenance-encoding scheme;
- We design efficient techniques for provenance decoding and verification at the base station;
- We extend the secure provenance encoding scheme and devise a mechanism that detects *packet drop attacks* staged by malicious forwarding sensor nodes;
- We perform a detailed security analysis and performance evaluation of the proposed provenance encoding scheme and packet loss detection mechanism.

The rest of the paper is organized as follows: Section 2 sets the problem background and describes the system, threat and security models. Section 3 introduces the provenance encoding scheme, whereas Section 4 outlines the scheme extension and the mechanism for identification of malicious nodes that stage packet drop attacks. Section 5 presents the security analysis of our methods. Section 6 provides an analytical performance evaluation, whereas Section 7 presents the experimental evaluation results for the proposed scheme. We survey related work in Section 8 and conclude with directions for future research in Section 9.

2 Background and System Model

In this section, we introduce the network, data and provenance models used. We also present the threat model and security requirements. Finally, we provide a brief primer on Bloom filters, their fundamental properties and operations.

2.1 Network Model

We consider a multihop wireless sensor network, consisting of a number of sensor nodes and a base station (BS) that collects data from the network. The network is modeled as a graph $G(N, L)$, where $N = \{n_i |, 1 \leq i \leq |N|\}$ is the set of nodes, and L is the set of links, containing an element $l_{i,j}$ for each pair of nodes n_i and n_j that are communicating directly with each other. Sensor nodes are stationary after deployment, but routing paths may change over time, e.g., due to node failure. Each node reports its neighboring (i.e. one hop) node information to the BS after deployment. The BS assigns each node a unique identifier $nodeID$ and a symmetric cryptographic key K_i. In addition, a set of hash functions $H = \{h_1, h_2, ..., h_k\}$ are broadcast to the nodes for use during provenance embedding.

2.2 Data Model

We assume a multiple-round process of data collection. Each sensor generates data periodically, and individual values are aggregated towards the BS using any existing hierarchical (i.e., tree-based) dissemination scheme [6]. A data path of D hops is represented as $< n_l, n_1, n_2, ..., n_D >$, where n_l is a leaf node representing the data source, and node n_i is i hops away from n_l. Each non-leaf node in the path aggregates the received data and provenance with its own locally-generated data and provenance.

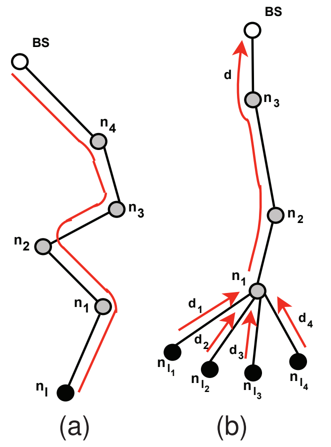


Fig. 1. Provenance graph for a sensor network.

Each data packet contains (i) a unique packet sequence number, (ii) a data value, and (iii) provenance. The sequence number is attached to the packet by the data source, and all nodes use the same sequence number for a given round [7]. The sequence number integrity is ensured through MACs, as discussed in Section 3.

2.3 Provenance Model

We consider *node-level* provenance, which encodes the nodes at each step of data processing. This representation has been used in previous research for trust management [1] and for detecting selective forwarding attacks [8]. Given packet d, its provenance is modeled as a directed acyclic graph $G(V, E)$ where each vertex $v \in V$ is attributed to a specific node $HOST(v) = n$ and represents the provenance record (i.e. nodeID) for that node. Each vertex in the provenance graph is uniquely identified by a vertex ID (VID) which is generated by the host node using cryptographic hash functions. The edge set E consists of directed edges that connect sensor nodes.

Definition 1 (Provenance): Given a data packet d, the provenance p_d is a directed acyclic graph $G(V, E)$ satisfying the following properties: (1) p_d is a subgraph of the sensor network $G(N, L)$; (2) for $v_i, v_j \in V$, v_i is a child of v_j if and only if HOST $(v_i) = n_i$ participated in the distributed calculation of d and/or forwarded the data to HOST (v_j) $=n_j$; (3) for a set $U = \{v_i\} \subset V$ and $v_j \in V$, U is a set of children of v_j if and only if HOST (v_j) collects processed/forwarded data from each HOST$(v_i \in U)$ to generate the aggregated result.

Figure 1 shows two provenance examples. In Figure 1(a), the leaf node n_l generates a data packet d, and each intermediate node aggregates its own sensory data with d and then forwards it towards the BS. Hence, the provenance corresponding to d is $< v_l, v_1, v_2, v_3 >$, which can be represented as a simple path. In Figure 1(b), the internal node n_1 generates the data d by aggregating data d_1 , ..., d_4 from n_{l_1} , ..., n_{l_4} and then passes d towards the BS. Here, n_1 is an aggregator and the aggregated provenance $< \{v_{l_1}, v_{l_2}, v_{l_3}, v_{l_4}\}, v_1, v_2, v_3 >$ is represented as a tree.

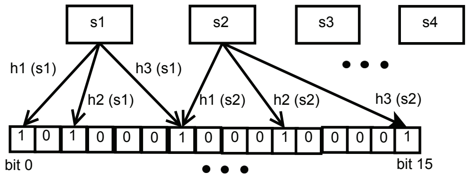


Fig. 2. A Bloom filter with $n = 4$, $m = 16$ and $k = 3$.

2.4 Threat Model and Security Objectives

We assume that the BS is trusted, but any other arbitrary node may be malicious. An adversary can eavesdrop and perform traffic analysis anywhere on the path. In addition, the adversary is able to deploy a few malicious nodes, as well as compromise a few legitimate nodes by capturing them and physically overwriting their memory. If an adversary compromises a node, it can extract all key materials, data, and codes stored on that node. The adversary may drop, inject or alter packets on the links that are under its control. We do not consider denial of service attacks such as the complete removal of provenance, since a data packet with no provenance records will make the data highly suspicious [5] and hence generate an alarm at the BS. Instead, the primary concern is that an attacker attempts to misrepresent the data provenance. Our objective is to achieve the following security properties:

- *Confidentiality*: An adversary cannot gain any knowledge about data provenance by analyzing the contents of a packet. Only authorized parties (e.g., the BS) can process and check the integrity of provenance.
- *Integrity*: An adversary, acting alone or colluding with others, cannot add or remove non-colluding nodes from the provenance of benign data (i.e. data generated by benign nodes) without being detected.
- *Freshness*: An adversary cannot replay captured data and provenance without being detected by the BS.

It is also important to provide *Data-Provenance Binding*, i.e., a coupling between data and provenance so that an attacker cannot successfully drop or alter the legitimate data while retaining the provenance, or swap the provenance of two packets. Although this problem is orthogonal to the method we propose, we address it in Section 3.3.

2.5 The Bloom Filter (BF)

The BF is a space-efficient data structure for probabilistic representation of a set of items $S = \{s_1, s_2, ..., s_n\}$ using an array of m bits with k independent hash functions $h_1, h_2, ..., h_k$. The output of each hash function h_i maps an item s uniformly to the range [0, m-1], i.e., an index in a m-bit array. The BF can be represented as $\{b_0, \ldots, b_{m-1}\}$. Initially all m bits are set to 0.

To insert an element $s \in S$ into a BF, s is hashed with all the k hash functions producing the values $h_i(s)(1 \leq i \leq k)$. The bits corresponding to these values are then set to 1 in the bit array. Figure 2 illustrates an example of BF insertion.To query the membership of an item s' within S, the bits at indices $h_i(s')(1 \leq i \leq k)$ are checked. If any of them is 0, then certainly $s' \notin S$. Otherwise, if all of the bits are set to 1, $s' \in S$ with high probability. There exists a possibility of error which arises due to *hashing collision* that makes the elements in S collectively causing indices $h_i(s')$ being set to 1 even if $s' \notin S$. This is called a *false positive*. Note that, there is no *false negative* in the BF membership verification.

Several BF variations that provide additional functionality exist. A Counting Bloom Filter (CBF) [9] associates a small counter with every bit, which is incremented/decremented upon item insertion/deletion. To answer approximate set membership queries, the *distance-sensitive Bloom filter* [10] has been proposed. However, aggregation is the only operation needed in our problem setting. The cumulative nature of the basic BF construction inherently supports the *aggregation* of BFs of a same kind, so we do not require CBFs or other BF variants.

3 Secure Provenance Encoding

We propose a distributed mechanism to encode provenance at the nodes and a centralized algorithm to decode it at the BS. The technical core of our proposal is the notion of *in-packet Bloom filter* (iBF) [11]. Each packet consists of a unique sequence number, data value, and an iBF which holds the provenance. We emphasize that our focus is on securely transmitting provenance to the BS. In an aggregation infrastructure, securing the data values is also an important aspect, but that has been already addressed in previous work (e.g., [12]). Our secure provenance technique can be used in conjunction with such work to obtain a complete solution that provides security for data, provenance and data-provenance binding, as shown in Section 3.3.

3.1 Provenance Encoding

For a data packet, *provenance encoding* refers to generating the vertices in the provenance graph and inserting them into the iBF. Each vertex originates at a node in the data path and represents the provenance record of the host node. A vertex is uniquely identified by the vertex ID (VID). The VID is generated per-packet based on the packet sequence number (seq) and the secret key K_i of the host node. We use a *block cipher function* to produce this VID in a secure manner. Thus for a given data packet, the VID of a vertex representing the node n_i is computed as

$$vid_i = generateVID(n_i, seq) = E_{K_i}(seq) \qquad (1)$$

where E is a secure block cipher such as AES, etc.

When a source node generates a packet, it also creates a BF (referred to as ibf_0), initialized to 0. The source then generates a vertex according to Eq. (1), inserts the VID into ibf_0 and transmits the BF as a part of the packet.

Upon receiving the packet, each intermediate node n_j performs data as well as provenance aggregation. If n_j receives data from a single child n_{j-1}, it aggregates the partial provenance contained in the packet with its own provenance record. In this case, the iBF ibf_{j-1} belonging to the received packet represents a partial provenance, i.e., the provenance graph of the sub-path from the source

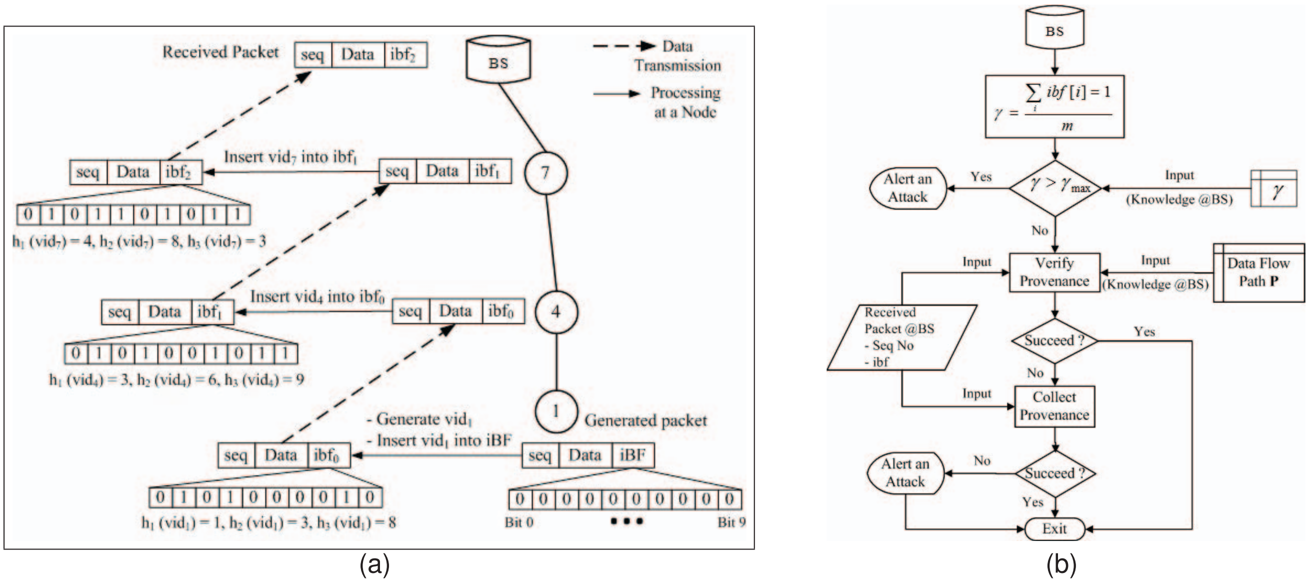


Fig. 3. (a) Mechanism for encoding provenance (node 1 is data source). (b) Provenance processing workflow at the BS upon receiving a packet.

upto n_{j-1}. On the other hand, if n_j has more than one child, it generates an aggregated provenance from its own provenance record and the partial provenance received from its child nodes. At first, n_j computes a BF ibf_{j-1} by bitwise-ORing the iBFs from its children. ibf_{j-1} represents a partial aggregated provenance from all of the children. In either case, the ultimate aggregated provenance is generated by encoding the provenance record of n_j into ibf_{j-1}. To this end, n_j creates a vertex using Eq. (1) and inserts the VID into ibf_{j-1} which is then referred to as ibf_j.

When the packet reaches the BS, the iBF contains the provenance records of all the nodes in the path i.e. the full provenance. We denote this final record by ibf.

Example: We illustrate the encoding mechanism in Figure 3(a). The data path considered is $< 1, 4, 7 >$, where node 1 is the data source. We use a 10-bit BF and a set of 3 hash functions $H = \{h_1, h_2, h_3\}$ for BF operations. When node 1 generates a data packet with sequence number seq, it creates the BF ibf_0 which is set to all 0's. The node then creates a vertex corresponding to its provenance record and computes the VID as $vid_1 = E_{K_1}(seq)$. To insert vid_1 into ibf_0, node 1 generates three indices as $h_1(vid_1) = 1$, $h_2(vid_1) = 3$, $h_3(vid_1) = 8$. The VID is then inserted by setting $ibf_0[1]$, $ibf_0[3]$, and $ibf_0[8]$ to 1. The updated ibf_0 along with the packet is then sent towards the BS.

Upon receiving the packet, node 4 performs provenance aggregation. Since the node has one child, it only aggregates its own provenance record with ibf_0. For this purpose, the node generates a VID vid_4; computes 3 indices as $h_1(vid_4) = 3$, $h_2(vid_4) = 6$, $h_3(vid_4) = 9$; and inserts vid_4 into ibf_0 by setting bits 3, 6, 9 of the iBF to 1. This updated iBF is referred to as ibf_1. The data packet with ibf_1 is then forwarded to node 7 which repeats the provenance aggregation steps. At the end, the BS receives the packet with the final iBF (ibf_2 from node 7) and stores this iBF for further processing.

3.2 Provenance Decoding

When the BS receives a data packet, it executes the *provenance verification* process, which assumes that the BS knows what the data path should be, and checks the iBF to see whether the correct path has been followed. However, right after network deployment, as well as when the topology changes (e.g., due to node failure), the path of a packet sent by a source may not be known to the BS. In this case, a *provenance collection* process is necessary, which retrieves provenance from the received iBF and thus the BS learns the data path from a source node. Afterwards, upon receiving a packet, it is sufficient for the BS to verify its knowledge of provenance with that encoded in the packet. Below we discuss these processes in detail:

Algorithm 1 ProvenanceVerification

```
Input: Received packet with sequence seq and iBF ibf.
Set of hash functions $H$, Data path $P' = < n'_{l_1}, ..., n'_1, ..., n'_p >$

$BF_c \leftarrow 0$   // Initialize Bloom Filter
for each $n'_i \in P'$ do
    $vid'_i$ = generateVID ($n'_i, seq$)
    insert $vid'_i$ into $BF_c$ using hash functions in $H$
endfor
if ($BF_c = ibf$) then
    return true   // Provenance is verified
endif
return false
```

Provenance Verification: The BS conducts the verification process not only to verify its knowledge of provenance but also to check the integrity of the transmitted provenance. Algorithm 1 shows the steps to verify provenance for a given packet. We assume that the knowledge of the BS about this packet's path is P'. At first, the BS initializes a Bloom filter BF_c with all 0's. The BF is then updated by generating the VID for each node in the path

P' and inserting this ID into the BF. BF_c now reflects the perception of BS about the encoded provenance. To validate its perception, the BS then compares BF_c to the received iBF ibf. The provenance verification succeeds only if BF_c is equal to ibf . Otherwise, if BF_c differs from the received iBF, it indicates either a change in the data flow path or a BF modification attack. The verification failure triggers the provenance collection process which attempts to retrieve the nodes from the encoded provenance and also to distinguish between the events of a path change and an attack.

Provenance Collection: As illustrated in Algorithm 2, the *provenance collection* scheme makes a list of potential vertices in the provenance graph through the ibf membership testing over all the nodes. For each node n_i in the network, the BS creates the corresponding vertex (i.e. v_i with VID vid_i) using Eq. (1). The BS then performs the membership query of vid_i within ibf. If the algorithm returns true, the vertex is very likely present in the provenance, i.e., the host node n_i is in the data path. Such an inference might introduce errors because of false positives (a node not on the route is inferred to be on the route). However, as we show later in Section 6, the false positive probability obtained is very low.

Once the BS finalizes the set of potential candidate nodes S = $< n'_{l_1}, ..., n'_1, n'_2, ..., n'_p >$, it executes the provenance verification algorithm on this set. This step is required to distinguish between the cases of a legitimate route change and that of malicious activity. If the verification succeeds, we decide that there was a natural change in the data path and we have been able to determine the path correctly. Otherwise, an attack has occurred.

A possible attack is the *all-one* attack where all bits in the provenance are set to 1, which implies the presence of all nodes in the provenance. To address the issue, we use a *density metric* γ introduced in [13]. γ reflects the number of 1's in the provenance (i.e. the iBF) as a fraction of the total size. To consider the provenance valid, we require that the density is equal or below a certain threshold: $\gamma \leq \gamma_{max}$. Such a requirement is reasonable since in a BF with n elements and k hash functions, there may be at most kn bits marked as '1'. Hence, we can always find an upper bound for the number of 1's in a BF. Thus, the maximum number of allowable 1's is $m\gamma_{max}$. Within this bound, an attacker may also *randomly flip some bits* to add or delete a legitimate node. The chance of being successful in this attack is very small since the attacker has to identify k bit positions corresponding to the node, which again change for each packet. If each bit is guessed randomly, the probability that the attacker guesses all of them correctly is given by $\frac{1}{2^m}$. Moreover, an attempt of blindly altering some bits is detected since the verification process at the end of the *provenance collection* phase does not succeed. A successful attack occurs when the bits set by the attacker (limited by γ_{max}) make all the k bits corresponding to a legitimate node turn out to be '1'. If the data provenance includes n nodes, the kn hash results may map to at least one and at most $m\gamma_{max}$ bits. Thus a smart attacker marks upto $(m\gamma_{max} - 1)$ bits. The total number of bit patterns by $(m\gamma_{max} - 1)$ hash computations is

$$B = \sum_{i=1}^{(m\gamma_{max}-1)} \binom{m}{i}$$

Randomly guessing one of them has $\frac{1}{B}$ chance of success. Hence, the manipulation attack has a very small success probability. The workflow shown in Fig. 3(b) summarizes the provenance decoding process.

Algorithm 2 ProvenanceCollection

```
Input: Received packet with sequence seq and iBF ibf.
Set of nodes ($N$) in the network, Set of hash functions $H$
1. Initialize
    Set of Possible Nodes $S \leftarrow \emptyset$
    Bloom Filter $BF_c \leftarrow 0$  // To represent S
2. Determine possible nodes in the path and build the representative BF
    for each node $n_i \in N$ do
      $vid_i$ = generateVID ($n_i, seq$)
      if ($vid_i$ is in $ibf$) then
        $S \leftarrow S \cup n_i$
        insert $vid_i$ into $BF_c$ using hash functions in $H$
      endif
    endfor
3. Verify $BF_c$ with the received iBF
    if ($BF_c = ibf$) then
      return $S$  // Provenance has been determined correctly
    else
      return NULL  // Indicates an in-transit attack
    endif
```

3.3 Scheme for Data-Provenance Binding

One of the important security challenges for a provenance scheme is to tie-up data and provenance. In an aggregation infrastructure, the data value is updated at each intermediate node which makes it a crucial problem to maintain the relationship between provenance and the intermediate data. A trivial solution can be based on making the provenance encoding mechanism dependent on the partial aggregation results (PAR) and append each PAR to the packet to verify the data-provenance binding at the BS. However, such an overhead nullifies the benefit of data aggregation. Hence, we formalize the problem in a slightly different way:

If the data aggregation result is verified at the BS, then the data-provenance coupling is ensured at each node in the routing path.

Since our concern is to devise a secure provenance scheme, we utilize secure in-network aggregation mechanisms to connect provenance with the intermediate aggregation results. Our objective is to incorporate our provenance scheme with a secure aggregation mechanism so that the aggregation verification process can also be used to check the data-provenance binding.

To serve this purpose, we can utilize an existing secure aggregation scheme such as [12], [14], [15]. To do so, we include some *Partial Provenance Information (PPI)* at each aggregation node so that the data-provenance binding is guaranteed through the data aggregation verification scheme at the BS. We adapt the verifiable in-network aggregation scheme proposed by Garofalakis et al. [12]. However, other similar schemes can be investigated and adapted to accommodate provenance information and hence, data-provenance binding. We first present a brief description of the scheme in [12], followed by a discussion on how it can be integrated with our proposed approach.

Background: The scheme in [12] allows parties in a distributed aggregate computation to verify that the final result has not been perturbed by more than a small error bound with high probability. The objective is to construct a verifiable random sample of given size p over the sensors' data values. The scheme ensures that the result computed by the aggregators is verifiably an unbiased random sample of the data. The sample is a general-purpose summary of the sensors' data that can be used to approximate verifiably a variety of different aggregation functions, not known beforehand. The key problem in the threat model is verifying the sampling procedure run by each aggregator.

Data aggregation is performed via three functions: an initializer $I(d)$ on data d generated by source node n_l, a merging function $\mathcal{M}$, and an evaluator function $\mathcal{E}$. $\mathcal{M}$ has the form $\langle z\rangle = \mathcal{M}(\langle x\rangle, \langle y\rangle)$, where $\langle x\rangle$ and $\langle y\rangle$ are multi-valued partial state records (PSRs) representing the intermediate state that will be required to compute an aggregate. $\langle z\rangle$ is the PSR resulting from the application of $\mathcal{M}$ to $\langle x\rangle$ and $\langle y\rangle$.

The basic technique combines Flajolet-Martin (FM) sketches and compact cryptographic signatures termed as authentication manifests (AM) to verify count aggregates. The solution from [12], *AM-Sample proof sketches*, collects a random sample by mapping $\langle dataRecord,\ sensorID\rangle$ elements to buckets with exponentially decreasing probabilities, using hash functions as in FM. Thus, given a uniformly randomizing hash function f over $\langle d_i, n_i\rangle$ pairs, where d_i is the data record at sensor n_i and a sample size p, an AM-Sample proof sketch is a pair $\langle L, \mathcal{A}\rangle$. Here, $\mathcal{A} = \{\langle l_1, d_1, n_1, s_{n_1}(d_1)\rangle, \ldots, \langle l_t, d_t, n_t, s_{n_t}(d_t)\rangle\}$ is a subset of t AMs $\langle l_i, d_i, n_i, s_{n_i}(d_i)\rangle$ with corresponding bucket levels $l_i = \mathrm{lsb}(f(d_i||n_i))$. The function $lsb(.)$ denotes the position of the least-significant '1' bit in the input binary string, $||$ denotes concatenation, and $s_{n_i}(d_i)$ represents a valid signature on data d_i by sensor n_i. A well-formed AM-Sample sketch stores exactly the AMs for $\langle dataRecord, sensorID\rangle$ elements at levels $\geq L$.

A concise description of the aggregation scheme for *AM-Sample proof sketches* is as follows. First, the algorithm computes the AMs and bucket levels for individual sensors as in Eq. (2).

$$I(d) = \langle 0, \{\langle \mathrm{lsb}(f(d||n_l)), d, n_l, s_{n_l}(d)\rangle\}\rangle \tag{2}$$

Next, these manifests are unioned up the aggregation topology, only keeping elements at the maximum level $\max(L_1,\ L_2)$ with every PSR merge. To keep the sketch size under control, the sampling rate drops by a factor of 2 when the sample size grows beyond $2p(1+\xi)$, where $\xi < 1$ denotes an error parameter [12]. Formally,

$$\mathcal{M}(\langle L_1, \mathcal{A}_1\rangle, \langle L_2, \mathcal{A}_2\rangle) = \langle L, \mathcal{A}(L, \mathcal{A}_1, \mathcal{A}_2)\rangle$$

where,

$$\langle L, \mathcal{A}(L, \mathcal{A}_1, \mathcal{A}_2)\rangle = \{\langle l, d, n, s_n(d)\rangle \in \mathcal{A}_1 \cup \mathcal{A}_2 : l \geq L\}$$

and

$$L = \begin{cases} max(L_1,\ L_2), & |\langle L, \mathcal{A}(L, \mathcal{A}_1, \mathcal{A}_2)\rangle| \leq (1+\xi)2p \\ max(L_1,\ L_2)+1 & \text{otherwise} \end{cases}$$

Finally, the evaluation at the BS is performed as

$$\mathcal{E}(\langle L,\ \mathcal{A}\rangle) = \{d : \langle l, d, n, s_n(d)\rangle \in \mathcal{A}\}$$

The AM-Sample proof sketches protect against the adversarial inflation of the collected random sample in two ways. First, through the use of authentication manifests for data tuples, the sketch prevents aggregators from forging new data, since all tuples are signed by a sensor. Second, AM signatures also prevent aggregators from migrating tuples across bucket levels (thereby biasing random sampling choices) since the level is determined through hashing by the signed tuple and sensor identifier. The scheme also prevents the aggregators from removing specific data from input PSRs, i.e., dropping child data during aggregation.

The more recent work in [15] deals with attacks against the *synopsis diffusion* and presents a lightweight *verification* algorithm to verify at BS if the computed aggregate is correct. The verification protocol computes several synopses verified independently through three phases. In the *query dissemination* phase, the BS broadcasts the name of the aggregation to compute and a random seed. In the *aggregation* phase, each node computes a subaggregate value based on the local value and the synopses of its children. The node also randomly selects a set of MACs from the MACs generated locally and the received ones from its children. Finally, in the *verification* phase, the BS computes the final synopsis using the messages from its child nodes and verifies the received MACs. This scheme can also be used in our work, but for brevity, we focus on the scheme from [12].

Extension for Data-Provenance Binding: To achieve data-provenance binding, we extend the above algorithm as follows. First, we modify the I function to

$$I(d) = \langle 0, \{\,\langle \mathrm{lsb}\,(f(d||n_l||b)\,), d, n_l, b, s_{n_l}(d||b)\,\rangle\,\}\rangle$$

Here, b represents the number of '1' bits in the generated or aggregated iBF at node n_l. Since an intermediate node n_i also aggregates its own sensory data with the child data, the merge function L should include a PSR corresponding to the data and the provenance record of n_i, as follows:

$$\begin{aligned}\langle L, \mathcal{A}(L, \mathcal{A}_1, \mathcal{A}_2)\rangle = \{\langle l, d, n, b, s_n(d||b)\rangle \in \mathcal{A}_1 \cup \mathcal{A}_2 : l \geq L\} \\ \cup\ \{\,\langle \mathrm{lsb}\,(h(d_i||n_i||b_i)\,), d_i, n_i, b_i, s_{n_i}(d_i||b_i)\,\rangle\,\}\end{aligned}$$

The modified *AM-Sample proof sketch* depends on data as well as provenance. While constructing the BF for provenance decoding, the BS counts the number of 1's in the BF after inserting the provenance record of each node and feeds the count value to the aggregation verification process. Thus, the aggregation verification succeeds only when both the data and the provenance are transmitted without perturbation. Note that, the order in which the nodes embedded their provenance record, i.e., the edges in the provenance graph might be useful in this context.

4 Detecting Packet Drop Attacks

We extend the secure provenance encoding scheme to detect packet drop attacks and to identify malicious node(s). We assume the links on the path exhibit natural packet loss and several adversarial nodes may exist on the path. For simplicity, we consider only linear data flow paths (i.e., as illustrated in Fig. 1(a)). Also, we do not address the issue of recovery once a malicious node is detected. Existing techniques that are orthogonal to our detection scheme can be used, which may initiate multipath routing [16] or build a dissemination tree around the compromised nodes [17].

We augment provenance encoding to use a packet-acknowledgement that requires the sensors to transmit more meta-data. For a data packet, the provenance record generated by a node will now consist of the node ID and an acknowledgement in the form of a sequence number of the lastly seen (processed/forwarded) packet belonging to that data flow. If there is an intermediate packet drop, some nodes on the path do not receive the packet. Hence, during the next round of packet transmission, there will be a mismatch between the acknowledgements generated from different nodes on the path. We utilize this fact to detect the packet drop attack and to localize the malicious node.

We consider a data flow path P where n_l is the only data source. We denote the link between nodes n_i and $n_{(i+1)}$ as l_i. We describe next packet representation, provenance encoding and decoding for detecting packet loss.

4.1 Data Packet Representation

To enable packet loss detection, a packet header must securely propagate the packet sequence number generated by the data source in the previous round. In addition, as in the basic scheme, the packet must be marked with a unique sequence number to facilitate per-packet provenance generation and verification. Thus, in the extended provenance scheme, any j^{th} data packet contains (i) the unique packet sequence number ($seq[j]$), (ii) the previous packet sequence number ($pSeq$), (iii) a data value, and (iv) provenance.

4.2 Provenance Encoding

Fig. 4 depicts the extended provenance encoding process. The provenance record of a node includes (i) the node ID, and (ii) an acknowledgement of the lastly observed packet in the flow. The acknowledgement can be generated in various ways to serve this purpose. In our solution, a node n_i creates a vertex v_i for every j^{th} packet it generates/forwards. The vertex ID vid_i is generated as:

$$\begin{aligned} vid_i &= generateVID\,(\,n_i,\, seq[j],\, pSeq_i\,) \qquad (3)\\ &= E_{K_i}(\,seq[j]\,||\,pSeq_i\,) \end{aligned}$$

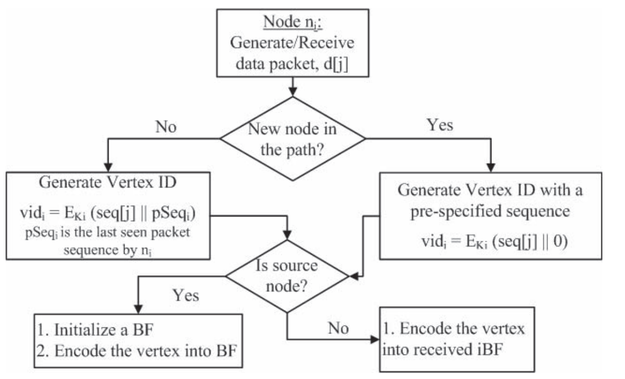

Node n_j:
Generate/Receive data packet, d[j]
New node in the path?
No
Yes
Generate Vertex ID
vid_i = E_Ki (seq[j] || pSeq_i)
pSeq_i is the last seen packet sequence by n_i
Generate Vertex ID with a pre-specified sequence
vid_i = E_Ki (seq[j] || 0)
Is source node?
Yes
No
1. Initialize a BF
2. Encode the vertex into BF
1. Encode the vertex into received iBF

Fig. 4. Extended provenance framework to detect packet drop attacks and identify malicious nodes.

where $pSeq_i$ is the knowledge of n_i about the sequence number of the previous packet in the flow. n_i updates the provenance of the packet by inserting vid_i into the iBF.

Note that, a node must maintain a per-flow record to store the previous packet sequence for each data flow that passed through the node. After a node n_i processes/forwards any j^{th} packet, it updates the $pSeq_i$ record for the corresponding data flow with the recently processed packet sequence, $seq[j]$. If a node receives a packet from a data flow for which it has no previous packet information, then it may use a *pre-specified special purpose* identifier, such as 0, as the previous packet sequence $pSeq_i$. This addresses the case of routing path changes where a new node in the path can use this special identifier for encoding provenance. Moreover, if a node does not receive packets from a data flow for a long time, it can erase the previous packet information for that flow to reduce space overhead. The node can get updated and maintain this flow-specific record when it receives packets from that flow more frequently.

4.3 Provenance Decoding at the BS

Not only the intermediate nodes, but also the BS stores and updates the latest packet sequence number for each data flow. Upon receiving a packet, the BS retrieves the preceding packet sequence ($pSeq$) transmitted by the source node from the packet header, fetches the last packet sequence for the flow from its local storage ($pSeq_b$), and utilizes these two sequences in the process of *provenance verification* and *collection*.

Provenance Verification: Similar to the basic scheme in Section 3, the BS first executes the provenance verification process upon receiving a packet. The BS knows (i) the current data path for the packet (decoded from the provenance of the previous packet in the flow), and (ii) the preceding packet sequence number forwarded by each node in the path. In this context, the BS assumes that each node in the path saw and forwarded the same packet in the last round, and that this packet's sequence number is the same one as recorded at the BS. Thus the verification is bound to fail when $pSeq$ and $pSeq_b$ do not match, which also indicates a possible packet loss and suffices to execute provenance collection process directly skipping the verification.

The provenance verification is performed according to algorithm 1, with the only difference that the BS now uses

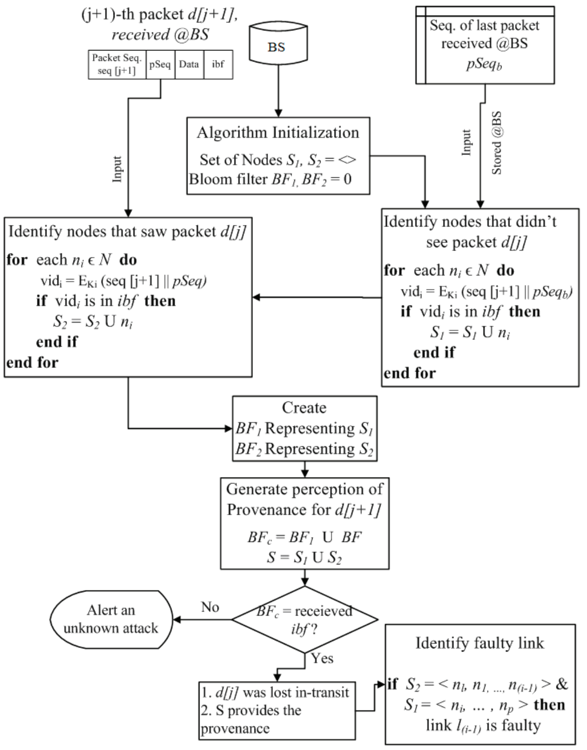

(j+1)-th packet d[j+1], received @BS
Packet Seq. seq [j+1]
pSeq
Data
ibf
BS
Seq. of last packet received @BS pSeq_b
Input
Input
Stored @BS
Algorithm Initialization
Set of Nodes S_1, S_2 = <>
Bloom filter BF_1, BF_2 = 0
Identify nodes that saw packet d[j]
for each n_i ∈ N do
vid_i = E_Ki (seq [j+1] || pSeq)
if vid_i is in ibf then
S_2 = S_2 U n_i
end if
end for
Identify nodes that didn't see packet d[j]
for each n_i ∈ N do
vid_i = E_Ki (seq [j+1] || pSeq_b)
if vid_i is in ibf then
S_1 = S_1 U n_i
end if
end for
Create
BF_1 Representing S_1
BF_2 Representing S_2
Generate perception of Provenance for d[j+1]
BF_c = BF_1 U BF
S = S_1 U S_2
Alert an unknown attack
No
BF_c = receieved ibf?
Yes
Identify faulty link
1. d[j] was lost in-transit
2. S provides the provenance
if S_2 = < n_l, n_1, ..., n_(i-1) > &
S_1 = < n_i, ... , n_p > then
link l_(i-1) is faulty

Fig. 5. Provenance Collection, Packet Loss Detection, and Malicious Node Identification.

Eq. (3) to create the VID for a node. Verification failure here indicates either a change in the data flow path, a packet drop attack or a BF modification attack, and triggers the provenance collection process.

Provenance Collection: Collection attempts to retrieve the nodes from the encoded provenance, confirm a packet loss and identify the malicious node that dropped the packet. It also distinguishes between the packet drop attack and other attacks that might have altered the iBF.

Note that, in case of a path change, the new nodes can be easily learnt through an iteration of *ibf* membership testing over all the nodes. During provenance encoding, every new node in the path uses a *special purpose* packet identifier (e.g., 0) as the previous packet sequence and generates its VID as $E_{K_i}(seq[j]||0)$. Therefore, to retrieve the new nodes in the path, the decoding scheme at the BS should perform an *ibf* membership testing over all the nodes, where the VID for each node will be generated using the *pre-specified* previous packet identifier, along with the nodeID and the packet sequence number, $seq[j]$.

For the remainder of the discussion, we assume that a data packet $d[j]$ has been dropped by an intermediate node n_i. Thus, the nodes $n_l, n_1, ..., n_i$ received $d[j]$ and updated their lastly seen packet sequences to $seq[j]$. On the contrary, nodes $n_{i+1}, ..., n_p$ as well as the BS did not observe $d[j]$, They have no information to update the preceding packet sequence, and they retain the same old identifier $seq[j-1]$. Upon receiving the next packet in the flow, $n_l, n_1, ..., n_{i-1}$ include $seq[j]$ in the provenance metadata, whereas $n_{i+1}, ..., n_p$ use $seq[j-1]$ for this purpose when computing their VIDs. However, the malicious node n_i may either (i) use $seq[j]$, or (ii) use $seq[j-1]$. Without any loss of generality, we assume that the malicious node encodes $seq[j-1]$ in the provenance BF.

Fig. 5 illustrates the provenance collection algorithm which can retrieve the nodes in a data path even in the presence of packet loss. Upon receiving the next packet (i.e. the $(j+1)^{th}$ packet), the BS checks the membership of all nodes in the network within the iBF using a two step process. The first query is performed with the identifier of the last packet ($pSeq_b$) recorded at the BS, and the next one with the previous packet sequence ($pSeq$) contained in the packet header. We denote the set of nodes found in the first and second step by $S_1 = < n'_i, ..., n'_p >$ and $S_2 = < n'_l, n'_1, ..., n'_{(i-1)} >$, respectively. Let the BFs constructed with S_1 and S_2 be BF_1 and BF_2, respectively. The final Bloom filter BF_c is constructed as a bitwise-OR of BF_1 and BF_2 and reflects the perception of the BS about the encoded provenance.

Using as input the set of potential candidate nodes $S = S_1 \; U \; S_2$, the BS executes a verification algorithm in order to distinguish between the packet drop attack and any other iBF modification attacks. If BF_c and the received iBF ibf match, the verification succeeds. In this case, we confirm the event of a packet loss and decide that the path constructed on the set of nodes S is equivalent to path P. Thus, we have been able to determine the provenance correctly. Otherwise, some unknown attack has occurred. Note that, if no packet drop attack occurred, the first query is sufficient to compute the provenance.

***Malicious Node Identification*:** If S represents the actual data flow path P, then $S_2 = < n_l, n_1, ..., n_{(i-1)} >$ and $S_1 = < n_i, ..., n_p >$. Thus, we can conclude that the link $l_{(i-1)}$ was the one where the packet was lost. However, if we would have assumed that the malicious node encodes $seq[j]$, then the BS would have detected l_i as the location of the loss. In either case, an adjacent link to the malicious node is identified, and the node can be marked a such. To confirm that the faulty link $l_{(i-1)}$ is where the packet loss occurred, the BS observes more packets. Whenever the BS identifies a packet loss and the responsible link $l_{(i-1)}$, it updates the empirical loss rate $el_{(i-1)}$ for the link. Assume that the *drop rate threshold* for a link is α, where α is greater than the natural loss rate of any link. If after a number of packet transmissions, $el_{(i-1)} > \alpha$, then the BS asserts that $l_{(i-1)}$ was the link where the packet was lost, and identifies n_i as malicious.

5 Security Discussion

In this section, we discuss the security properties of the proposed provenance scheme.

Confidentiality.

Claim 1: It is computationally infeasible for an attacker to gain information about the sensor nodes included in the provenance by observing data packets.

Justification: The confidentiality of the scheme is achieved through two factors: the use of BF and the use of encryption keys. When one-way hash functions are used to insert elements in the BF, the identities of the inserted elements cannot be reconstructed from the BF representation. An attacker may collect a large sample of iBFs to infer some common patterns of the inserted elements. If the attacker has the knowledge of the complete element space (i.e. provenance records of all the nodes) and the hashing schemes, it can try a dictionary attack by testing for the presence of every element and obtain a probabilistic answer to what elements are carried in a given iBF. However, the elements inserted in the iBF, i.e., provenance records of the nodes, depend on a per-packet variable - *sequence number*, and also there is a secret key that is used in deriving the node VIDs that are inserted in the iBF. For legitimate nodes, these secrets are unknown to the attacker, as each key K_i is shared only between the node and the BS. To increase the level of security, we can use pseudo-random functions (PRFs) seeded with the secret key and produce a different key instance at each epoch [18]. Therefore, the shared key is not directly exposed, and each instance key is used only once. Thus, even if an adversary obtains plaintexts and corresponding ciphertexts for one epoch, the confidentiality at other time epochs is preserved. To conclude, an attacker cannot gain any information through the observation of packets and the encoded provenance. □

Integrity.

Claim 2: An attacker, acting alone or colluding with others, cannot successfully add or legitimate nodes to the provenance of data generated by the compromised nodes.

Justification: Attacker(s) may attempt to generate fake data and construct the provenance including some innocent nodes $< n'_l, n'_1, n'_2, ..., n'_p >$ to make them responsible for false data and consequently to mark them as untrustworthy. However, the provenance embedding process requires the node specific secret K_i for cryptographic computation of the corresponding VID, and the attackers do not know the key for the legitimate nodes. Hence, this attack will fail. □

Claim 3: An attacker or a set of cooperative attackers cannot selectively add or remove nodes from the provenance of data generated by benign nodes.

Justification: Assume that n_e and n_m collude to execute an attack. A benign packet with provenance $< n_l, ..., n_1, n_2, ..., n_p >$ is routed through n_e. For this packet, n_e wants to remove n_2 from the provenance and to replace it with another legitimate node n'_2, utilizing any knowledge from n_m. When the packet reaches n_e, it contains the partial provenance $< n_l, ..., n_1, ..., n_e >$ encoded in the iBF ibf_{pp}. To remove n_2 from provenance, at first n_e has to construct the Bloom filter BF_2 containing the provenance record of n_2. Then, by performing a bitwise-AND of the bitwise negation of BF_2 with ibf_{pp}, n_e removes the information of n_2 from the provenance. Assume the modified iBF is ibf'_{pp}. To add n'_2 to the provenance after the removal of n_2, the BF corresponding to n'_2 should be built and then OR-ed with ibf'_{pp}.

In both cases, the attackers have to construct a BF representing an uncompromised node. This requires the knowledge about secrets of these legitimate nodes which are unknown to n_e and n_m. Furthermore, the provenance record of a node changes dynamically for each packet. So the attackers cannot utilize any old BF.

However, n_e and n_m can collude to remove benign nodes more intelligently, where n_m (in or outside the path) reports its observation of the iBF state to n_e. Upon receiving the packet, n_e zeroes all the 1's added to the iBF since n_m's observation, and thus removes provenance records of the corresponding nodes. Since our data-provenance binding solution adds PPI to data at each intermediate (i.e. aggregator) node, this attack fails the data aggregation verification at the BS and hence, the attack is detected. □

Claim 4: A malicious aggregator cannot selectively drop a child node from the provenance.

Justification: As illustrated in Fig. 1(b), there are two types of data aggregation. The cluster head (e.g. n_1) aggregates data from all of the nodes in its cluster whereas an intermediate node (e.g. n_2) aggregates its sensed data with the data received from child node (i.e. n_1). Thus, we consider two scenarios:

(i) Aggregator n_1 drops the incoming data from a child node (e.g., n_{l_2}) and computes the aggregated data and provenance excluding it. The scheme from [12] prevents nodes from dropping child data during aggregation. Our data-provenance binding solution integrates PPI with the partially aggregated data. Hence, it detectd when the data or provenance record of a child is dropped.

(ii) Intermediate node n_2 discards the data and partial provenance received from child node n_1. Since this attack represents a *packet dropping attack*, it is detected by the BS with the scheme described in section 4. Node n_2 cannot selectively remove the provenance of n_1. However, n_2 might also forward the data as it is but discard the partial provenance. This will destroy the data-provenance binding which will be detected as discussed in Section 3.3. □

Freshness.

Claim 5: Provenance replay attacks are detected by our proposed scheme.

Justification: Since provenance encoding depends on a packet specific information, the value of the constructed iBF varies from packet to packet. Hence, malicious attempts to associate a previously captured iBF with a more recent data packet (benign/fake) is detected at the BS. □

6 Performance Analysis

We present an analysis of the space and energy overhead of our scheme. We use the following benchmarks:

(i) We adapt the generic secure provenance framework $SProv$ [5] to sensor networks. In this lightweight version of the scheme, referred to as SSP, we simplify the provenance record at a node n_i as $P_i =< n_i, hash(D_i), C_i >$, where $hash(D_i)$ is a cryptographic hash of the updated data, and C_i contains an integrity checksum as $Sign(hash(n_i, hash(D_i)|C_{i-1}))$.

(ii) We also consider a MAC-based provenance scheme, referred to as *MP*, where a node transmits the nodeID and a MAC computed on it as the provenance record.

6.1 Space Complexity

To implement SSP, we use SHA-1 (160 bit) for cryptographic hash operations and the TinyECC library [19] to generate 160-bit digital signatures (ECDSA). The nodeID has length 2 bytes, thus the length of each provenance record is 42 bytes. For MP, we use TinySec library [20] to compute a 4-byte CBC-MAC. Hence, a provenance record has 6 bytes in this case. As each node in the path encodes its own provenance record, the provenance size increases linearly with the number of hops. For a D-hop path, the provenance is $42D$ bytes in SSP and $6D$ bytes in MP.

Since our approach is based on the BF, the provenance length depends on parameter selections for the BF. The false positive probability for a BF is defined as [21]

$$P_{fp} = \frac{n_a - n}{n_t - n}$$

where n_t is the total number of distinct elements in the element space, n is the number of elements actually encoded in the BF and n_a is the number of elements retrieved by querying the BF. Let m be the BF size, k the number of hash functions and D the maximum number of nodes in any path. The false positive probability is equal to that of getting 1 in all the k array positions computed by the hash functions while querying the membership of an element that was not inserted in the BF. The probability is [22]

$$P_{fp} = (1 - (1 - \frac{1}{m})^{kD})^k \approx (1 - e^{-\frac{kD}{m}})^k \tag{4}$$

For a given m and D, the number of hash functions that minimizes the false positives is computed as [22]

$$k_{opt} = \frac{m}{D} ln2 \tag{5}$$

Given D and a desired false positive probability P_{fp}, the required number of bits m can be computed by substituting the optimal value of k in Eq. (4) and then simplifying it to

$$ln(P_{fp}) = -\frac{m}{D} * (ln2)^2 \ \Rightarrow m = \frac{-D * ln(P_{fp})}{(ln2)^2} \tag{6}$$

This means that to maintain a fixed false positive probability, the length of a BF should grow with the number of elements. For example, if we consider $P_{fp} = 0.02$ and a 14-hop path, the BF size m is computed as 114 bits and k_{opt} = 6. Thus, a 120-bit (15 byte) BF is sufficient to encode provenance while maintaining low false positives. In practice, we bound P_{fp} by a small constant $\delta\,(> 0)$ such that $P_{fp} < \delta$. To find the appropriate value of m we have

$$ln(P_{fp}) > ln\delta \ \Rightarrow -\frac{m}{D} * (ln2)^2 > ln\delta \ \Rightarrow m < \frac{Dln\frac{1}{\delta}}{(ln2)^2}$$

6.2 Energy Consumption

For a D-hop path, SSP has to transmit $42 * D$ bytes (= $336 * D$ bits), MP transmits $6 * D$ bytes (= $48 * D$ bits) whereas our scheme requires transmitting m bits. SSP, MP and our scheme consume a radio energy proportional to $(336{*}D)$, $(48{*}D)$ and $\frac{ln\frac{1}{\delta}}{(ln2)^2}{*}D$, respectively. Although all of the terms are proportional to D, the constant coefficient in the first two terms are much larger than the last one. For example, if we set $\delta = 10^{-4}$ then the coefficient in our scheme is 19.17 which is much smaller than the coefficients in SSP and MP. Another part of overhead comes from the signature, MAC and hash computations. However, in sensor networks, computation overhead is much smaller than communication, and adds only marginal energy consumption [23].

6.3 Detection of Packet Drop Attacks

Provenance is used to detect packet loss attacks, and to identify the malicious node(s). As discussed in Section 4, the first step in identifying the malicious node is to detect the link where the loss occurred. The detection error depends on the BF parameters and the analysis from Section 6.1 applies to this case as well, with the only difference that the element space is larger now due to the addition of packet sequence information in the node VID. Hence, a larger BF is required to keep the false positive rate small.

Since packet drop attacks directly reduce the amount of legitimate data throughput, we also analyze our scheme to provide the theoretical bounds for guaranteed end-to-end throughput and for attack detection rate. Let ρ_i be the natural packet loss rate (i.e., in the absence of attacks) of link l_i, where ρ_i's are i.i.d. random variables with maximum value ρ. Let α denote the per-link drop rate threshold. The theoretical bounds are computed under the condition that the empirical loss rate converges to its true value within a small uncertainty interval ϵ. The detection rate of the proposed scheme, i.e., the number of data packets transmitted by the source before reaching the converging condition is computed as follows:

Theorem 1: *Given the threshold $\alpha = \rho + \epsilon$ and the allowed false positive σ, the scheme requires $\frac{ln(\frac{2}{\sigma})}{2\epsilon^2(1-\rho-\epsilon)^D}$ packets to be transmitted by the source to achieve the converging condition.*

Proof. At first, we determine the number of packet transmissions required to estimate the drop rate of a single link l_i within a certain accuracy interval. We consider that the actual packet loss rate of link l_i is r_i^*, and the empirical loss rate is r_i. We compute the number of packets needed to estimate the packet drop rate with a (ϵ_{r_i}, σ) accuracy such that $Pr(|r_i^* - r_i| > \epsilon_{r_i}) < \sigma$. In other words, with probability at least $(1 - \sigma)$ the estimated $r_i \in (r_i^* - \epsilon_{r_i}, r_i^* + \epsilon_{r_i})$. Then we compute the total number of packets needed to achieve a (ϵ_{r_i}, σ) accuracy for the loss rate estimation (i.e., r_i) of every link.

We define each instance of a data packet arriving at node n_i as a random variable of l_i. We assume that a node correctly embeds its provenance record whenever it forwards a data packet. Using Maximum Likelihood Estimation and Hoeffding's inequality, we obtain

$$\begin{aligned} &Pr(|r_i^* - r_i| > \epsilon_{r_i}) \leq 2\exp -2N_i\epsilon_{r_i}^2 \\ \Longrightarrow\ &N_i = \frac{ln(\frac{2}{\sigma})}{2\epsilon_{r_i}^2} \geq \frac{ln(\frac{2}{\sigma})}{2\epsilon^2} \end{aligned} \tag{7}$$

where $\epsilon_{r_i} \leq \epsilon$.

Now we compute the number of packets needed to give an estimate with (ϵ_{r_i}, σ)-accuracy for every link in a D-hop path. When each packet transmitted by the source reaches node $n_{(D-1)}$, it provides a trial for every link l_i belonging to the path. Therefore, transmitting N_i packets to $n_{(D-1)}$ also suffices to give other links enough trials,

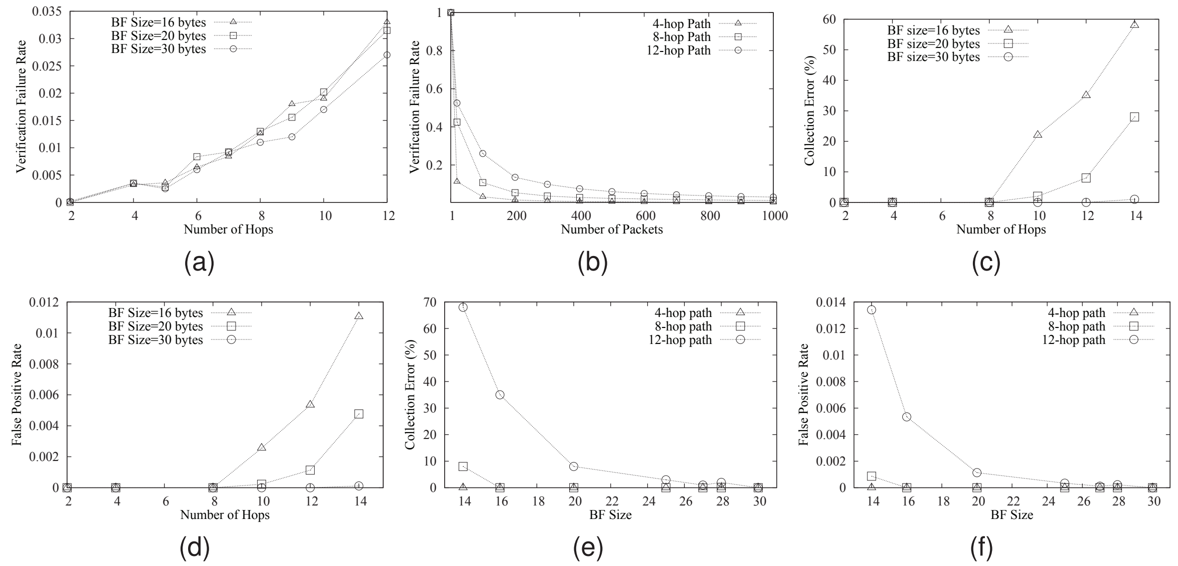


Fig. 6. (a) Provenance VFR vs path length. (b) VFR variation with time as network stabilizes. (c)(d)(e)(f) Collection Error and False Positive Rate for various path lengths and BF sizes.

which requires a total of

$$N_{(D-1)}\frac{1}{(1-\alpha)^D} = \frac{ln(\frac{2}{\sigma})}{2\epsilon^2(1-\rho-\epsilon)^D}$$

packets transmitted from the source. □

Under the converging condition, a link drops packets at no more than α rate. Thus at any time, the combined rate of packet drop induced by z malicious nodes would be at most $z\alpha$. Hence, given a path of length D, an adversary in control of z intermediate nodes can cause an end-to-end packet loss rate of at most $z\alpha$ without being detected.

7 Simulation Results

We implemented and tested the proposed techniques using the TinyOS simulator (TOSSIM) [24]. We have used the *micaz* energy model and PowerTOSSIM z [25] plugin to TOSSIM to measure the energy consumption. We consider a network of 100 nodes and vary the network diameter from 2 to 14. All results are averaged over 100 runs. First, we look at how effective the secure provenance encoding scheme (introduced in Section 3) is in detecting provenance forgery and path changes. Next, we investigate the accuracy of the proposed method for detecting packet loss (which was presented in Section 4). Finally, we measure the energy consumption overhead of securing provenance.

7.1 Provenance Decoding Error

Provenance decoding retrieves the provenance from the in-packet BF and consists of *verification* and *collection* phases. To quantify the accuracy and efficiency of our provenance scheme, we measure the decoding error in both the above phases, i.e., *verification* and *collection error*.

Algorithm 1 shows that the verification fails when the provenance graph in the packet does not match the local knowledge at the BS. This may happen when there is a data flow path change or upon a BF modification attack. Provenance *verification failure rate (VFR)* measures the ratio of packets for which verification fails. Fig. 6(a) shows the VFR for paths of 2 to 12 hops with various BF sizes. For each path length, the VFR is averaged over 1000 distinct paths. The results show that the provenance verification process fails only for a very small fraction of packets. Thus, for most packets the *lightweight verification* process is sufficient to retrieve the provenance. The more costly provenance collection process is executed only for a very few packets when verification fails. As expected, VFR increases linearly with the increase of the path length. On the other hand, VFR is not significantly influenced by BF size, proving that even small BF sizes provide good protection. Fig 6(b) shows the variation of VFR over time, as the number of packet transmissions increases. As the network gets stable with time, the data paths do not change often, and hence the VFR approaches 0.

Fig. 6(c) and 6(d) plot the percentage of *provenance collection error* for different number of hops and the corresponding BF false positive rates, respectively. Recall that, the collection phase is executed when provenance verification fails. Fig. 6(e) and 6(f) show the collection error corresponding to various BF sizes and the related false positives, respectively. The number of hash functions used is determined using Eq. (5). The resulting false positive rates vary from $0 \sim 0.013$ and it is observed that the collection error becomes negligible when the false positive rate drops at or below 10^{-4}. It is also seen that a BF size of 16 bytes is enough to ensure no decoding error for up to 8-hop paths. The empirical BF size required is much less than the theoretical one(~ 20 bytes for a 8-hop path).

7.2 Detection of Packet Drop Attacks

In these experiments, we consider one malicious node in every data path considered, the natural link loss rate is set to $\rho = 0.01$, the malicious link loss rate to $\alpha = 0.06$, and the maximum allowed threshold for false positives in attack detection to $\sigma = 0.03$. The BF sizes are varied from 16 to 35 bytes (note that this is slightly larger than for

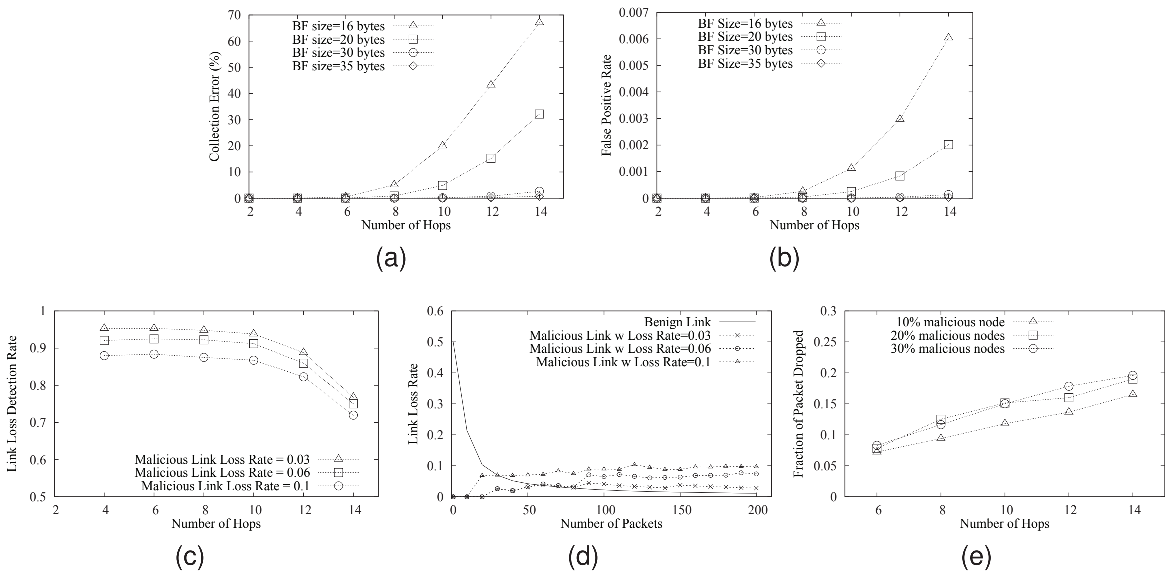


Fig. 7. (a) Percentage of Collection Error (b) False Positive Rates of extended provenance scheme. (c) Success rate of detecting packet drop for various malicious link loss rates. (d) Accuracy of malicious link identification over time. (e) End-to-end packet drop rate for various percentages of malicious nodes deployed in the network.

the basic scheme, because the packet sequence information must now be included as well in the BFs). The percentages of provenance collection error and corresponding false positive rates for the extended provenance scheme are shown in Fig. 7(a) and 7(b), respectively. Fig. 7(a) shows that the provenance collection error for the extended scheme depends on BF sizes and follows the same pattern as in the basic scheme. As expected, the errors for the same BF sizes are higher compared to the basic scheme, due to the extended (doubled) element space for the received iBF which increases the hash collisions and consequently the error rates. With a suitably chosen BF size (e.g. 30 bytes), collection errors can be kept low for any path lengths. Thus, the collection error does not affect much the accuracy of the malicious node identification process. The false positives in the error cases, as shown in Fig. 7(b), do not have significant changes compared to those of the basic scheme.

Fig. 7(c) illustrates the success of our provenance scheme in detecting packet losses. The success rate, termed as *Link Loss Detection Rate*, is measured as a ratio of the number of packet losses detected to the actual packet losses. The BF size is set to 25 bytes, one malicious node is placed in every data path considered, and the malicious link loss rate is varied as 0.03, 0.06, 0.1. Intuitively, the link loss detection rate decreases with an increase of the link loss rate. When the link loss rate increases, the probability of consecutive packet losses by the malicious nodes also increases. Our provenance scheme cannot distinguish between a single packet loss and multiple consecutive packet losses and thus counts the consecutive packet losses as a single one. Hence, as the malicious link loss rate increases, the link loss detection rate by our scheme degrades. However, even though we do not achieve a 100% detection rate, the success probability we obtain is high (75% in the worst case).

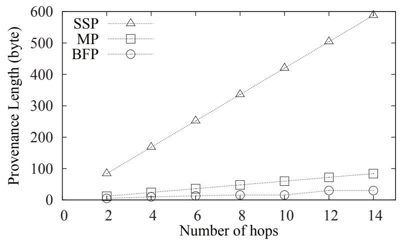
Fig. 7(d) shows the accuracy of the *malicious link identification* process over time and how it leads to the detection of packet drop attacks. The figure plots the link loss rates over packet transmissions in order to show the convergence of link statistics to their actual values. For an uncompromised node, the link loss rate should converge to the natural loss rate whereas for a malicious node the link statistics should tend towards a significantly higher loss rate which confirms the packet drop attack. We consider an arbitrary 14 hop path where n_3 is malicious and controls the link l_3. As earlier, we consider a natural link loss rate $\rho = 0.01$ and 3 different malicious link loss rates 0.03, 0.06, 0.1. The results show that eventually the packet drop attack is detected successfully. However, there is a probability of errors since in the earlier stage the loss rate of malicious links seem to be much less than the actual packet drop rate, while the loss rate of the benign link seems high.

Fig. 7(e) presents the degradation of data throughput by the time the attack is detected in robust settings, where 10%, 20%, and 30% of the total nodes are malicious. Drop rates over paths are bounded by the sum of natural loss rates of the intermediate links and malicious loss rates of any malicious links in a considered data flow path. As expected, the data throughput at the BS degrades with the increasing number of malicious nodes in the data flow path.

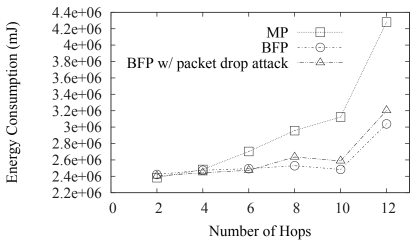
7.3 Space Complexity and Energy Consumption

Fig. 8(a) compares SSP, MP and our provenance mechanism in terms of bytes required to transmit provenance. The provenance length in SSP and MP increases linearly with the path length. For our scheme, we empirically determine the BF size which ensures no decoding error. Although the BF size increases with the expected number of elements to be inserted, the increasing rate is not linear. We see that even for a 14-hop path, a 30 byte BF is sufficient for provenance decoding without any error.

We also measure the energy consumption for both the basic provenance scheme and the extended scheme for packet drop detection, while varying hop counts. For packet



(a)



(b)

Fig. 8. (a) Provenance length (b) Energy consumption

drop attack, we set the malicious link loss rate as 0.03. Note that, modern sensors use *ZigBee* specification for high level communication protocols which allows upto 104 bytes as data payload. Hence, **SSP and MP can be used to embed provenance (in data packet) for maximum 2 and 14 nodes, respectively**. Figure 8(b) shows aggregate energy consumption over 1000 packet transmissions. The results confirm the energy efficiency of our solutions.

8 Related Work

Pedigree [26] captures provenance for network packets in the form of per packet tags that store a history of all nodes and processes that manipulated the packet. However, the scheme assumes a trusted environment which is not realistic in sensor networks. ExSPAN [27] describes the history and derivations of network state that result from the execution of a distributed protocol. This system also does not address security concerns and is specific to some network use cases. SNP [28] extends network provenance to adversarial environments. Since all of these systems are general purpose network provenance systems, they are not optimized for the resource constrained sensor networks.

Hasan et al. [5] propose a chain model of provenance and ensure integrity and confidentiality through encryption, checksum and incremental chained signature mechanism. Syalim et al. [29] extend this method by applying digital signatures to a DAG model of provenance. However, these generic solutions are not aware of the sensor network specific assumptions, constraints etc. Since provenance tends to grow very fast, transmission of a large amount of provenance information along with data will incur significant bandwidth overhead, hence low efficiency and scalability. Vijaykumar et al. [30] propose an application specific system for near-real time provenance collection in data streams. Nevertheless, this system traces the source of a stream long after the process has completed. Closer to our work, Chong et al. [31] embed the provenance of data source within the dataset. While it reflects the importance of issues we addressed, it is not intended as a security mechanism, hence, does not deal with malicious attacks. Besides, practical issues like scalability, data degradation, etc. have not been well addressed. In our earlier work [32], secure transmission of the provenance requires several distinct packet transmissions. The underlying assumption is that provenance remains the same for at least a flow of packets. Our work relinquishes that assumption.

While BFs are commonly used in networking applications, *iBF*s have only recently gained more attention being utilized in applications such as credential based data path security [13], IP traceback [33], source routing and multicast [34], [35] etc. The basic idea in these works is to encode the link identifiers constituent to the packet routing path into an iBF. However, the encoding of the whole path is performed by the data source, whereas the intermediate routers check their membership in the iBF and forward the packet further based on this decision. This approach is infeasible for sensor networks where the paths may change due to several reasons. Moreover, an intermediate router only checks its own membership which may leave several integrity attacks such as *all-one attack*, *random bit flips* etc., undetected. Our approach resolves these issues by encoding the provenance in a distributed fashion.

9 Conclusion

We addressed the problem of securely transmitting provenance for sensor networks, and proposed a light-weight provenance encoding and decoding scheme based on Bloom filters. The scheme ensures confidentiality, integrity and freshness of provenance. We extended the scheme to incorporate data-provenance binding, and to include packet sequence information that supports detection of packet loss attacks. Experimental and analytical evaluation results show that the proposed scheme is effective, light-weight and scalable. In future work, we plan to implement a real system prototype of our secure provenance scheme, and to improve the accuracy of packet loss detection, especially in the case of multiple consecutive malicious sensor nodes.

Acknowledgments. This work has been partially funded by NSF awards CNS-1111512 and CNS-0964294.

References

[1] H. Lim, Y. Moon, and E. Bertino, "Provenance-based trustworthiness assessment in sensor networks," in *Proc. of Data Management for Sensor Networks*, 2010, pp. 2–7.
[2] I. Foster, J. Vockler, M. Wilde, and Y. Zhao, "Chimera: A virtual data system for representing, querying, and automating data derivation," in *Proc. of the Conf. on Scientific and Statistical Database Management*, 2002, pp. 37–46.
[3] K. Muniswamy-Reddy, D. Holland, U. Braun, and M. Seltzer, "Provenance-aware storage systems," in *Proc. of the USENIX Annual Technical Conf.*, 2006, pp. 4–4.
[4] Y. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," *SIGMOD Record*, vol. 34, pp. 31–36, 2005.
[5] R. Hasan, R. Sion, and M. Winslett, "The case of the fake picasso: Preventing history forgery with secure provenance," in *Proc. of FAST*, 2009, pp. 1–14.
[6] S. Madden, J. Franklin, J. Hellerstein, and W. Hong, "TAG: a tiny aggregation service for ad-hoc sensor networks," *SIGOPS Operating Systems Review*, no. SI, Dec. 2002.

[7] K. Dasgupta, K. Kalpakis, and P. Namjoshi, "An efficient clustering based heuristic for data gathering and aggregation in sensor networks," in *Proc. of Wireless Communications and Networking Conference*, 2003, pp. 1948–1953.

[8] S. Sultana, E. Bertino, and M. Shehab, "A provenance based mechanism to identify malicious packet dropping adversaries in sensor networks," in *Proc. of ICDCS Workshops*, 2011, pp. 332–338.

[9] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, Jun. 2000.

[10] A. Kirsch and M. Mitzenmacher, "Distance-sensitive bloom filters," in *Proc. of the Workshop on Algorithm Engineering and Experiments*, 2006, pp. 41–50.

[11] C. Rothenberg, C. Macapuna, M. Magalhaes, F. Verdi, and A. Wiesmaier, "In-packet bloom filters: Design and networking applications," *Computer Networks*, vol. 55, no. 6, pp. 1364 – 1378, 2011.

[12] M. Garofalakis, J. Hellerstein, and P. Maniatis, "Proof sketches: Verifiable in-netwok aggregation," in *ICDE*, 2007, pp. 84–89.

[13] T. Wolf, "Data path credentials for high-performance capabilities-based networks." in *Proc. of ACM/IEEE Symp. on Architectures for Networking and Communications Systems.*, 2008, pp. 129–130.

[14] H. Chan, A. Perrig, and D. Song, "Secure hierarchical in-network aggregation in sensor networks," in *Proc. of the conf. on Computer and communications security (CCS)*, 2006, pp. 278–287.

[15] S. Roy, M. Conti, S. Setia, and S. Jajodia, "Secure data aggregation in wireless sensor networks," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 3, pp. 1040–1052, 2012.

[16] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: attacks and countermeasures," in *Proc. of Intl. Workshop on Sensor Network Protocols and Applications*, 2003, pp. 113–127.

[17] S. Marti, T. J. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks," in *Proc. of the Intl. Conf. on Mobile Computing and Networking*, 2000, pp. 255–265.

[18] S. Papadopoulos, A. Kiayias, and D. Papadias, "Secure and efficient in-network processing of exact sum queries," in *Proc. of International Conference on Data Engineering*, 2011, pp. 517–528.

[19] A. Liu and P. Ning, "TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks," in *Proc. of IPSN*, 2008, pp. 245–256.

[20] C. Karlof, N. Sastry, and D. Wagner, "Tinysec: a link layer security architecture for wireless sensor networks," in *Proc. of the Intl. Conf. on Embedded networked sensor systems*, 2004, pp. 162–175.

[21] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, pp. 422–426, 1970.

[22] M. Mitzenmacher, "Compressed bloom filters," in *Proc. of ACM Symp. on Principles of Distributed Computing*, 2001, pp. 144–150.

[23] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical en-route filtering of injected false data in sensor networks," in *Proc. of INFOCOM*, 2004, pp. 839–850.

[24] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: accurate and scalable simulation of entire tinyos applications," in *Proc. of the Intl. Conf. on Embedded networked sensor systems*, 2003, pp. 126–137.

[25] E. Perla, A. Catháin, R. S. Carbajo, M. Huggard, and C. M. Goldrick, "Powertossim z: realistic energy modelling for wireless sensor network environments," in *Proc. of the ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, 2008, pp. 35–42.

[26] A. Ramachandran, K. Bhandankar, M. Tariq, and N. Feamster, "Packets with provenance," Georgia Tech, Tech. Rep. GT-CS-08-02, 2008.

[27] W. Zhou, M. Sherr, T. Tao, X. Li, B. Loo, and Y. Mao, "Efficient querying and maintenance of network provenance at internet-scale," in *Proc. of ACM SIGMOD*, 2010, pp. 615–626.

[28] W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. Loo, and M. Sherr, "Secure network provenance," in *Proc. of ACM SOSP*, 2011, pp. 295–310.

[29] A. Syalim, T. Nishide, and K. Sakurai, "Preserving integrity and confidentiality of a directed acyclic graph model of provenance," in *Proc. of the Working Conf. on Data and Applications Security and Privacy*, 2010, pp. 311–318.

[30] N. Vijayakumar and B. Plale, "Towards low overhead provenance tracking in near real-time stream filtering," in *Proc. of the Intl. Conf. on Provenance and Annotation of Data (IPAW)*, 2006, pp. 46–54.

[31] S. Chong, C. Skalka, and J. A. Vaughan, "Self-identifying sensor data," in *Proc. of IPSN*, 2010, pp. 82–93.

[32] S. Sultana, M. Shehab, and E. Bertino, "Secure provenance transmission for streaming data," *IEEE TKDE*, 2012.

[33] R. Laufer, P. Velloso, D. Cunha, I. Moraes, M. Bicudo, M. Moreira, and O. Duarte, "Towards stateless single-packet ip traceback," in *Proc. of IEEE LCN*, 2007, pp. 548–555.

[34] P. Jokela, A. Zahemszky, C. Esteve, S. Arianfar, and P. Nikander, "Lipsin: line speed publish/subscribe inter-networking," in *Proc. of ACM SIGCOMM*, 2009, pp. 195–206.

[35] A. Ghani and P. Nikander, "Secure in-packet bloom filter forwarding on the netfpga," in *Proc. of the European NetFPGA Developers Workshop*, 2010.

Salmin Sultana is pursuing a PhD in Computer Engineering at Purdue University. Her research interests include data provenance, security and fault tolerance of distributed systems. She is an ACM student member and member of the Center for Education and Research in Information Assurance and Security (CERIAS).

Dr. Gabriel Ghinita is an Assistant Professor with the Dept. of Computer Science, University of Massachusetts Boston. His research interests focus on privacy-preserving transformation of microdata, private queries in location based services and privacy-preserving sharing of sensitive datasets. Dr. Ghinita serves as reviewer for top journals and conferences such as IEEE TPDS, IEEE TKDE, IEEE TMC, VLDBJ, VLDB, WWW, ICDE and ACM SIGSPATIAL GIS.

Dr. Mohamed Shehab is an associate professor in the Department of Software and Information Systems, College of Computing and Informatics, University of North Carolina at Charlotte. He is the director of the Lab of Information Integration Security and Privacy. His research interests lie in network and information security, in the design and implementation of distributed access-control protocols to cope with the requirements of emerging distributed social networks, mobile applications, web services, and peer-to-peer environments. He is the recipient of the Google Research Award, in addition his research has been funded by US National Science Foundation (NSF), DOD, and Google. He has maintained an active leadership roles in access control and privacy by serving in conference and workshop organizing roles, and on program committee of premiere security and privacy conferences. He was a guest editor for IEEE Internet Computing Special Issue (Security and Privacy in Social Networks). Shehab received a PhD in computer engineering from Purdue University. He is a member of the IEEE Computer Society and the ACM.

Dr. Elisa Bertino is Professor of Computer Science at Purdue University, and serves as Research Director of the Center for Education and Research in Information Assurance and Security (CERIAS) and Director of Cyber Center (Discovery Park). Previously, she was a faculty member and department head at the Department of Computer Science and Communication of the University of Milan. Her main research interests include security, privacy, digital identity management systems, database systems, distributed systems, and multimedia systems. She served as editor in chief of the VLDB Journal and editorial board member of ACM TISSEC and IEEE TDSC, and will be serving as editor in chief of IEEE TDSC starting from January 2014. She coauthored the book Identity Management - Concepts, Technologies, and Systems. She is a fellow of the IEEE and a fellow of the ACM. She received the 2002 IEEE Computer Society Technical Achievement Award for outstanding contributions to database systems and database security and advanced data management systems and the 2005 IEEE Computer Society Tsutomu Kanai Award for pioneering and innovative research contributions to secure distributed systems.