Secure Data Provenance Compression Using Arithmetic Coding in Wireless Sensor Networks

Syed Rafiul Hussain*, Changda Wang†, Salmin Sultana*, and Elisa Bertino*
* Department of Computer Science, Purdue University
Email: {hussain1, ssultana, bertino}@purdue.edu
†School of Computer Science and Communication Engineering, Jiangsu University
Email: changda@ujs.edu.cn

***Abstract*—Since data are originated and processed by multiple agents in wireless sensor networks, data provenance plays an important role for assuring data trustworthiness. However, the size of the provenance tends to increase at a higher rate as it is transmitted from the source to the base station and is processed by many intermediate nodes. Due to bandwidth and energy limitations of wireless sensor networks, such increasing of provenance size slows down the network and depletes the energy of sensor nodes. Therefore, compression of data provenance is an essential requirement. Existing lossy compression schemes based on Bloom filters or probabilistic packet marking approaches have high error rates in provenance-recovery. In this paper, we address this problem and propose a distributed and lossless arithmetic coding based compression technique which achieves a compression ratio higher than that of existing techniques and also close to Shannon's entropy bound. Unlike other provenance schemes, the most interesting characteristic of our scheme is that the provenance size is not directly proportional to the number of hops, but to the occurrence probabilities of the nodes that are on a packet's path. We also ensure the confidentiality, integrity, and freshness of provenance to prevent malicious nodes from compromising the security of data provenance. Finally, the simulation and testbed results provide a strong evidence for the claims in the paper.**

Keywords*—*Wireless sensor network, provenance, compression, arithmetic coding, security.

I. Introduction

Wireless sensor networks (WSN) are the backbone of many critical systems, such as cyber-physical systems, health and environmental monitoring, weather forecasting, and surveillance. In these application domains, data sources vary from miniature body-worn sensors to external sensors, e.g., video cameras, positioning devices etc. Such diversity of data sources necessitates the assurance of data trustworthiness so that only reliable information is provided to applications. As an example, consider battlefield surveillance systems and mission critical applications which need high confidence data in order to support accurate decisions. Since provenance summarizes the history of the ownership of data and the actions performed on these, it is an effective tool for evaluating data trustworthiness. Recent research [1] highlights the key contribution of provenance in systems (e.g., SCADA systems for critical infrastructure) where the use of untrustworthy data may lead to wrong control decisions.

In a multi-hop sensor network, data provenance allows the base station to trace the source and forwarding sensor nodes of an individual data packet since its generation. To ensure data quality and trustworthiness, it is crucial to record the provenance of each data packet, including information about each node in the data flow path. However, energy and bandwidth limitations, tight storage, and resource constraints of sensor nodes make the collection of data provenance challenging. Hence, several lightweight data provenance schemes have been proposed for WSN [2]–[7]. In these schemes, data provenance is represented as a directed graph, where each vertex represents the provenance record of a node that is on the data flow path and each edge indicates the direction of data transmission between two nodes. One major issue with these schemes is that the provenance size increases with the number of nodes in data flow path. As a result, per-packet provenance transmission in sensor networks incurs bandwidth and energy exhaustion. To reduce the provenance size, some approaches [5], [7] adopt lossy compression schemes that however drop critical information while compressing provenance record. Some of them do not even include the edges that indicate directed connections among sensor nodes and thus fail to provide accurate packet path topologies. Therefore, it is necessary to devise a lightweight provenance solution that provides lossless provenance without introducing any significant overhead. Furthermore, the environments where sensor networks are deployed are often untrusted and sensors 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 an efficient provenance encoding and decoding mechanism that is able to compress the provenance as much as possible while assuring its security.

Current research focuses mostly on the provenance of the workflow, its modeling, querying, curated databases [8], [9], leaving the security issues of the path provenance unexplored. In this paper, we propose a secure, distributed and lossless provenance compression scheme where each node on a packet's data flow path encodes its provenance record using arithmetic coding [10]. Upon receiving a packet, the base station decodes the provenance and verifies its validity.

Existing approaches use fixed size data structures [6], [7] or cryptographic operations [11] that result in high decoding error rates, or in unconstrained provenance expansion or in provenance data loss. By contrast, our scheme is lossless and achieves a compression ratio close to Shannon's theoretical entropy bound for provenance. Unlike other approaches, the provenance size in our scheme is not directly proportional to the number of hops, but depends only on the occurrence probabilities of nodes that appear on a packet's path. The

978-1-4799-7575-4/14/$31.00 ©2014 IEEE

larger the probabilities are, the smaller the provenance size is. Thus, our scheme can save more energy and bandwidth while transmitting the provenance across the network. In addition, we ensure provenance security, i.e., confidentiality, integrity and freshness. Our specific contributions are:

- An arithmetic coding based, distributed, and lossless provenance encoding mechanism for WSN. Our scheme also supports data aggregation.
- An efficient technique for provenance decoding and verification at the base station.
- A secure mechanism for assigning and sharing the occurrence probabilities of a particular node that are used in arithmetic coding. Our mechanism ensures confidentiality so that no malicious node can decode the provenance information of other nodes.
- A secure packet sequence number generation mechanism to be used along with AM-FM sketch technique [12] to ensure the integrity and freshness of the provenance.
- A detailed security analysis of our provenance scheme and its performance evaluation through theoretical analysis, simulation and testbed experiments.

The rest of the paper is organized as follows: Section II introduces the system model and relevant preliminary concepts. Section III gives an overview of our approach. Section IV and V describe our proposed encoding, decoding, and binding mechanisms for simple and aggregated provenance, respectively. Section VI discusses implementation details and network dynamics. Section VII analyzes the performances of our scheme. Section VIII provides a detailed security analysis. Section IX and X present the simulation and experimental evaluation of our protocol, respectively. Section XI surveys related work and Section XII concludes the paper.

II. System Model and Background

We introduce our system model in this section, i.e, network, data, provenance and adversary models considered in this paper. We also provide a brief primer on arithmetic coding based data compression mechanism.

A. Network Model

We consider a multi-hop sensor network, consisting of a number of nodes and a base station (BS) that collects data from the network. The BS has no constraints with respect to energy, storage space, security, and computational capability. The network is modeled as a directed graph, $G(N, L)$, where $N = \{n_i | 1 \leq i \leq |N|\}$ is the set of nodes and $L = \{l_{ij} | 1 \leq i, j \leq |N|\}$ is the set of links between nodes. $|N|$ denotes the cardinality of set N and l_{ij} denotes a directed edge from node n_i to n_j. Sensor nodes are stationary after deployment, but routing paths may change due to node failure, link quality degradation, resource optimization etc. However, nodes can also be dynamic provided that their mobility is predictable.

The BS assigns each node a unique ID, n_i, a packet counter, $count_i$, a symmetric cryptographic key, k_i, and a seed, s_i, that is used for random number generation. Also, the BS itself maintains a packet counter, $count'_i$, for each node n_i. Initially, $count_i$ and $count'_i$ are both set to 0. When node n_i generates a packet d, it increments the value of $count_i$ by 1. Upon receiving the packet d, the BS increments the n_i specific counter $count'_i$ by 1 in order to synchronize.

B. Data Model

Assume that some of the sensor nodes in a network generate data periodically. We refer to these nodes as *source nodes*. A node may receive data from another node and forward it to the one that appears next on the path. We call this node a *forwarder node*. If a node aggregates data received from multiple nodes, it forwards the aggregated packet to the next node. We refer to such a node as *aggregator node*. Each data packet contains: (i) a unique sequence number, (ii) source node information, (iii) data value, (iv) provenance, and (v) message authentication code (MAC). Any existing routing protocol can be used to transmit data from the source to the BS. The path traversed by a data packet forms a tree rooted at the BS.

In our model, a packet d, generated at node n_i has a sequence number seq computed as $E_{k_i}(count_i \mid\mid t)$, where E is an encryption function, $||$ denotes concatenation and t is the timestamp of d's generation. *Source node information* is computed as $E_{k_i}(n_i \mid\mid count_i)$. Since the BS knows each node's current $count_i$, it maintains a list of such $|N|$ encryptions, i.e., what each node may send as the *source node information* if it generates a packet. Upon receiving a packet, the BS retrieves the *source node* by comparing the received *source node information* with the listed items. As k_i is secret, only the BS can determine the source of the packet. Also, no node other than the BS can tell if two packets are from the same source because the *source node information* changes with the increment of $count_i$. The MAC included in the packet is not used in provenance encoding or decoding, and thus is not a part of the provenance. It is used only to ensure the provenance's integrity.

C. Provenance Model

The definition of provenance varies with different application domains. In the context of WSN, provenance refers to where a data packet is generated and how it is delivered to the BS [6], [7]. In this paper, we consider *node-level provenance* which encodes the nodes that are involved at each step of data processing. Given a data packet d, the provenance, p_d, is modeled as a directed acyclic graph $G(V, E)$. Each vertex $v_x \in V$, where $1 \leq x \leq |V|$, is attributed to a specific node n_i and represents the provenance record (i.e., node ID) for that node. We refer to this relation as $HOST(v_x) = n_i$, i.e., node n_i is the host of v_x. Each edge $e_{xy} \in E$ represents a directed edge from vertex v_x to v_y, where $1 \leq y \leq |V|$. As in [6], [7], we formally define data provenance as follows:

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_x, $v_y \in V$, v_y is a child of v_x if and only if $HOST(v_y)$ forwards d to $HOST(v_x)$; (3) $U \subset V$ is a set of children of $v_x \in V$ if and only if for each $v_y \in U$, $HOST(v_x)$ receives forwarded data from $HOST(v_y)$.

Fig. 1 shows two different kinds of provenance. In Fig. 1(a), data are generated at leaf node n_3 and are forwarded by the internal nodes towards the BS. We call such provenance *simple provenance* and represent it as a path $\langle n_3, n_2, n_1 \rangle$. In Fig. 1(b), data are aggregated while being forwarded towards the BS, i.e., the *aggregator node* n_3 generates a new data packet by aggregating data from n_5 and n_6, and then passes

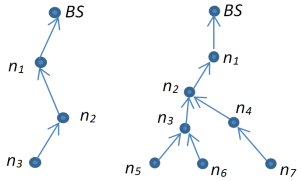


Fig. 1: (a) Simple provenance (b) Aggregated provenance

the new packet to node n_2. Here, the provenance is called *aggregated provenance* and is represented as a tree $\langle((((n_5)(n_6)), n_3)(n_7, n_4)), n_2), n_1\rangle$. A recursive representation $\langle((b)(c)), a\rangle$ is used to denote a tree topology, where a denotes the root, (b) and (c) denote the left and right subtrees, respectively.

D. Adversary Model

During the network operations, every node but the BS may be compromised by adversaries. An adversary can eavesdrop in the network and collect confidential information by means of packet sniffing, traffic analysis etc. It can compromise legitimate nodes and extract critical information such as keys, codes, or data. It may also use these nodes as malicious ones to perform attacks cooperatively, e.g., providing false route information, dropping, injecting or altering packets. The BS cannot distinguish these compromised nodes from the benign ones. However, the BS needs to be assured about the confidentiality, integrity, and freshness of provenance so that it can secure the trustworthiness of data provenance in presence of potential attackers. Our objective is to achieve the following security properties:

- **Confidentiality:** An adversary cannot learn any information about data provenance by analyzing the contents of the packet.
- **Integrity:** (i) An adversary cannot inject counterfeit information in the provenance or remove any benign node. (ii) A malicious node cannot drop packets without being identified.
- **Freshness:** Packet replay attacks are detectable.

E. Arithmetic Coding

Arithmetic coding [10], [13] is a lossless data compression technique that assigns short codewords to more probable data symbols and longer codewords to less probable ones. Each codeword is represented by a half-open subinterval of the half-open unit interval $[0, 1)$. Enough bits are used for a codeword to distinguish the corresponding subinterval from all other possible subintervals. For illustration, let the number of unique symbols be M. We assume that the symbols are independent and identically distributed with *occurrence probability*, op_m ($\neq 0$), and *cumulative occurrence probability*, cp_m.

$$\sum_{m=1}^{M} op_m = 1$$
$$cp_m = \sum_{i=1}^{m} op_i$$

where $m = 1, 2, \ldots, M$.

1) Encoder: For a given string of k random symbols, $S = \{s_1, s_2, \ldots, s_k\}$, the arithmetic coding process creates a sequence of nested intervals of the form $[L_x, H_x)$, where $x = 1, 2, \ldots, k$ and L_x, H_x are real numbers such that $0 \leq L_x \leq L_{x+1}, L_x \leq H_x$, and $H_{x+1} \leq H_x$. The intervals are computed using the following recursive formulas:

$$[L_0, H_0) = [0, 1)$$
$$H_x = L_{x-1} + cp_{s_x} \times (H_{x-1} - L_{x-1})$$
$$L_x = L_{x-1} + (cp_{s_x} - op_{s_x}) \times (H_{x-1} - L_{x-1})$$

where s_x is the xth symbol in S, and op_{s_x}, cp_{s_x} are its corresponding *occurrence* and *cumulative occurrence probabilities*.

The final outcome of the compression of S, denoted as $[L_f, H_f)$, is the kth subinterval $[L_k, H_k)$.

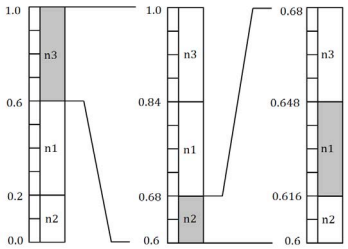


Fig. 2: Example of arithmetic coding

Fig. 2 shows an example of arithmetic encoding procedure for the simple provenance in Fig. 1(a). To conform with arithmetic coding, each node in the network is considered as a symbol and the node *occurrence probabilities* are chosen arbitrarily. The encoding procedure starts by dividing the half-open unit interval $[0, 1)$ into $M = 3$ half-open subintervals: $[0, 0.2)$ for n_2, $[0.2, 0.6)$ for n_1 and $[0.6, 1)$ for n_3. As n_3 is the first node on the path, its interval $[0.6, 1)$ is further divided into $M = 3$ subintervals $[0.6, 0.68)$, $[0.68, 0.84)$, and $[0.84, 1.0)$. Note that the ratios of the subintervals are same as the original *cumulative occurrence probabilities*. In second iteration, the next node n_2 is found within the subinterval $[0.6, 0.68)$. So, this subinterval now gets divided into $[0.6, 0.616)$, $[0.616, 0.648)$, and $[0.648, 0.68)$. Finally, the last node n_1 falls into the subinterval $[0.616, 0.648)$ and thereby encodes the provenance $\langle n_3, n_2, n_1\rangle$ as $[0.616, 0.648)$.

2) Decoder: The decoding process recovers the data symbols in the same sequence as they are encoded. Any number in the $[L_f, H_f)$ range can be used to retrieve the first symbol of S. We denote this number as $code_1$. Every time a symbol is retrieved, its effect is removed from the current $code$ value in order to recover the next symbol. Here, op_{x-1}, and cp_{x-1} are the *occurrence* and *cumulative occurrence probabilities* of $(x-1)$th decoded symbol, respectively.

$$code_1 = \frac{(L_f + H_f)}{2}$$
$$code_x = \frac{code_{x-1} - (cp_{x-1} - op_{x-1})}{op_{x-1}}$$

To illustrate the decoding process, we continue the example of Section II-E1. When the BS receives the interval $[0.616, 0.648)$, it computes $code_1$ as 0.632. As this value resides in the interval $[0.6, 1)$, the first node n_3 is decoded. From the above equation, $code_2$ results to be 0.08 that falls in $[0, 0.2)$ and thus decodes node n_2. Finally, the value of $code_3$, 0.4, reveals node n_1 as the third node. As soon as the BS retrieves n_1, it stops decoding because n_1 is the node from which it has received the packet. So, the BS decodes the provenance of the packet as $\langle n_3, n_2, n_1\rangle$.

III. Overview of Our Approach

We develop a distributed mechanism to encode provenance at sensor nodes and a centralized approach to decode that at the BS. Each node on the data flow path encodes its provenance by exploiting arithmetic coding and thus achieves a lossless provenance compression. When the BS receives the encoded provenance, it decompresses the *code* to retrieve the nodes that are on the path.

Determining occurrence probabilities: To determine the *occurrence probabilities* of sensor nodes, the BS observes the network for a specific time period which we call *training phase*. During this period, the BS counts for each node n_i the number of times it appears on the packets' paths which we represent as *occurrence frequency*, of_i, of n_i. From these of_i values, the BS also computes the *total occurrence frequency*, of, and for each node n_i its *occurrence probability*, op_i.

$$of = \sum_{i=1}^{|N|} of_i$$
$$op_i = \frac{of_i}{of}$$

The *cumulative occurrence probability*, cp_i, for each node n_i is calculated according to the following equations:

$$cp_0 = 0$$
$$cp_i = cp_{i-1} + op_i$$

For security purpose, the BS shuffles the probability intervals in order to randomize their positions in the total probability range. The details are discussed in Section VIII.

Upon receiving each packet in the *training phase*, the BS examines the packet's path and learns which node appears after another on the path. At the end of *training phase*, the BS knows the total count of n_i's appearance following the node n_j on packets' paths where $j = 1, 2, \ldots, |N|$ and $j \neq i$. We refer to this frequency as *association frequency* of node n_i with n_j and denote it with of_{ij}. Note that the *total association frequency*, $\sum_{i=1}^{|N|} of_{ij}$ of node n_j is equal to its *occurrence frequency* of_j because the number of times n_j appears on packets' paths is equal to the number of times the other nodes receive packets from it. Using these of_{ij} values, the BS computes for each node n_i its *association probability*, op_{ij}, where n_j is a child of node n_i.

$$op_{ij} = \frac{of_{ij}}{of_j}$$

The *cumulative association probability*, cp_{ij}, for each node n_i is calculated according to the following equations:

$$cp_{0j} = 0$$
$$cp_{ij} = cp_{(i-1)j} + op_{ij}$$

After computing the probabilities as discussed above, the BS informs node n_i of its op_i, cp_i, and for each of its children n_j, the probabilities op_{ij}, and cp_{ij}.

Encoding at sensor nodes: In our proposed scheme, the provenance encoding procedures in *source node* and *forwarder node* are different.

• Source node: If node n_i generates a packet, it uses its op_i, and cp_i to compute the interval $[L_1, H_1)$ according to the equations in Section II-E.

$$[L_0, H_0) = [0, 1)$$
$$H_1 = L_0 + cp_i \times (H_0 - L_0)$$
$$L_1 = L_0 + (cp_i - op_i) \times (H_0 - L_0)$$

• Forwarder node: If node n_i is the xth node on a packet's path, and receives the provenance $[L_{x-1}, H_{x-1})$ from the $(x-1)$th node n_j, it uses op_{ij}, and cp_{ij} to compress the provenance into the range $[L_x, H_x)$ according to the following equations:

$$H_x = L_{x-1} + cp_{ij} \times (H_{x-1} - L_{x-1})$$
$$L_x = L_{x-1} + (cp_{ij} - op_{ij}) \times (H_{x-1} - L_{x-1})$$

Decoding at the BS: When the BS receives a packet, it gets the final encoded interval $[L_f, H_f)$. The BS then decodes the provenance in a similar way discussed in Section II-E by using the probability values saved in its storage. The details of the decoding process are discussed in Section IV-B.

IV. Scheme for Simple Provenance

In this section, we discuss in detail the encoding, decoding and binding schemes for *simple provenance*.

A. Provenance Encoding

Upon generating or receiving a data packet, each node n_i adds a vertex to the provenance tree of that packet. Our

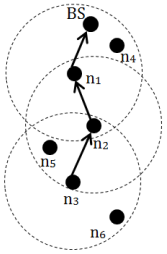


Fig. 3: Network topology

provenance compression scheme based on arithmetic coding is presented in Algorithm 1. Here, input $[L_{x-1}, H_{x-1})$ represents the encoded provenance from the *source node* to $(x-1)$th node on the packet's path. For the *source node*, the input $[L_0, H_0)$ is $[0, 1)$. Output $[L_x, H_x)$ denotes the encoded provenance from the *source node* to the current encoding node n_i.

We use floating point numbers in our algorithm to represent probability values and intervals. From the given example in Section II-E1, it can be seen that the $[low, high)$ intervals shrink at each iteration and require the precision to increase. Since floating point number supports a limited number of bits to represent the digits after its decimal point, it cannot accommodate this increasing precision. To avoid this situation, we use a *buffer*. Whenever the most significant digits of low and $high$ match, we shift out those from both of the variables to make space for future precisions. The shifted out digits are then saved in the *buffer*. For example, if low and $high$ values are respectively 0.616 and 0.648 at some point, we shift out 6 and save it in the *buffer* which results the new low and $high$

Algorithm 1 Provenance encoding

1: **Input**: $[L_{x-1}, H_{x-1})$
2: **Output**: $[L_x, H_x)$
3: $low = L_{x-1}$, $high = H_{x-1}$
4: $range = high - low$
5: *buffer* $= \emptyset$
6: **if** n_i is a *source node* **then**
7: $high$ = $low + cp_i \times range$
8: low = $low + (cp_i - op_i) \times range$
9: **else if** n_i is a *forwarder node* receiving packet from n_j **then**
10: $high$ = $low + cp_{ij} \times range$
11: low = $low + (cp_{ij} - op_{ij}) \times range$
12: **end if**
13: **while** MSD(low) = MSD($high$) **do**
14: /*MSD(x) returns the most significant digit of x*/
15: *buffer* = *buffer* $\bigcup MSD(low)$
16: low = ShiftL(low, 1)
17: $high$ = ShiftL($high$, 1)
18: /*ShiftL(x, y) function returns x shifted by y digits in the left*/
19: **end while**
20: L_x = low
21: H_x = $high$

TABLE I: Occurrence and cumulative occurrence probabilities

Nodes	op_i	cp_i	Range
n_1	0.35	0.35	$[0, 0.35)$
n_2	0.25	0.6	$[0.35, 0.6)$
n_3	0.15	0.75	$[0.6, 0.75)$
n_4	0.12	0.87	$[0.75, 0.87)$
n_5	0.08	0.95	$[0.87, 0.95)$
n_6	0.05	1	$[0.95, 1)$

to be 0.16 and 0.48, respectively. This part of the algorithm is presented from line 13 to 19 in Algorithm 1.

Here, we explain our *provenance encoding* procedure with a detailed example. To demonstrate how the algorithm works, we introduce some new nodes in the *simple provenance* shown in Fig. 1(a). The modified topology along with some new nodes n_4, n_5 and n_6 is shown in Fig. 3. After the *training phase*, let the *occurrence probabilities* and *cumulative occurrence probabilities* of nodes n_1 through n_6 be as shown in Table I.

In Table II, we present the *association probabilities* with respect to the nodes n_1, n_2 and n_3 only, because the packet we consider for the example traverses the path $\langle n_3, n_2, n_1 \rangle$. From the table, for instance, we can say that if node n_1 transmits a packet, the probability that the packet is received by node n_2 is 0.1. The *cumulative association probabilities* according to Table II are shown in Table III.

In our example, we assume that n_3 is the *source node*. Since the *occurrence probability* range of n_3 is $[0.6,\ 0.75)$ (see Table I), n_3 sends these values as $[L_1,\ H_1)$ to the next node. When node n_2 receives this provenance, it uses its *association probability* and *cumulative association probability* with respect to n_3 to compute $[L_2,\ H_2)$. According to our algorithm, n_2 transmits $[0.6, 0.72)$ as provenance to n_1. Upon receiving this, n_1 similarly uses its *association probability* and *cumulative association probability* with respect to n_2 and computes $[L_3, H_3)$ as $[0.6, 0.696)$. This is the first time the most significant digits in low and $high$ are same. So '6' is

TABLE II: Association probabilities

Nodes	n_1	n_2	n_3	n_4	n_5	n_6	BS
n_1	–	0.1	0	0.2	0	0	0.7
n_2	0.8	–	0.1	0	0.1	0	0
n_3	0	0.8	–	0	0.2	0	0

TABLE III: Cumulative association probabilities

Nodes	n_1	n_2	n_3	n_4	n_5	n_6	BS
n_1	–	.1	.1	.3	.3	.3	1
n_2	.8	–	.9	.9	1	1	1
n_3	0	.8	–	.8	1	1	1

shifted out to *buffer* and the updated $[L_3, H_3)$ is $[0, 0.96)$. The BS then receives $[0, 0.96)$ as the final interval and $\{6\}$ in the *buffer*.

Note that, arithmetic coding as described in Section II-E1 uses only *occurrence probabilities* to encode provenance at each node along the path and thus increases the size of the *buffer* at a higher rate. This is because it considers all possible nodes in the network at each step and, as a result, shrinks the $[low, high)$ interval faster. For example, in the above scenario, if node n_2 uses its op_2 and cp_2 values to encode its provenance after receiving $[0.6, 0.75)$ from n_3, the new interval is $[0.6525, 0.69)$ which immediately shifts out '6' to the *buffer*. But in this case, our proposed approach stores the value in *buffer* one step later and thus reduces the *buffer* overhead. For this significant advantage, we introduce *association probability* while encoding provenance at *forwarder nodes*.

Algorithm 2 Provenance decoding

1: **Input**: $[L_f, H_f)$, *buffer*
2: **Output**: Provenance P
3: low = L_f
4: $high$ = H_f
5: **if** *buffer* $= \emptyset$ **then**
6: $code$ = $\frac{(low+high)}{2}$
7: **else**
8: $code$ = ShiftR(*buffer* + $\frac{(low+high)}{2}$, $|$*buffer*$|$)
9: /*ShiftR(x, y) function returns x shifted by y digits in the right*/
10: **end if**
11: **for** $i = 1$ to $|N|$ **do**
12: **if** $(cp_i - op_i) \leq code < cp_i$ **then**
13: $source$ = n_i
14: break
15: **end if**
16: **end for**
17: dc = $source$ /*source decoded*/
18: P = $P \bigcup dc$
19: R_{dc} = $op_{(dc)}$ /*probability range of decoded*/
20: L_{dc} = $cp_{(dc)}$ - $op_{(dc)}$ /*lower end of probability range*/
21: **while** $dc \neq$ the node from which BS received packet **do**
22: /*remove effect of decoded node*/
23: $code$ = $\frac{(code - L_{dc})}{R_{dc}}$
24: **for** $i = 1$ to $|N|$ **do**
25: **if** $(cp_{i(dc)} - op_{i(dc)}) \leq code < cp_{i(dc)})$ **then**
26: P = $P \bigcup n_i$
27: L_{dc} = $cp_{i(dc)}$ - $op_{i(dc)}$
28: R_{dc} = $op_{i(dc)}$
29: dc = n_i
30: break
31: **end if**
32: **end for**
33: **end while**

B. Provenance Decoding

When the BS receives a packet, it decompresses the provenance using the encoded interval $[L_f,\ H_f)$ and *buffer*.

According to the example in Section IV-A, the BS receives $[0.0,\ 0.96)$ as the final interval $[L_f,\ H_f)$ and $\{6\}$ in *buffer*. At first, it decodes the *source node information* as described in Section II-B. So, it learns that n_3 is the *source node*. The BS then computes $code_1$ as 0.648 by taking the sum of 6 and mid of $[0.0,\ 0.96)$ and shifting the sum $|$*buffer*$|$ times right, where $|$*buffer*$|$ denotes the number of digits shifted into *buffer*. As node n_3's probability interval is $[0.6, 0.75)$, it verifies the source node to be n_3.

Every time the BS decodes a node, that node's effect is removed from $code$ value. So, after removing n_3's effect, $code_2$ is computed as 0.32. In Table III, it can be seen that n_2's probability interval in terms of receiving packet from n_3 is $[0,\ 0.8)$. So, n_2 is decoded. Removing n_2's effect results $code_3$ to be 0.4. As the *association probability* interval of n_1 with n_2 is $[0,\ 0.8)$, n_1 is also decoded. At this point, the BS ends decoding because it receives packet from n_1.

C. Provenance Binding

In order to defend attacks by adversaries, any unauthorized modification of packet content or its associated provenance record needs to be detected. Simply encrypting the packet content or its provenance is not feasible due to the high cost of encryption. Another approach is to bind together the packet and its provenance by MAC and then encrypt the MAC with the private key of each node on the packet's path. Traditional MAC schemes, e.g., MD5 or SHA-1, are designed for centralized scenarios. In such cases, each node in the packet's path generates an independent MD5 or SHA-1. If we chain all the MACs together, the combined size of the resultant MAC increases in proportion to the number of nodes on the path. Such a MAC chain incurs extra requirements of energy and channel bandwidth.

To address this issue, we adopt a provenance binding technique, a direct application of AM-FM proof sketch [12], which constructs a verifiable random sample of size k over the sensor's data values and ensures that the result computed by the aggregators is verifiably unbiased. With such distributed message digest mechanism, we bind a packet and its provenance together at each node along its path. This approach restrains the MAC size from growing beyond a range $[(1-\varepsilon)k, (1+\varepsilon)2k]$, where ε $(\varepsilon < 1)$ is the false positive rate related to k. When AM-FM sketch digests infinite number of messages into a finite data set, collisions may occur which result in false positives. However, any unauthorized modification of packet content or provenance can be detected with a certain statistical confidence by using AM-FM sketch if the false positive rate is insignificant.

V. Scheme for Aggregated Provenance

For aggregated provenance as shown in Fig. 1(b), node n_3 needs to combine its own provenance record with the provenance intervals received from nodes n_5 and n_6. Assume that the intervals received from two branches are $[L_{b_1},\ H_{b_1})$ and $[L_{b_2},\ H_{b_2})$, respectively. The solution that we propose for *aggregated provenance* is as follows:

Provenance encoding: Node n_3 encodes itself into both $[L_{b_1},\ H_{b_1})$ and $[L_{b_2},\ H_{b_2})$ to create the new ranges $[L'_{b_1},\ H'_{b_1})$ and $[L'_{b_2},\ H'_{b_2})$, respectively. It then chooses real numbers $r_1 \in [L'_{b_1},\ H'_{b_1})$ and $r_2 \in [L'_{b_2},\ H'_{b_2})$, and sends $(r_1)(r_2)$ along with the interval $[L'_{b_1},\ H'_{b_1})$ to the next hop. The encoding procedure is presented in Algorithm 3 where $\mathcal{A}$ is the number of branches of an *aggregator node* n_i.

Algorithm 3 Aggregated provenance encoding

1: **Input**: $[L_{b_1}, H_{b_1})$, $[L_{b_2}, H_{b_2})$,..., $[L_{b_\mathcal{A}}, H_{b_\mathcal{A}})$
2: **Output**: $(r_1)(r_2)\ldots(r_\mathcal{A})[L_b, H_b)$
3: **for** x=1 to $\mathcal{A}$ **do**
4: n_i encodes itself with $[L_{b_x}, H_{b_x})$ using Algorithm 1 and results $[L'_{b_x}, H'_{b_x})$
5: n_i chooses a real number $r_x \in [L'_{b_x}, H'_{b_x})$
6: **end for**
7: $[L_b, H_b)$ = $[L'_{b_1}, H'_{b_1})$
8: prepend $r_1, r_2, \ldots r_\mathcal{A}$ to $[L_b, H_b)$

Provenance decoding: When the BS receives the aggregated packet, it first decodes the interval $[L_b, H_b)$ according to Algorithm 2. The (r_1) and (r_2) values are then decoded until they reach a node on the path retrieved from $[L_b, H_b)$, i.e., until they reach the *aggregator node* n_3. The BS then builds a subtree rooted at n_3 along with the paths decoded from $(r_1), (r_2)$ and thus retrieves the *aggregated provenance*.

Algorithm 4 Aggregated provenance decoding

1: **Input**: $(r_1)(r_2)\ldots(r_\mathcal{A})[L_b, H_b)$
2: **Output**: Aggregated provenance AP
3: P = decode $[L_b, H_b)$ using Algorithm 2
4: $AP = AP \bigcup P$
5: **for** i=1 to $\mathcal{A}$ **do**
6: let $code = r_i$
7: P = decode $code$ using Algorithm 2 until it reaches a node on the path retrieved from $[L_b, H_b)$
8: $AP = AP \bigcup P$
9: **end for**

Provenance binding for aggregated scenario is similar to that of *simple provenance*.

VI. Implementation Details

For the purpose of simplicity, we use floating point numbers to explain our algorithms. But in our actual implementation, we use *integer arithmetic coding* [10] which is the most practical and efficient solution to manage the *buffer* and precision requirements. The main idea remains same, but we scale the initial $[L_0, H_0)$ range from $[0, 1)$ to $[0,\ 2^{16}-1)$ as we use 2-byte integers. Also, we consider *occurrence frequencies* instead of *occurrence probabilities* for provenance encoding and decoding.

Two exceptional conditions, *overflow* and *underflow* still may arise while doing *integer arithmetic coding*. *Overflow* occurs when $[low, high)$ values become greater than the capacity of integer variables. And underflow occurs when scaling of the current range causes several nodes to be mapped into the same range due to the closeness of $high$ and low values. To prevent *underflow*, we cannot use more than 14 bits to represent *cumulative occurrence frequencies* [10]. And *overflow* can be prevented by using a 4-byte size long variable as $code$.

Handling network update: As mentioned in Section II-A, we assume that sensor nodes are stationary after deployment.

But the routing paths may change over time due to node failure, link quality degradation, resource optimization etc. To handle such changes of routing paths as well as the topology, after a certain amount of network operations (e.g., after receiving every 1000 packets), the BS informs all the sensor nodes in the network about their updated *occurrence probabilities* and *association probabilities*. So, the model we use for arithmetic coding in our proposed provenance scheme is *semi-adaptive*.

VII. Performance analysis

In this section, we analyze the performance of our scheme by computing the entropy of provenance records. We also present the time and space complexities for provenance encoding and decoding.

Entropy of simple provenance: To encode the provenance, the number of bits required in our scheme by a *source node* n_j is $H_j = \lceil -\log op_j \rceil$, and by a *forwarder node* n_i is $H_{ij} = \lceil -\log op_{ij} \rceil$, where n_j is the child of n_i. Hence, the entropy of the arithmetic coding can be derived from the number of bits required to represent an interval r, i.e., $H_r = \lceil -\log r \rceil$. Note that the size of the final interval is equal to the product of the *source node*'s *occurrence probability* and *forwarder nodes'* corresponding *association probabilities*:

$$r = op_j \times \prod_{l_{ji} \in E} op_{ji}$$

So, H_r can be computed as follows:

$$H_r = \left\lceil -\log(op_j \times \prod_{l_{ji} \in E} op_{ji}) \right\rceil = \left\lceil -\log op_j - \sum_{l_{ji} \in E} \log op_{ji} \right\rceil$$

Thus, the provenance size of a packet is directly proportional to r. In other words, it depends on the *occurrence probabilities* of the nodes that are on a packet's path, but not on the number of nodes.

Entropy of aggregated provenance: *Aggregator nodes* incur overhead by prepending a real number for each of their branches. Assume that there are R *aggregator nodes* and each *aggregator node* $n_r^\star$ has b_r ($b_r > 1$) merged-in branches, where $1 \leq r \leq R$. So, the number of extra bits required for all *aggregator nodes* are:

$$-\sum_{r=1}^{R} \sum_{q=1}^{b_r - 1} \lceil \log op_{n_r^\star j} \rceil$$

where $op_{n_r^\star j}$ is the *association probability* of node $n_r^\star$ with respect to its child node n_j at the branch q.

Time complexity: While encoding at node n_i, the $[low, high)$ values can be computed in $O(1)$ time. However, shifting out similar most significant bits of low and $high$ requires $O(B)$ operations in the worst case where B is the maximum number of bits used to represent low and $high$. To decode the provenance, the BS successively removes the effect of the last retrieved symbol from $code$ value. It also needs to locate the current $code$ value in the probability ranges of $|N|$ nodes. So, if there are k nodes on a packet's path, the BS requires $O(k \cdot |N|)$ operations. However, if we use an efficient data structure, e.g., hash table at the BS which takes $O(1)$ operation to locate the $code$ value, the time complexity can be reduced to $O(k)$.

Space complexity: A node n_i has to store its *occurrence frequencies* and for each of its child n_j, the *association frequencies*. Therefore, if n_i has a number C of children, its space complexity is $O(C)$. The BS requires $O(|N|^2)$ space to store *occurrence frequencies* and *association frequencies* for all the nodes in the network.

VIII. Security Analysis

We justify the following security properties of our scheme:

Claim 1: An adversary cannot learn any information about the provenance from an encoded interval.

Justification: In our approach, provenance is an encoded interval defined by two real numbers. Now, assume that an adversary learns an encoded interval associated with a packet by eavesdropping in the network. Without knowing the *source node* and its probabilities and also the *association probabilities* of all *forwarder nodes* of that packet, the adversary is unable to decode the provenance. However, if the adversary once becomes successful in guessing a packet's actual path, it may decode the provenance of all subsequent packets that follow the same path since using the fixed probability ranges of *source nodes* results in identical encoded intervals. We prevent such leak of information by introducing randomness in the nodes' *occurrence probability* ranges. As mentioned in Section II-A, the BS shares with each node n_i a seed s_i (encrypted by symmetric key k_i) that generates a series of random numbers. When node n_i generates its pth packet, it computes the pth random number r_p ($0 < r_p < 1$) from the shared seed and uses this r_p as its cp_i value. Since op_i remains same, the change in cp_i does not affect the subsequent encoding other than generating different intervals even for the same paths. Hence, if an attacker eavesdrops in the network and gets encoded intervals of two packets that traverse same path, it cannot identify their paths' similarity. And on the BS side, when the BS receives the pth packet generated from node n_i, it produces the same random number r_p using the seed s_i and replaces the current cp_i value with that one. In this way, without any communication overhead, the BS and node n_i synchronize every time n_i generates a packet and leave cp_i unpredictable for any adversary. Thus, it is not possible for an adversary to extract useful information about the provenance by just seeing the encoded interval.

Claim 2: A malicious node cannot inject counterfeit information in the provenance or remove any benign node.

Justification: An attacker may attempt to generate fake provenance including some innocent nodes to make them responsible for malicious data. However, adding an innocent node into the provenance requires the attacker to know the target node's *association probabilities* and *cumulative association probabilities*. But the BS shares these information with each node in the network secretly using its corresponding key. Also, due to the confidentiality of encoded intervals, a malicious node cannot learn these probabilities for any target node. Hence, it is not possible to inject a benign node in the provenance. Removal of a benign node is more difficult because it requires decoding up to the target node and encoding again excluding the target node.

Claim 3: A malicious node cannot selectively drop packets generated by benign nodes without being detected.

Justification: According to our packet sequence number generation technique as in Section II-B, $count_i$ and $count'_i$ are both set to 0 initially. When node n_i generates a new packet, $count_i$ is incremented by 1 and when the packet arrives at the BS, $count'_i$ is also incremented by 1. Assume that at some point the values of both $count_i$ and $count'_i$ are c. Node n_i generates a new packet d at time t for which the sequence number is $E_{k_i}(c \mid\mid t)$. As a result, the value of $count_i$ is increased to $c+1$. Until the packet arrives at the BS, $count'_i$ holds the value c. Now, assume that this packet is dropped by a malicious node on its way to the BS. If node n_i generates another packet d' at time t', and assigns it the sequence number $E_{k_i}((c+1) \mid\mid t')$, and this d' successfully arrives at the BS, the decrypted $count_i$, i.e., $(c+1)$ does not match with c. Thus, the BS detects that the packet d is dropped.

Claim 4: Packet replay attacks are detectable.

Justification: Assume that the timestamp when node n_i generates a packet d is t. So, the sequence number of d is $E_{k_i}(c \mid\mid t)$ where c is the current value of $count_i$. Once the BS receives the packet, it can detect any repeated arrival of d by examining the $count_i$ value c. And to prevent delayed attacks, if the packet d arrives at the BS at timestamp t', it is accepted only if $(t' - t)$ is within a reasonable tolerance. Otherwise, it is considered as a replay packet. The BS can set the tolerance range for each *source node* according to network conditions.

IX. Simulation

We have evaluated the performance of our arithmetic coding based provenance scheme (*ACP*) through simulation. We have used TinyOS 2.1.2 TOSSIM simulator [14] and POWERTOSSIMZ *micaz* as energy model for the implementation of our scheme. In our simulation, we consider a sensor network of 100 stationary nodes with IDs 0 through 99 and vary the network diameter from 2 to 14 hops. The node with ID 99 is assumed to be the BS. Some nodes are randomly selected as *source nodes*. During the *training phase*, we let each node in the network to send $10 \sim 100$ packets to the BS. Upon receiving these packets, the BS computes the *occurrence frequencies* for each node in the network. Note that each data point of the simulation results are obtained by taking the average of 5 simulation runs.

We compare our approach with the following schemes:

Bloom filter based provenance scheme (BFP): This scheme [6], [7] uses a fixed size Bloom filter (BF) to encode the provenance of a packet. It embeds all the nodes on a packet's path in the BF using a set of hash functions.

Generic secure provenance scheme (SPS): For comparison purpose, we adapt the generic secure provenance scheme [11] for use in WSN. Here, we simplify the provenance record of a node n_i as P_i = $\langle\ n_i,\ hash(D_i),\ C_i\ \rangle$, where $hash(D_i)$ is a cryptographic hash of the updated data D_i, and C_i contains integrity checksum computed as $Sign(hash(n_i, hash(D_i) \mid\mid C_{i-1}))$. As each node n_i on the packet's path appends its P_i with the received provenance, the size of the provenance increases linearly. To implement *SPS*, we use SHA-1 (160 bit) for cryptographic hash operations and the TinyECC library [15] to generate 160-bit digital signatures. The node ID is 2 bytes in size and thus the increase in the size of the provenance at each hop is 42 bytes.

MAC based provenance scheme (MP): We also consider a MAC-based provenance scheme [6], [7] where a node transmits its node ID and a MAC computed on it as the provenance record. We use TinySec library [16] to compute a 4-byte CBC-MAC. Hence, the provenance size increases by 6 bytes at each hop.

A. Performance Metrics

We analyze the performance of our proposed provenance scheme using the following performance metrics:

(a) *Average provenance size (APS)*: Along with the packet, the BS receives $[L_f, H_f)$ and *buffer* as the encoded data provenance. Assume that the sizes of these elements are denoted by S_{L_f}, S_{H_f}, and S_{buffer}, respectively. In our scheme, we consider the size of the provenance to be $(S_{L_f} + S_{H_f} + S_{buffer})$. We have used 16 bits to represent each of L_f and H_f. The size of the *buffer* varies with different packets' paths and so is the provenance size. For *BFP*, the size of the provenance [6], [7] is equal to the size of the Bloom filter (S_{BF}).

We compute APS for m packets $p_1, p_2, \ldots, p_m$ as follows:

$$APS = \frac{\sum_{i=1}^{m} PS_i}{m}$$

where PS_i denotes the provenance size for packet p_i.

(b) *Total energy consumption (TEC)*: If there are $n_1, n_2, n_3, \ldots, n_{|N|}$ nodes in the network, TEC is computed as follows:

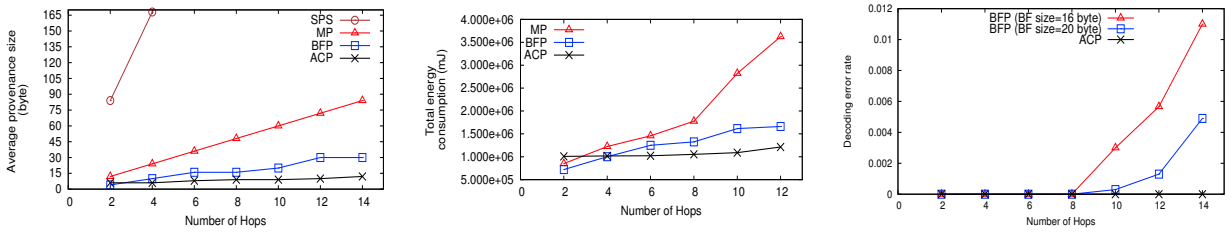
$$TEC = \sum_{i=1}^{|N|} EC_i$$

where EC_i represents the energy consumed by node n_i and $|N|$ is the total number of nodes in the network.

(c) *Decoding error rate (DER)*: It is defined as the percentage of packets for which the BS decodes the provenance incorrectly. If the BS receives m packets and among them d packets are incorrectly decoded, DER is computed as $\frac{d}{m}$.

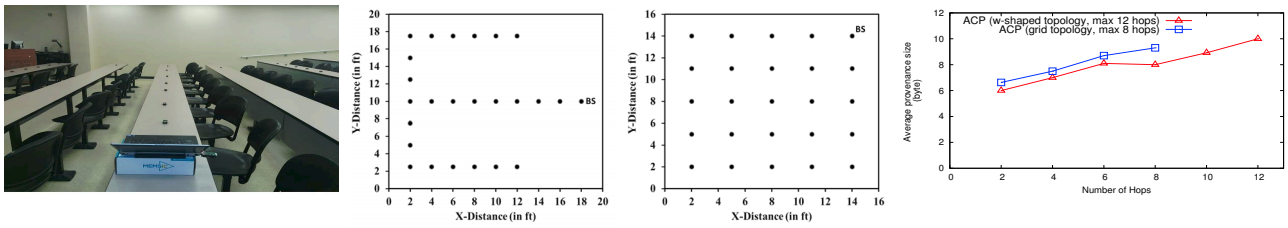
B. Simulation Results

Fig. 4(a) presents the provenance size for the *ACP*, *BFP*, *MP* and *SPS* schemes with respect to the number of hops a packet traverses. In both *SPS* and *MP*, each node appends its encoded provenance to the received one. As a result, the provenance size increases linearly with the path length. However, the *SPS* approach appends 42 bytes at each hop and thus results the provenance size to increase at a much higher rate than that of *MP*. Also in *BFP*, the size of the provenance increases with the number of nodes traversed, but not as fast as in *MP*. As shown in Fig. 4(a), to correctly decode the provenance of a packet that traverses 12 hops, a 30-byte size Bloom filter is required. The provenance size in our scheme is 2 bytes larger than *BFP* for a 2-hop network. However, our scheme demonstrates better performance than *BFP* in reducing the provenance size as the number of hops increases. We see that *ACP* requires only 12-byte provenance for a 14-hop network which is almost one-third of the provenance size in *BFP*. We achieve such compression because the provenance size in our scheme is determined by the *occurrence probabilities* of the nodes on a path. If a path is frequently used, the provenance size of subsequent packets using that path becomes smaller.



(a) (b) (c)

Fig. 4: (a) Average provenance size, (b) Total energy consumption, (c) Decoding error rate



(a) (b) (c) (d)

Fig. 5: (a) Test-bed implemented in a classroom, (b) Placement of nodes in w-shaped topology, (c) Placement of nodes in grid topology, (d) Average provenance size

Fig. 4(b) illustrates the total energy consumption in ACP, BFP, and MP for different hop counts over 100 packet transmissions. We see that the energy consumption in *MP* and *BFP* increases at a higher rate than that of our *ACP* scheme due to their larger provenance size. Hence, the results in Fig. 4(b) shows that *ACP* is more energy efficient than *BFP* and *MP*.

Fig. 4(c) compares decoding error rates of *ACP* and *BFP*. Due to collisions in hash functions, bloom filter of smaller size causes more decoding errors than larger one in *BFP*. On the contrary, our scheme does not introduce any decoding error.

X. Experiments

To further assess our scheme, we deployed it in a testbed containing 25 sensor motes. In this section, we evaluate the performance with respect to the average provenance size as mentioned in Section IX-A.

A. Experimental Setup

We have used TelosB sensor motes to port the implementation. TelosB has a 8MHz TI MSP430 micro-controller, 2.4 GHz radio, 10 KB RAM, and 1 MB external flash for data logging. We placed TelosB motes in an indoor environment (Fig. 5(a)) in a w-shaped topology (Fig. 5(b)) and a grid topology (Fig. 5(c)) with network areas of 20x20 ft^2 and 16x16 ft^2, respectively. We controlled the transmission power of the motes (i.e., set at power level 2) to ensure multi-hop communication in the network. All motes were battery powered and a special mote was used as the BS to collect statistical information. We connected the BS to a laptop through USB port in order to collect the statistical data from the network. In the *training phase*, each node sent 100 packets to the BS. Upon receiving the packets, the BS computed the *occurrence frequencies* of respective nodes.

B. Experimental Results

Fig. 5(d) presents the average provenance size results in the testbed experiments. Since the number of nodes to which a particular node may forward packets is less in w-shaped topology, the *association probabilities* of *forwarder nodes* result to be higher compared to grid topology. Hence, the average provenance size for w-shaped topology is slightly smaller than that of the grid topology.

These results reflect the trends observed in the simulation results shown in Fig. 4(a) and also confirm the claim that the provenance size does not increase with hop counts, but depends on the *occurrence probabilities* of the nodes on the packets' paths.

XI. Related Work

Extensive research has been undergoing on network provenance. However, due to limited computational ability, energy constraints, and low bandwidth, conventional provenance schemes [17], [18] cannot be directly applied in WSN.

Salmin et al. [6], [7] extend the work of Shebaro et al. [5] by proposing a lightweight secure provenance scheme based on in-packet Bloom filter. In this approach, all nodes on a packet's path are embedded in the BF using a set of hash functions. Upon receiving a packet, the BS retrieves the nodes on the path with a certain false positive rate. Such false positive rate depends on the size of the BF. The larger the BF size is, the lower the false positive rates are. Alam et

al. [2] propose an energy-efficient provenance transmission and construction scheme based on probabilistic incorporation of node identities. In this scheme, the provenance is scattered into several packets and thus incurs high decoding error rates if data flow path changes frequently. Moreover, this scheme does not address any security issues. All of these approaches minimize the size of the provenance by keeping only the nodes' IDs, but discard the edges' information that refers to the packets' transmissions. Hence, they are considered as lossy provenance compression techniques which share the following drawbacks: (i) only nodes' IDs are recorded in the data provenance, (ii) the exact path of a packet cannot be properly recovered as edges are discarded, (iii) provenance decoding has false positive, and (iv) the provenance size increases with the number of nodes traversed in order to keep the false positive rate under a given threshold.

Hasan et al. [11] propose a chain model for the provenance. As discussed in Section IX, each node uses checksum, and incremental chained signature mechanism to construct its provenance and thus results in the increase of the provenance size by 42 bytes. Hence, the provenance size in this scheme increases at a much higher rate than that of our scheme.

Dynamic Markov model [19] and Bayesian network [20] based techniques compress data in a centralized environment by assigning a unique and finite codeword to each path. Wall et al. [21] and Witten et al. [10] discuss stream arithmetic coding for centralized environment that uses fixed precision register. However, these techniques cannot be applied to the sensor networks where the environment is distributed.

XII. Conclusion

In this paper, we propose an arithmetic coding based, secure provenance scheme which compresses the provenance in a distributed and lossless manner. The arithmetic coding ensures the compression ratio to be close to Shannon's entropy bound and thus saves more energy and bandwidth compared to other existing approaches. We ensure the confidentiality of data provenance through a secure way of assigning and sharing probability values of each node. We also ensure the integrity and freshness by using MAC and a secure packet sequence number generation mechanism, respectively. The simulation and testbed results show that our scheme outperforms the known compact or lightweight schemes with respect to energy consumption and provenance size.

Acknowledgment

The work reported in this paper has been partially supported by the Purdue Cyber Center, by the National Science Foundation under grant CNS-1111512, and by the Bajian Project for Selected Researchers in Jiangsu University under contract number 1213000013.

References

[1] H.-S. Lim, Y.-S. Moon, and E. Bertino, "Provenance-based trustworthiness assessment in sensor networks," in *Proceedings of the Seventh International Workshop on Data Management for Sensor Networks*, 2010, pp. 2–7.

[2] S. I. Alam and S. Fahmy, "A practical approach for provenance transmission in wireless sensor networks," *Ad Hoc Networks*, vol. 16, no. 0, pp. 28 – 45, 2014.

[3] E. Dawson, D. Wong, and D. Ma, "Secure feedback service in wireless sensor networks," in *Information Security Practice and Experience*. Springer Berlin Heidelberg, vol. 4464, pp. 116–128.

[4] S. C. Misra, I. Woungang, S. Misra, A.-H. Jallad, and T. Vladimirova, "Data-centricity in wireless sensor networks," in *Guide to Wireless Sensor Networks*. Springer London, 2009, pp. 183–204.

[5] B. Shebaro, S. Sultana, S. R. Gopavaram, and E. Bertino, "Demonstrating a lightweight data provenance for sensor networks," in *ACM Conference on Computer and Communications Security*, 2012, Conference Paper, pp. 1022–1024.

[6] S. Sultana, G. Ghinita, E. Bertino, and M. Shehab, "A lightweight secure provenance scheme for wireless sensor networks," in *2012 IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS)*, 2012, pp. 101–108.

[7] S. Sultana, G. Ghinita, E. Bertino, and M. Shehab, "A lightweight secure scheme for detecting provenance forgery and packet drop attacks in wireless sensor networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 99, no. PrePrints, p. 1, 2014.

[8] I. T. Foster, J.-S. Vöckler, M. Wilde, and Y. Zhao, "Chimera: Avirtual data system for representing, querying, and automating data derivation," in *Proceedings of the 14th International Conference on Scientific and Statistical Database Management*, ser. SSDBM '02, 2002, pp. 37–46.

[9] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer, "Provenance-aware storage systems," in *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, ser. ATEC '06, 2006, pp. 4–4.

[10] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[11] R. Hasan, R. Sion, and M. Winslett, "The case of the fake picasso: Preventing history forgery with secure provenance," in *Proccedings of the 7th USENIX Conference on File and Storage Technologies*, ser. FAST, 2009, pp. 1–14.

[12] M. Garofalakis, J. M. Hellerstein, P. Maniatis, and IEEE, "Proof sketches: Verifiable in-network aggregation," in *IEEE 23rd International Conference on Data Engineering*, pp. 971–980.

[13] J. S. Vitter, P. G. Howard, P. G. Howard, and J. S. Vitter, "Arithmetic coding for data compression," in *Information Processing and Management*, 1994, pp. 749–763.

[14] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: accurate and scalable simulation of entire tinyos applications," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, ser. SenSys '03, 2003, pp. 126–137.

[15] A. Liu and P. Ning, "Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks," in *International Conference on Information Processing in Sensor Networks. IPSN '08.*, April 2008, pp. 245–256.

[16] C. Karlof, N. Sastry, and D. Wagner, "Tinysec: A link layer security architecture for wireless sensor networks," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, ser. SenSys '04, 2004, pp. 162–175.

[17] W. Zhou, M. Sherr, T. Tao, X. Li, B. T. Loo, and Y. Mao, "Efficient querying and maintenance of network provenance at internet-scale," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD, 2010, pp. 615–626.

[18] W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. T. Loo, and M. Sherr, "Secure network provenance," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11, 2011, pp. 295–310.

[19] G. V. Cormack and R. N. S. Horspool, "Data compression using dynamic markov modelling," *Comput. J.*, vol. 30, no. 6, pp. 541–550, Dec. 1987.

[20] A. M. Scott Davies, "Bayesian networks for lossless dataset compression," in *Proceedings of the Fifth International Conference on Knowledge Discovery in Databases*. AAAI Press, 1999, pp. 387–391.

[21] L. Wall, K. Ferens, and W. Kinsner, "Real-time dynamic arithmetic coding for low bit-rate channels," in *WESCANEX 93. 'Communications, Computers and Power in the Modern Environment.' Conference Proceedings., IEEE*, Conference Proceedings, pp. 381–391.