VEBEK: *V*irtual *E*nergy-*B*ased *E*ncryption and *K*eying for Wireless Sensor Networks

A. Selcuk Uluagac* Raheem A. Beyah† Yingshu Li† John A. Copeland*

*CSC Lab, School of ECE
Georgia Institute of Technology
Atlanta, GA 30332, USA
{selcuk,jcopeland}@ece.gatech.edu

†CAP Group, Dept. of CS
Georgia State University
Atlanta, GA 30303, USA
{rbeyah,yli}@cs.gsu.edu

***Abstract*—Designing cost-efficient, secure network protocols for Wireless Sensor Networks (WSNs) is a challenging problem because sensors are resource-limited wireless devices. Since the communication cost is the most dominant factor in a sensor's energy consumption, we introduce an energy-efficient Virtual Energy-Based Encryption and Keying (VEBEK) scheme for WSNs that significantly reduces the number of transmissions needed for rekeying to avoid stale keys. In addition to the goal of saving energy, minimal transmission is imperative for some military applications of WSNs where an adversary could be monitoring the wireless spectrum. VEBEK is a secure communication framework where sensed data is encoded using a scheme based on a permutation code generated via the RC4 encryption mechanism. The key to the RC4 encryption mechanism dynamically changes as a function of the residual virtual energy of the sensor. Thus, a one-time dynamic key is employed for one packet only and different keys are used for the successive packets of the stream. The intermediate nodes along the path to the sink are able to verify the authenticity and integrity of the incoming packets using a predicted value of the key generated by the sender's virtual energy, thus requiring no need for specific rekeying messages. VEBEK is able to efficiently detect and filter false data injected into the network by malicious outsiders. The VEBEK framework consists of two operational modes (VEBEK-I and VEBEK-II), each of which is optimal for different scenarios. In VEBEK-I, each node monitors its one-hop neighbors where VEBEK-II statistically monitors downstream nodes. We have evaluated VEBEK's feasibility and performance analytically and through simulations. Our results show that VEBEK, without incurring transmission overhead (increasing packet size or sending control messages for rekeying), is able to eliminate malicious data from the network in an energy efficient manner. We also show that our framework performs better than other comparable schemes in the literature with an overall $60\%-100\%$ improvement in energy savings without the assumption of a reliable medium access control (MAC) layer.**

***Index Terms*—Security, WSN Security, VEBEK, Virtual Energy-Based Keying, Resource constrained devices**

I. Introduction

Rapidly developed WSN technology is no longer nascent and will be used in a variety of application scenarios. Typical application areas include environmental, military, and commercial enterprises [1]. For example, in a battlefield scenario, sensors may be used to detect the location of enemy sniper fire or to detect harmful chemical agents before they reach troops. In another potential scenario, sensor nodes forming a network under water could be used for oceanographic data collection, pollution monitoring, assisted navigation, military surveillance, and mine reconnaissance operations. Future improvements in technology will bring more sensor applications into our daily lives and the use of sensors will also evolve from merely capturing data to a system that can be used for real-time compound event alerting [2].

From a security standpoint, it is very important to provide authentic and accurate data to surrounding sensor nodes and to the sink to trigger time-critical responses (e.g., troop movement, evacuation, first response deployment) [3]. Protocols should be resilient against false data injected into the network by malicious nodes. Otherwise, consequences for propagating false data or redundant data are costly, depleting limited network resources and wasting response efforts.

However, securing sensor networks poses unique challenges to protocol builders because these tiny wireless devices are deployed in large numbers, usually in unattended environments, and are severely limited in their capabilities and resources (e.g., power, computational capacity, and memory). For instance, a typical sensor [4] operates at the frequency of 2.4 GHz, has a data rate of 250Kbps, 128KB of program flash memory, 512KB of memory for measurements, transmit power between 100μW and 1mW, and a communications range of 30m to 100m. Therefore, protocol builders must be cautious about utilizing the limited resources onboard the sensors efficiently.

In this paper, we focus on keying mechanisms for WSNs. There are two fundamental key management schemes for WSNs: *static* and *dynamic*. In static key management schemes, key management functions (i.e., key generation and distribution) are handled statically. That is, the sensors have a fixed number of keys loaded either prior to or shortly after network deployment. On the other hand, dynamic key management schemes perform keying functions (rekeying) either periodically or on demand as needed by the network. The sensors dynamically exchange keys to communicate. Although dynamic schemes are more attack-resilient than static ones, one significant disadvantage is that they increase the communication overhead due to keys being refreshed or redistributed from time to time in the network. There are many reasons for key refreshment, including: updating keys after a key revocation has occurred, refreshing the key such that it does not become stale, or changing keys due to dynamic changes in the topology. In this paper, we seek to minimize the overhead associated with refreshing keys to avoid them becoming stale. Because the communication cost is the most dominant factor in a sensor's energy consumption [5] [6], the message transmission cost for rekeying is an important issue in a WSN deployment (as analyzed in the next section). Furthermore, for certain WSN applications (e.g., military applications), it may be very important to minimize the number of messages to decrease the probability of detection if deployed in an enemy territory. That is, being less ”chatty” intuitively decreases the number of opportunities for malicious entities to eavesdrop or intercept packets.

The purpose of this paper is to develop an efficient and secure communication framework for WSN applications. Specifically, in this paper we introduce Virtual Energy-Based Encryption and Keying (VEBEK) for WSNs, which is primarily inspired by our previous work [7]. VEBEK's secure communication framework provides a technique to verify data in line and drop false packets from malicious nodes, thus maintaining the health of the sensor network. VEBEK dynamically updates keys without exchanging messages for key renewals and embeds integrity into packets as opposed to enlarging the packet by appending message authentication codes (MACs). Specifically, each sensed data is protected using a simple encoding scheme based on a permutation code generated with the RC4 encryption scheme and sent toward the sink. The key to the encryption scheme dynamically changes as a function of the *residual virtual energy of the sensor*, thus requiring no need for rekeying. Therefore, a one-time dynamic key is used for one message generated by the source sensor and different keys are used for the successive packets of the stream. The nodes forwarding the data along the path to the sink are able to verify the authenticity and integrity of the data and to provide non-repudiation. The protocol is able to continue its operations under dire communication cases as it may be operating in a high-error-prone deployment area like under water. VEBEK unbundles key generation from other security services, namely authentication, integrity, and non-repudiation; thus, its flexible modular architecture allows for adoption of other encryption mechanisms if desired. The contributions of this paper are as follows: (1) a dynamic en-route filtering mechanism without that does not exchange explicit control messages for rekeying; (2) provision of one-time keys for each packet transmitted to avoid stale keys; (3) a modular and flexible security architecture with a simple technique for ensuring authenticity, integrity and non-repudiation of data without enlarging packets with MACs; (4) a robust secure communication framework that is operational in dire communication situations and over unreliable MACs. Both analytical and simulation results verify the feasibility of VEBEK. We also illustrate that VEBEK is significantly more energy efficient than other comparable schemes in the literature with an overall 60%−100% improvement.

The paper proceeds as follows. To motivate our work, a preliminary analysis of the rekeying cost with and without explicit control messages is given in Section II. Section III discusses the semantics of VEBEK. VEBEK's different operational modes are discussed in Section IV. An analytical framework and performance evaluation results including a comparison with other relevant works are given in Section V. Section VI summarizes the design rationale and benefits of the VEBEK framework. Related work is presented in Section VII. Finally, Section VIII concludes the paper.

II. Background and Motivation

One significant aspect of confidentiality research in WSNs entails designing efficient *key management* schemes. This is because regardless of the encryption mechanism chosen for WSNs, the keys must be made available to the communicating nodes (e.g., sources, sink(s)). The keys could be distributed to the sensors before the network deployment or they could be re-distributed (rekeying) to nodes on demand as triggered by keying events. The former is *static key* [8] management and the latter is *dynamic key* [9] management. There are myriads of variations of these basic schemes in the literature. In this work, we only consider dynamic keying mechanisms in our analysis since VEBEK uses the dynamic keying paradigm. The main motivation behind VEBEK is that the communication cost is the most dominant factor in a sensor's energy consumption [5] [6]. Thus, in this section, we present a simple analysis for the rekeying cost with and without the transmission of explicit control messages. Rekeying with control messages is the approach of existing dynamic keying schemes whereas rekeying without extra control messages is the primary feature of the VEBEK framework.

Dynamic keying schemes go through the phase of rekeying either periodically or on demand as needed by the network to refresh the security of the system. With rekeying, the sensors dynamically exchange keys that are used for securing the communication. Hence, the energy cost function for the keying process from a source sensor to the sink while sending a message on a particular path with dynamic key-based schemes can be written as follows (assuming computation cost, E_{comp}, would approximately be fixed):

$$E_{Dyn} = (E_{K_{disc}} + E_{comp}) * E[\eta_h] * \frac{\chi}{\tau} \tag{1}$$

where χ is the number of packets in a message, τ is the key refresh rate in packets per key, $E_{K_{disc}}$ is the cost of shared-key discovery with the next hop sensor after initial deployment, and $E[\eta_h]$ is the expected number of hops. In dynamic key-based schemes, τ may change periodically, on-demand, or after a node-compromise. A good analytical lower bound for $E[\eta_h]$ is given in [10] as

$$E[\eta_h] = \frac{D - t_r}{E[d_h]} + 1 \tag{2}$$

where D is the end-to-end distance (m) between the sink and the source sensor node, t_r is the approximated transmission range (m), and $E[d_h]$ is the expected hop distance (m) [11]. An accurate estimation of $E[d_h]$ can be found in [11]. Finally, $E_{K_{disc}}$, can be written as follows:

$$E_{K_{disc}} = \{E[N_e] * E_{node}) * M - 2 * E_{node}\} \tag{3}$$

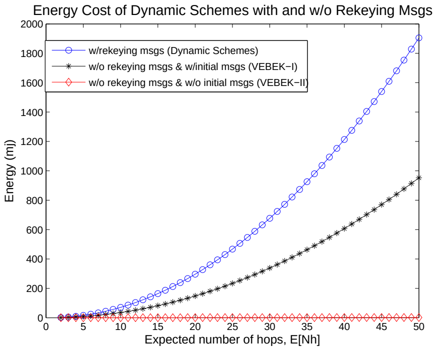


Fig. 1. Keying cost of dynamic key-based schemes based on $E[nh]$ vs. VEBEK.

$$E_{node} = E_{tx} + E_{rx} + E_{comp} \tag{4}$$

where E_{node} is the approximate cost per node for key generation and transmission, $E[N_e]$ is the expected number of neighbors for a given sensor, M is the number of key establishment messages between two nodes, and E_{tx} and E_{rx} are the energy cost of transmission and reception, respectively. Given the transmission range of sensors (assuming bi-directional communication links for simplicity), t_r, total deployment area, A, total number of sensors deployed, N, $E[N_e]$ can be computed as

$$E[N_e] = \frac{N * \pi * t_r^2}{A} \tag{5}$$

On the other hand, VEBEK does rekeying without messages. There are two operational modes of VEBEK (VEBEK-I and VEBEK-II). The details of these modes are given in Section IV. However, for now it suffices to know that VEBEK-I is representative of a dynamic system without rekeying messages, but with some initial neighborhood info exchange whereas VEBEK-II is a dynamic system without rekeying messages and without any initial neighborhood info exchange. Using the energy values given in [4], Figure 1 shows the analytical results for the above expressions. For both VEBEK modes, we assume there would be a fixed cost of E_{comp} [1] because VEBEK does not exchange messages to refresh keys, but for VEBEK-I, we also included the cost of $E_{K_{disc}}$.

With this initial analysis, we see that dynamic key-based schemes, in this scenario, spend a large amount of their energy transmitting rekeying messages. With this observation, the VEBEK framework is motivated to provide the same benefits of dynamic key-based schemes, but with low energy consumption. It does not exchange extra control messages for key renewal. Hence, energy is only consumed for generating the keys necessary for protecting the communication. The keys are dynamic; thus, one key per packet is employed. This makes VEBEK more resilient to certain attacks (e.g., replay attacks, brute-force attacks, masquerade attacks).

III. Semantics of VEBEK

The VEBEK framework is comprised of three modules: Virtual Energy-Based Keying, Crypto, and Forwarding.

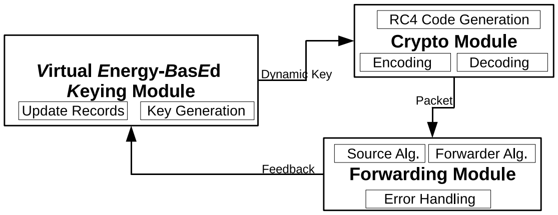


Fig. 2. Modular structure of VEBEK framework.

The virtual energy-based keying process involves the creation of dynamic keys. Contrary to other dynamic keying schemes, it does not exchange extra messages to establish keys. A sensor node computes keys based on its residual *virtual energy* of the sensor. The key is then fed into the crypto module.

The crypto module in VEBEK employs a simple encoding process, which is essentially the process of permutation of the bits in the packet according to the dynamically created permutation code generated via RC4. The encoding is a simple encryption mechanism adopted for VEBEK. However, VEBEK's flexible architecture allows for adoption of stronger encryption mechanisms in lieu of encoding.

Lastly, the forwarding module handles the process of sending or receiving of encoded packets along the path to the sink.

A high level view of the VEBEK framework and its underlying modules are shown in Figure 2. These modules are explained in further detail below. Important notations used are given in Table I.

A. Virtual Energy-based Keying Module

The virtual energy-based keying module of the VEBEK framework is one of the primary contributions of this paper. It is essentially the method used for handling the keying process. It produces a dynamic key that is then fed into the crypto module.

In VEBEK, each sensor node has a certain virtual energy value when it is first deployed in the network. The rationale for using virtual energy as opposed to real battery levels as in our earlier work, DEEF [7], is that in reality battery levels may fluctuate and the differences in battery levels across nodes may spur synchronization problems, which can cause packet drops. These concerns have been addressed in VEBEK and are discussed in detail in the performance evaluation section (Section V).

After deployment, sensor nodes traverse several functional states. The states mainly include node-stay-alive, packet reception, transmission, encoding and decoding. As each of these actions occur, the virtual energy in a sensor node is depleted. The current value of the virtual energy, E_{vc}, in the node is used as the key to the key generation function, F. During the initial deployment, each sensor node will have the same energy level E_{ini}, therefore the initial key, K_1, is a function of the initial virtual energy value and an initialization vector (IV). The IVs are pre-distributed to the sensors. Subsequent keys, K_j, are a function of the current virtual energy, E_{vc}, and the previous key K_{j-1}. VEBEK's virtual energy-based

[1] A more rigorous analysis is presented in Section V.

TABLE I
NOTATIONS USED

E_{tx}	Tx energy	E_{sens}	Sensing energy	E_{Fw}	Forwarding energy	P_{drop}	Drop probability
E_{rx}	Rx energy	E_{sa}	Staying alive energy	E_{Kdisc}	Key discovery energy	φ	Synch ratio
E_{comp}	Computation energy	E_{vc}	Virtual cost	E_{Dyn}	Dynamic keying cost	l	packet size
E_{enc}	Encoding energy	E_p	Perceived energy	E_{So}	Source node energy	N	# of nodes
E_{dec}	Decoding energy	E_b	Bridge energy	$E[\eta_h]$	Expected # of hops	r	# of watched nodes

keying module ensures that each detected packet[2] is associated with a new unique key generated based on the transient value of the virtual energy. After the dynamic key is generated, it is passed to the crypto module, where the desired security services are implemented. The process of key generation is initiated when data is sensed; thus, no explicit mechanism is needed to refresh or update keys. Moreover, the dynamic nature of the keys makes it difficult for attackers to intercept enough packets to break the encoding algorithm. The details are given in Algorithm 1. As mentioned above, each node computes and updates the transient value of its virtual energy after performing some actions. Each action (or state traversal) on a node is associated with a certain predetermined cost. Since a sensor node will be either forwarding some other sensor's data or injecting its own data into the network, the set of actions and their associated energies for VEBEK includes packet reception (E_{rx}), packet transmission (E_{tx}), packet encoding (E_{enc}), packet decoding (E_{dec}) energies, and the energy required to keep a node alive in the idle state (E_a).[3] Specifically, the transient value of the virtual energy, E_v, is computed by decrementing the total of these predefined associated costs, E_{vc}, from the previous virtual energy value.

Algorithm 1 Compute Dynamic Key

```
ComputeDynamicKey($E_{vc}, ID_{clr}$)
begin
$j \leftarrow tx_{cnt}^{ID_{clr}}$
if $j = 1$ then
    $K_j \leftarrow F(E_{ini}, IV)$
else
    $K_j \leftarrow F(K_{(j-1)}, E_{vc})$
end if
return $K_j$
end
```

The exact procedure to compute virtual cost, E_{vc}, slightly differs if a sensor node is the originator of the data or the forwarder (i.e., receiver of data from another sensor). In order to successfully decode and authenticate a packet, a receiving node must keep track of the energy of the sending node to derive the key needed for decoding. In VEBEK, the operation of tracking the energy of the sending node at the receiver is called *watching* and the energy value that is associated with the watched sensor is called *Virtual Perceived Energy* (E_p) as in [7]. More formal definitions for watching are given as follows.

Definition 1: Given a finite number of sensor nodes, N ($N = \{1, .., N\}$), deployed in a region, *watching* is defined as a node's responsibility for monitoring and filtering packets coming from a certain (configurable) number of sensor nodes, r, where $r <= N$. $\lessdot$ is used to denote the watching operation.

Definition 2: Given a sensor node i, the total number of watched nodes, r, which the node is configured to watch, constitutes a *watching list*, WL_i for node i and $WL_i = (1, 2, .., r)$. Node i watches node k if $ID_k \in WL_i$.

Deciding which nodes to watch and how many depends on the preferred configuration of the VEBEK authentication algorithm, which we designate as the operational mode of the framework. Specifically, we propose two operational modes VEBEK-I and VEBEK-II and they are discussed in the next section.

When an event is detected by a source sensor, that node has remained alive for t units of time since the last event (or since the network deployment if this is the first event detected). After detection of the event, the node sends the l-bit length packet toward the sink. In this case, the following is the virtual cost associated with the source node:

$$E_{vc} = l * (e_{tx} + e_{enc}) + t * e_a + E_{synch} \tag{6}$$

In the case where a node receives data from another node, the virtual perceived energy value can be updated by decrementing the cost associated with the actions performed by the sending node using the following cost equation. Thus, assuming that the receiving node has the initial virtual energy value of the sending node and that the packet is successfully received and decoded associated with a given source sensor, k, the virtual cost of the perceived energy is computed as follows:

$$E_p^k = l * (e_{rx} + e_{dec} + e_{tx} + e_{enc}) + t * 2 * e_a \tag{7}$$

where in both the equations, the small $e's$ refer to the one bit energy costs of the associated parameter. However, E_{synch} in (6) refers to a value to synchronize the source with the watcher-forwarders toward the sink as watcher-forwarder nodes spend more virtual energy due to packet reception and decoding operations, which are not present in source nodes. Hence, $E_{synch} = l*(e_{rx}+e_{dec})+e_a*t$. The watching concept is illustrated with an example in Figure 3. In the figure, there is one source sensor node, A, and other nodes B, C, and D are located along the path to the sink. Every node watches its downstream node, i.e., B watches A ($B \lessdot A$); C watches B ($C \lessdot B$); D watches C ($D \lessdot C$). All the nodes have the initial virtual energy of 2000mJ and as packets are inserted into the network from the source node (A) over time, nodes decrement their virtual energy values. For instance, as shown in Figure 3, node A starts with the value of 2000mJ as the first

[2] Indeed, the same key can be used for a certain number of transmissions, n, to further save energy.

[3] The set of actions can be extended to include other actions depending on the WSN application or functionality of the network.

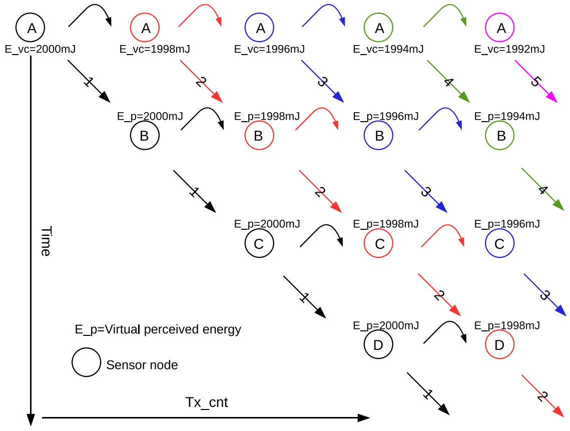


Fig. 3. An illustration of the watching concept with forwarding.

key to encode the packet (key generation based on the virtual energies is explained in the crypto module). Node A sends the first packet and decrements its virtual energy to 1998mJ. After node B receives this first packet, it uses the virtual perceived energy value (E_p=2000mJ) as the key to decode the packet, and updates its E_p (1998mJ) after sending the packet. When the packet travels up to the sink, the virtual energy becomes a *shared dynamic cryptic credential* among the nodes.

B. Crypto Module

Due to the resource constraints of WSNs, traditional digital signatures or encryption mechanisms requiring expensive cryptography is not viable. The scheme must be simple, yet effective. Thus, in this subsection we introduce a simple encoding operation similar to that used in [7]. The encoding operation is essentially the process of permutation of the bits in the packet according to the dynamically created permutation code via the RC4 encryption mechanism. The key to RC4 is created by the previous module (virtual energy-based keying module). The purpose of the crypto module is to provide simple confidentiality of the packet header and payload while ensuring the authenticity and integrity of sensed data without incurring transmission overhead of traditional schemes. However, since the key generation and handling process is done in another module, VEBEK's flexible architecture allows for adoption of stronger encryption mechanisms in lieu of encoding.

The packets in VEBEK consists of the ID (i-bits), type (t-bits) (assuming each node has a type identifier), and data (d-bits) fields. Each node sends these to its next hop. However, the sensors' ID, type, and the sensed data are transmitted in a pseudo random fashion according to the result of RC4. More specifically, the RC4 encryption algorithm takes the key and the packet fields (byte-by-byte) as inputs and produces the result as a permutation code as depicted in Figure 4. The concatenation of each 8-bit output becomes the resultant permutation code. As mentioned earlier, the key to the RC4 mechanism is taken from the core virtual energy-based keying module, which is responsible for generating the dynamic key according to the residual virtual energy level. The resultant permutation code is used to encode the $\langle ID|type|data\rangle$ message. Then, an additional copy of the ID is also transmitted in the clear along with the encoded message. The format of the final packet to be transmitted becomes $Packet = [ID, \{ID, type, data\}_k]$ where $\{x\}_k$ constitutes encoding x with key k. Thus, instead of the traditional approach of sending the hash value (e.g., message digests, message authentication codes) along with the information to be sent, we use the result of the permutation code value locally. When the next node along the path to the sink receives the packet, it generates the local permutation code to decode the packet.

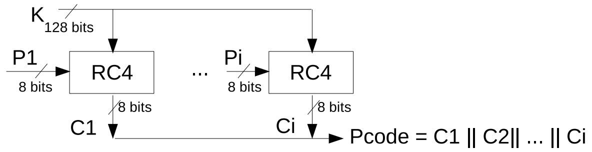


Fig. 4. An illustration of the use of RC4 encryption mechanism in VEBEK.

TABLE II
EXAMPLE ENCODING OPERATIONS

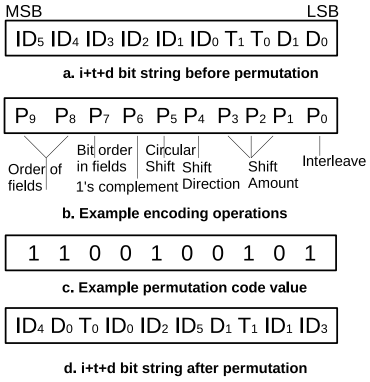
Order of fields in pkt		**1's complement**	
ID, Type, Data	00	Yes	1
ID, Data, Type	01	No	0
Data, ID, Type	10	**Circular Shift**	
Data, Type, ID	11	Yes	1
Order of bits in field		No	0
Little Endian	0	**1-bit interleave**	
Big Endian	1	Yes	1
Shift Direction		No	0
Left	1	**Shift Amount**	
Right	0		

Another significant step in the crypto module involves how the permutation code dictates the details of the encoding and decoding operations over the fields of the packet when generated by a source sensor or received by a forwarder sensor.

Specifically, the permutation code P can be mapped to a set of actions to be taken on the data stream combination. As an example, the actions and their corresponding bit values can include simple operations such as shift, interleaving, taking the 1's complement, etc. Other example operations can be seen in Table II.

For example, if a node computed the following permutation code $P = \{1100100101\}$, the string in Figure 5.a becomes the string in Figure 5.d before it is transmitted. The receiver will perform the same operations (since the inputs to RC4 are stored and updated on each sensor) to accurately decode the packet. To ensure correctness, the receiver compares the plaintext ID with the decoded ID. Moreover, although it is theoretically possible (1 in 2^{i+t+d}) for a hacker to accurately inject data, it becomes increasingly unlikely as the packet grows.

The benefits of this simple encoding scheme are: 1) since there is no hash code or message digest to transmit, the packet size does not grow, avoiding bandwidth overhead on



a. i+t+d bit string before permutation

b. Example encoding operations

c. Example permutation code value

d. i+t+d bit string after permutation

Fig. 5. Illustration of a sample encoding operation.

an already resource constrained network, thus increasing the network lifetime; 2) the technique is simple, thus ideal for devices with limited resources (e.g., PDAs); and 3) the input to the RC4 encryption mechanism, namely the key, changes dynamically without sending control messages to rekey.

C. *Forwarding Module*

The final module in the VEBEK communication architecture is the forwarding module. The forwarding module is responsible for the sending of packets (reports) initiated at the current node (source node) or received packets from other sensors (forwarding nodes) along the path to the sink. The reports traverse the network through forwarding nodes and finally reach the terminating node, the sink. The operations of the forwarding module are explained in this subsection.

1) Source Node Algorithm: When an event is detected by a source node the next step is for the report to be secured. The source node uses the local virtual energy value and an IV (or previous key value if not the first transmission) to construct the next key. As discussed earlier, this dynamic key generation process is primarily handled by the VEBEK module. The source sensor fetches the current value of the virtual energy from the VEBEK module. Then, the key is used as input into the RC4 algorithm inside the crypto module to create a permutation code for encoding the $\langle ID|type|data\rangle$ message. The encoded message and the cleartext ID of the originating node are transmitted to the next hop (forwarding node or sink) using the following format: $[ID, \{ID, type, data\}_{Pc}]$, where $\{x\}_{Pc}$ constitutes encoding x with permutation code Pc. The local virtual energy value is updated and stored for use with the transmission of the next report.

2) Forwarder Node Algorithm: Once the forwarding node receives the packet it will first check its watch-list to determine if the packet came from a node it is watching. If the node is not being watched by the current node, the packet is forwarded without modification or authentication. Although this node performed actions on the packet (received and forwarded the packet), its local virtual perceived energy value is not updated. This is done to maintain synchronization with nodes watching it further up the route. If the node is being watched by the current node, the forwarding node checks the associated current virtual energy record (Algorithm 2) stored for the sending node and extracts the energy value to derive the key. It then authenticates the message by decoding the message and comparing the plaintext node ID with the encoded node ID. If the packet is authentic, an updated virtual energy value is stored in the record associated with the sending node. If the packet is not authentic it is discarded. Again, the virtual energy value associated with the current sending node is only updated if this node has performed encoding on the packet.

Algorithm 2 Forwarding Node Algorithm with Communication Error Handling

```
Forwarder($currentNode, WatchedNode, UpstreamNode$)
begin
$i \leftarrow currentNode; enc \leftarrow 0; WL_i \leftarrow WatchList$
$k \leftarrow WatchedNode; src \leftarrow 0; j \leftarrow 0$
$E_{rx_i}, \langle ID_{clr}, \{msg\}_K\rangle \leftarrow ReceivePacket()$
if $ID_{clr} \in WL_i$ then
    while $(keyFound = 0) and (j <= thresHold)$ do
        $E^k_{p_i} \leftarrow FetchVirtualEnergy(i, ID_{clr}, enc, src)$
        $K \leftarrow ComputeDynamicKey(E^k_{p_i}, ID_{clr})$
        $Pc \leftarrow RC4(K, ID_{clr})$
        $E_{dec_i}, Msg_{ID} \leftarrow decode(Pc, \{msg\}_K)$
        if $ID_{clr} = Msg_{ID}$ then
            $keyFound \leftarrow true$
        else
            $j++$
            $E^k_{p_i} \leftarrow E^k_{p_i} - E_{tx_i} - E_{enc_i} - E_{rx_i} - E_{dec_i} - 2 * E_{a_i}$
        end if
    end while
    if $keyFound = true$ then
        if $j > 1$ then
            $reEncode \leftarrow true$
        else
            if $E_{b_i} > 0$ then
                $reEncode \leftarrow true$
            else
                $reEncode \leftarrow false$
            end if
        end if
        if $reEncode = true$ then
            $enc \leftarrow 1$
            $E_{b_i} \leftarrow FetchVirtualEnergy(i, ID_{clr}, enc, src)$
            $K \leftarrow ComputeDynamicKey(E_{b_i}, ID_{clr})$
            $Pc \leftarrow RC4(K, ID_{clr})$
            $E_{enc_i}, \{msg\}_{Pc} \leftarrow encode(Pc, msg)$
            $packet \leftarrow \langle ID_{clr}, \{msg\}_{Pc}\rangle$
            $E_{tx_i} \leftarrow$ ForwardPacket()
            $E_{b_i} \leftarrow E_{b_i} - E_{tx_i} - E_{enc_i} - E_{rx_i} - E_{dec_i} - 2 * E_{a_i}$
        else
            ForwardPacket() //Without any modification
        end if
    else
        DropPacket() //Packet not valid
    end if
else
    ForwardPacket() //Without any modification
end if
end
```

3) Addressing Communication Errors via Virtual Bridge Energy: In VEBEK, to authenticate a packet, a node must keep track of the virtual energy of the sending node to derive the key needed for decoding. Ideally, once the authenticating node has the initial virtual energy value of the sending node, the value can be updated by decrementing the cost associated with the actions performed by the sending node using the cost equations defined in the previous sub-sections on every successful packet reception. However, communication errors may cause some of the packets to be lost or dropped. Some errors may be due to the deployment region (e.g., underwater shadow zones) while operating on unreliable underlying protocols (e.g., MAC protocol). For instance, ACK or data packets can be lost and the sender may not be able to determine which one actually was lost. Moroever, malicious packets inserted by attackers who impersonate legitimate sensors will be dropped intentionally by other legitimate sensors to filter the bad data out of the network. In such communication errors or intentional packet drop cases, the virtual energy value used to encode the next data packet at the sending node may differ from the virtual energy value that is stored for the sending node at its corresponding watching node. Specifically, the node that should have received the dropped packet and the nodes above that node on the path to the sink lose synchronization with the nodes below (because the upper portion never sees the lost packet and does not know to decrement the virtual energy associated with servicing the lost transmission). If another packet were to be forwarded by the current watching node using its current virtual energy, the upstream node(s) that watch this particular node would discard the packet. Thus, this situation needs to be resolved for proper functioning of the VEBEK framework.

To resolve potential loss of packets due to possible communication errors in the network, all the nodes are configured to store an additional virtual energy value, which we refer to as the *Virtual Bridge Energy,* E_{b_i}, value to allow resynchronization (*bridging*) of the network at the next watching sensor node that determines that packets were lost.

Definition 3: Given a node, i, bridging is defined as the process of encoding the incoming packet coming from any sensor node in WL_i for the upstream sensor node, j, with the key generated using the local copy of E_{b_i}.

That is, as subsequent packets generated from the node of interest pass through the next watching node, the next watching node will decode the packet with the virtual perceived energy key of the originating node and re-encode the packet with the virtual bridge energy key, thus the network will be kept synchronized. It is important to note that once this value is activated for a watched node, it will be always used for packets coming from that node and used even if an error does not occur for the later transmissions of the same watched node. The watching node always updates and uses this parameter to keep the network bridged.

Another pertinent point is the determination of packet loss by the first upstream watching node who will bridge the network. The VEBEK framework is designed to avoid extra messages and not increase the packet size to determine packet loss in the network. Thus, the next watching node tries to find the correct value of the virtual perceived energy for the key within a window of virtual energies. For this, a sensor is configured with a certain *VirtualKeySearchThreshold* value. That is, the watching node decrements the predefined virtual energy value from the current perceived energy at most virtualKeySearchThreshold times. When the node extracts the key successfully, it records the newest perceived energy value and associates it with the sender node (lines $7-18$ in Algorithm 2). This approach may also be helpful in severe packet loss cases (i.e., bursty errors) by just properly configuring the virtualKeySearchThreshold value. However, if the watcher node exhausts all of the virtual energies within the threshold, it then classifies the packet as malicious.

The combined use of virtual perceived and bridge energies assure the continued synchronization of the network as whole. The forwarding node algorithm including the handling of communication errors is shown in Algorithm 2.

IV. Operational Modes of VEBEK

The VEBEK protocol provides three security services: Authentication, integrity, and non-repudiation. The fundamental notion behind providing these services is the watching mechanism described before. The watching mechanism requires nodes to store one or more records (i.e., current virtual energy level, virtual bridge energy values, and Node-Id) to be able to compute the dynamic keys used by the source sensor nodes, to decode packets, and to catch erroneous packets either due to communication problems or potential attacks. However, there are costs (communication, computation, and storage) associated with providing these services. In reality, applications may have different security requirements. For instance, the security need of a military WSN application (e.g., surveiling a portion of a combat zone) may be higher than that of a civilian application (e.g., collecting temperature data from a national park). The VEBEK framework also considers this need for flexibility and thus, supports two operational modes: *VEBEK-I* and *VEBEK-II*. The operational mode of VEBEK determines the number of nodes a particular sensor node must watch. Depending on the vigilance required inside the network, either of the operational modes can be configured for WSN applications. The details of both operational modes are given below. The performance evaluation of both modes is given in Section V.

A. VEBEK-I

In the VEBEK-I operational mode, all nodes watch their neighbors; whenever a packet is received from a neighbor sensor node, it is decoded and its authenticity and integrity are verified. Only legitimate packets are forwarded toward the sink. In this mode, we assume there exists a short window of time at initial deployment that an adversary is not able to compromise the network, because it takes time for an attacker to capture a node or get keys. During this period, route initialization information may be used by each node to decide which node to watch and a record r is stored for each of its 1-hop neighbors in its watch-list. To obtain a neighbor's initial energy value, a network-wise master key can be used to transmit this value during this period similar to the shared-key discovery phase of other dynamic key management schemes. Alternatively, sensors can be pre-loaded with the initial energy value.

When an event occurs and a report is generated, it is encoded as a function of a dynamic key based on the virtual energy of the originating node, and transmitted. When the packet arrives at the next-hop node, the forwarding node extracts the key of the sending node (this could be the originating node or another forwarding node) from its record. (The virtual perceived energy value associated with the sending node and decodes the packet). After the packet is decoded successfully, the plaintext ID is compared with the decoded ID. In this process, if the forwarding node is not able to extract the key successfully, it will decrement the predefined virtual energy value from the current perceived energy (line 16 in Algorithm 2) and tries another key before classifying the packet as malicious (because packet drops may have occurred due to communication errors). This process is repeated several times; however, the total number of trials that are needed to classify a packet as malicious is actually governed by the value of virtualKeySearchThreshold. If the packet is authentic, and this hop is not the final hop, the packet is re-encoded by the forwarding node with its own key derived from its current virtual bridge energy level. If the packet is illegitimate, the packet is discarded. This process continues until the packet reaches the sink. Accordingly, illegitimate traffic is filtered before it enters the network.

Re-encoding at every hop refreshes the strength of the encoding. Recall that the general packet structure is $[ID, \{ID, type, data\}_k]$. To accommodate this scheme, the ID will always be the ID of the current node and the key is derived from the current node's local virtual bridge energy value. If the location of the originating node that generated the report is desired, the packet structure can be modified to retain the ID of the originating node and the ID of the forwarding node.

VEBEK-I reduces the transmission overhead as it will be able to catch malicious packets in the next hop, but increases processing overhead because of the decode/encode that occurs at each hop.

B. VEBEK-II

In the VEBEK-II operational mode, nodes in the network are configured to only watch some of the nodes in the network. Each node randomly picks r nodes to monitor and stores the corresponding state before deployment. As a packet leaves the source node (originating node or forwarding node) it passes through node(s) that watch it probabilistically. Thus, VEBEK-II is a statistical filtering approach like SEF [12] and DEF [13]. If the current node is not watching the node that generated the packet, the packet is forwarded. If the node that generated the packet is being watched by the current node, the packet is decoded and the plaintext ID is compared with the decoded ID. Similar to VEBEK-I, if the watcher-forwarder node cannot find the key successfully, it will try as many keys as the value of virtualKeySearchThreshold before actually classifying the packet as malicious. If the packet is authentic, and this hop is not the final destination, the original packet is forwarded unless the node is currently bridging the network. In the bridging case, the original packet is re-encoded with the virtual bridge energy and forwarded. Since this node is bridging the network, both virtual and perceived energy values are decremented accordingly. If the packet is illegitimate, which is classified as such after exhausting all the virtual perceived energy values within the virtualKeySearchThreshold window, the packet is discarded. This process continues until the packet reaches the sink.

This operational mode has more transmission overhead because packets from a malicious node may or may not be caught by a watcher node and they may reach the sink (where it is detected). However, in contrast to the VEBEK-I mode, it reduces the processing overhead (because less re-encoding is performed and decoding is not performed at every hop). The trade-off is that an illegitimate packet may traverse several hops before being dropped. The effectiveness of this scheme depends primarily on the value r, the number of nodes that each node watches. Note that in this scheme, re-encoding is not done at forwarding nodes unless they are bridging the network.

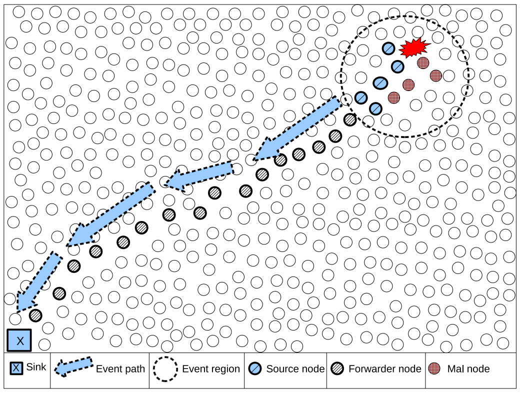


Fig. 6. Simulation topology with GTSNetS.

V. Performance Analysis

In this section we evaluate the effectiveness of the VEBEK framework via both simulations and analysis.

A. Assumptions

Due to the broadcast nature of the wireless medium used in sensor networks, attackers may try to eavesdrop, intercept, or inject false messages. In this paper we mainly consider the false injection and eavesdropping of messages from an outside malicious node; hence, similar to [12], insider attacks are outside the scope of this paper. This attacker is thought to have the correct frequency, protocol, and possibly a spoofed valid node ID. Throughout this work, the following assumptions are also made:

- Directed Diffusion [14] routing protocol is used, but others such as [15] can also be used. According to specifics of Directed Diffusion, after the sink asks for data via interest messages, a routing path is established from the sources in the event region to the sink. We assume that the path is fixed during the delivery of the data and the route setup is secure.
- The routing algorithm is deployed on an unreliable MAC. The network may experience ACK or data packet drops.

- The sensor network is densely populated such that multiple sensors observe and generate reports for the same event.
- Sensors are assumed to have the same communication ranges and may have different initial battery supplies.

B. *Simulation Parameters*

We use the Georgia Tech Sensor Network Simulator (GT-SNetS) [16], which is an event-based object-oriented sensor network simulator with C++, as our simulation platform to perform the analysis of the VEBEK communication framework. The topology used for the simulation is shown in Figure 6, while the parameters used in the simulation are summarized in Tables III and IV. Nodes were distributed randomly in the deployment region and on average, the distance between the source nodes and the sink was around 25 35 hops. The virtualKeySearchThreshold value was 15 [17]. The energy costs for different operations in the table are computed based on the values given in [4]. However, the costs for encoding and decoding operations are computed based on the the reported values of the implementation of RC4 [18] on real sensor devices.

TABLE III
GENERAL SIMULATION PARAMETERS

# of Nodes	500	SensSize	32 bytes
Area	1000x1000 m	RecvInterval	50s
# of Watched	(0..60)	SensRate	30s
Link Rate	250Kbps	SimTime	3000s
Range	75 m	#of Mal Node	(0..10)

TABLE IV
ENERGY RELATED PARAMETERS

E_{rx}	85.1μJ	E_{dec}	15.5μJ
E_{tx}	78μJ	E_{enc}	15.5μJ
E_{sens}	36μJ	Voltage	3V
E_{sa}	18.6μJ		

C. *Attack Resilience*

In this sub-section, the performance of VEBEK is analyzed when there are malicious source nodes in the data collection field who insert bad packets into the network. Specifically, the analytical basis of the VEBEK framework's resilience against malicious activities is formulated. Then, this theoretical basis is verified with the simulation results. We compare VEBEK-I and VEBEK-II considering the drop probability vs. number of hops. We also take a closer look at VEBEK-II and how it is affected by the parameter, r (the number of records).

In VEBEK-I and VEBEK-II, in order for an attacker to be able to successfully inject a false packet, an attacker must forge the packet encoding (which is a result of dynamically created permutation code via RC4). Given that the complexity of the packet is 2^l where l is the sum of the ID, TYPE, and DATA fields in the packet, the probability of an attacker correctly forging the packet is:

$$P_{forge} = \frac{1}{2^{packetsize}} = \frac{1}{2^l} \tag{8}$$

Accordingly, the probability of the hacker incorrectly forging the packet and therefore the packet being dropped (p_{drop-I}) is:

$$P_{drop-I} = 1 - P_{forge} \tag{9}$$

Since VEBEK-I authenticates at every hop, forged packets will always be dropped at the first hop with a probability of P_{drop-I}.

On the other hand, VEBEK-II statistically drops packets along the route. Thus, the drop probability for VEBEK-II ($P_{drop-II}$) is a function of the effectiveness of the watching nodes as well as the ability for a hacker to correctly guess the encoded packet structure. Accordingly, the probability of detecting and dropping a false packet at one hop when randomly choosing r records (nodes to watch) is:

$$P_{drop-II} = \frac{r}{N} * (1 - P_{forge}) \tag{10}$$

Thus, the probability to detect and drop the packet when choosing r records after h hops is:

$$P^{r,h}_{drop-II} = 1 - (1 - p_{drop-II})^h \tag{11}$$

Moreover, even if one false packet successfully makes it to the sink, we assume that the sink has enough resources to determine which data to process and accept.

Figure 7 shows both the theoretical and simulation results for VEBEK-II based on the above equations for a varying number of watched nodes, r, in the WSN. Note that VEBEK-I is not shown in this figure because it eliminates malicious data immediately. The x-axis represents the number of hops a malicious packet travels before it has been detected and taken out of the network. As can be seen from the figure, VEBEK-II is able to eliminate malicious packets from the WSN within 15 hops with 0.5 probability when nodes watch 25 randomly chosen nodes (r value). However, if more storage is available on the sensors, then VEBEK-II can detect and remove malicious packets within 15 hops with 0.90 probability when r is 60. A similar trend is observed in the same figure with the simulation results.

On the other hand, Figure 8 presents the comparison of VEBEK-I (VI in the figure) and VEBEK-II (VII in the figure) via simulation in terms of their filtering efficiency. The x-axis represents the number of watched nodes (r) that each node is configured to watch in VEBEK-II and the y-axis shows the percent of in-network malicious packet dropped with varying number of malicious nodes in the simulation. As expected, we see that VEBEK-I is always able to filter malicious packets from the network with its 100% filtering efficiency. This is mainly due to the fact that malicious packets are immediately taken out from the network at the next hop. However, the filtering efficiency of VEBEK-II is closely related to the number of nodes (r) that each node watches. The more nodes watched by other nodes, the more efficient VEBEK-II is with filtering malicious data. Additionally, as seen when r is equal to 40,

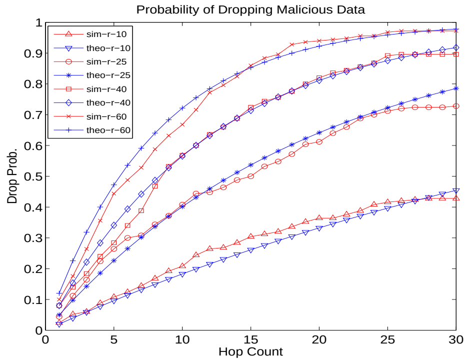


Fig. 7. Theoretical and simulation results with varying number of watched nodes.

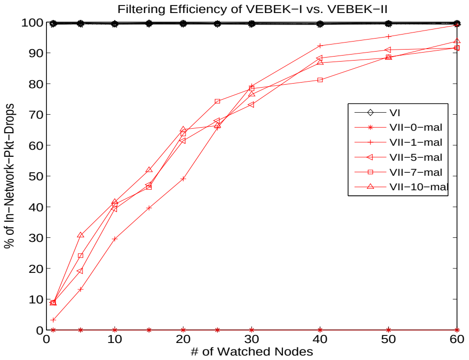


Fig. 8. Comparison of filtering efficiency for VEBEK-I and VEBEK-II with varying number of malicious nodes.

it is possible to achieve almost 90% filtering efficiency. This particular observation with VEBEK-II is significant because for some WSN applications, energy can be saved by properly configuring the *r* parameter. Finally, with respect to Figure 8, we observe that the VEBEK framework is independent of the number of malicious nodes as the framework still filters the malicious data from the network successfully.

D. Energy Consumption of VEBEK-I and VEBEK-II

In this sub-section we look at the associated costs to transmit valid data in VEBEK-I and VEBEK-II.

In both operational modes, there is a single cost (E_{So}) to stay-alive, sense the event, encode the packet, and transmit the packet ($E_{sa}, E_{sens}, E_{enc}, E_{tx}$) at the source sensor. Thus,

$$E_{So} = E_{sens} + E_{enc} + E_{tx} + E_{sa} \tag{12}$$

Additionally, there is a recurring forwarding cost (E_{FW}) to marshal the packet through the network depending on the number of hops. In VEBEK-I, this cost is

$$E_{FW} = E_{rx} + E_{dec} + E_{enc} + E_{tx} + E_{sa} \tag{13}$$

for all of the intermediate nodes since all of the nodes perform the same operations. Hence, the average cost to transmit a packet in VEBEK-I using $E[\eta_h]$ from (2) is:

$$E_{FW_I} = E_{So} + (E[\eta_h] * E_{FW}) \tag{14}$$

On the other hand, in VEBEK-II the cost of $E_{FW_{II}}$ consists of E_{FW_w} and $E_{FW_{nw}}$ for variable fractions of the forwarding nodes depending on the number of nodes each node chose to watch, where $E_{FW_w} = E_{FW}$ and $E_{FW_{nw}} = E_{rx} + E_{tx} + E_{sa}$. Hence, the average cost to transmit a packet using VEBEK-II is:

$$E_{FW_{II}} = E_{So} + (E[\eta_{h_w}] * E_{FW_w}) + (E[\eta_{h_{nw}}] * E_{FW_{nw}}) \tag{15}$$

where $E[\eta_{h_w}]$ and $E[\eta_{h_{nw}}]$ represent the expected number of nodes along the path who are watchers and non-watcher nodes, respectively. The values for these expectations can be computed given the total expected number of hops with $E[\eta_h]$ from (2) where $E[\eta_h] = E[\eta_{h_w}] + E[\eta_{h_{nw}}]$ for $i = 1, 2, 3, .., \eta_h$.

Let $X_i = 1$ if the i^{th} sensor is a watcher and let $X_i = 0$ otherwise for a given path to the sink with probabilities $P\{p = 1\} = \frac{r}{N}$, $P\{q = 0\} = \frac{N-r}{N}$, and N sensors. Then, $X_i \sim Bernoulli(p)$ i.i.d. random variables and $\eta_{h_w} = X_1 + ... + X_{\eta_h}$.

$$E[\eta_{h_w}] = E[\sum_{i=1}^{\eta_h} X_i] = E[E[\sum_{i=1}^{\eta_h} X_i | \eta_h]] \tag{16}$$

Hence, by the independence of X_i and η_h;

$$E[\eta_{h_w}] = E[\eta_h] * E[X_i] = \frac{r}{N} * E[\eta_h] \tag{17}$$

With a similar reasoning, an expression for the expected number of non-watchers, $E[\eta_{h_{nw}}]$, can be written as follows.

$$E[\eta_{h_{nw}}] = E[\eta_h] * E[X_i] = \frac{N-r}{N} * E[\eta_h] \tag{18}$$

Implementing these costs inside the GTSNetS simulator, we have evaluated the energy performance of the scheme both for VEBEK-I and VEBEK-II and plotted the results. In all the figures, the x-axis represents the number of malicious nodes while the y-axis is the energy consumption. Different values for the number of watched nodes (*r*) were analyzed for VEBEK-II. Furthermore, two attack scenarios were considered: Attack-Scenario-1 and Attack-Scenario-2. VEBEK-I and VEBEK-II are abbreviated as VI and VII in the figures.

In Attack-Scenario-1, less powerful malicious nodes are assumed. The total number of healthy source nodes that collect the event information and send it toward the sink is assumed to be fixed, whereas the number of malicious nodes are increased over time. Letting i be the number of healthy source nodes and j be the number of malicious nodes, in Attack-Scenario-1, $j \leq i$, where $i = n$ and $n > 0$. Figures 9−11 show the results for Attack-Scenario-1. As seen from the computation costs (i.e., E_{enc}, E_{dec}) (Figure 9), VEBEK-II's consumption is less than that of VEBEK-I. The primary reason for this behavior stems from decoding and re-encoding of packets at every hop in the network for VEBEK-I. Also, as the number of watched nodes (*r*) increases, VEBEK-II's computation cost increases because more packets are processed for the filtering operation. On the other hand, the more malicious nodes in the system, the more resources are consumed to filter the increased number of malicious packets in the network. As for the transmission costs (i.e., E_{tx}, E_{rx}) in Figure 10, VEBEK-I

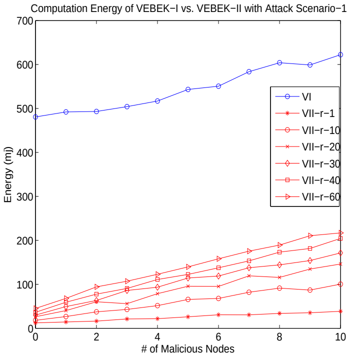


Fig. 9. Computation costs (Attack-Scenario-1).

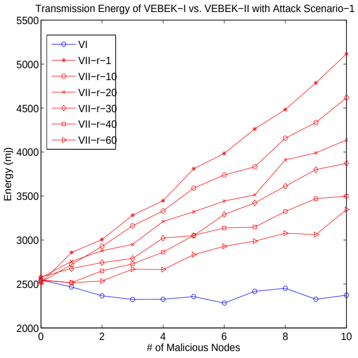


Fig. 10. Transmissions costs (Attack-Scenario-1).

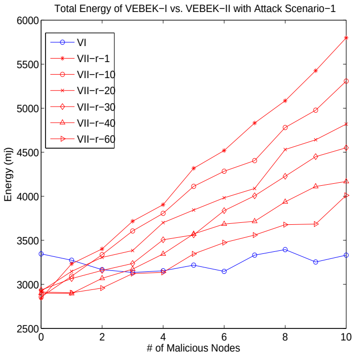


Fig. 11. Total energy cost (Attack-Scenario-1).

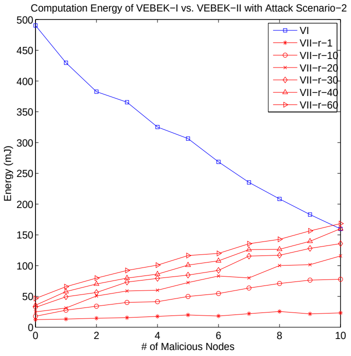


Fig. 12. Computation costs (Attack-Scenario-2).

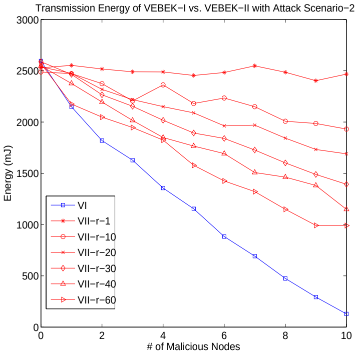


Fig. 13. Transmissions costs (Attack-Scenario-2).

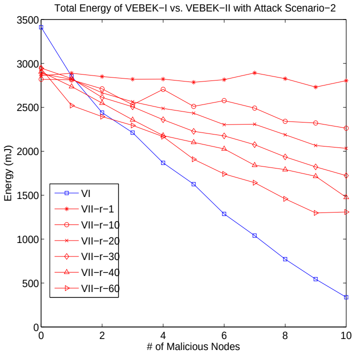


Fig. 14. Total energy costs (Attack-Scenario-2).

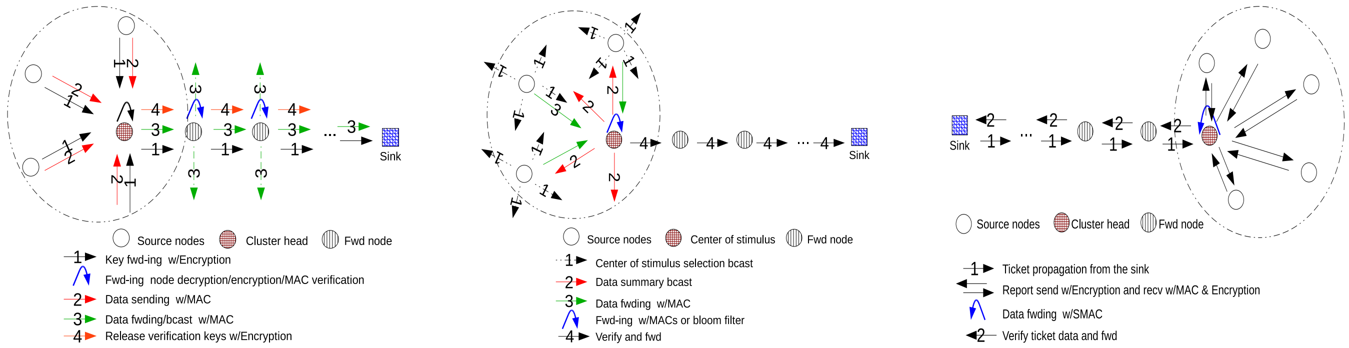
is better as the nodes are able to catch and drop malicious packets and do not let malicious packets traverse the network. As r decreases, fewer nodes are watched by the sensors. Thus, the transmission cost increases in the network because more traffic traverses the network as a result of less filtering capability with smaller r values. Furthermore, as the number of malicious nodes increases in the network, the transmission cost increases due to more malicious traffic. Finally, analyzing the results for the total energy consumption, we see that the total energy consumption in the network exhibits a similar behavior as transmission costs because the overall energy consumption is greatly dominated by the transmission costs. Moreover, we observe that the total energy consumption for VEBEK-II is smaller than VEBEK-I up to a certain number of malicious nodes (1 and 2) for certain values of r (all watching values at 1 malicious node; and watching values of 30, 40, and 60 at 2 malicious nodes). The implication of this result is interesting. If the deployment region is a relatively safe environment ($<$ 2 malicious nodes in our scenario), a similar filtering efficiency of VEBEK-I can be achieved using VEBEK-II (100% for VEBEK-I vs. 99% for VEBEK-II with $r = 60$) (Figure 8) if more storage is available on the nodes. This can be accomplished while consuming less energy than VEBEK-I (3400mJ for VEBEK-I vs. 2800 mJ for VEBEK-II).

In Attack-Scenario-2, more powerful malicious nodes are assumed. For instance, they can jam the signal and not allow healthy nodes to transmit. Over time, more powerful nodes are assumed to replace the number of healthy source nodes. Hence, $j = 0, 1, 2, .., n$ and $i = n, n{-}1, n{-}2, .., 0$ where again $n > 0$. Figures $12{-}14$ present the results for Attack-Scenario-2. In all the figures, it is possible to observe the same patterns as Attack-Scenario-1. The only difference is the downward slope with some of the plots. This is attributed to the fact that the ratio of the healthy traffic diminishes in this attack scenario as the number of bad packets increases due to the number of malicious nodes in the network.

So, if a more secure application is desired or if the WSN application is deployed in an hostile environment, then VEBEK-I is recommended because VEBEK-I provides security services at every hop. VEBEK-I also watches fewer nodes in comparison to VEBEK-II. Thus, the lower storage requirement (i.e., fewer watched nodes) and providing security at every hop make VEBEK-I well suitable for military WSN applications where immediate reaction to enemy units is necessary. However, the downside of the VEBEK-I operational mode is its high processing costs. On the other hand, if the deployment region is expected to be a relatively safe environment, which may be true for some civilian WSN applications, then VEBEK-II can be utilized. But, as discussed above, to provide a comparable level of vigilance to the network, this operational mode uses much more storage than VEBEK-I.

E. Comparison of VEBEK-II with Other Statistical Schemes

In this sub-section, we evaluate the energy performance of VEBEK-II with other ”en-route dynamic filtering” works in the literature. We focus on statistical schemes because they


Sink
Source nodes
Cluster head
Fwd node
1 Key fwd-ing w/Encryption
Fwd-ing node decryption/encryption/MAC verification
2 Data sending w/MAC
3 Data fwding/bcast w/MAC
4 Release verification keys w/Encryption
Source nodes
Center of stimulus
Fwd node
1 Center of stimulus selection bcast
2 Data summary bcast
3 Data fwding w/MAC
Fwd-ing w/MACs or bloom filter
4 Verify and fwd
Source nodes
Cluster head
Fwd node
1 Ticket propagation from the sink
Report send w/Encryption and recv w/MAC & Encryption
Data fwding w/SMAC
2 Verify ticket data and fwd

(a) DEF. (b) SEF. (c) STEF.

Fig. 15. Illustrations of DEF, SEF, and STEF.

have received a lot of attention in recent years. Specifically, we compare the expected energy costs of DEF [13], SEF [12], and STEF [19][4] with that of VEBEK-II because VEBEK-II is the statistical mode of the VEBEK framework. First, we briefly summarize each protocol and discuss their drawbacks. Then, the comparison results are presented. An illustration of each protocol is given in Figure 15.

In the Dynamic En-route Filtering scheme (DEF) by Yu and Guan [13], a legitimate report is endorsed by multiple sensing nodes using their own authentication keys. Before deployment, each node is preloaded with a seed authentication key and $l + 1$ secret keys randomly chosen from a global key pool. Before sending reports, the cluster head disseminates the authentication keys to forwarding nodes encrypted with secret keys that will be used for endorsing. The forwarding nodes stores the keys if they can decrypt them successfully. Later, cluster heads send authentication keys to validate the reports. The DEF scheme involves the usage of authentication keys and secret keys to disseminate the authentication keys; hence, it uses many keys and is complicated for resource-limited sensors.

Ye et al. proposed statistical en-route filtering (SEF) [12]. In SEF, each sensing report is validated by multiple keyed message authentication codes (MACs). Specifically, each node is equipped with some number of keys that are drawn randomly from the global key pool. First, a center of stimulus is selected among the source sensor nodes in the event region. Then, once a report is generated by a source node, a MAC is appended to the report. Next, another upstream node that has the same key as the source can verify the validity of the MAC and filters the packet if the MAC is invalid. However, the downside of SEF is that the nodes must store keys and packets are enlarged by MACs. Although the authors suggest the use of bloom-filters to decrease the MAC overhead, SEF is a static key-based scheme and it inherits all the downsides of static key management schemes.

The scheme, Secure Ticket-Based En-route Filtering (STEF) [19], by Krauss et al. proposes using a ticket concept, where tickets are issued by the sink and packets are only forwarded if they contain a valid ticket. If a packet does not contain a valid ticket, it is immediately filtered out. STEF is similar in nature to SEF and DEF. The packets contain a MAC and cluster heads share keys with their immediate source sensor nodes in their vicinity and with the sink. The downside of STEF is its one way communication in the downstream for the ticket traversal to the cluster head.

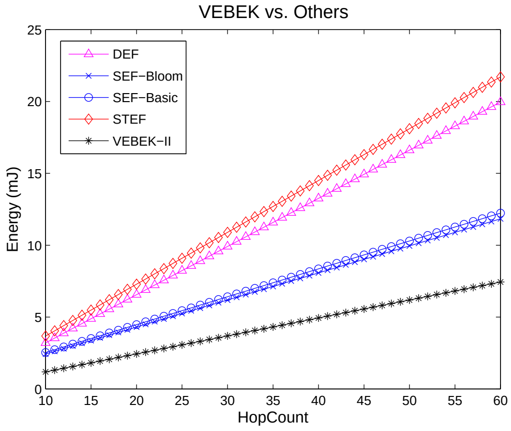


Fig. 16. Comparison of VEBEK, DEF[13], SEF[12], and STEF[19].

Since DEF and SEF are probabilistic schemes, a comparison of each scheme with VEBEK-II in terms of their energy consumption is presented in Figure 16. The results are generated for one round of communication from a source node to the sink, which is assumed to be located n hops away from the source node. The x-axis represents the hop count and is varied, while the y-axis is the energy. To simplify the comparisons, we assumed that all the nodes in DEF, SEF and VEBEK-II would have the necessary keying material with 0.7 probability to do the desired security features imposed by the specific protocol in a benign environment (no malicious nodes). We also assumed that the protocols that use hashing and encryption mechanisms would use MD5 and RC4, respectively. The real sensor implementation values for these crypto mechanisms are taken from [18] and [20]. Another necessary assumption was that all protocols would work in perfect communication cases without packet loss because only the VEBEK framework has been designed with handling communication error cases and it would not be meaningful to compare VEBEK with

[4] Although STEF is not a statistical approach, we included in our comparison because it is a relevant en-route filtering study.

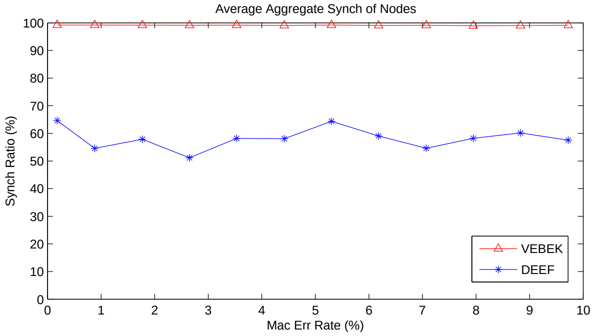


Fig. 17. Synchronization ratio of nodes along the path to the sink.

others when others were not designed to handle errors. As can be seen, VEBEK-II is better than all the schemes, exhibiting a performance improvement of 60%−100% in energy consumption than the closest scheme, SEF. We note that all other schemes provide a nice framework for filtering malicious data en-route; however, the other schemes exchange many messages, involve the use of many keys, and do not have any mechanism to cope with packet loss.

Moreover, we analyze how VEBEK improves the synchronization problems that may occur due to communication errors in our previous work, DEEF [7]. Since DEEF is based on generating communication keys with real battery levels, packet drops may cause the nodes to easily loose synchronization with other nodes along the path to the sink. To analyze the synchronization problem, we define *synchronization ratio* as a metric to measure the performance of the VEBEK framework during packet drops. Specifically, we denote the synchronization ratio, φ, as follows:

$$\varphi = \sum_{i=1}^{\eta_{h_w}} \frac{\gamma_i}{\gamma_i + \varepsilon_i} \tag{19}$$

where i is the node, γ is the number of forwarded-watched packets, ε is the number of dropped-watched packets, and η_{h_w} is the number of watcher nodes between the source and the sink. Figure 17 presents the simulation results of the synchronization ratio with respect to DEEF and VEBEK. As can be seen, VEBEK outperforms DEEF and it is able to keep its synchronization even in dire communication scenarios. The x-axis is the the percent of the packets that are dropped due to communication errors.

VI. Related Work

En-route dynamic filtering of malicious packets has been the focus of several studies, including dynamic en-route filtering (DEF) by Yu and Guan [13], statistical en-route filtering (SEF), [12], and Secure Ticket-Based En-route Filtering (STEF) [19]. As the details are given in the performance evaluation section (Section V) where they were compared with the VEBEK framework, the reader is referred to that section for further details as not to replicate the same information here. Moreover, Ma's work [21] applies the same filtering concept at the sink and utilizes packets with multiple MACs appendedand. A work [22] proposed by Hyun and Kim uses relative location information to make the compromised data meaningless and to protect the data without cryptographic methods. In [23], using static pairwise keys and two MACs appended to the sensor reports, ”an interleaved hop-by-hop authentication scheme for filtering of injected false data” was proposed by Zhu et al to address both the insider and outsider threats. However, the common downside of all these schemes is that they are complicated for resource-constrained sensors and they either utilize many keys or they transmit many messages in the network, which increases the energy consumption of WSNs. Also, these studies have not been designed to handle dire communication scenarios unlike VEBEK. Another significant observation with all of these works is that a realistic energy analysis of the protocols was not presented. Lastly, the concept of dynamic energy-based encoding and filtering was originally introduced by the DEEF [7] framework. Essentially, VEBEK has been largely inspired by DEEF. However, VEBEK improves DEEF in several ways. First, VEBEK utilizes virtual energy in place of actual battery levels to create dynamic keys. VEBEK's approach is more reasonable because in real life, battery levels may fluctuate and the differences in battery levels across nodes may spur synchronization problems, which can cause packet drops. Second, VEBEK integrates handling of communication errors into its logic, which is missing in DEEF. Lastly, VEBEK is implemented based on a realistic WSN routing protocol, i.e., Directed Diffusion [14], while DEEF articulates the topic only theoretically.

Another crucial idea of this paper is the notion of sharing a dynamic cryptic credential (i.e., virtual energy) among the sensors. A similar approach was suggested inside the SPINS study [24] via the SNEP protocol. In particular, nodes share a secret counter when generating keys and it is updated for every new key. However, the SNEP protocol does not consider dropped packets in the network due to communication errors. Although another study, Minisec [25], recognizes this issue, the solution suggested by the study still increases the packet size by including some parts of a counter value into the packet structure. Finally, one useful pertinent work [6] surveys cryptographic primitives and implementations for sensor nodes.

VII. Conclusion and Future Work

Communication is very costly for wireless sensor networks (WSN)s and for certain WSN applications. Independent of the goal of saving energy, it may be very important to minimize the exchange of messages (e.g., military scenarios). To address these concerns, we presented a secure communication framework for WSNs called ***V***irtual ***E***nergy-***B***ased ***E***ncryption and ***K***eying (VEBEK).

In comparison with other key management schemes, VEBEK has the following benefits: (1) it does not exchange control messages for key renewals and is therefore able to save more energy and is less chatty; (2) it uses one key per message so successive packets of the stream use different keys - making VEBEK more resilient to certain attacks (e.g., replay attacks, brute-force attacks, masquerade attacks); and (3) it unbundles key generation from security services, providing a flexible modular architecture that allows for an easy adoption of different key-based encryption or hashing schemes.

We have evaluated VEBEK's feasibility and performance through both theoretical analysis and simulations. Our results show that different operational modes of VEBEK (I and II) can be configured to provide optimal performance in a variety of network configurations depending largely on the application of the sensor network. We also compared the energy performance of our framework with other en-route malicious data filtering schemes. Our results show that VEBEK performs better (in the worst case between 60%−100% improvement in energy savings) than others while providing support for communication error handling, which was not the focus of earlier studies. Our future work will address insider threats and dynamic paths.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: A survey," *Computer Networks (Elsevier) Journal*, vol. 38, no. 4, pp. 393–422, Mar. 2002.

[2] C. Vu, R. Beyah, and Y. Li, "A composite event detection in wireless sensor networks," in *Proc. of the IEEE IPCCC*, April 2007.

[3] S. Uluagac, C. Lee, R. Beyah, and J. Copeland, "Designing secure protocols for wireless sensor networks," *Lecture Notes in Computer Science, Wireless Algorithms, Systems, and Applications (WASA).*, vol. 5258, pp. 503–514, 2008.

[4] Xbow, "Crossbow technology," 2008. [Online]. Available: http://www.xbow.com/

[5] G. J. Pottie and W. J. Kaiser, "Wireless integrated network sensors," *Commun. ACM*, vol. 43, no. 5, pp. 51–58, 2000.

[6] R. Roman, C. Alcaraz, and J. Lopez, "A survey of cryptographic primitives and implementations for hardware-constrained sensor network nodes," *Mobile Networks and Applications, Springer*, vol. 12, no. 4, pp. 231–244, August 2007.

[7] H. Hou, C. Corbett, Y. Li, and R. Beyah, "Dynamic energy-based encoding and filtering in sensor networks," in *Proc. of the IEEE MILCOM*, October 2007.

[8] L. Eschenauer and V. D. Gligor, "A key-management scheme for distributed sensor networks," in *Proc. of IEEE Symposium on Security and Privacy*, 2002, pp. 41–47.

[9] M. Eltoweissy, M. Moharrum, and R. Mukkamala, "Dynamic key management in sensor networks," *IEEE Communications Magazine*, vol. 44, no. 4, pp. 122–130, April 2006.

[10] M. Zorzi and R. Rao, "Geographic random forwarding (geraf) for ad hoc and sensor networks: multihop performance," *Mobile Computing, IEEE Transactions on*, vol. 2, no. 4, pp. 337–348, Oct.-Dec. 2003.

[11] M. Vuran and I. Akyildiz, "Cross-layer analysis of error control in wireless sensor networks," *Proc. of IEEE SECON*, vol. 2, pp. 585–594, Sept. 2006.

[12] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical en-route filtering of injected false data in sensor networks," *IEEE JSAC*, vol. 23, no. 4, pp. 839–850, April 2005.

[13] Z. Yu and Y. Guan, "A dynamic en-route scheme for filtering false data injection in wireless sensor networks," *Proc. of IEEE INFOCOM*, pp. 1–12, April 2006.

[14] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *Proc. of ACM MOBICOM*, August 2002, pp. 56–67.

[15] K. Akkaya and M. Younis, "A survey on routing protocols for wireless sensor networks," *Elsevier Ad Hoc Networks Journal*, vol. 3, pp. 325–349, May 2005.

[16] G. T. S. N. Simulator, "Gtsnets," 2007.

[17] S. Uluagac, R. Beyah, and J. Copeland, "Secure source-Based loose time Synchronization (sobas) for wireless sensor networks," Communications Systems Center, School of ECE, Georgia Institute of Technology, Tech. Rep., 2009. [Online]. Available: http://users.ece.gatech.edu/ selcuk/sobas-csc-techreport.pdf

[18] R. V. et al., "Encryption overhead in embedded systems and sensor network nodes: modeling and analysis," in *Proc. of ACM CASES '03*, 2003, pp. 188–197.

[19] C. Kraub, M. Schneider, K. Bayarou, and C. Eckert, "Stef: A secure ticket-based en-route filtering scheme for wireless sensor networks," *The 2nd Int. Conf. on Availability, Reliability and Security (ARES)*, pp. 310–317, April 2007.

[20] M. Passing and F. Dressler, "Experimental performance evaluation of cryptographic algorithms on sensor nodes," Oct. 2006, pp. 882–887.

[21] M. Ma, "Resilience of sink filtering scheme in wireless sensor networks," *Elsevier Comput. Commun.*, vol. 30, no. 1, pp. 55–65, 2006.

[22] J. Hyun and S. Kim, "Low energy consumption security method for protecting information of wireless sensor networks," *LNCS Advanced Web and Network Technologies, and Applications*, vol. 3842, pp. 397–404, 2006.

[23] S. Zhu, S. Setia, S. Jajodia, and P. Ning, "An interleaved hop-by-hop authentication scheme for filtering of injected false data in sensor networks," in *Proc. of IEEE Symposium on Security and Privacy*, 2004.

[24] A. Perrig, R. Szewczyk, V. Wen, D. Cullar, and J. Tygar, "Spins: Security protocols for sensor networks," in *Proc. of MOBICOM'01*, 2001.

[25] M. Luk, G. Mezzour, A. Perrig, and V. Gligor, "Minisec: A secure sensor network communication architecture," *6th International Symposium on Information Processing in Sensor Networks (IPSN)*, pp. 479–488, April 2007.

Arif Selcuk Uluagac is a Ph.D. candidate in the School of ECE at the Georgia Institute of Technology as a member of the Communications Systems Center. He received his B.Sc. in CompEng. from the Turkish Naval Academy and M.Sc. degrees in ECE from Carnegie Mellon University in 1997 and 2002, respectively. He received "2007 Outstanding ECE Graduate Teaching Assistant Award" from the School of ECE at Georgia Institute of Technology. He is a student member of IEEE, ACM, and ASEE.

Raheem Beyah is an Assistant Professor in the Department of Computer Science at Georgia State University where he leads the Georgia State Communications Assurance and Performance Group (CAP). He is also an Adjunct Professor in the School of ECE at the Georgia Institute of Technology. He received his Bachelor of Science in Electrical Engineering from North Carolina A&T State University in 1998. He received his Masters and Ph.D. in Electrical and Computer Engineering from the Georgia Institute of Technology in 1999 and 2003, respectively. His research interests include network security, wireless networks, and network traffic characterization and performance. Dr. Beyah received the National Science Foundation CAREER award in 2009. He is a member of ACM, NSBE, and a senior member of IEEE.

Yingshu Li received her Ph.D. and M.S. degrees from the Department of Computer Science and Engineering at University of Minnesota-Twin Cities. She received her B.S. degree from the Department of Computer Science and Engineering at Beijing Institute of Technology, China. Dr. Li is currently an Assistant Professor in the Department of Computer Science at Georgia State University. Her research interests include Optimization in Networks, Wireless Sensor Networks, Wireless Networking and Mobile Computing, Approximation Algorithm Design and Computational Biology. Dr. Li is the recipient of an NSF CAREER Award.

John A. Copeland holds the John H. Weitnauer, Jr., Chair as a professor in the School of ECE at the Georgia Institute of Technology (Ga.Tech), and is a Georgia Research Alliance Eminent Scholar. Dr. Copeland was VP Techn. at Hayes (1985-1993), and VP Eng.Techn. at Sangamo Weston, Inc. (1982-1985) and served at Bell Labs (1965-1982). He founded Lancope, Inc. (2000), and invented the StealthWatch network security monitoring system. Dr. Copeland received B.S., M.S. and Ph.D. degrees in physics from the Ga.Tech. He has been awarded 48 patents and has published over 100 technical articles. In 1970 he received the IEEE's Morris N. Liebmann Award. He is a Fellow of the IEEE.