Leveraging Channel Diversity for Key Establishment in Wireless Sensor Networks

Technical Report

December 2005

Matthew J. Miller
Department of Computer Science,
and Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
mjmille2@uiuc.edu

Nitin H. Vaidya
Department of Electrical and Computer Engineering,
and Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
nhv@uiuc.edu

***Abstract*— As the use of sensor networks increases, security in this domain becomes a very real concern. One fundamental aspect of providing confidentiality and authentication is key distribution. While public-key encryption has provided these properties historically, sensors are resource constrained and benefit from symmetric key approaches.**

In this work, we propose a novel protocol for symmetric key distribution that uses the multiple channels available on sensor hardware. This channel diversity, along with spatial diversity of device locations, allows neighboring sensors to establish secure link keys from plaintext keys that are broadcast by sensors in their neighborhood. In particular, we show that using even *one* extra channel during the initialization procedure significantly improves the security of key establishment.

Via analysis and simulation, we show that our protocol performs well in terms of network connectivity and resilience to colluding malicious devices, when compared to previous work. We show that our protocol can achieve over 90% connectivity among neighboring sensors with link keys that are uncompromised even when 80% of the devices in the network are malicious and collude. Finally, we present a thorough discussion of the comparative advantages and disadvantages of our approach compared to previous techniques.

I. Introduction

As sensor networks become more ubiquitous, security becomes a major concern. It is also an excellent opportunity for researchers to integrate security in the initial stages of protocol design, rather than an added afterthought as has occurred in many previous network protocols. To this end, an important aspect of sensor network security is key establishment. Using secure keys is the foundation of many other aspects of security such as encryption for confidentiality and message authentication for integrity.

Certain properties of sensor networks make the key establishment problem unique compared to protocols for other types of networks:

- *Sensors are resource constrained:* The sensors are generally assumed to be small devices that are barely noticeable in most environments. This implies that resources such as memory, computation power, and transmission rates are much more constrained than in desktops and laptops. As an example, Mica Motes [2] have a CPU speed of 4 MHz, a few hundred kilobytes of memory, and a bitrate of 19.2 kbps. From a security perspective, this means that symmetric keys are preferable to asymmetric keys and, ideally, only a small portion of memory is devoted to key material [3]–[7].
- *Packets are broadcast over the air:* This is true for wireless networks of all types and means that it is much easier for an adversary to tap into a device's communication channel. Thus, it is assumed that an attacker can overhear any packet transmitted in its vicinity.
- *Deployment may be large in scale:* Networks may be on the order of thousands of sensors. Thus, it is preferable to localize key establishment as much as possible.
- *Topology may be uncontrolled:* It is envisaged that sensors may, for example, be tossed out of an airplane to monitor an area. In such a case, there is no advance knowledge of which sensors will be neighbors in the network. Thus, it is conceivable that a device is equally likely to be neighbors with any of the N sensors being deployed. If a sensor wishes to share a secret key with each of its neighbors, it needs to store $N-1$ in memory, creating a scalability problem given the memory size of the sensors and the potentially large scale of the network.
- *Deployment may be in hostile territory:* Sensors may be used to monitor enemy areas, therefore it is possible that an adversary may be able to populate the network with a significant number of its own devices.
- *Planned incremental additions maybe desired:* Sensor networks may be long-lived and require the owner to deploy new sensors as older ones fail (whether maliciously or due to battery exhaustion). Additionally, it may be desirable for an owner to occasionally "upgrade" the network by increasing the density of the sensors to get better sensing coverage. In this work, we assume that incremental additions are relatively rare events that can be planned reasonably well in advance.

This is an extended version of a paper (edited for space considerations) that appears in IEEE Infocom 2006 [1]. This work is partially funded by a NSF fellowship.

Given these properties, we design a key establishment protocol based on symmetric cryptography that can scale to hundreds or thousands of sensors without any prior knowledge of sensor locations and demonstrate resilience to the threat model described in Section III-B. In this paper, we focus on distributing pairwise keys between one-hop neighbors in a sensor network. Pairwise keys are important in sensor networks for a couple of reasons. First, when sensors are sending their data to a sink, it allows secure aggregation of data at each hop since a sensor shares a secret key with each of its children as well as its parent. Second, pairwise keys can be used to authenticate a hash chain commitment that can then be used to do, for example, authenticated broadcasts [8].

The design space for pairwise key distribution lies between two extremes. On one end, each sensor could be loaded with $N-1$ keys prior to deployment such that it shares a secret key with every other sensor in the network. This scheme offers the most security since no information about the keys is ever broadcast and a compromised sensor gives an attacker no information about keys being used by other sensor pairs. However, this scheme suffers greatly from a scalability viewpoint since a sensor may need to store thousands of keys, of which it probably only uses a small subset.

At the other extreme, each sensor is given one key which it shares with every other sensor in the network. From a scalability viewpoint, this scheme is excellent since a sensor only stores one key regardless of the size of the network. However, this scheme offers little resilience to an attacker since if one device is compromised, the communication between every pair of sensors is also compromised.

The major contributions of this work are as follows. First, we present a novel, distributed protocol that requires a small amount of memory for storage and, with high probability, ensures each link in the network shares a unique key. Through analysis, we show the properties of our protocol and demonstrate that it is feasible within the resource constraints of current sensor hardware.

Second, we show that diversity of sensor channels and location can be a benefit to sensor network security. While such diversity has been widely used to improve performance in wireless networks (e.g., increasing spatial reuse, decreasing bit errors), to our knowledge, this is the first work to apply these concepts to symmetric key establishment. In particular, we show that the location diversity of randomly deployed sensors greatly improves resilience to adversarial hardware that is deployed in the same manner. It is also demonstrated that using only *one* extra channel during initialization (i.e., using two channels instead of one) greatly improves security.

This paper is organized as follows. In Section II we review related work. Section III gives some background information relevant to our protocol. Section IV presents our proposed protocol. Properties of the protocol are demonstrated via analysis in Section V and via simulation in Section VI. We compare our protocol to other key establishment mechanisms in Section VIII. In Section VII we discuss extensions to the protocol to support incremental sensor deployment. Section IX discusses future work and Section X concludes the paper.

II. Related Work

In [9], the SPINS security architecture for sensor networks is presented. It includes a key establishment protocol that requires two sensors to get a pairwise key from a trusted server with which both of the sensors share a secret key. A disadvantage of this approach is that the server may become a bottleneck in large networks. The LEAP architecture [10] provides a method of establishing pairwise keys provided sensors cannot be compromised during a short initialization phase after deployment and the sensor hardware can be trusted to completely erase keying material after initialization.

Eschenauer and Gligor [6] were among the first to consider key predistribution for sensor networks. In their work, referred to as the basic scheme, sensors are loaded with randomly chosen keys out of a master key pool prior to deployment. After deployment, a sensor can securely communicate with its neighbors if it shares at least one key in common with the neighbor. Chan et al. [3] extend the basic scheme to require neighbors to share q keys in common before a link is possible. This improves security at the cost of decreased connectivity. Their work also proposes the idea of using multiple node disjoint paths to strengthen security. This is a different form of diversity than what we propose, but demonstrates how the concept can improve security in the form of diverse path selection. Other schemes propose that keys be distributed deterministically based on a sensor's ID [11], [12].

Du et al. [4] adapt a key predistribution scheme originally proposed by Blom [13] for sensor networks by using finite fields to generate multiple key spaces that can be randomly deployed to sensors. Liu et al. [5] extend a key distribution method proposed by Blundo et al. [14] that uses polynomial based distribution methods. In Section VIII-A, we further discuss the comparative advantages and disadvantages of our approach compared to key predistribution schemes in general.

The work most similar to ours is that of Anderson et al. [7]. Their protocol is based on the assumption that the number of adversary devices in the network at the time of key establishment is very small (in their results, less than 3% of the devices are adversaries). Thus, during initialization, a sensor u broadcasts a randomly generated plaintext key, k_u, that is overheard by all its one-hop neighbors (including adversaries). Each of u's neighbors replies with the message $\{v, k_{uv}\}_{k_u}$, where v is the neighbor's ID and k_{uv} is a pairwise key randomly generated by v.[1] After this exchange, u and v use key k_{uv} for communication. Power control is used to reduce the number of devices that overhear the key exchange.

In Section VIII-B, we discuss in detail the differences between our work and Anderson's. We briefly mention these differences here. First, our protocol is much more resilient to eavesdropping by attacking devices since we leverage channel

[1]The notation $\{M\}_k$ denotes a message, M, encrypted using key k.

diversity and utilize location diversity more.[2] Second, a link cannot be authenticated in Anderson's scheme since u or v has no way to verify the sender of the messages. In contrast, we provide mechanisms that allow a trusted source to authenticate sensor IDs and broadcasted keys. Refer to Section VIII-B for more detail about these differences.

III. Background

A. System Model

We assume that the sensors are deployed such that their locations are distributed uniformly at random in a desired area. In our model, the network is relatively dense (e.g., more than ten one-hop neighbors per sensor) so there are multiple neighbors which a sensor can overhear.

Our analysis also assumes a radio model that can be represented by unit disks and that links are symmetric; so if A can hear B, then B can also hear A. As part of future work, discussed in Section IX, we plan to implement the protocol do determine the effects of a more realistic physical layer. We assume that the radio can communicate on multiple, non-interfering channels, but can only listen to or transmit on a single channel at any given time. For example, devices such as Mica Mote sensors [2] and any 802.11 compliant hardware [15] can use frequency-division multiple access (FDMA) to achieve this.

B. Threat Model

In this work, an attacker's primary objective is to learn the link key that a legitimate pair of sensors is using for communication. If the attacker is able to learn this key, then the encryption and authentication for the link is no longer secure. As in previous work [3]–[7], we consider denial-of-service to be beyond the scope of this paper. See [16] for a discussion of possible solutions to address such attacks in sensor networks.

The attacker has two means by which it attempts to learn link keys. The first is by compromising legitimate sensors and learning all the keying material that is stored on the device. In this case, obviously all communication with the compromised sensor becomes insecure. However, learning the keying material of the compromised sensor may also assist the attacker in learning the link keys being used by non-compromised sensors. In this work, we assume that attackers can compromise sensors *any* time after deployment. We note that this is a stronger attack model than is assumed in some key establishment protocols [10] in which an attacker can only compromise sensors *after* some initialization time.

The second method by which an attacker may attempt to learn link keys is listening to the plaintext keys exchanged during the initialization phase, discussed in Section IV. Since all the information needed to create link keys is broadcast in plaintext at some point during the initialization, the attacker may be able to reconstruct link keys by eavesdropping.

As in [7], we assume that the hardware that an attacker deploys is similar to the legitimate sensor hardware in the network. Thus, any radio hardware an attacker uses has a receive threshold equal to or larger than that of the sensors in the network. That is, for a packet transmitted at a certain power level, the attacker's radio cannot receive a packet if it is farther away than the distance that sensors can receive the packet. As another result of this assumption, the attacker cannot execute wormhole attacks whereby colluding devices can propagate data across the network via an out-of-band channel.

In this work, we do not investigate coordinated channel assignment and switching protocols that an adversary may use with multiple radio devices or colluding single radio devices. Such scenarios are an interesting avenue for future work. However, we *do* consider the effects of an attacker adding more devices that collude by combining the knowledge of their overheard packets.

C. Bloom Filters [17]

In our protocol, we use Bloom filters [17] to communicate sets of keys. In this section, we give a brief overview of the operation of Bloom filters. For more information, the reader is encouraged to peruse any of the numerous tutorials or surveys available on the subject (e.g., [18]).

Each sensor is preloaded with the same h one-way hash functions [19], $H^1_{BF}, H^2_{BF}, \ldots, H^h_{BF}$.[3] Given an input, each of these hash functions maps an object to a value between 1 and s according to a uniform distribution. The Bloom filter is a bit vector of s bits. Initially, every bit is set to 0. Given an object, v_i, it is placed in the filter by setting the bits $H^1_{BF}(v_i), H^2_{BF}(v_i), \ldots, H^h_{BF}(v_i)$ to 1. Thus, for each object (say, $i = 1$ to n) added to the filter, up to h bits are set to 1. After receiving a Bloom filter, a sensor can check to see if an object, v_j, is in the filter by testing if the bits $H^1_{BF}(v_j), H^2_{BF}(v_j), \ldots, H^h_{BF}(v_j)$ are all set to 1. If this test is true, then the object is considered to be in the filter. False positives occur when each of the h values for an object, v_j, maps to a bit that was set to 1 by the hash function for some object other than v_j. In Section V, we investigate what values of h and s are appropriate for our scheme to avoid false positives with high probability. However, our scheme is designed to be robust against occasional false positives as described in Section IV-E.

D. Merkle Trees [20]

Another cryptographic primitive used in our protocol is Merkle trees [20]. We use Merkle trees to provide authentication that the set of keys being broadcast by a sensor was generated by a trusted source prior to deployment.

Assume that a trusted source has m objects that it wishes to distribute among untrusted sensors. The goal of a Merkle tree is that when a sensor claims to have a certain object, the value of that object can be authenticated without contacting

[2]We note that the goal in [7], unlike our work, is not to make it difficult for a nearby attacker to compromise a link nor to operate in hostile environments where there may be a large number of adversaries.

[3]A one-way function, H, is easy to compute (i.e., given object x, $H(x)$ can be computed in polynomial time), but hard to invert (i.e., given $H(x)$, no polynomial time algorithm exists to find y such that $H(y) = H(x)$).

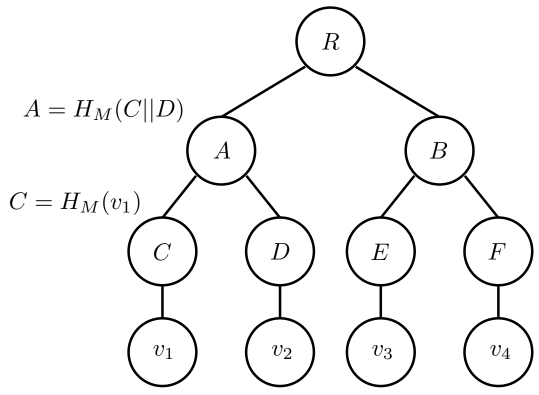

Fig. 1. An example Merkle tree [20] for four objects: v_1, v_2, v_3, v_4. The leaf nodes are generated by doing a one-way hash, H_M, on the corresponding object. The interior nodes are generated by doing a one-way hash, H_M, on a concatenation of the children of a node.

the trusted source. To do this, the trusted source generates the Merkle tree for the objects prior to deployment and then loads each sensor with the root of the tree as well as the $\lg m$ interior nodes of the tree needed to verify the authenticity of each object distributed to the sensor.

Figure 1 gives an example of a Merkle tree for four objects: v_1, v_2, v_3, v_4. First, each leaf node of the tree is generated by hashing one of the objects. For example, $C = H_M(v_1)$, where H_M is a one-way hash function [19]. Each of the interior nodes of the tree is generated by hashing a concatenation of the node's left and right child. For example, $B = H_M(E||F)$. This continues until the root, R, is generated.

Each sensor that wishes to later verify the authenticity of objects is loaded with R by the trusted source. As an example, consider a sensor that is given object v_2 by the trusted source. The sensor is then also loaded with the values necessary to authenticate v_2 (i.e., C and B). When the sensor wishes to verify the authenticity of v_2, it can do so by transmitting v_2 along with the values of C and B. With these values, any sensor that knows R can verify the authenticity of v_2 by checking that $R = H_M(H_M(C||H_M(v_2))||B)$.

IV. Protocol Description

A. Overview

We begin with a brief overview of the details that are presented in Section IV-B through IV-D. Table I provides a key for the notation used in this section.

Prior to deployment, each sensor is loaded with a unique, non-overlapping set of α keys by a trusted source. Unlike previous work [3], [6], this set of keys is known only to that sensor and *not* part of a larger shared pool of keys. Along with these keys, the sensors are loaded with the Merkle tree nodes necessary to authenticate the Bloom filter of their α keys (as discussed in Section IV-B, the actual keys could be authenticated rather than the Bloom filter at the cost of increased overhead).

The sensors are then deployed uniformly at random over a given area. On a common channel, each sensor broadcasts the Bloom filter of the α keys with which it was loaded along with the Merkle values necessary to authenticate the filter. This allows sensors to verify that future keys received from a given neighbor were given to that neighbor by the trusted source. In Section IV-B, we discuss this aspect of the protocol in depth as well as how to deal with attacking devices that attempt to rebroadcast legitimate keys and generate arbitrary keys.

TABLE I
Protocol Notation

Notation	Description
c	Number of non-interfering channels available
α	Number of keys broadcast by each sensor during initialization
λ	Maximum number of advertisement keys from any one of a sensor's neighbors
γ	Cumulative number of keys a sensor includes in its advertisement from neighbors
η	Minimum number of advertised keys a sensor must share with a neighboring sensor to engage in communication
h	Number of hash functions per Bloom filter
s	Size of the Bloom filter (bits)

During initialization, sensors switch their radios to a channel chosen uniformly at random and choose a non-deterministic amount of time to listen to the channel before switching to another channel. Also during this time, the sensors choose non-deterministic times to broadcast each of their α keys in plaintext. The key broadcast times are independent of the channel switching times. This procedure continues until all sensors have had a chance to broadcast all of their keys. Each key is sent on the channel to which the sensor is currently listening at the chosen broadcast time. During this initialization phase, sensors store every key that they transmit as well as every key that they overhear in broadcasts by their neighbors.

At the end of the initialization phase, all sensors switch their radio to a common channel, and perform a key discovery phase during which a sensor tries to establish a unique key to communicate with each neighbor. For the discovery phase, each sensor hashes all its known keys into a Bloom filter and broadcasts the filter. Every time a sensor overhears a filter, it searches the Bloom filter of its own keys to determine which keys it has in common with the sender of the overheard filter.

After the key discovery phase, the key establishment phase is performed. During key establishment, a sensor broadcasts a separate message for each filter it overheard in the key discovery phase. In this message, the sensor includes a Bloom filter indicating the keys it believes it has in common with the sender of the original Bloom filter along with a random nonce encrypted by a link key composed of those shared keys. If the sensor receives a single acknowledgment with a properly encrypted, incremented nonce value, a link key has been established with that neighbor. This process continues until a sensor has a unique key for each of its neighbors.

At this point, with high probability, the sensor shares a secret key with each of it neighbors and only needs to store its link keys, α preloaded keys, and Merkle nodes for authentication. In Section VII, we discuss how sensors can be incrementally added after the initial deployment of the network. Now, we describe the protocol in detail.

B. Predeployment Phase

We now describe how keys for sensors in the initial deployment are loaded onto each device. In Section VII, we discuss how this phase of the protocol can be extended to allow incremental sensor additions after this initial deployment.

A trusted authority generates α keys per sensor that are loaded on each sensor before deployment. The keys generated for each sensor are unique to that sensor and *not* part of a global key pool as has been done in previous work [3], [6] (i.e., a sensor's set of keys do not overlap with another sensor's set of keys). Once the key sets are generated for the sensors, the trusted source computes the Bloom filter for each sensor's set of keys as described in Section III-C. Finally, the trusted source creates a Merkle tree, described in Section III-D, as follows. The leaves of this Merkle tree are generated by hashing a concatenation of the sensor's ID with the Bloom filter of its keys (discussed in Section IV-B), denoted as BF.[4] Thus, the Merkle leaf for that sensor is $H_M(ID||H_M(BF))$.[5]

At this point, each sensor is loaded with the α keys that the trusted source generated for that device, the root of the Merkle tree, and the $\lg N$ interior Merkle nodes needed to authenticate the sensor's Bloom filter of its keys, where N is the number of sensors deployed. After deployment, every sensor listens to a common channel. During this period, each sensor broadcasts the Bloom filter of its α preloaded keys along with the $\lg N$ Merkle values needed to authenticate its Bloom filter and ID. Every sensor stores the ID and Bloom filters for each of its neighbors for the duration of the key distribution procedure.

When the key broadcasting begins, a sensor only accepts a key from a neighbor if the key exists in the Bloom filter associated with the neighbor's ID. This scheme is vulnerable to two attacks. First, an attacker may try to assume another sensor's identity by rebroadcasting the packets containing a legitimate sensor's ID, Bloom filter, and Merkle values and then rebroadcasting keys that it hears the legitimate sensor broadcast during key initialization. In Section IV-D, we explain why an attacker could benefit from such a strategy. However, such an attack can be *detected* if any legitimate sensor overhears a key broadcast twice claiming to be from the same ID since each key is to be broadcast only once. Alternatively, the malicious device can be detected if the sensor that is the victim of identity theft overhears the attacker using its keys and ID. Such detection will happen with high probability in relatively dense sensor networks. It is an area of future work to quantify the optimal strategy for an attacker to attempt to rebroadcast legitimate packets while remaining undetected.

[4]Alternatively, the leaves of the Merkle tree could be a hash of *each of the sensor's keys* instead of the Bloom filter of the keys. This would increase the protocol's storage and communication overhead, but eliminates the possibility of false positives and increases robustness against attacks that generate arbitrary keys. Our work does not explore this approach.

[5]If there is some knowledge of a sensor's post-deployment location *prior* to deployment, then the techniques from [21] can be used to reduce the amount of communication required to verify that a sensor's leaf is legitimate.

The second type of attack is for the malicious devices to generate arbitrary keys which hash to all 1's in a legitimate sensor's Bloom filter. In Section V, we analyze the effort required by an attacker to generate these arbitrary keys and see that somewhere on the order of tens to hundreds of arbitrary keys must be generated, on average, for an attacker to create such a key. Also, increasing the size of the Bloom filter can greatly increase the effort required to generate arbitrary keys. In applications that require greater immunity to this attack, at the expense of storage and communication overheard, each *key* could be placed as a leaf in the Merkle tree, resulting in $\alpha \times N$ leaves for the tree instead of only N leaves. This would allow each broadcasted key to be authenticated individually.

C. Initialization Phase

The goal of the initialization phase is that each pair of neighbors knows a unique subset of keys at the end of the process. The length of the initialization process depends on how large α must be to achieve the desired probability that all links have a unique set of keys and how much contention there is for the channel. We assume that broadcasts are sent using CSMA/CA to reduce collisions. In Section V, we analyze what values of α are appropriate for a deployment.

Neighbor pairs are expected to share a unique subset of keys because of the channel and spatial diversity in the network. The primary form of diversity is from the channel switching of the protocol. The only constraints we make on the channel switching algorithm are: (1) each of a sensor's α keys are broadcast during the initialization phase, (2) that each broadcast from a sensor is, on average, overheard by a subset of d/c neighbors, where d is the expected number of one-hop neighbors of the sensor and c is the number of channels available, and (3) a different subset of neighbors overhears each of a sensor's broadcasts, with high probability. Thus, channel selection gives diversity in the subset of neighbors that overhears each of a sensor's key broadcasts.

This is illustrated in Figure 2 where E is a neighbor of A, B, and C. We refer to a key that is known by both A and B as a *shared key*. In Section IV-D and Section IV-E, we elaborate on how the shared keys are used to generate a *link key* for the sensor pair. The link key is the secret symmetric key that A and B use for secure communication. When C broadcasts a key, there is a $(1/c)^2$ probability that A and B are both listening to the channel on which C broadcasts. Given that A and B are listening to the same channel on which C broadcasts, there is a $1-(1/c)$ probability that E is *not* listening to that channel. When this occurs, A and B have a shared key that can keep the link secure against eavesdropping by E.

The spatial diversity comes from the fact that two neighbors, say A and B, may overhear broadcasts from a unique set of common neighbors. Consider the scenario in Figure 2 where A and B are one-hop neighbors that are both within range of C and D. However, C and D are *not* within range of each other. Thus, for example, if one of A and B's shared keys comes from a broadcast by C, then D will not know that key and, hence, A and B's link can use the overheard key from C

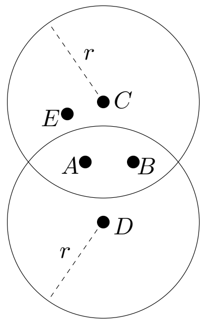

Fig. 2. One-hop neighbors A and B can both overhear broadcasts from C and D. However, C cannot overhear broadcasts from D and vice versa. E is a one-hop neighbor of A, B, and C.

to secure the link from being compromised by D. Similarly, if one of A and B's shared keys comes from D's broadcast, then the link can be made secure against eavesdropping by C. Therefore, over the course of several broadcasts, the link between A and B is expected to eventually be secure against eavesdropping by both C and D with high probability.

Another form of diversity from the wireless channel comes from uncorrelated packet loss. Thus, even if A, B, and E all listen to the channel on which C broadcasts a key, E may receive a packet in error that A and B receive correctly. In this case, A and B learn a key that E does not know. Packet loss can occur for a couple of reasons. First, it may be lost due to noise and interference degrading the received signal.

If two sensors do end up in close proximity to each other, our protocol still provides some security from diversity. There are two cases to consider. The first is when two sensors are close enough to have the same set of one-hop neighbors, but still far enough apart to have uncorrelated packet loss. In this case, the sensors may still receive a different set of keys since they are switching to different channels *and* experiencing different packet losses. The second case is when the sensors are within the coherence distance of each other. In this case, the diversity in key sets still exists since the sensors are switching to a different set of channels.

D. Key Discovery Phase

When the initialization phase completes after the specified time, in order to determine a link key, sensors must discover which keys are known by its neighbors. To reduce communication overheard, we use Bloom filters [17] to advertise key sets in a compact manner.

A sensor creates a Bloom filter for advertising its keys by including each of the α keys that it transmitted *and* some of the keys it overheard from the broadcasts of others. We denote the set of keys that a sensor u overheard from other neighbors as K_u. Sensor u then chooses a subset of size γ to advertise in its filter (i.e., γ is a predetermined constant that, with high probability, is less than $|K_u|$). Since the sensor knows the source of each of its overheard keys, there is an upper limit, λ, placed on how many keys in this advertised subset come from any one neighbor. This avoids giving disproportionate influence to any one neighbor in establishing link keys. The sensor creates its advertisement by placing each of these $\alpha+\gamma$ keys into an s-bit Bloom filter. Thus, to place key k in the filter, a sensor sets the bits $H^1_{BF}(k), H^2_{BF}(k), \ldots, H^h_{BF}(k)$ to one. A sensor also stores the h values associated with *all* of the keys that it overheard or transmitted during the initialization phase so that it can check its key hashes for inclusion in the Bloom filters of its neighbors. Using CSMA/CA the sensors broadcast their s-bit filters.

Upon receiving a filter from one of its neighbors, a sensor checks to determine which subset of keys it shares with the sender of the filter. Assume that sensor u overhears an advertisement from sensor v. Node u checks to see which of its known keys are included in v's filter. For each key that all h associated bits are set in the filter, u adds the key to the list of keys it potentially shares with v. Because Bloom filters are susceptible to false positives (but not false negatives), u may add a key to the list that is unknown to v. In Section IV-E, we describe how this list of keys is verified. After a filter has been received from each of u's neighbors, it has a list of keys that it believes to be shared with each neighbor. For security, we can require that a sensor only participates in the key establishment phase with neighbors that share some minimum number of keys, η, with the sensor. η can be determined based on the expected number of keys a link should share such that, with high probability, their subset of shared keys is unique. We note that η is similar to the q value in [3].

Now, u attempts to determine a unique subset of keys that it shares with each neighbor and creates a link key that will be used for future communication. The set of keys that sensor u believes it shares with sensor v is denoted as K_{uv}. To attempt to create a shared key with v, u chooses a subset of K_{uv} of size η. It creates the link key, k_{uv}, as the hash of these η shared keys: $k_{uv} = hash(k_1||k_2||\ldots||k_\eta)$, where $k_1, k_2, \ldots k_\eta \in K_{uv}$. Node u will attempt to verify k_{uv} during the key establishment phase described in Section IV-E.

If a sensor determines that it does not share at least η keys with some neighbor, then there are a few options. First, it can choose not to use the link. In a dense network, not being able to use a small number of links may be acceptable. Alternatively, the sensors can request that the sensors do another initialization procedure at a later time to attempt to add the links,[6] though such a technique would require some authentication mechanism for the request to avoid attackers spuriously sending such requests. Another option is to use one of the multipath reinforcement mechanisms described in [3] if there is enough trust among its neighbors to do so.[7]

E. Key Establishment Phase

The final phase of the protocol is the key establishment phase. At this point, each sensor knows of a subset of keys

[6]If another link key initialization procedure occurs, sensors that established link keys previously do not attempt to re-establish a new link key.

[7]Multipath reinforcement allows a sensor pair, A and B, that do not share η keys to use m shared neighbors, $nbr_1, \ldots, nbr_m$ to establish keys if A and B both already share η with each of these m neighbors. See [3] for more details.

that it believes it shares with each of its neighbors (sometimes this belief may be erroneous as discussed below). Furthermore, as we show in Section V and Section VI, with high probability, this subset of keys is unique to that sensor pair. Each sensor has a list of unique filters received in the key discovery phase. We refer to this as the *filter list*. In key establishment, a sensor, u, challenges its neighbor, v, with the key k_{uv} that it generated in the previous phase. Specifically, u sends a random nonce, RN, encrypted by key k_{uv} to v along with a Bloom filter containing the η keys that compose k_{uv}. This packet is called a Link Request ($LREQ$). Thus, $LREQ = (v||u||\{RN\}_{k_{uv}}||BF(k_{uv}))$, where $BF(k_{uv})$ is a Bloom filter containing the keys that compose k_{uv}. We note that the size of $BF(k_{uv})$ should be much smaller than the Bloom filters send during the advertisement phase since, typically, $\eta \ll \alpha + \gamma$.

When v receives the $LREQ$, it searches K_v to determine if it believes it knows every key in $BF(k_{uv})$. If v can decrypt the $LREQ$ correctly using the keys that u used to compose k_{uv},[8] it will reply with a Link Reply packet ($LREP$). The form of this packet is $LREP = (u||v||\{RN+1\}_{k_{uv}})$. Once u receives the $LREP$ and verifies that the incremented value of RN is correct, the link key is established and, with high probability, k_{uv} is known only to u and v.

It is possible that v is not able to decode u's $LREQ$ correctly for one of two reasons. First, there could have been a false positive when u determined its shared keys from v's advertisement and it used the key that caused this false positive to compose k_{uv}. The second reason is that there was a false positive when v determined the keys that composed k_{uv} from the Bloom filter in u's $LREQ$. If one of these two event occur, then v can reply with its own $LREQ$ to u using some different subset of keys that it believes the two sensors share. If the sensors are unable to agree on a shared key after some specified number of $LREQ$ exchanges, then then can resort to one of the methods described in the previous section (i.e., do not establish the link, request another link key initialization procedure, or use multipath reinforcement [3]).

V. Analysis

In this section, we present analysis, followed by simulation in Section VI. We use the notation in Table I and Table II. We consider two sensors, u and v, that have d_{uv} common neighbors. We are interested in the probability that the link key generated by u and v is known by another sensor, w, in the system. Thus, our analysis does not consider multiple, colluding adversary devices. However, the simulation in Section VI, addresses this scenario. In our analysis, w has d_{uvw} neighbors that are shared by u, v, and w, which is a subset of the neighbors shared by u and v (i.e., $d_{uvw} \leq d_{uv}$). In our analysis, we only consider sensors u, v, w, and the d_{uv} shared neighbors of u and v. In a real network, there may be many sensors within range of one of the sensors (i.e., u, v, or w), but not the other two; this is not captured by our analysis. In Section VI, we simulate the protocol in a more realistic setting; our analysis is only intended to give some intuition into the protocol performance. We then derive the probability that u and v share at least one key in their key establishment phase that is unknown to w. Our analysis is based on worst case bounds, with some probability (discussed in Appendix I), and we use the notation $LB(\mu)$ and $UB(\mu)$ to refer to the lower and upper Chernoff bounds [22] for the sum of Bernoulli random variables, where the sum has mean μ. To preserve the flow of the paper, the exact values of the Chernoff bounds and probability with which the bounds hold are discussed and derived in Appendix I. In the analysis, we assume an ideal MAC layer with no collisions. In Section VI, we simulate the protocol using a more realistic MAC layer.

TABLE II
Analysis Notation

Notation	Description
d_{uv}	Shared one-hop neighbors of sensors u and v
d_{uvw}	Shared one-hop neighbors of sensors u and v that are also shared by sensor w
p_e	Probability of a packet loss
p_h	Probability a sensor in overhears a key that is broadcast by one of neighbors
$LB(\mu)$	Lower Chernoff bound on the sum of Bernoulli random variables, where μ is the mean of the sum
$UB(\mu)$	Upper Chernoff bound on the sum of Bernoulli random variables, where μ is the mean of the sum
fp	False positive rate for Bloom filter
p_a	The probability an attacker creates an arbitrary key that will be accepted as a legitimate key

The probability that a sensor hears a key broadcast by one of its neighbors, p_h, is:

$$p_h = \frac{1}{c}(1-p_e) \tag{1}$$

where c is the number of channels and p_e is the probability of a packet loss. Now, we discuss how to select values for γ, η, and λ as a function of the bounds on the number of keys a sensor overhears from its neighbor when each sensor is preloaded with α keys (refer to Table I for parameter definitions). As we will show later, we select these values such that, with high probability, two sensors create a link key that cannot be known by any other sensors in the network.

Recall from Section IV-D, γ is the total number of keys in the Bloom filter of a sensor's advertisement that were overheard from its neighbors. Additionally, the total number of keys in the Bloom filter of the sensor's advertisement is $\alpha + \gamma$. Thus, in order to ensure that, with high probability, u and v each have at least γ overhead keys to include in the Bloom filters of their advertisements, we set γ to be the lower bound on the number of keys overheard by all neighbors:

$$\gamma = \lfloor d_{uv} \cdot LB(\alpha p_h) \rfloor \tag{2}$$

If γ is larger than this, then we can no longer assume that both u and v are able to advertise $\alpha + \gamma$ keys in their Bloom filters (as is necessary in our analysis).

Given that all sensors are able to advertise $\alpha + \gamma$ keys in their Bloom filters, we next focus on selecting η, the number

[8]We assume there are well-known fields in the encrypted part of the $LREQ$ so that a sensor can verify that it correctly decrypted the random nonce.

of keys that u and v must share to have a link. Clearly we can set η no larger than the lower bound on the expected number of keys that two sensors share. This lower bound is explained as follows. Without loss of generality, look at the advertisement that u broadcasts to v. Node v knows, with high probability, at least $LB(\alpha p_h)$ of the keys that u originated. Of the γ keys that u heard from other neighbors and included in the Bloom filter of the advertisement, there is a p_h probability that v overheard each of these keys and, thus, knows at least $LB(\gamma p_h)$ of these γ keys.[9] Therefore, we have:

$$\eta = \lfloor LB(\alpha p_h) + LB(\gamma p_h) \rfloor \tag{3}$$

As mentioned earlier, the values for the Chernoff bounds (LB and UB) are derived in Appendix I.

Finally, we need to select λ, the maximum number of keys in the Bloom filter of a sensor's advertisement from any one of its neighbors (i.e., the maximum number of w's transmitted keys that are included in the advertisement Bloom filter that u sends to v). Notice from Equation 2, that the value of γ is based on receiving *at least* $LB(\alpha p_h)$ keys from each of u or v's d_{uv} neighbors. Thus, when u or v compose their advertisement, it can include $LB(\alpha p_h)$ keys from each of its neighbors in its Bloom filter. Since u and v do not know which of their neighbors may be malicious, it is best to minimize the maximum number of keys any one neighbor can have in Bloom filter of their advertisement. Thus, we set λ to be:

$$\lambda = \lfloor LB(\alpha p_h) \rfloor \tag{4}$$

If λ is smaller, we can no longer ensure that u and v can create an advertisement with γ overheard keys in the Bloom filter. If λ is larger, then, in the worst case, the knowledge that w has of u and v's shared keys is increased.

Now, we can calculate the probability that w knows all of u and v's η shared keys. Among these η keys, w could have originated at most λ as discussed above. Thus, the probability that w knows all of u and v's keys is given by:

$$\left(\frac{d_{uvw}}{d_{uv}} \times \frac{\lceil UB(d_{uvw} \alpha p_h) \rceil}{d_{uvw} \alpha} \right)^{\eta - \lambda} \tag{5}$$

where $UB(d_{uvw})$ is the maximum number of keys a sensor (e.g., w) overhears during the link key initialization procedure.

Thus, the probability that the link key of u and v is not known to any of u and v's neighbors is:

$$\left(1 - \left(\frac{d_{uvw}}{d_{uv}} \times \frac{\lceil UB(d_{uvw} \alpha p_h) \rceil}{d_{uvw} \alpha} \right)^{\eta - \lambda} \right)^{d_{uv}} \tag{6}$$

Finally, we look at the Bloom filter size necessary for our protocol. For brevity, we do not explain the equations used to analyze Bloom filters. The reader is referred to [18] for a detailed derivation and explanation of these equations. We use η as an example of the number of objects we wish to place in a Bloom filter. The minimum filter size (in *bits*), s, required for η objects to achieve a false positive rate of fp is:

$$s \approx \eta \left(\frac{\ln(fp)}{\ln(0.6185)} \right) \tag{7}$$

and the optimal number of hashes per object, h, is given by:

$$h \approx \ln 2 \left(\frac{s}{\eta} \right) \tag{8}$$

As mentioned in Section IV-B, attackers may attempt to create arbitrary keys that map to all ones in a legitimate sensor's Bloom filter. We now determine how many arbitrary keys an attacker must generate, on average, before it can forge a key in such a manner. The probability, p_a, that an attacker generates an arbitrary key that maps to one h times is:

$$p_a \leq \left(\frac{\eta h}{s} \right)^h \approx 1.6968^{\ln(fp)} \tag{9}$$

Thus, the average number of keys the attacker must create before generating a key that will be accepted as legitimate is $(1/p_a)$. For $fp = 10^{-3}$, an attacker has to generate around 38 keys on average. However, at the cost of increasing the size of the Bloom filter, setting $fp = 10^{-5}$ requires the attacker to generate 440 keys on average.

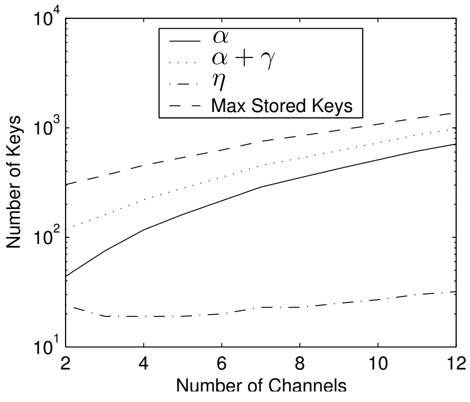

Fig. 3. The size of α, γ, η, and the maximum number of keys stored by a sensor as a function of the number of channels.

A. Numerical Results

We now provide some numerical results from Matlab based on the above analysis. The default values used in these tests are: $d_{uv} = 10$, $d_{uvw} = 7$, $p_e = 0.1$, $c = 3$, and the link is unique, according to Equation 6, with probability 0.99999.

In Figure 3, we see how some of the analytical values change as a function of the number of channels. Note that the vertical-axis is log scale. We see that increasing the number of channels while maintaining a fixed probability of a unique link can greatly increase α (and hence the advertisement size), while the increase in η is modest.

Next, we are interested in the byte overhead required for the advertisement packet and the $LREQ$ packet. Figure 4 shows the byte overhead required as a function of the desired false

[9]We note this ignores the fact that some of the keys that u includes in the advertisement Bloom filter originated from v and, hence, v knows these keys with probability one. This causes the value of η to increase and since we claim a lower bound it is acceptable to ignore keys transmitted by v that are included in u's advertisement Bloom filter.

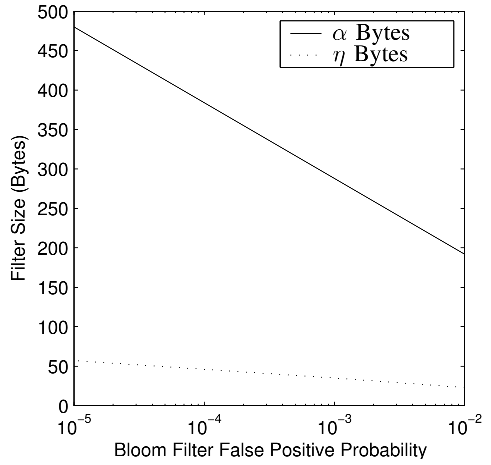

Fig. 4. Number of bytes required to advertise a sensor's keys and the shared keys for a link as a function of the desired false positive rate.

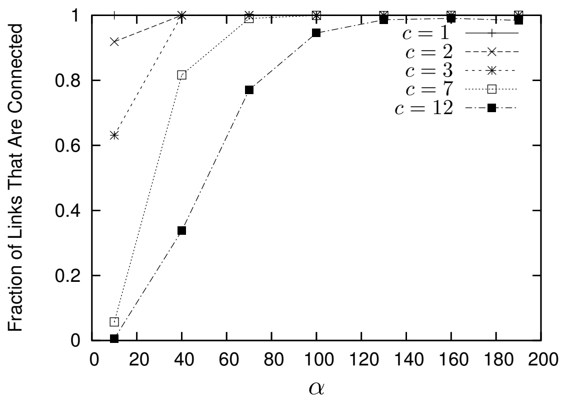

Fig. 5. Connectivity of legitimate sensors vs. α.

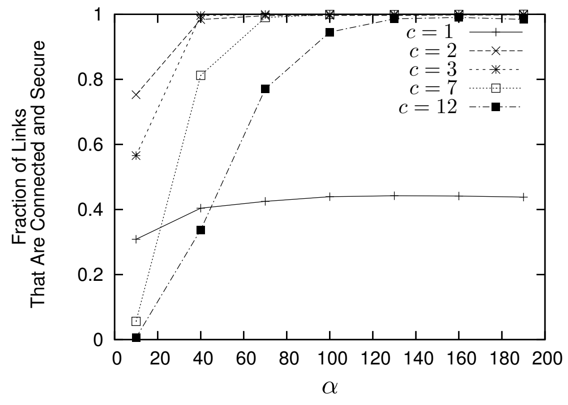

Fig. 6. Secure connectivity of legitimate sensors vs. α.

positive rate of the Bloom filter. While the byte overhead for the advertisement filter is quite large for a sensor network, we note that the packet can be fragmented for efficiency. Also, we note that other predistribution schemes [3], [6] require similar, if not more, overhead for sharing key knowledge even if they do not explicitly evaluate this metric.

VI. Simulation

We simulated our protocol with *ns-2* [23]. In each test, 50 sensors are placed uniformly at random such that the density of the network (i.e., the expected number of one-hop neighbors per sensor) is 10. Each data point is the average of 30 test runs. The default values used in the simulations for the protocol are: $\alpha = 100$, $\gamma = 100$ (i.e., the advertisement size, $\alpha + \gamma$, is 200 keys), and $\eta = 10$ (i.e., a sensor pair must share at least 10 advertised keys for their link to be considered "connected"). We set $\lambda = \alpha$; in future work, we plan to thoroughly investigate the effects of the λ parameter.

To implement the channel switching and key broadcasting discussed in Section IV-C, we use the following algorithms. Sensors are assumed to be synchronized and at fixed intervals, all of the sensors switch to a new channel uniformly at random. Within each fixed interval, if a sensor has not broadcast all of its α keys, it chooses a time uniformly at random to broadcast *one* of its remaining keys.

We set 30% of the sensors, chosen uniformly at random, to be controlled by a malicious entity. In our tests, these malicious sensors follow the same protocol as the uncompromised sensors, however, they collude in their knowledge of plaintext keys learned during the initialization procedure. Thus, collusion between malicious devices is global; we do not restrict them to being neighbors to share their knowledge. Thus, the attacker is able to compromise a link between two legitimate sensors if their set of shared keys is found within the superset of keys known by *all* the colluding malicious sensors.

In our tests, we vary α, c (the number of channels), and the percentage of colluding malicious sensors and consider the following metrics:

Connectivity: defined as the fraction of links between legitimate sensors in the network that share at least η advertised keys.

Secure Connectivity: the fraction of links between legitimate sensors that use a key set with at least one key that it unknown to the colluding malicious sensors.

In Figure 5, we see how increasing α improves the connectivity of legitimate sensor pairs for different values of c. This is expected since sensor pairs will share more keys when their total number of known keys increases. We also note that the connectivity improves with a smaller number of channels, since the probability of a sensor overhearing a neighbor's broadcast is increased.

However, as shown in Figure 6, using a smaller number of channels is not always good from a security perspective. In Figure 5, using $c = 1$ gave a connected topology for all values of α. However, in Figure 6, we see that less than half of the connected links for $c = 1$ remain uncompromised by the attacker. All of the other values of $c > 1$ are much more resilient to attacker compromise, though they require a larger value of α for all of the links to be connected.

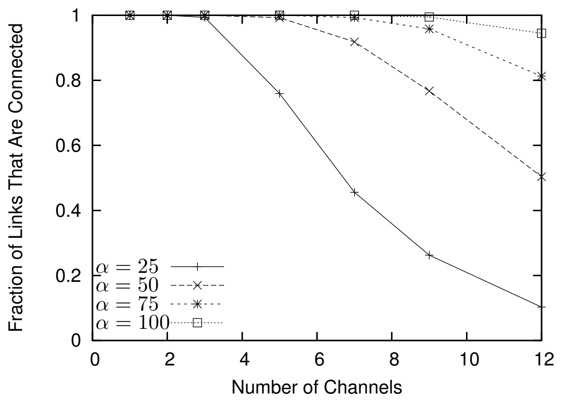

Fig. 7. Connectivity of legitimate sensors vs. the number of channels.

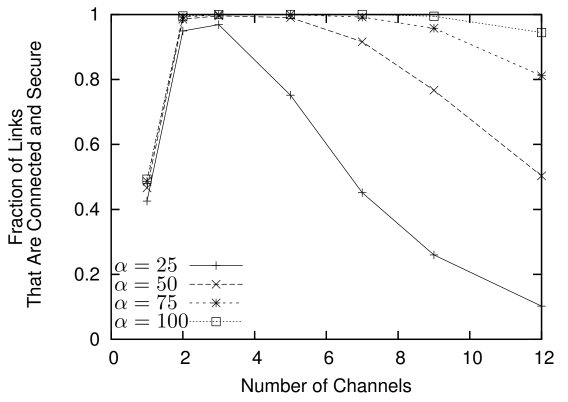

Fig. 8. Secure connectivity of legitimate sensors vs. the number of channels.

In Figure 7 and Figure 8, we look more closely at the affect of the number of channels on connectivity and security. As expected, the connectivity of the network drops as the number of channels increases as shown in Figure 7. The more interesting result is in Figure 8, which shows the large benefit obtained from channel diversity. In particular, by adding *one* extra channel (i.e., $c = 2$), the protocol's security is greatly increased. We note that the benefit from using multiple channel diversity is not possible in the Anderson's work [7], which only uses one channel to broadcast plaintext keys.

At this point, it is evident that setting $c = 2$ provides the significant gain in link security (compared to $c = 1$) while maintaining very high network connectivity (compared to larger values of c). Thus, one may wonder if there is any utility in setting $c > 2$. To answer this question, we refer the reader to Figure 9, which shows the fraction of connected links between legitimate sensors that are secure against the colluding malicious devices as a function of the number of such devices in the network. For brevity, we omit a graph showing the connectivity of the protocol in these settings, but note that the connectivity is greater that 90% in each of these tests and for $c \leq 7$, the connectivity is greater than 99% for each test. Thus, all the values of c shown in Figure 9 provide much higher connectivity than is seen in the results for some

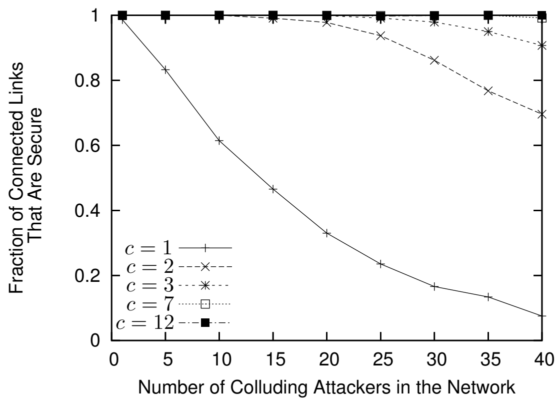

Fig. 9. Fraction of links between legitimate sensors that are secure vs. the number of colluding attacker sensors.

other key predistribution schemes (e.g., [3], [6]).

Figure 9 shows that having more channels allows connected links to be more secure against the colluding malicious devices. In particular, for $c = 2$, when there are about 15 malicious devices in the network (30% of the sensors), some of the links are no longer secure. For $c = 3$, about 25 malicious devices, 50% of the sensors, are necessary to start compromising some of the legitimate links. For $c = 7$ and $c = 12$, even with 40 malicious devices in the network, virtually all of the links between legitimate sensors are secure. We would like to emphasize that this last scenario corresponds to ***80%*** of the sensors in the network being malicious, which is an extremely hostile setting. Recall that in Anderson's work [7], only up to about ***3%*** of the sensors were malicious.

VII. Incremental Sensor Deployment

We assume that incremental sensor deployment is done in a planned manner rather than a completely ad hoc fashion. When the network is initially set up, the owner has accurate knowledge of how many incremental deployments will occur during the lifetime of a sensor as well as the maximum number of new sensors that will be deployed each time. The new sensors are deployed in batches rather than individually.

When new devices are added after the initial deployment, sensors start another link key initialization procedure. A link key initialization procedure after the initial deployment could be triggered by one of several means. It could be at regular, predetermined intervals when the incremental deployment will happen. Alternatively, the new sensors could request the initialization procedure on demand by broadcasting their authenticated Bloom filters. Finally, a trusted source could send packets to the sensors telling them when to start the procedure. As mentioned earlier, sensors that already have established link keys do not attempt to create a new link key during subsequent link key initialization procedures.

Having discussed how the initialization, key discovery, and key establishment phases can be triggered for incremental deployment, we now propose a modification to the predeployment phase to allow authentication between existing and new sensors as well as provide a new key set for existing sensors in a space-efficient manner.

First, we focus on authentication between existing and new sensors since it has a relatively simple solution. Because incremental deployments are planned, it is assumed that the network administrator plans no more than I incremental deployment over the lifetime of a sensor. Furthermore, for the i-th incremental deployment, the network owner knows in advance that no more than N_i sensors will be deployed at that time. Thus, whenever a sensor is deployed along with its associated Merkle tree values, it is also loaded with the roots of the next I Merkle trees that will be generated by the trusted source for future deployments. The trusted source is able to generate these I Merkle roots in advance since the maximum number of sensors that will be associated with each of these I Merkle trees is known in advance. This allows an existing sensor to authenticate newly deployed sensors. To allow newly deployed sensor to authenticate existing sensors, the new sensors are loaded with the previous $I-1$ Merkle roots that were generated for prior deployments. Given that sensors are loaded with these roots by the trusted source, they can now authenticate the keys of sensors deployed over the previous $I-1$ generations or I generations in the future.

The second issue that needs to be addressed is how an existing sensor can generate a new set of authenticatable keys to broadcast in subsequent deployments. If a sensor continues using the same α keys for every initialization phase, this gives the attacker the chance to learn more of the sensor's preloaded keys. If the attacker records previously heard $LREQ/LREP$ handshakes, then it may be able to break existing link keys as more of the preloaded keys are learned. Also, we seek to avoid using $\alpha \times I$ extra storage per sensor. Thus, we present a scheme which only require $\alpha + 2I - 1$ extra storage per sensor. To do this, the Merkle tree generated in Section IV-B is modified to be a two-level Merkle tree as follows.

Each sensor is again loaded with α unique, secret values. However, rather than use these values directly as keys to broadcast, they are used to create a hash chain to generate key values. Specifically, each sensor is loaded with the secret values $sv_1^1, sv_2^1, \ldots, sv_\alpha^1$. The first set of keys a sensor broadcasts is generated by applying a one-way hash function, H_{key}, to each of these α secret values. Thus, the first key broadcast when the sensor is initially deployed is $H_{key}(sv_1^1)$ and the last key broadcast during the sensor's first initialization procedure is $H_{key}(sv_\alpha^1)$. When a new batch of sensors is deployed, the existing sensor must generate keys to use for its second initialization procedure using a new set of secret values, $sv_1^2, sv_2^2, \ldots, sv_\alpha^2$. This is done by applying a hash function (not necessarily one-way), H_{sv}, to each of its secret values to generate a new set of secret values. That is, $sv_j^2 = H_{sv}(sv_j^1)$ for $j = 1, \ldots, \alpha$. The keys for the second initialization phase are then created by applying the one-way hash function H_{key} to each of the sv_j^2 values. So, the first key broadcast for the second initialization procedure is $H_{key}(sv_1^2)$ and the last key is $H_{key}(sv_\alpha^2)$. Figure 10 illustrates this process. Note that in between initialization procedures, a sensor needs only store α secret values rather than $\alpha \times I$ keys.

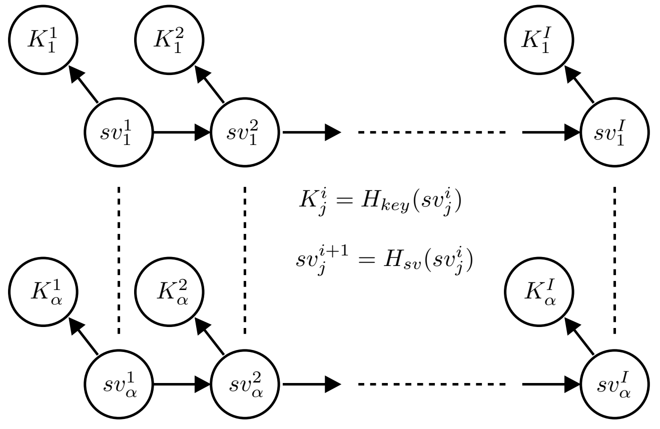

Fig. 10. Key generation from secret values for incremental deployment. K_j^i denotes the j-th key broadcast by a sensor in the i-th initialization phase. H_{key} is a one-way hash function and H_{sv} is a different hash function that is not necessarily one-way.

Now that we have specified a way for a trusted source to generate each of a sensor's I set of keys in advance while only using α storage on the sensor, we must create the Merkle tree used to authenticate the Bloom filter for each set of keys. To do this, we extend the Merkle tree discussed in Section IV-B by making each leaf node the root of another Merkle tree.[10] Thus, each sensor in a given deployment generation has its own unique second Merkle tree. This second Merkle tree has I leaf nodes, one for each of the I authenticated Bloom filters corresponding to its key sets. Each sensor is then loaded with the $2I-1$ nodes from its second Merkle tree to authenticate its Bloom filters along with the $\lg N$ nodes from the primary Merkle tree to authenticate the root of its second Merkle tree.

An example of a two-level Merkle tree is shown in Figure 11. In this example, $N = 4$ and $I = 4$. Without loss of generality, consider the second sensor in the deployment. It is loaded with the four Bloom filters necessary to authenticate its key sets, BF_1^2, BF_2^2, BF_3^2, and BF_4^2. The filters are then hashed to form the leaves of its local Merkle tree. This local Merkle tree is constructed as described in Section III-D to generate root R_2. This process is repeated for the three other sensors as well. Using these four roots as leaves, the primary Merkle tree is constructed with root R_0. The second sensor is then loaded with all the nodes from the subtree rooted at R_2 as well as the $\lg N$ nodes from the primary tree necessary to authenticate root R_0. The sensor must be loaded with all of the nodes rooted at R_2 in order to ensure each of its I Bloom filters can be authenticated.

VIII. Discussion

We now discuss our protocol in relation to the key predistribution method (e.g., [3]–[6]) as well as the approach of Anderson et al. [7]. In some scenarios, these methods may be

[10]Though this structure is actually just one larger Merkle tree, we refer to it as a two-level tree for ease of explanation.

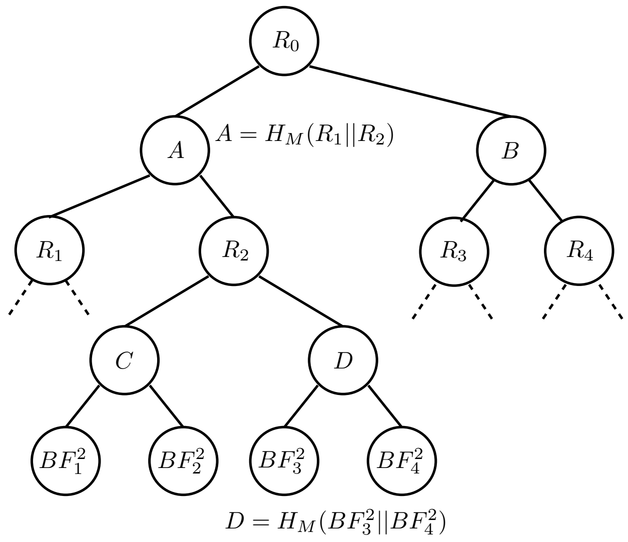

Fig. 11. Two-level Merkle tree for incremental deployment. In this example, there are four sensors and four Bloom filters per sensor for the sets of keys on that device. BF_j^i refers to the Bloom filter for the keys of the i-th sensor for the j-th initialization procedure.

preferable to our protocol. However, we believe properties of our protocol make desirable it in many environments.

A. Comparison with Predistribution Schemes

We begin with some comparative advantages of our scheme:

- *Network Connectivity:* As shown in Section VI, our protocol is able to achieve close to 100% network connectivity in many settings *without using multipath reinforcement*, which requires a sensor pair to rely on other sensors to establish their link key. Allowing a sensor to communicate with all of its neighbors is desirable from a performance perspective since it gives more options for forwarding a packet over a high quality link [24].
- *Localizing Damage from Sensor Compromise:* In the other predistribution protocols, every time a sensor is captured, the entire network becomes slightly less secure since the attacker learns more about the network's global key pool. By contrast, our protocol localizes the damage caused by compromised devices. If an attacker captures many sensors in one region of the network, it does not learn anything about the link keys being used in another region of the network. This is because our protocol forms link keys based on the key sets of nearby sensors rather than from a network-wide key set.

Some comparative disadvantages of our scheme include:

- *Multihop Key Sharing:* In some applications (e.g., [25], [26]), it may be desirable for key establishment to result in shared keys between sensors that are multiple hops away from each other. Predistribution schemes provide this property since a sensor is as likely to share keys with one-hop neighbors as it is to share keys with sensors in other regions of the network. Our protocol is localized and, therefore, only establishes keys between neighboring sensors. As mentioned in Section IX, adapting our protocol to establish keys with sensors multiple hops away is an area we will pursue for future work.
- *Key Set Authentication Overhead and Vulnerability:* While key predistribution schemes do have significant overhead to advertise their key sets, our protocol has the added overhead of sending Merkle nodes to authenticate the Bloom filter of the key set that will be broadcast by a sensor. However, we note that other sensor protocols have been proposed with require of $O(\lg N)$ overhead associated with Merkle trees (e.g., [21]) and suggested methods to improve this overhead if location information is available. Additionally, our protocol introduces a vulnerability that is not present in key predistribution schemes whereby an attacker could generate arbitrary keys that will be accepted as legitimate when broadcast. However, we have discussed methods to address this problem, such as increasing the Bloom filter size or increasing the Merkle tree size, in Section IV-B.

B. Comparison with Anderson et al. [7]

Compared to Anderson's protocol, our protocol is more complex and has more overhead. Additionally, our protocol requires the availability of multiple channels, which we do not view as a disadvantage since current sensors [2] already have this capability. However, by in large, we feel that our protocol offers significant comparative advantages:

- *Greatly Increased Security:* Any adversary that compromises our protocol would also be able to compromise Anderson's protocol [7]. However, there are many cases where Anderson's protocol is compromised but our protocol is not. Anderson's protocol can provide security no greater than the $c = 1$ case that is simulated in Section VI. In the same section, we show that $c > 1$ significantly improves resilience to colluding malicious sensors.
- *Increased Link Authentication:* From the description in Section II, it is easy to see that Anderson's scheme is vulnerable to identity theft whereby a malicious device claims a legitimate sensor's ID and creates link keys with neighbors using this ID. In our protocol, we preload the sensors with data necessary to authenticate their ID and key set by a trusted source. We note that the authentication mechanisms that we use could be adapted for use in Anderson's protocol.

IX. Future Work

The first area of future work we plan to pursue is implementing the protocol on sensor hardware to determine its viability in a realistic setting, as indicated in Section III-A. Next, we plan to augment the protocol to allow key establishment among sensors multiple hops away from each other, as discussed in Section VIII-A. Another area, as mentioned in Section III-B, is to explore stronger threat models where an attacker coordinates its channel switching protocol among multiple radios. Finally, we will more thoroughly investigate the schemes an attacker can use do compromise our authentication scheme described in Section IV-B. In particular, it is interesting to consider the optimal strategy an attacker can use to rebroadcast a legitimate sensor's keys and authentication data without being detected.

X. Conclusion

In this work, we have proposed a novel method of symmetric key establishment for a sensor network that uses channel diversity, as well as spatial diversity, to create link keys for one-hop neighbors. Establishing such keys is important because public keys are too computationally intensive for many sensors. Sharing a symmetric key with neighbors allows for secure aggregation as well as transmitting data used to authenticate hash chains, for example.

Via analysis and simulation, we show that our protocol performs very well in terms of network connectivity and resilience to colluding malicious devices compared to previous work. One result is that using even *one* extra channel for broadcasting keys during the initialization phase significantly improves security. From a numerical perspective, our simulations demonstrate that our protocol can achieve over 90% connectivity among neighboring sensors with link keys that are uncompromised even when 80% of the devices in the network are malicious and collude.

References

[1] M. J. Miller and N. H. Vaidya, "Leveraging Channel Diversity for Key Establishment in Wireless Sensor Networks," in *IEEE Infocom 2006*, April 2006.

[2] Crossbow Technology Inc., `http://www.xbow.com`.

[3] H. Chan, A. Perrig, and D. Song, "Random Key Predistribution Schemes for Sensor Networks," in *IEEE Security and Privacy Symposium 2003*, May 2003.

[4] W. Du, J. Deng, Y. S. Han, and P. K. Varshney, "A Pairwise Key Predistribution Scheme for Wireless Sensor Networks," in *ACM Computer and Communications Security (CCS) 2003*, October 2003.

[5] D. Liu and P. Ning, "Establishing Pairwise Keys in Distributed Sensor Networks," in *ACM Computer and Communications Security (CCS) 2003*, October 2003.

[6] L. Eschenauer and V. D. Gligor, "A Key-Management Scheme for Distributed Sensor Networks," in *ACM Computer and Communications Security (CCS) 2002*, November 2002.

[7] R. Anderson, H. Chan, and A. Perrig, "Key Infection: Smart Trust for Smart Dust," in *IEEE International Conference on Network Protocols (ICNP) 2004*, October 2004.

[8] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "Efficient and Secure Source Authentication for Multicast," in *ISOC NDSS 2001*, February 2001.

[9] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, "SPINS: Security Protocols for Sensor Networks," *Wireless Networks (WINET)*, vol. 8, no. 5, September 2002.

[10] S. Zhu, S. Setia, and S. Jajodia, "LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks," in *ACM Computer and Communications Security (CCS) 2003*, October 2003.

[11] S. Zhu, S. Xu, S. Setia, and S. Jajodia, "Establishing Pairwise Keys for Secure Communication in Ad Hoc Networks: A Probabilistic Approach," in *IEEE International Conference on Network Protocols (ICNP) 2003*, November 2003.

[12] J. Newsome, E. Shi, D. Song, and A. Perrig, "The Sybil Attack in Sensor Networks: Analysis & Defenses," in *ACM ISPN 2004*, April 2004.

[13] R. Blom, "Non-Public Key Distribution," in *Advances in Cryptology - CRYPTO 1982*, August 1982.

[14] C. Blundo, A. D. Santis, A. Herzberg, S. Kutten, U. Vaccaro, and M. Yung, "Perfectly-Secure Key Distribution for Dynamic Conferences," in *Advances in Cryptology - CRYPTO 1992*, August 1992.

[15] IEEE 802.11, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 1999.

[16] A. D. Wood and J. A. Stankovic, "Denial of Service in Sensor Networks," *IEEE Computer*, vol. 35, no. 10, October 2002.

[17] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, July 1970.

[18] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[19] G. Tsudik, "Message Authentication with One-Way Hash Functions," in *IEEE Infocom 1992*, May 1992.

[20] R. C. Merkle, "A Certified Digital Signature," in *Advances in Cryptology - CRYPTO 1989*, August 1989.

[21] W. Du, R. Wang, and P. Ning, "An Efficient Scheme for Authenticating Public Keys in Sensor Networks," in *ACM MobiHoc 2005*, May 2005.

[22] T. H. Cormen, C. E. Leiserson, , R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.

[23] `ns-2` – The Network Simulator, `http://www.isi.edu/nsnam/ns`.

[24] A. Cerpa, J. L. Wong, M. Potkonjak, and D. Estrin, "Temporal Properties of Low Power Wireless Links: Modeling and Implications on Multi-Hop Routing," in *ACM MobiHoc 2005*, May 2005.

[25] S. Zhu, S. Setia, S. Jajodia, and P. Ning, "An Interleaved Hop-by-Hop Authentication Scheme for Filtering Injected False Data in Sensor Networks," in *IEEE Security and Privacy Symposium 2004*, May 2004.

[26] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical En-route Filtering of Injected False Data in Sensor Networks," in *IEEE Infocom 2004*, March 2004.

Appendix I
Chernoff Bounds

This section provides a brief review of Chernoff bounds. More detailed information is available from many sources (e.g., [22]). As denoted in Table II, $LB(\mu)$ refers to the lower bound on the sum of the Bernoulli random variables, where μ is the mean of the sum.

Lower Chernoff bounds give the following inequality:

$$\Pr[X < (1-\delta)\mu] < \exp\left(-\frac{\mu\delta^2}{2}\right) \quad \text{when } 0 \leq \delta < 1 \tag{10}$$

where $LB(\mu) = (1-\delta)\mu$ and the right-hand side is the probability with which this bound is violated.

To demonstrate the application of this bound, we use $LB(\alpha p_h)$ from Section V (α and p_h are defined in Table I and Table II, respectively) as an example. We set δ from Equation 10 to be:

$$\delta = \sqrt{\frac{2\beta \ln \alpha}{\alpha p_h}} \tag{11}$$

where β controls the probability that the bound is violated (i.e., influences how "high" the high probability is), as discussed below. Thus, using the δ from Equation 11, we get:

$$LB(\alpha p_h) = (1-\delta)\alpha p_h = \left(1 - \sqrt{\frac{2\beta \ln \alpha}{\alpha p_h}}\right)\alpha p_h \tag{12}$$

and to avoid violating the condition that $0 \leq \delta < 1$ from Equation 10, we must enforce that:

$$p_h > \frac{2\beta \ln \alpha}{\alpha} \tag{13}$$

Using Equation 10 and δ from Equation 11, this gives us:

$$\Pr[X < LB(\alpha p_h)] < \frac{1}{\alpha^{\beta}} \tag{14}$$

which says that the bound is tighter for larger values of α. For our numerical results, we use $\beta = 1$.

The upper Chernoff bound gives the following inequalities:

$$\Pr[X > (1+\delta)\mu] < \begin{cases} \exp\left(-\frac{\mu\delta^2}{3}\right) & \text{when } \delta < 1 \\ \exp\left(-\frac{\mu\delta^2}{4}\right) & \text{when } \delta < 2e-1 \\ 2^{-(1+\delta)\mu} & \text{when } \delta > 2e-1 \end{cases} \tag{15}$$

Using the tightest bound from Equation 15 (i.e., $\delta < 1$), $UB(d_{uvw}\alpha p_h)$ from Table II is obtained as:

$$\begin{aligned} UB(d_{uvw}\alpha p_h) &= (1+\delta) d_{uvw}\alpha p_h \\ &= \left(1 + \sqrt{\frac{3\beta \ln(d_{uvw}\alpha)}{d_{uvw}\alpha p_h}}\right) d_{uvw}\alpha p_h \end{aligned} \tag{16}$$

and to avoid violating the condition that $\delta < 1$ from Equation 15, we must enforce that:

$$p_h > \frac{3\beta \ln(d_c\alpha)}{d_c\alpha} \tag{17}$$

Using Equation 15 and the bound from Equation 16, we get:

$$\Pr[X > UB(d_{uvw}\alpha p_h)] < \frac{1}{(d_{uvw}\alpha)^\beta} \tag{18}$$

and, again, we use $\beta = 1$ in our numerical results.