Message-In-a-Bottle: User-Friendly and Secure Key Deployment for Sensor Nodes*

Cynthia Kuo
Department of Engineering and Public Policy
Carnegie Mellon University
cykuo@cmu.edu

Mark Luk
Department of Electrical and Computer Engineering
Carnegie Mellon University
mluk@cmu.edu

Rohit Negi
Department of Electrical and Computer Engineering
Carnegie Mellon University
negi@ece.cmu.edu

Adrian Perrig
Department of Electrical and Computer Engineering
Carnegie Mellon University
perrig@cmu.edu

Abstract

Existing protocols for secure key establishment all rely on an unspecified mechanism for initially deploying secrets to sensor nodes. However, no commercially viable *and* secure mechanism exists for initial setup. Without a guarantee of secure key deployment, the traffic over a sensor network cannot be presumed secure.

To address this problem, we present a user-friendly protocol for the secure deployment of cryptographic keys in sensor networks. We propose a collection of five techniques to prevent an attacker from eavesdropping on key deployment. To demonstrate feasibility for real-world use, we implement our protocol on Telos motes and conduct a user study.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: Security and Protection

General Terms

Security, Human Factors

Keywords

Key deployment, sensor network, Faraday cage, wireless communication, human error

* This research was supported in part by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389 and MURI W 911 NF 0710287 from the Army Research Office, and grants CNS-0347807 and CNS-0627357 from the National Science Foundation, and a grant from Bosch. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of ARO, Bosch, CMU, NSF, or the U.S. Government or any of its agencies.

1 Introduction

Deploying cryptographic keys in a secure manner to sensor nodes is a prerequisite for secure sensor network operation. If the cryptographic keys are compromised during key setup, attackers can access the data transmitted – even if secure data communication protocols are used.

Unfortunately, numerous protocols for secure key establishment all rely on a pre-existing shared secret. The secret is magically loaded by an unspecified mechanism. For example, the TinySec [14] authors state that key "distribution is relatively simple; nodes are loaded with the shared key before deployment." SPINS and the plethora of random key pre-distribution papers also rely on unspecified key distribution mechanisms [6, 10, 11, 18, 24]. Outside of the academic literature, the ZigBee [34] security specification provides two suggestions for loading keys onto sensor nodes. First, cryptographic keys could be sent in the clear, "resulting in a brief moment of vulnerability." Alternatively, factories could imprint keys on sensor nodes, but customers may not trust the keys.

This paper proposes a secure mechanism for initial key setup. Our protocol deploys cryptographic keys (either shared secret symmetric keys or authentic public asymmetric keys) to wireless sensor nodes. Without loss of generality, we consider the problem of setting up a unique shared secret key between the base station and each node. The unique keys can be used to bootstrap group keys or public/private key pairs.

Secure key deployment in sensor networks is uniquely challenging. We designed our protocol to be easy to use despite the demanding requirements of large-scale, secure network deployments:

- **No Physical Interfaces.** Commodity wireless sensors may not be equipped with physical interfaces, such as USB connectors, screens, or keypads. This serves two purposes. First, removing physical interfaces reduces per-node manufacturing costs. Second, it protects the nodes from environmental hazards. Sensor nodes are ideal for deployment in extreme environmental conditions, such as on major highways or bridges, on naval vessels, under water, and in fire-prone wilderness [12, 21, 28, 32]. Therefore, key setup must take place over the wireless communication interface.

- **Secure Key Deployment, Wirelessly.** Wireless communication is vulnerable to eavesdropping and the injection of malicious traffic. For example, suppose an organization uses Diffie-Hellman to secure key setup. The exchange may be compromised with an active man-in-the-middle attack. Thus, resistance to eavesdropping and injection attacks is critical.
- **Key Deployment by Non-experts.** At first glance, relying on factory-installed keys is an attractive option for key deployment. However, pre-installed keys cannot be trusted – unless the entire distribution chain is secured, from factory to customer. Since this is unlikely, keys will be deployed by each customer, and non-expert personnel will be managing the installation process. The process must be simple, secure, and tolerant of human error.
- **Batch Deployment for Multiple Nodes.** Proponents of sensor networks envision large-scale deployments for various monitoring applications. A viable setup mechanism must support key deployment for multiple nodes – and require minimal effort and time.

Our protocol, Message-In-a-Bottle (MIB), also supports networks with less stringent requirements. However, large-scale networks realize MIB's greatest benefits.

Contribution. We propose Message-In-a-Bottle (MIB), a user-friendly protocol for initial key deployment in sensor networks. MIB is the first secure key setup protocol designed for low-cost, commodity sensor nodes. It enables fast, secure wireless key deployment for multiple nodes.

Outline. In the next section, we define our problem space, assumptions, and attacker model. We then characterize MIB for single node deployment before describing the protocol for multiple node deployment. Section 3 introduces MIB for single node deployment. This is followed by a security analysis, an implementation, and a user study in Sections 4, 5, and 6, respectively. Section 7 subsequently extends MIB for key deployment onto multiple nodes. We compare our scheme to related work in Section 8.

2 Problem Definition, Assumptions and Attacker Model

2.1 Problem Definition

Consider the following problem: a customer receives a shipment of new sensor nodes. Using wireless communication, how can a shared secret be set up between a trusted base station and each new uninitialized node? A viable solution provides each of the following properties:

- **Key secrecy.** An attacker has (at best) a negligible probability of compromising the shared secret.
- **Key authenticity.** A new node receives the (unaltered) key that the base station intended for it to receive.
- **Forward secrecy.** Compromising the key on one node does not compromise the keys on previously deployed nodes.
- **Demonstrative identification.** Users physically manipulate devices in such a way that they are certain which devices are communicating.
- **Robust to user error.** A system should be intuitive for non-experts. It should also be difficult for the human installer to introduce a vulnerability into the system. A human error should result in failing to set up a key – not in a key compromise.
- **Cost effective.** Secure setup of wireless sensor nodes requires additional hardware on each node or specialized setup hardware. Selecting the most cost effective approach depends on a variety of factors, such as the cost of per-node hardware, the cost of specialized setup hardware, and the number of nodes that will be deployed. Generally, specialized hardware will become more cost effective as the number of deployed nodes increases.
- **No public key cryptography.** The implementation of public key cryptography adds large amounts of software code and may be prone to energy-draining Denial of Service attacks.

2.2 Assumptions

We first assume that installation personnel are trustworthy. These installers may not be security experts, but we do assume that they can accurately follow simple directions, such as "if the red light is lit, discard the sensor node."

Second, we assume the trusted computing base consists of the base station and the devices used to perform key deployment (i.e., the keying device and keying beacon introduced in Section 3).

Finally, we assume that nodes employ secure communication protocols after the initial key setup. For example, MiniSec [20] could be used to provide an authenticated and/or encrypted communication channel.

2.3 Attacker Model

The goal of the attacker is to compromise the keys shared by the base station and the sensor nodes.

We adopt the Dolev-Yao attacker model, where an attacker is present in the network and can overhear, intercept, and inject any messages into the radio communication channel. An attacker could take the form of a more powerful device, such as a laptop equipped with a directional antenna. The attacker is assumed to be present before, during, and after key deployment. Such an omnipotent attacker is in line with other research in this area [2, 22].

3 Protocol for Key Deployment

MIB enables a sensor network to establish a secret with a new node *in the absence of prior shared secrets between the two parties*. MIB relies solely on the wireless communication interface, ensuring that our approach is applicable to any commodity wireless sensor node. In the following section, we first present an overview of MIB. This is followed by a detailed description of our protocol.

3.1 MIB Participants

Five parties participate in MIB: a base station, a keying device, a keying beacon, an uninitialized node, and a user. The goal is to establish a shared secret key between the base station and the new uninitialized node while satisfying the properties in Section 2.1. Below, we introduce the participants in MIB and provide an intuition about how they interact with one another.

Base Station. The base station, a PC-class machine that controls the entire network, delegates key deployment to two devices: the keying device and the keying beacon. The base station is not directly involved with key deployment.

New Node. The new node can be in one of three states: uninitialized, initialized, or rejected. When node M is first powered on or reset, it has no prior keying information and is *uninitialized*. Once the node receives a key, it is *initialized*. The key is the shared secret between the base station and the new node. If an error occurred during key deployment, the node is *rejected*. Rejected nodes do not share a valid secret key with the base station.

Keying Device. During key deployment, the keying device and the uninitialized node are placed inside a Faraday cage. The keying device sends keying information to the node when the Faraday cage is closed. Later, the new node uses the keying information to derive the key.

Keying Beacon. During key deployment, the keying beacon remains outside of the Faraday cage. The keying beacon has three purposes: 1) detect when the Faraday cage is closed; 2) jam the communication channel to prevent an eavesdropper from overhearing keying information leaked from the Faraday cage; and 3) inform the user of the status and outcome of the deployment.

User. The user of MIB is the person (e.g., technician) who performs key deployment.

3.2 Overview of MIB Protocol

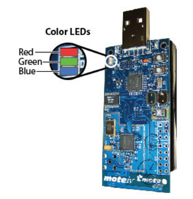
Initially, the keying device and keying beacon exchange authenticated heartbeat messages. The messages are used to determine whether the Faraday cage is open. In addition, the heartbeat messages are also used to maintain loose time synchronization.

To initiate deployment, the user places the keying device and the new node inside the Faraday cage. The keying beacon remains outside of the Faraday cage. Once the Faraday cage is closed, the keying device and the keying beacon can no longer exchange heartbeat messages.

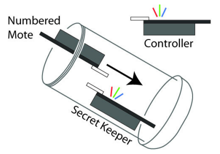
Inside the Faraday cage, the keying device begins key deployment. The keying device sends a key to the new node in several segments, although one segment is withheld. We refer to messages exchanged inside the Faraday cage as *shielded messages*. Meanwhile, the keying device monitors the background noise in the Faraday cage. If the background noise exceeds a certain threshold (which indicates that the protocol is under attack or that the Faraday cage is faulty), key deployment is aborted.

Outside of the Faraday cage, the keying beacon jams the wireless frequency of the shielded messages during key deployment; this overpowers any shielded messages that may leak. After a set time period, the keying beacon indicates to the user that key deployment has completed. The user opens the Faraday cage. If the protocol encountered an error, the keying device informs the keying beacon, who informs the user. If the protocol executed as expected, the keying device sends the following information to the keying beacon: the last key segment, the time periods in which shielded messages were exchanged, and a random validation string. The keying beacon uses the time periods to confirm that the jamming and the key exchange occurred simultaneously. The keying beacon then sends the last key segment to the new node. The new node reconstructs the key using all of the segments and computes the validation string. After the keying beacon verifies the new node's response, it informs the user that deployment succeeded.

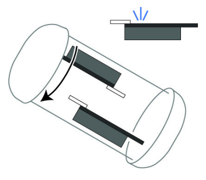
Each mote has three color LEDs: Red, Green, and Blue.
You will be working with three types of motes: Controller, Numbered Mote, Secret Keeper.



Step 1.
Verify that all three LEDs (**red**, **green**, and **blue**) on the **Controller** and the **Secret Keeper** are blinking simultaneously.
Gently place the **Numbered** mote in the container.



Step 2.
Screw the cap on the container snugly.
When no radio signals can escape the container, the **Controller's blue** LED will be lit.



Step 3.
Wait until the **Controller** begins to **blink** its **blue** LED. This should take about 5 seconds.

Step 4.
Open the container and remove the **Numbered** mote. Record the result on the Results page.

- If the **Controller's green** LED is lit, encryption keys **were** successfully exchanged.
- If the **Controller's red** LED is lit, the key exchange process was **not** successful.
- It does not matter what color (if any) the **Numbered** mote is lit.

Figure 1: Instructions for Key Deployment
These are the instructions that we provided to end users. Note that we modified the terminology for end users: the keying device is called the Secret Keeper; the keying beacon is called the Controller; and the new node is called the Numbered mote.

From a user's perspective, the key deployment process is simple. The directions from our user study are shown in Figure 1. Non-expert users can use MIB with little to no training. Moreover, MIB is intuitive. Placing the keying device and the new node in the Faraday cage provides demonstrative identification. The user knows precisely which devices are engaged in key deployment. In addition, the Faraday cage provides assurances that the new node receives the key that the keying device intended for it to receive.

3.2.1 *Why Is a Faraday Cage Insufficient?*

A reader may object that a Faraday cage is sufficient for secure key deployment, as suggested by Castelluccia and Mutaf [4]. Theoretically, this may be true. However, real-world factors introduce subtleties that must be addressed for security purposes. In practice, a Faraday cage is imperfect; it cannot block radio signals completely. It only attenuates signals. There are also usability issues: end users must seal the Faraday cage completely; they must not open the Faraday cage during key deployment; and they must reliably judge whether key deployment succeeded. MIB addresses these factors so that key deployment is both secure and practical.

3.2.2 *Protecting Shielded Messages*

Because a Faraday cage cannot block radio signals completely, MIB leverages five techniques to protect shielded messages.

1. The Faraday cage greatly attenuates the shielded message transmissions.
2. All shielded messages are transmitted at minimum power.
3. The keying beacon jams the wireless frequency of the shielded messages – at full power – outside of the Faraday cage. If any wireless signals leak from the Faraday cage, the keying beacon's stronger signal will interfere with and overpower the shielded messages.
4. The radios of many sensor nodes transmit in spread spectrum to achieve a high signal processing gain. Shielded messages do not use spread spectrum.
5. The deployed secret key is a function of all the shielded messages. In order to obtain the key, attackers need to overhear *all* of the shielded messages.

3.3 Detailed Description

We use the notation in Figure 2 to describe our protocol and cryptographic operations. Note that the ID of a node can be used to refer to the node itself (e.g., A represents both the node A and the ID of node A). We now describe the details of our key deployment protocol. Figure 5 shows a timeline of the protocol.

$A \rightarrow B: \langle M \rangle$	A sends message M to B.
S	Base station.
D	Keying Device.
B	Keying Beacon.
M	New sensor node to receive key.
$\mathscr{X}_{MS}$	Master Secret known only to S.
c	Counter used by the keying device to disambiguate between new nodes.
K_D	Deployment key used by keying device D to bootstrap new keys.
K_{DB}	Encryption key between D and B
K'_{DB}	MAC key between D and B.
K_M	Key to deploy onto node M.
N_A	A nonce generated by device A.
$PRF_K(X)$	A pseudo-random function computed over ID of node X, keyed by K.
$H(X)$	A cryptographic hash function computed over X.
$E_K(M)$	Encryption of message M with key K.
$MAC_K(M)$	Message authentication code of message M with key K.

Figure 2: Notation

Setup. Prior to key deployment, the base station assigns keys to the keying device D and the keying beacon B. This step represents the delegation of key deployment from the base station to these two devices.

Since the keying device and the keying beacon do not share any prior secrets with the base station, deploying keys onto these nodes may be difficult. We suggest using specialized keying devices that possess physical interfaces, such as USB connectors, for securely transmitting keys.

$S:$	$K_D = PRF_{\mathscr{X}_{MS}}(D)$
	$K_{DB} = PRF_{\mathscr{X}_{MS}}(D\|\|B\|\|0)$
	$K'_{DB} = PRF_{\mathscr{X}_{MS}}(D\|\|B\|\|1)$
$BS \rightarrow D:$	$\langle K_D, K_{DB}, K'_{DB}, initial\_timestamp \rangle$
$BS \rightarrow B:$	$\langle K_{DB}, K'_{DB}, initial\_timestamp \rangle$
$D:$	$c = 0$
User :	User places D inside of the Faraday cage and B outside of the Faraday cage

Figure 3: Base Station Sets Up Keying Device and Keying Beacon

As shown in Figure 3, the base station S derives a set of keys for these devices using a pseudo-random function PRF, keyed with a master secret $\mathscr{X}_{MS}$ known only to the base station. First, the base station derives an encryption key K_{DB} and MAC key K'_{DB} for communication between D and B. Next, S derives *deployment key* K_D for the keying device, which is used by D to derive secret keys for the new nodes. After each key deployment, D replaces the deployment key with the hash of the key $K_D = H(K_D)$. This creates a hash chain of deployment keys, ensuring that a new deployment key is used for each node. Using a hash chain provides forward secrecy, since an attacker cannot recreate previously deployed keys. Finally, D uses a counter c to track how many times K_D has been updated.

After completing the initial setup, the user places the keying device inside the Faraday cage. The keying beacon remains outside.

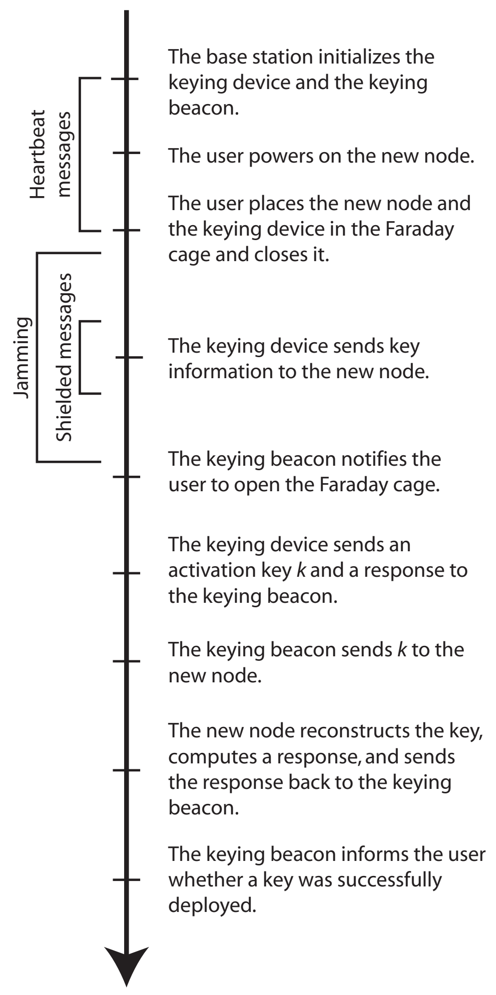
Step 1 of Key Deployment: Exchange Heartbeat Messages.

$B \rightarrow D:$	$\langle timestamp, MAC_{K'_{DB}}(B\|\|timestamp) \rangle$
$D \rightarrow B:$	$\langle MAC_{K'_{DB}}(D\|\|timestamp) \rangle$

Figure 4: Heartbeat Messages

The keying device and keying beacon exchange the authenticated heartbeat messages shown in Figure 4 to determine whether the Faraday cage is closed. Heartbeat messages require strong authentication; without authentication, an attacker can masquerade as the keying device to the keying beacon.

During the setup phase, the base station sent an initial timestamp to both devices. This established loose time synchronization. The heartbeat messages in this phase maintain time synchronization. The keying beacon sends its current timestamp, which is a monotonically increasing counter. The keying device adjusts its internal timer and replies immediately. Note that three rounds of communication are required to ensure mutual authentication and freshness.



The base station initializes the keying device and the keying beacon.
The user powers on the new node.
The user places the new node and the keying device in the Faraday cage and closes it.
The keying device sends key information to the new node.
The keying beacon notifies the user to open the Faraday cage.
The keying device sends an activation key k and a response to the keying beacon.
The keying beacon sends k to the new node.
The new node reconstructs the key, computes a response, and sends the response back to the keying beacon.
The keying beacon informs the user whether a key was successfully deployed.
Heartbeat messages
Jamming
Shielded messages

Figure 5: MIB Protocol Timeline

Step 2: Place New Node *M* inside Faraday Cage. The user initiates the key deployment protocol by turning on new node *M* and placing it inside the Faraday cage. Once the user closes the Faraday cage, the keying device and keying beacon can no longer exchange authenticated heartbeats. Both devices move to the next step in the protocol.

Step 3a: Deploy Cryptographic Keys.

After the Faraday cage is closed, the keying device D performs three tasks: (1) generates key K_M, which will be deployed onto the uninitialized node M; (2) transmits keying information to M, and 3) monitors background noise.

All messages exchanged in this step are shielded messages. MIB protects the shielded messages in several ways. First, the Faraday cage greatly attenuates the signal strength of the messages. Also, shielded messages are transmitted at minimum power and without spread spectrum encoding.

The shielded messages are shown in Figure 6. The keying device D generates key K_M by computing the pseudo-random function PRF over M, keyed with the current value of deployment key K_D. K_D is then replaced by the hash of K_D. In addition, the keying device transmits the hash of K_M,

$$\begin{array}{ll}
D \rightarrow * : & \langle Key\_Deployment\_Hello || D || c \rangle \\
M \rightarrow D : & \langle Key\_Deployment\_Hello\_Ack || D || c || M \rangle \\
D : & K_M = PRF_{K_D}(M) \\
 & r_i \overset{R}{\leftarrow} \{0,1\}^{80} \quad \forall i \in [1, s] \\
 & k = K_M \oplus r_1 \oplus r_2 \oplus \ldots \oplus r_s \\
 & h = H(K_M) \\
D \rightarrow M : & \langle h \rangle \\
D : & K_D = H(K_D) \\
 & c{+}{+} \\
for\ i = 1 \ldots s & \\
\quad D \rightarrow M : & \langle r_i \rangle \\
\quad D : & t_i = current\ time \\
\quad M \rightarrow D : & \langle H(r_i) \rangle
\end{array}$$

Figure 6: Shielded Messages

or h, to the new node M as a commitment; M can later verify the correctness of K_M against h.

Next, the keying device generates s segments of keying information and transmits them to the new node as follows. First, it generates s random nonces $r_1, r_2, \ldots, r_s$. It then repeatedly XORs key K_M with each value of r_i to generate an activation key k. Finally, the random nonces $r_1, r_2, \ldots, r_s$ are sent to M over s rounds of communication, along with counter c. M uses these messages to reconstruct K_M, as $K_M = k \oplus r_1 \oplus r_2 \ldots \oplus r_s$. Thus, an attacker must overhear all s messages to compromise the key. The time periods of these message exchanges are also recorded as $(t_1, t_2, \ldots, t_s)$.

The keying device D's third task is to monitor the background noise to ensure that the Faraday cage is indeed closed. If the Faraday cage is not attenuating the signals as required in the protocol (e.g., the Faraday cage is not closed properly), the keying device will sense the presence of the keying beacon and abort key deployment.

D considers the key deployment to be a success as long as M participates in the protocol as expected.

Step 3b: Outside the Faraday Cage. Meanwhile, the keying beacon remains outside of the Faraday cage. The keying beacon times how long the Faraday cage is closed and jams at full power. The jamming has two purposes. First, it prevents an eavesdropping attack, since the white noise overpowers any leaked shielded messages. It also enables the keying device to determine when the Faraday cage is opened prematurely. The jamming begins at time t_s and lasts for several seconds, which is more than enough time for the keying device to complete Step 3a. After several seconds, B notifies the user to open the Faraday cage.

Step 4: Key Activation and Verification. After the user opens the Faraday cage, M's key must be activated and verified. If the keying device and keying beacon both agree that no errors occurred in the previous steps, they activate key K_M. Finally, the keying beacon verifies that M possesses the correct key when M responds with the correct validation string. After verification, M considers itself as initialized. The messages exchanged in this step are shown in Figure 7.

Key activation reveals the last segment of keying information to the new node. First, if D considers the deployment to be successful, it encrypts activation key k and sends it to

```
    $D:$          $rp = MAC_{K_M}(k)$
                  $m = \{k, rp, t_1, t_s\}$
    $D \rightarrow B:$    $\langle E_{K_{DB}}(m),\ MAC_{K'_{DB}}(E_{K_{DB}}(m)) \rangle$
// If $B$ agrees there are no errors, and it was jamming during $t_1$ to $t_s$
    $B \rightarrow M:$    $\langle k \rangle$
    $M:$          $K_M = (r_1 \oplus r_2 \oplus ... \oplus r_m) \oplus k$
// $M$ verifies if $H(K_M) == h$
    $M \rightarrow B:$    $\langle MAC_{K_M}(k) \rangle$
// $B$ verifies response. If $MAC_{K_M}(k) == rp$
    $B:$           signals success
```

Figure 7: Key Activation and Verification Messages

the keying beacon B, along with the time periods in which D was sending protocol messages. B considers the deployment to be successful if it was jamming during the same time periods. This comparison requires loose time synchronization, which is established in the heartbeat messages. If B was not jamming during the appropriate time periods, it signals an error and aborts key deployment. Otherwise, B decrypts activation key k and forwards k to M. Upon receiving activation key k, new node M has all the information needed to generate K_M. To verify its correctness, node M hashes K_M and compares it against the commitment h received in Step 3.

Next, B verifies that M has the correct key. If M responds correctly with $rp = MAC_{K_M}(k)$, B flashes a success sequence on its LEDs, indicating to the user that key deployment succeeded.

If the keying device or the keying beacon recognized any errors during the key deployment process, B flashes a failure sequence on its LEDs.

After Key Deployment. When an initialized node M needs to communicate securely to base station S using key K_M, it identifies itself with its ID M, counter c, and the keying device's ID D. Since the base station possesses the master secret $\mathscr{X}_{MS}$, it can regenerate K_M as follows: $K_D = H^c(PRF_{\mathscr{X}_{MS}}(D))$, and $K_M = PRF_{K_D}(M)$.

Subsequently, a base station may wish to rekey the device. To ensure that new key K'_M originate from the base station, rekey messages must be authenticated and encrypted under the current key. This prevents attackers from masquerading as the keying device and performing malicious rekeying. If the current key is compromised, a manual reset of node is necessary.

4 Security Analysis

In this section, we discuss potential security issues and how MIB handles them.

Obtain key K_M through eavesdropping. To obtain key K_M, an eavesdropping attacker must overhear all of the shielded messages and the activation key k. However, the five techniques proposed in Section 3.2.2 guarantee that any device outside the Faraday cage cannot eavesdrop on all shielded messages. A rigorous analysis of this claim, based on radio transmission power and the attenuation factor of our Faraday cage, is presented in Section 5.2. Because an attacker cannot eavesdrop on all of the shielded messages, we ensure that K_M remains secret.

Inject a malicious key onto the new node. An attacker may attempt to deploy his own key onto new nodes by injecting malicious traffic. As a result, the attacker can accomplish two objectives. First, the new node would be unable to communicate securely with the base station, resulting in a denial of service attack. Second, the new node may mistake the attacker for the base station, allowing the attacker to control the new node.

An attacker can launch an attack before or after the new node is placed inside the Faraday cage. When an attack occurs beforehand, the attacker masquerades as the keying device and deploys his own key onto the new node. The attacker also poses as the keying beacon to send a bogus activation key to the new node. The new node then changes from an uninitialized state to an initialized state. Once this happens, it refuses to accept a key from the legitimate keying device inside the Faraday cage. The keying device detects the anomaly and notifies the user.

Alternatively, an attacker may interfere with legitimate key deployment by sending a bogus activation key k to the new node after the Faraday cage is opened. This simply causes the node to compute an incorrect key. The new node detects this attack when it compares the hash of the incorrect key to the commitment it received in the Faraday cage. If the hash does not match, it discards the bogus activation key k and waits for the correct one.

Compromise the network by compromising the keying device. If the keying device is compromised, an attacker cannot obtain previously deployed keys. Recall that the keying device D uses deployment key K_D to generate a unique key K_M for new node M. Once M receives K_M, the keying device replaces K_D with the cryptographic hash of K_D. Since it is computationally infeasible to invert a hash function, an attacker who obtains K_D cannot regenerate previous values of K_D. Consequently, compromising the keying device only yields the current value of K_D; all previously generated keys remain safe.

Initiate key deployment before the Faraday cage is closed. If an attacker leads the keying device to believe that the Faraday cage is closed when it is not, he will be able to eavesdrop on the (un)shielded messages. There are two ways to achieve this attack: 1) jamming the keying device and keying beacon so that all heartbeat messages are dropped, or 2) allowing the keying device to send its heartbeats but induce collisions on the return heartbeat from the keying beacon.

In the first attack, the keying beacon jams with full power and the keying device monitors the background noise whenever the Faraday cage is supposedly closed. The keying device aborts key deployment if the noise level rises above a predetermined threshold.

In the second attack, the attacker induces collisions only on the return heartbeat. This causes the keying device to send shielded messages before the keying beacon begins jamming. The keying beacon would detect this inconsistency when it compares the time periods it was jamming to the time periods in which shielded messages were exchanged $(t_1, t_2, ..., t_s)$.

Prevent the keying beacon from jamming during key deployment. The keying beacon begins jamming when it no longer hears the keying device's heartbeat messages. Successfully posing as the keying device allows an attacker to listen for leaked shielded messages. This attack is prevented by the challenge-response heartbeat protocol that provides for strong one-way authentication of the keying device to the keying beacon.

In addition, the time periods in which shielded messages are exchanged should fall within the time periods in which the keying beacon is jamming. Even if an attacker finds a way to masquerade as the keying device, the time periods will not match.

Wait for the user to make an error. A patient attacker could wait for the user to make a mistake. However, MIB was designed to detect user errors and fail safely.

For example, suppose the user closes the Faraday cage improperly or opens the cage prematurely. An attacker would like to exploit the error and eavesdrop on shielded messages that leak from the Faraday cage. However, the keying beacon jams during the transmission of shielded messages, and the keying device is monitors the background noise. If the cage is not closed properly or opened prematurely, the keying device will observe that the background noise exceeds a predetermined threshold. Key deployment will be aborted.

Alternatively, the user may confuse the keying beacon and keying device, placing the keying beacon inside the Faraday cage instead of the keying device. We address this issue in Section 7.

5 Implementation

This section describes our implementation of MIB on Moteiv's Telos motes. Telos motes feature TI MSP430 micro-controllers and ChipCon CC2420 radios. We first explain the software design and implementation, which was written in nesC and executed on TinyOS. Next, we measured the radio signal strength of our implementation. Using these measurements, we present a mathematical analysis on the likelihood of an eavesdropper obtaining the key.

5.1 Protocol Implementation

Primitives. Because of stringent energy and computation constraints on sensor nodes, we select the following cryptographic primitives. Skipjack was chosen as the blockcipher because of efficient computation and low memory footprint [15]. To make encryption as flexible as possible, we set Skipjack's block size to 64 bits. In addition, because Skipjack makes a fine pseudo-random permutation (PRP), we can also use Skipjack as a PRF. We use 80-bit symmetric keys for Skipjack; Lenstra and Verheul recommended that such keys are considered to be secure until 2012, even against resourceful adversaries [16].

To meet the stringent memory constraint of sensor nodes, we construct the encryption, MAC, and hash function with the same block cipher. We use cipher block chaining (CBC) to construct the encryption and MAC functions, and Matyas-Meyer-Oseas mode to construct the hash function.

Keying beacon. The keying beacon has three purposes: 1) detect when the Faraday cage is closed; 2) jam the communication channel to prevent an eavesdropper from overhearing keying information leaked from the Faraday cage; and 3) inform the user of the status of the deployment. The keying beacon's state diagram is shown in Figure 8.

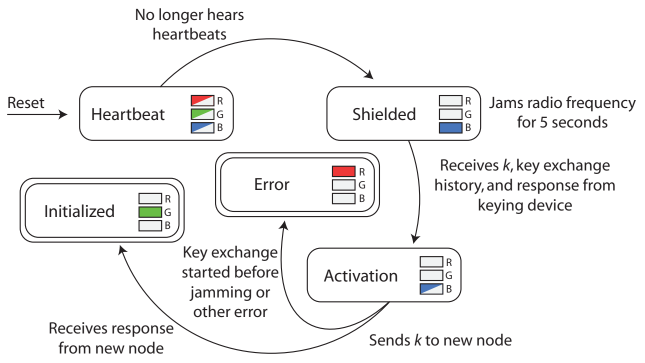

No longer hears heartbeats
Reset
Heartbeat
R
G
B
Shielded
Jams radio frequency for 5 seconds
Error
Initialized
Receives k, key exchange history, and response from keying device
Key exchange started before jamming or other error
Activation
Receives response from new node
Sends k to new node

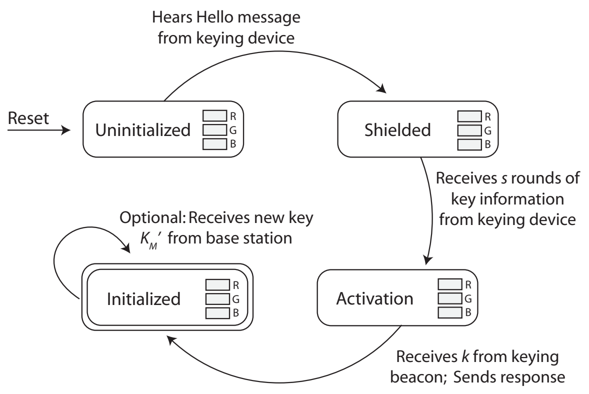
Figure 8: State Diagram for Keying Beacon
A fully colored box indicates that the LED is lit. A half colored box indicates the LED is blinking.

The keying beacon starts in the Heartbeat state. In this state, it blinks all three LEDs to indicate that it is exchanging heartbeat messages with the keying device. Once the Faraday cage is closed, the keying beacon misses several consecutive heartbeats from the keying device. This causes the keying beacon to transition into the Shielded state. In the Shielded state, the keying beacon illuminates its blue LED and jams at full power. After 5 seconds, the beacon transitions into the Activation state. It blinks its blue LED and waits for the user to open the Faraday cage. Once it receives the keying device's message, it forwards the activation key to the new node and waits for its response. A correct response causes the keying beacon to transition into the Initialized state. The successful deployment is indicated by illuminating the green LED. Any error causes the keying beacon to transition into the Error state, where the red LED is lit.

The protocol was designed with simplicity as one of the main goals. The user only needs to monitor the LEDs on the keying beacon.

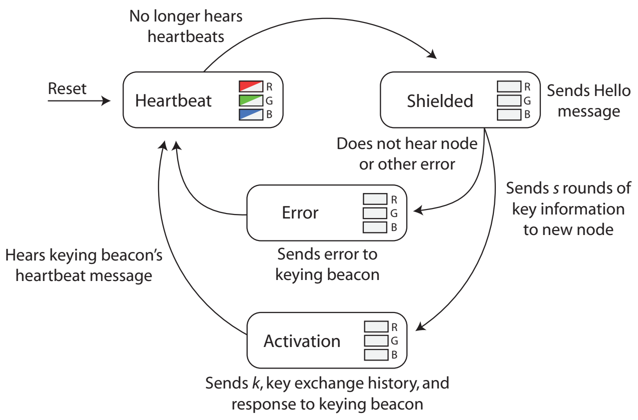
New node. As shown in Figure 9, an uninitialized node powers on and waits for keying information from the keying device. After it receives enough keying information to compute key K_M, it responds to the keying beacon with a validation string.

Keying device. The keying device computes and deploys key K_M onto the new node, as illustrated in Figure 10. Like the keying beacon, the keying device starts in the Heartbeat state. It blinks all three LEDs and exchanges heartbeats with the keying beacon. After the user closes the Faraday cage and the keying device no longer hears heartbeat messages, it enters the Shielded state. In this state, the keying device locates the new node and transmits multiple rounds of keying information to the new node. If this is successful, the key-



Hears Hello message from keying device
Reset
Uninitialized
R
G
B
Shielded
R
G
B
Receives s rounds of key information from keying device
Optional: Receives new key K_M' from base station
Initialized
R
G
B
Activation
R
G
B
Receives k from keying beacon; Sends response

Figure 9: State Diagram for New Node



No longer hears heartbeats
Reset
Heartbeat
R
G
B
Shielded
R
G
B
Sends Hello message
Does not hear node or other error
Error
R
G
B
Sends s rounds of key information to new node
Hears keying beacon's heartbeat message
Sends error to keying beacon
Activation
R
G
B
Sends k, key exchange history, and response to keying beacon

Figure 10: State Diagram for Keying Device
A fully colored box indicates that the LED is lit. A half colored box indicates the LED is blinking.

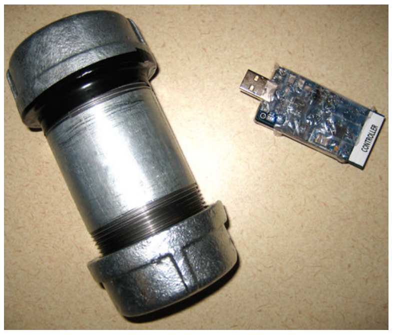


Figure 11: Our Implementation Using Galvanized Steel Pipe

ing device transitions into the Activation state when the user opens the Faraday cage. In the Activation state, the keying device sends an activation key and a validation string to the keying beacon. If an error occurred in the Shielded state, the keying device transitions into the Error state instead, and an error message is sent to the keying beacon. When the keying beacon is reset, the keying device returns to the Heartbeat state.

5.2 Radio Measurement

We measured the strength of the keying device's signal inside and outside of the Faraday cage. The data were used to confirm that (1) the uninitialized node inside the Faraday cage receives the keying device's transmissions without difficulty and (2) an attacker outside of the Faraday cage cannot eavesdrop on the shielded messages. We also compare the security implications when the keying beacon fails to jam while shielded messages are exchanged and when the keying beacon jams as specified.

Our assumptions follow. We assume that the keying device always broadcasts at the minimum transmit power of $P_t = -24$ dBm [7] of the CC2420 chip. We also assume that the keying beacon broadcasts Gaussian white noise in a 5 MHz bandwidth around the carrier frequency of the keying device at a power of 5 dBm, so as to be within the maximum power spectral density (PSD) limit specified for the 2.5 GHz ISM band. Our measurements of the galvanized steel pipe used as a Faraday cage show that the minimum attenuation through the pipe is $L_{cage} = 84$ dB. Further, since the keying device and the new node are placed next to each other, the attenuation between them is $L_{DM} = 3$ dB in our testing environment. We assume that the eavesdropper may have an antenna of gain 10 dB, which is typical of a cell-site antenna.

Key Reception at New Node. Thus, the received *Signal-to-Interference Ratio (SIR)* of the sensor node is -24 dBm -3 - (5 dBm-84) = 52 dB while the *Signal-to-Noise Ratio (SNR)* is -24 dBm-3-(-106 dBm) = 79 dB. Note that the noise level of -106 dBm is obtained from the thermal noise PSD of -113 dBm/MHz [26] and the link bandwidth of 5 MHz. Both are significantly more than the minimum SNR of 1 dB required [7] for a packet error rate of 1%, so that the key reception at the sensor is lossless.

Key Reception at Eavesdropper without Jamming. The minimum SNR required to achieve channel capacity for 250 kbps over a 5 MHz channel (based on the specifications of the CC2420 chip) is given by 250 kbps = 5 MHz $\times \log_2(1+SNR)$. Thus, the required SNR is -15 dB. Since channel capacity requires an ideal code, which is not present in the CC2420 chip, this value of SNR is the most optimistic value from the point of view of the eavesdropper.

We define RS_e to be the eavesdropper's required radio sensitivity in order to overhear shielded messages, measured in dBm. With thermal noise of -106 dBm, and assuming an eavesdropper antenna gain of 10 dB, this implies that RS_e must be at least -106 dBm -10 = -116 dBm.

Assuming that the Faraday cage works perfectly, we define d_{min} to be the minimal distance between the keying device and eavesdropper, beyond which eavesdropping will fail. RS_e can be calculated using d_{min} by the free space loss equation [26] as follows.

$$RS_e = P_t - 20\log_{10}\left(\frac{4\pi d_{min}}{\lambda}\right) - L_{cage}$$

Note that free-space propagation occurs over short distances, as in our scenario. Over short distances, there is no fading. Further, if there is shadowing, it will only serve to increase the attenuation. Since the carrier wavelength $\lambda = 12$ cm and we set RS_e to be -116 dBm, we calculate $d_{min} = 2.5$ cm. In other words, in order for the eavesdropper to overhear shielded messages, he must be within 2.5 cm of the keying device.

If we relax our assumption and allow for leakage from the Faraday cage, it is slightly easier for the attacker to eavesdrop, as d_{min} increases from 2.5 cm to 19 cm. We arrive at this conclusion based on the following calculation. First, according to the Telos data sheet, the keying beacon senses leakage from the Faraday cage if the signal level outside the cage is above 90 dBm [23]. In that case, the Faraday cage must have an attenuation of at least -24 dBm+90 dBm = 66 dB for the leakage to go undetected. Repeating the d_{min} calculation with the reduced $L_{cage} = 66$ dB and all other variables unmodified, we find that $d_{min} = 19$ cm.

In summary, eavesdropping is not possible as long as the Faraday cage functions properly and a 2.5 cm radius around the keying apparatus is physically secure. Even if the Faraday cage leaks, an eavesdropper would not succeed as long as a 19 cm radius around the Faraday cage is free of other devices. While 19 cm is a small distance for inspection, we should not reply on human operators to maintain constant vigilance. As we will describe in the following section, jamming precludes the exploitation of human error.

Key Reception at Eavesdropper with Jamming. With a jammer (i.e., keying beacon) of power 5 dBm, even if we assume the smaller attenuation of 66 dB of the cage, the SIR at any point outside the cage will be smaller than -24 dBm - 66 - 5 dBm = -95 dB. This is substantially smaller than the minimum SNR of -15 dB to achieve capacity. Thus, eavesdropping will not be possible. More importantly, this is true even if the eavesdropper uses a high-gain antenna, since both keying-signal as well as the jammer signal will be equally amplified.

6 User Study

We conducted a user study to evaluate whether users could successfully use our protocol for key deployment.

6.1 Methodology

Our protocol was designed to be simple and accessible to a wide range of individuals. For the user study, we recruited university students and staff who are not colorblind.[1]

We did not expect that users would have any prior knowledge or experience with sensor nodes. We first described the nodes' capabilities. We then introduced the concept of encryption and the need for secure key deployment.

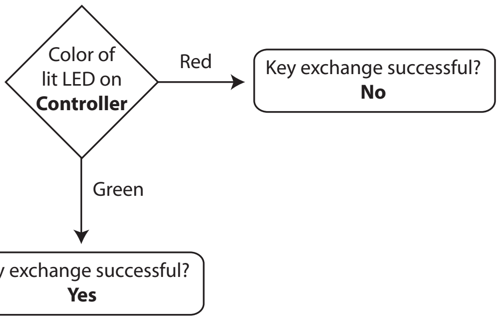


Figure 12: Flowchart Provided with Sheet for Recording Results

Each user was given a sheet with installation directions, a sheet on which to record the results, and four nodes for key deployment. The installation directions are shown in Figure 1. The sheet for recording results included the flowchart in Figure 12. Of the four nodes, two 'good' nodes were configured correctly for key deployment. The remaining two nodes simulated a variety of errors and attacks:

1. From the user's point of view, the keying beacon turns red, and the new node behaves exactly as expected. (None of the LEDs on the new node blinks). There are several possible causes for this situation: the new node is defective and failed to complete the key exchange process; the keying beacon failed to jam while the shielded messages were exchanged; shielded messages leaked from the Faraday cage (because the Faraday cage was improperly closed, the user opened the Faraday cage too early, or the Faraday cage itself is leaky); or the keying device heard jamming signals during key deployment.
2. From the user's point of view, the keying beacon turns red, but the new node's green LED is also lit. This situation occurs if the new node is defective or malicious. While unlikely, we included this scenario to confuse the user with a conflicting signal.

For each of the four nodes, participants followed the installation directions and determined whether a key was successfully deployed. The order of the nodes was randomly assigned.

6.2 Results

Because of the simplicity of MIB, participants were able to follow our directions with ease. Initially, many of the participants were apprehensive about working with the sensor nodes. Afterwards, several participants described the protocol as "simple," "easy," or even "self-evident."

We tested twenty individuals, including undergraduate students, graduate students, and staff members. Eight participants were female; twelve were male. Participants' ages ranged from 18 to over 50. Of the twenty participants, nineteen correctly categorized the four nodes. The remaining participant carelessly circled an incorrect option while rushing to finish the study. Our results are summarized in Table 1.

[1]Our implementation relies on users' ability to distinguish between different colored LEDs. Most users will use the LED colors for differentiation, but colorblind users could use the *locations* of lit LEDs. Our test nodes were sheathed in bubble wrap for protection, obscuring the LED locations.

Summary	
Total number of participants	20
Number of motes tested by each participant	4
Total number of errors	1
Motes tested by *each participant*	
Number of 'good' motes	2
Number of 'bad' motes	2
Errors	
Number of false negatives (identifying an unsuccessful key deployment as successful)	0
Number of false positives (identifying a successful key deployment as unsuccessful)	1

Table 1: Summary of User Study Results

7 MIB for Multiple Nodes

Our user study demonstrated that people are able to follow MIB's instructions quickly and accurately. However, we noticed that several participants fumbled with the ends of the steel pipe. In addition, a few participants commented that it would be tedious to configure many nodes in this manner.

As a next step, we extended MIB for key deployment on multiple nodes. We placed the following constraints on the extension:

1. From the user's perspective, no additional work should be required. Deployment for multiple nodes should be the same as deployment for one node.
2. MIB should be able to handle an arbitrary number of nodes.
3. It should be easy to deploy group keys or individual keys. For the remainder of this paper, we focus on deploying individual keys.
4. The Faraday cage should be easy to open and close.

To optimize the ergonomics of the Faraday cage, we developed the design shown in Figure 13, which was custom-manufactured on a high-precision CNC machine. Compared with the galvanized steel pipe, this container is much easier to open and close. To open the container, the lid is simply lifted off. To close the container, the lid is placed on the body of the container. The weight of the lid ensures that the container is sealed shut.

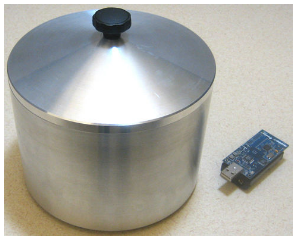


Figure 13: Bigger and Ergonomically-friendly Faraday Cage
This Faraday cage may resemble a storage container for baking goods, but it is heavier, air-tight, and attenuates radio signals.

Placing multiple new nodes in the Faraday cage introduces two significant challenges. First, the keying device and the keying beacon need some way to determine when the protocol has finished. In particular, the keying beacon must jam and notify the user to open the Faraday cage at the appropriate times. Times will vary, depending on the number of nodes in the Faraday cage. The keying device also needs to know when all the nodes have received their keys so that it can generate and send the activation keys and validation strings to the keying beacon. Thus, the keying device and keying beacon must know how many new nodes are placed in the Faraday cage. Second, the nodes must be counted without user intervention. Users may miscount the number of nodes – especially as the number of nodes increases.

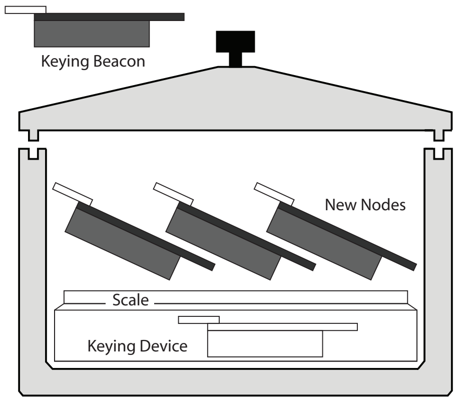


Figure 14: MIB Setup for Multiple Nodes
The Faraday Cage fits multiple nodes and a scale. The keying device is attached to the scale. Weight is used to calculate the number of new nodes.

MIB uses a scale to count the number of new nodes. The number of nodes is calculated using the weight of one node. (For example, each Telos mote weights approximately 60 grams.) A batch can only contain nodes of the same type. The keying device is then attached to the scale to obtain the reading. An illustration of the setup is shown in Figure 14. Note that attaching the keying device to the scale prevents one of the error scenarios we presented in Section 4, where the user accidentally swaps the keying device and the keying beacon. Since the keying device is physically attached to the scale, we design the physical casing such that the keying beacon cannot be connected to the scale.

On the protocol level, a few changes are required to support multiple nodes.

1. The keying device sends the most recent node count to the keying beacon in the heartbeat message.
2. The keying beacon estimates the time needed for shielded messages exchange and jams for the duration.
3. The keying device verifies that the number of responsive nodes in the Faraday cage matches node count from the scale. If these numbers do not match, the keying device aborts the protocol and returns an error.
4. The keying device assigns a MIB index to each node to facilitate key assignment. (We assume that each node receives a unique key. Of course, this could be changed to a group key.)

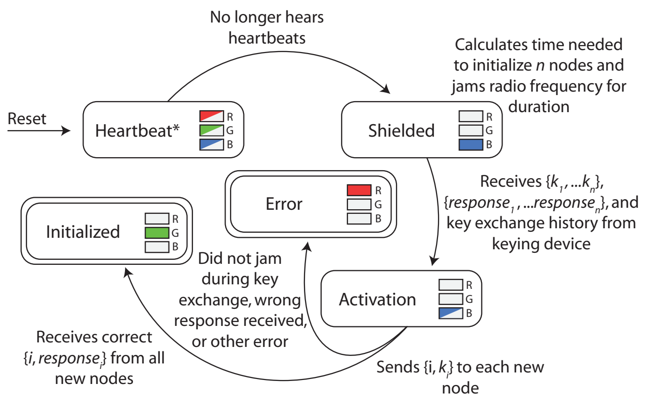

No longer hears heartbeats
Calculates time needed to initialize n nodes and jams radio frequency for duration
Reset
Heartbeat*
R G B
Shielded
Error
Initialized
Receives {k1,...kn}, {response1,...responsen}, and key exchange history from keying device
Did not jam during key exchange, wrong response received, or other error
Activation
Receives correct {i, responsei} from all new nodes
Sends {i, ki} to each new node

Figure 15: State Diagram for Keying Beacon: Multiple Nodes
* For multiple node setup, the heartbeat is extended to include the node count from the scale.

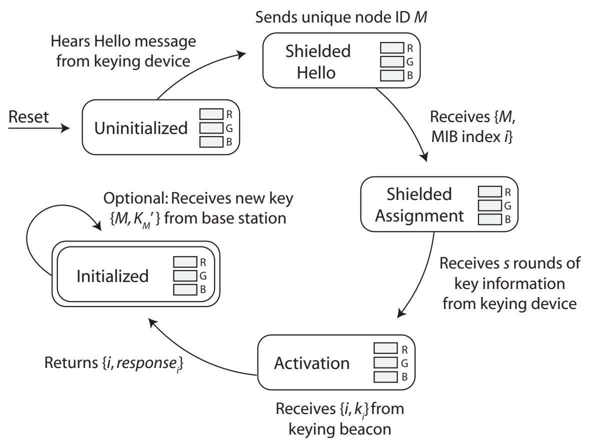

Sends unique node ID M
Hears Hello message from keying device
Shielded Hello
Reset
Uninitialized
Receives {M, MIB index i}
Optional: Receives new key {M, KM'} from base station
Shielded Assignment
Initialized
Receives s rounds of key information from keying device
Returns {i, response}
Activation
Receives {i, ki} from keying beacon

Figure 17: State Diagram for New Node: Multiple Nodes

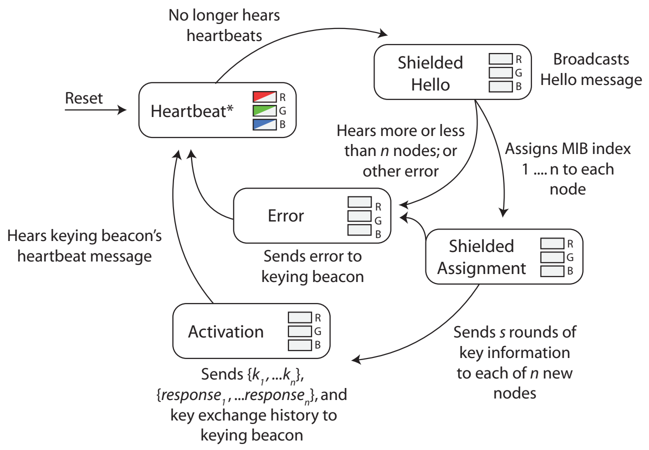

No longer hears heartbeats
Shielded Hello
Broadcasts Hello message
Reset
Heartbeat*
Hears more or less than n nodes; or other error
Assigns MIB index 1 .... n to each node
Error
Hears keying beacon's heartbeat message
Sends error to keying beacon
Shielded Assignment
Activation
Sends s rounds of key information to each of n new nodes
Sends {k1,...kn}, {response1,...responsen}, and key exchange history to keying beacon

Figure 16: State Diagram for Keying Device: Multiple Nodes
* For multiple node setup, the heartbeat is extended to include the node count from the scale.

These modifications are reflected in Figures 15, 16, and 17. Because LEDs have limited feedback capability, the entire batch of nodes must be discarded if an error occurs.

8 Discussion

8.1 Uses for MIB

Keys deployed in MIB can be used to bootstrap any other key establishment mechanism that requires pre-installed keys. For example, they can be used to securely download a pseudo-random key pool used in random key predistribution [6, 10, 11, 19, 25]. Alternatively, initial keys can be used to bootstrap keys for SPINS, TinySec, ZigBee, or MiniSec.

8.2 Variations on MIB

MIB implements key deployment over the wireless communication channel. However, the protocol could be adapted to use the sensors themselves. For example, a network comprised of light sensors could implement MIB using light as the communication channel. Then, any solid, opaque container takes the place of the Faraday cage; the function of the Faraday cage changes from attenuating radio signals to blocking the escape of light. The keying beacon could also radiate light instead of jamming radio signals. Similarly, MIB can be adapted for other modalities, such as acoustic or infrared.

8.3 Objections to MIB

Isn't the Faraday cage sufficient? Realistically, we cannot expect a Faraday cage to be perfect; even the equipment used in RF testing can only attenuate radio waves [8].

Thus, MIB relies on the Faraday cage to attenuate the radio waves and takes additional measures to safeguard the shielded messages.

Isn't it unrealistic to assume that there will be no USB (or other) hardware interface? Sensor networks will often be deployed under extreme conditions, for applications such as traffic monitoring [28], structural health monitoring [21], or water quality monitoring [32]. A physical interface becomes a vulnerable point in the body of a sensor. Already, some sensors are completely encased in hardened shells to withstand harsh conditions [28]. Physical interfaces also increase per-node manufacturing costs, particularly if they need to be protected. From a manufacturer's perspective, it is undesirable to create an additional hardware interface that is used only for key deployment.

What about factory-installed keys? There are three major arguments against factory-installed keys:

1. Customers would have to be confident that attackers could not access or tamper with the sensor nodes anywhere along the entire distribution chain.
2. Customers would have to trust the manufacturer to manage keys properly (e.g., the manufacturer does not keep a copy of the keys, or nodes initialized for one customer have not been accidentally delivered to another).
3. Manufacturers do not want to assume liability for key management.

What if an attacker slipped in a malicious node for "setup"? Earlier, we assumed that the new uninitialized node is trusted. We now briefly discuss possible attacks and countermeasures if we remove this assumption.

If a new uninitialized node is under control of the attacker, key secrecy is compromised. The secrecy provided by the shielded messages is eliminated. Furthermore, in multiple node deployment, the malicious "new uninitialized node" may eavesdrop on keys sent to other nodes.

To prevent such attacks, we use software attestation to verify code integrity on all new nodes. Software attestation techniques, such as SWATT [29], allow an external verifier to examine code integrity on an untrusted computing device without hardware extensions. SWATT employs a challenge-response based verification function that computes a checksum over the code memory of the untrusted device. The verification function is constructed in such a way that if an attacker modifies the expected code content, either the checksum response would be incorrect, or the execution time of the verification procedure takes longer than expected.

When using MIB to deploy keys for batches of single nodes, SWATT may be used as follows. The new uninitialized node is the untrusted device. The keying device acts as the verifier. Once the Faraday cage is closed and the keying device and the new node identify each other, the keying device initiates SWATT by sending a challenge. After receiving the checksum response from the new node, the keying device verifies the correctness of the checksum response and the duration of execution. Note that the keying device must know the specifications of the untrusted node's hardware for SWATT to be effective. If the new node passes verification, key deployment continues. Otherwise, key deployment is aborted, and an error is reported to the keying beacon.

Software attestation may also be used for batches of multiple nodes. Suppose one malicious node is hidden in a batch of new nodes. Once it has been placed in the Faraday cage, the malicious node has two options: remain silent or respond to the keying device. If the node remains silent, its presence will be detected. Using the scale, the keying device will know that there are n nodes in the Faraday cage and detect when fewer than n nodes respond. If a malicious node responds to keying device, the keying device will run SWATT on the node. SWATT will then detect the malicious code. In both cases, the keying device will report that key setup failed.

9 Comparison with Related Work

Researchers have proposed numerous sensor network key deployment schemes, such as ZigBee [34], SPINS [24], LEAP [33], Transitory Master Key [9], and random key predistribution [6, 10, 11, 19, 25]. Unfortunately, *all* of these approaches rely on an unspecified secure mechanism to set up the initial secret key in each sensor node.

Some exceptions are Shake Them Up [4], On-off Keying [2], and Key Infection [5]. Like MIB, these sensor network key establishment schemes do not rely on pre-shared secrets; hence we will discuss them in detail in this section. We will also compare MIB against out-of-band-based approaches proposed for key setup in ubiquitous computing settings: Resurrecting Duckling [31], Talking to Strangers [1], and Seeing-is-Believing [22].

	Message-In-a-Bottle	Resurrecting Duckling [31]	Talking to Strangers [1]	Seeing-is-Believing [22]	On-off Keying [2]	Key Infection [5]	Shake Them Up [4]
Security							
Key secrecy	Y	Y	-	-	-	N	N
Key authenticity	Y	Y	Y	Y	N	N	Y
Ease-of-use							
Demonstrative identification	Y	Y	Y	Y	N	N	Y
Robust to user error	Y	Y	Y	Y	Y	Y	N
Costs							
No per-node extra hardware	Y	N	N	Y	Y	Y	Y
No specialized setup hardware	N	Y	Y	N	Y	Y	Y
No public key cryptography	Y	Y	N	N	N	Y	N

Table 2: Comparison of Different Key Deployment Techniques
A '-' signifies that this property is not applicable.

We will discuss each key deployment scheme with respect to several relevant properties listed in Section 2.1: key secrecy, key authenticity, demonstrative identification, robustness to user error, cost effectiveness, and no public key cryptography. To compare cost effectiveness, we discuss two properties: no per-node specialized hardware, and no specialized setup hardware. Table 2 summarizes our comparison.

Resurrecting Duckling sets up a secure shared key through the out-of-band channel of physical contact [30, 31]. Because the side channel is assumed to be secure, the key exchanged over this medium is secret and authentic. Unfortunately, this scheme requires a specialized hardware interface for physical contact.

Talking to Strangers relies on a location-limited channel, such as audio or infrared, as an out-of-band channel to setup a public key [1]. Like Resurrecting Duckling, this scheme relies on specialized hardware on each device. In addition, Talking to Strangers requires public key cryptography, which is expensive for computationally constrained sensor nodes.

In Seeing-is-Believing, an installation device equipped with a camera or a bar code reader reads a public key on each device that is encoded as a 2D barcode [22]. Again, since the side channel is assumed to be secure, the key exchanged over this medium is authentic. Although Seeing-is-Believing does not require special hardware per node, a setup device with specialized hardware is needed. In addition, nodes perform expensive asymmetric cryptographic operations.

In On-off Keying, the presence of an RF signal represents a binary '1,' while its absence represents a binary '0' [2, 3].

Assuming that an attacker cannot cancel RF signals, the attacker can only modify authentic messages by changing 0's to 1's – but not the inverse. By carefully selecting the encoding scheme, On-off Keying ensures that the attacker is unable to modify a packet during transmission. On-off keying does not achieve key authenticity since it requires each user to know the signal strength threshold that differentiates between a '1' and a '0.' This value itself needs to be authenticated, but the authors did not specify such a method. In addition, since this scheme lacks feedback to the user, it also does not achieve demonstrative identification. Finally, On-off Keying requires public key cryptography.

Key Infection simply sends secret keys in the clear, assuming that an attacker arrives at a later point in time [5]. Designed for simplicity and cost effectiveness, this scheme cannot defend against a determined adversary. If the attacker is actually present during key deployment, she may eavesdrop on the deployed key, violating key secrecy. An attacker may also inject her own keys, violating key authenticity. The lack of user feedback means demonstrative identification is absent.

Shake Them Up [4] sets up shared keys between two nodes by requiring the user to hold one in each hand and shake them. These two nodes exchange identical packets, and rely on the fact that the adversary cannot distinguish between messages sent by either device. These two devices, however, could be distinguished using radio fingerprinting [27]. Thus, key secrecy may be violated. Shake Them Up is also not robust against user error. Tired after deploying several nodes, a human technician may deploy nodes without sufficient shaking.

Smart-Its Friends [13] and Are You with Me [17] are two related schemes that use movement to establish a secret key. In addition to the drawbacks of Shake Them Up, these schemes require an accelerometer on each node to measure movement. Because Shake Them Up makes one fewer assumption than these schemes, Smart-Its Friends and Are You with Me are not included in Table 2.

As illustrated in Table 2, MIB achieves all but one of the listed properties. Key secrecy and authenticity are attained because the five techniques described in Section 3.2.2 ensure that an attacker may not eavesdrop or inject its own key onto the new node. Demonstrative identification is achieved since the user knows that the node in the Faraday cage is the node which receives a key. MIB is robust to user error: any human error (e.g., premature opening of the Faraday cage) results in a failed deployment – rather than key compromise. Furthermore, MIB only requires symmetric cryptographic operations.

MIB requires a special Faraday cage and key deployment nodes with an additional USB interface. However, we argue that MIB is still cost effective because it does not require any specialized hardware per node. This tradeoff is a favorable one: one specialized Faraday cage and two deployment nodes can be used to perform key deployment on many sensor nodes. For large deployments, specialized setup hardware is more economical than additional per-node hardware.

10 Conclusion

Prior sensor network security architectures assume that communicating nodes possess an authenticated public key or a shared secret key. The initial distribution of keys in a secure fashion is vital to the security of a sensor network – yet no viable approach has been proposed to date.

We explore the technical and human factors that make initial key distribution challenging. For example, a Faraday cage can only attenuate radio signals in the real world; it cannot block signals completely. In addition, a protocol must be easy for end users to execute correctly – and resistant to inevitable errors.

We propose Message-In-a-Bottle, a user-friendly protocol for key deployment in high-security environments. We designed MIB with a combination of security, usability, and economic properties to ensure its applicability to real-world applications. The security properties of MIB include key secrecy, key authenticity, and forward secrecy. The usability of MIB has been validated by the low error rates in our user study. The economic viability of MIB is based on using the wireless channel for setup, avoiding additional per-node costs for special setup hardware. These characteristics make MIB a cost effective, usable, and secure solution for commodity sensor nodes.

11 Acknowledgments

We would like to thank Evan Gaustad, our shepherd Richard Han, and our anonymous reviewers for their generous assistance. Also, we thank Jeff Potter for suggesting the MIB name.

12 References

[1] D. Balfanz, D. Smetters, P. Stewart, and H. C. Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *Symposium on Network and Distributed Systems Security (NDSS)*, Feb. 2002.

[2] M. Cagalj, S. Capkun, and J.-P. Hubaux. Key agreement in peer-to-peer wireless networks. *Proceedings of the IEEE (Special Issue on Security and Cryptography)*, 94(2), 2006.

[3] M. Cagalj, S. Capkun, R. Rengaswamy, I. Tsigkogiannis, M. Srivastava, and J.-P. Hubaux. Integrity (I) codes: Message integrity protection and authentication over insecure channels. In *IEEE Symposium on Security and Privacy*, May 2006.

[4] C. Castelluccia and P. Mutaf. Shake them up! a movement-based pairing protocol for cpu-constrained devices. In *Proceedings of ACM/Usenix Mobisys*, 2005.

[5] H. Chan, R. Anderson, and A. Perrig. Key infection: Smart trust for smart dust. In *Proceedings of IEEE International Conference on Network Protocols*, May 2004.

[6] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *IEEE Symposium on Security and Privacy*, May 2003.

[7] ChipCon Products from Texas Instrments. *CC2420 Data Sheet*.

[8] Concentric Technology Solutions. RF Shield Box. `http://www.rfshieldbox.com/RF_Products.htm`.

[9] J. Deng, C. Hartung, R. Han, and S. Mishra. A practical study of transitory master key establishment for wireless sensor networks. In *Proceedings of IEEE/CreateNet Conference on Security and Privacy for Emerging Areas in Communication Networks (SecureComm)*, 2005.

[10] W. Du, J. Deng, Y. S. Han, and P. K. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communication Security*, pages 42–51, Oct. 2003.

[11] L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM Conference on Computer and Communication Security*, pages 41–47, Nov. 2002.

[12] C. Hartung, R. Han, C. Seielstad, and S. Holbrook. Firewxnet: A multitiered portable wireless system for monitoring weather conditions in wildland fire environments. In *The Fourth International Conference on Mobile Systems, Applications, and Services (MobiSys)*, June 2006.

[13] L. E. Holmquist, F. Mattern, B. Schiele, P. Alahuhta, M. Beigl, and H.-W. Gellersen. Smart-its friends: A technique for users to easily establish connections between smart artefacts. In *Proceedings of Ubicomp 2001*, 2001.

[14] C. Karlof, N. Sastry, and D. Wagner. TinySec: A link layer security architecture for wireless sensor networks. In *Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2004.

[15] Y. W. Law, J. Doumen, and P. Hartel. Survey and benchmark of block ciphers for wireless sensor networks. *ACM Transactions on Sensor Networks*, 2(1):65–93, February 2006.

[16] A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[17] J. Lester, B. Hannaford, and B. Gaetano. Are you with me? - using accelerometers to determine if two devices are carried by the same person. In *Proceedings of Pervasive 2004*, 2004.

[18] D. Liu and P. Ning. Establishing pairwise keys in distributed sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communication Security*, pages 52–61, Oct. 2003.

[19] D. Liu, P. Ning, and W. K. Du. Group-based key predistribution in wireless sensor networks. In *Proceedings of WiSe*, Apr. 2005.

[20] M. Luk, G. Mezzour, A. Perrig, and V. Gligor. Minisec: A secure sensor network communication architecture. In *Proceedings of ACM and IEEE Conference on Information Processing in Sensor Networks (IPSN)*, Apr. 2007.

[21] J. P. Lynch and K. Loh. A summary review of wireless sensors and sensor networks for structural health monitoring. *Shock and Vibration Digest*, 38(2):91–128, 2005.

[22] J. McCune, A. Perrig, and M. K. Reiter. Seeing-Is-Believing: Using camera phones for human-verifiable authentication. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2005.

[23] Moteiv Corp. *Telos Data Sheet*.

[24] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. *Wireless Networks*, 8(5):521–534, Sept. 2002.

[25] M. Ramjumar and N. Memon. An efficient key predistribution scheme for ad hoc network security. In *Proceedings of IEEE Journal of Selected Areas in Communications*, Mar. 2005.

[26] T. Rappaport. *Wireless Communications: Principles & Practice*. Prentice-Hall, 2001.

[27] K. B. Rasmussen and S. Capkun. Implications of radio fingerprinting on the security of sensor networks. In *Proceedings of IEEE SecureComm*, 2007.

[28] Sensys Networks. Components: VSN240 vehicle sensor nodes. `http://www.sensysnetworks.com/vsn240.html`.

[29] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. Swatt:software-based attestation for embedded devices. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2004.

[30] F. Stajano. The Resurrecting Duckling - What Next? In *Proceedings of Security Protocols Workshop 2000*, 2000.

[31] F. Stajano and R. Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Security Protocols, 7th International Workshop*. Springer Verlag, 1999.

[32] YSI Environmental. Environmental products. `http://www.ysi.com/index.html`.

[33] S. Zhu, S. Setia, and S. Jajodia. Leap: efficient security mechanisms for large-scale distributed sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communication Security*, pages 62–72. ACM Press, 2003.

[34] ZigBee Alliance. ZigBee Specification. Technical Report Document 053474r06, Version 1.0, ZigBee Alliance, June 2005.