MiniSec: A Secure Sensor Network Communication Architecture*

Mark Luk, Ghita Mezzour, Adrian Perrig
CyLab, Carnegie Mellon University
Pittsburgh, PA
mluk@cmu.edu, gmezzour@cmu.edu, adrian@cmu.edu

Virgil Gligor
University of Maryland
College Park, MD
gligor@eng.umd.edu

ABSTRACT

Secure sensor network communication protocols need to provide three basic properties: data secrecy, authentication, and replay protection. Secure sensor network link layer protocols such as TinySec [10] and ZigBee [24] enjoy significant attention in the community. However, TinySec achieves low energy consumption by reducing the level of security provided. In contrast, ZigBee enjoys high security, but suffers from high energy consumption.

MiniSec is a secure network layer that obtains the best of both worlds: low energy consumption and high security. MiniSec has two operating modes, one tailored for single-source communication, and another tailored for multi-source broadcast communication. The latter does not require per-sender state for replay protection and thus scales to large networks. We present a publicly available implementation of MiniSec for the Telos platform, and experimental results demonstrate our low energy utilization.

Categories and Subject Descriptors: D.4.4 [**Communications Management**]: Network communication; D.4.6 [**Security and Protection**]: Cryptographic controls.

General Terms: Security, Design.

Keywords: Sensor network security, secure communication architecture.

1. INTRODUCTION

Considerable attention had been paid to developing secure sensor network communication protocols. Unfortunately, existing technologies, such as TinySec and ZigBee, are unable to achieve low energy consumption while simultaneously providing the three important properties of secure communication: secrecy, authentication, and message replay protection.

TinySec, a popular secure link layer protocol, achieves low energy consumption and memory usage. Unfortunately, it also sacrifices on the level of security. For example, it employs a single network-wide key, such that every node in the network can masquerade as any other node. Second, TinySec does not attempt to protect against replay attacks.

ZigBee provides a higher level of security than TinySec since it is not restricted to a network-wide key. By keeping a per-message counter as the Initialization Vector (IV), ZigBee protects against message replay attacks. However, ZigBee is an expensive protocol. First, ZigBee sends the entire 8-byte IV with each packet, resulting in high communication overhead and high energy consumption by the radio. Also, ZigBee requires per-sender state, which consumes a large amount of memory as the number of participants increases.

In this paper, we present MiniSec, a secure network layer protocol for wireless sensor networks. We achieve the best of both worlds: lower energy consumption than TinySec, and a high level of security like ZigBee. We accomplish this by leveraging three techniques. First, we employ a block cipher mode of operation that provides both secrecy and authenticity in only one pass over the message data. Second, we send only a few bits of the IV, while retaining the security of a full-length IV per packet. In contrast, previous approaches require two passes over the plaintext (one for encryption and one for authentication) and transmission of the full-length IV. Third, we exploit the fundamental distinctions between unicast and broadcast communication, providing two energy-optimized communication modes. In unicast mode, we reduce the radio's energy consumption by using synchronized counters and performing extra computation. Although radically different from conventional networking protocols, such a scheme is desirable in this setting because of the stringent energy constraints of sensor networks. In broadcast mode, we employ a Bloom-filter based replay protection mechanism that avoids per-sender state.

MiniSec's improvement in energy consumption comes at the cost of a modest increase in memory size, which is a desirable tradeoff in sensor nodes. Note that TinySec had been developed for the Mica2 motes in 2003, when memory constraints were much more stringent than they are today. The current trend in mote development reveals that memory size has been consistently increasing, while energy constraints remain as stringent as ever. Thus, the design tradeoffs in MiniSec make it well-suited for current state-of-the-art sensor devices.

We present MiniSec as a complete solution to secure sensor network communication, and implement the protocol for the Telos motes [18]. To the best of our knowledge, MiniSec is the first gen-

*This research was supported in part by CyLab at Carnegie Mellon under grant DAAD19-02-1-0389 from the Army Research Office, and grants CNS-0347807 and CCF-0424422 from the National Science Foundation, and by a gift from Bosch. Virgil Gligor's work was supported by the US Army Research Laboratory under the Cooperative Agreement DAAD19-01-2-0011 for the Collaborative Technology Alliance for Communications and Networks. 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.

IPSN'07, April 25-27, 2007, Cambridge, Massachusetts, USA.

eral purpose security protocol available for this popular platform. Furthermore, MiniSec's source code is publicly available [1], and it can be easily ported to other platforms.

We evaluate MiniSec's performance against TinySec [10]. Our results show that under most circumstances, our energy consumption of security related tasks is about $\frac{1}{3}$ of the energy consumption TinySec, yet we are still able to provide a higher level of security.

The main contributions of this paper are s follows.

- We introduce MiniSec, the first fully-implemented general purpose security protocol for the Telos sensor motes.
- MiniSec simultaneously achieves low energy overhead and a high level of security (data secrecy, authentication, and replay protection). This is the first protocol to achieve replay protection in the sensor network broadcast setting.
- We measure the performance of MiniSec and show that, under most real-world scenarios, MiniSec outperforms other comparable systems.
- The source code of MiniSec is publicly available. We provide a turnkey system for sensor network developers that seamlessly integrates into TinyOS applications.

2. BACKGROUND

Several encryption modes exist that achieve secrecy and authentication. We select OCB [20] as our encryption mode since it is especially well-suited for the stringent energy constraints of sensor nodes. In addition to OCB, MiniSec also uses Bloom filters [4] and loose time synchronization to minimize energy consumption. In concert, Bloom filters and loose time synchronization can be used to provide efficient replay protection in broadcast communication.

In this section, we briefly review OCB and Bloom filters.

OCB. OCB, or Offset CodeBook, is a block-cipher mode of operation that features authenticated encryption. Given a plaintext of arbitrary length, OCB generates a ciphertext that simultaneously provides authenticity and data secrecy. OCB is provably secure, and is parameterized on a block cipher of blocksize n and a tag of length τ. τ is defined such that an adversary is able to forge a valid ciphertext with probability of $2^{-\tau}$.

OCB operates as follows. Let M be an arbitrary length message that needs to be encrypted and authenticated, H be an optional message header that only requires authentication, K be the encryption key (which is the key used by the underlying block cipher), and N be a non-repeating nonce. First, OCB takes in M, K, and N, and generates the ciphertext core C. Concurrently, using the plaintext M, ciphertext C, and H, OCB generates tag of length τ. The final output of $OCB_K(N,M,H)$ is the tuple (C,tag). To decrypt a ciphertext (C,tag), the receiver performs the reverse process on C to arrive at plaintext M. Then, the receiver ensures that the tag is as expected. If the receiver computes a different tag than the one in the ciphertext, the ciphertext is considered to be invalid.

OCB is especially well suited for sensor nodes. First, OCB avoids ciphertext expansion. Given an arbitrary length messages M, OCB generates ciphertext with length of $|M|+\tau$. Disregarding the tag, the ciphertext core has the same length as the plaintext. Second, OCB has superior performance, since it provides secrecy and authenticity in one pass of the block cipher. TinySec and ZigBee provide the same security guarantees, but requires two passes of the block cipher: one pass achieves secrecy with CBC-encryption, and another pass achieves authenticity with CBC-MAC. Consequently, since TinySec and SNEP almost doubles the amount of computation, the energy consumption also doubles. By comparing the number of block cipher calls, OCB only requires $\lceil \frac{|M|}{n} \rceil$ block cipher calls [20], while CBC-encryption and CBC-MAC together take between $2\lceil \frac{|M|}{n} \rceil + 1$ to $2\lceil \frac{|M|+1}{n} \rceil + 4$ block cipher calls, depending on padding (recall that $|M|$ is the message length, and n is the blocksize).

Bloom filter. The Bloom filter is a space-efficient data structure used for fast probabilistic membership tests [4]. It features two functions: membership addition and membership query. It is possible to have false positives, where a query returns true even though the element is not in the set. However, false negatives, where a query returns false when the element is in fact a member of the set, is not possible.

A Bloom filter requires an array of m bits and k hash functions. The array is first initialized to all zeros. Each hash function maps an element to one of the m array positions. To add an element, the element is hashed by all k hash functions, and the k subsequent array positions are set to one. To query for an element, we use the k hash functions to arrive at the k bit positions. If any of them were zero, then the element is not in the set. If all bits were set to one, then either the element is indeed in the set, or all k bits were set to one by insertion of other elements. The latter demonstrates a false positive. The more elements are added to a Bloom filter, the higher the probability of a false positive.

Bloom filters are well suited in the severe resource-constrained environment of sensor nodes because of space and time advantages. The space advantage is apparent, since it only requires $O(n \log n)$ bits to store a fingerprint of n messages. Bloom filters also exhibit the desirable property that adding and querying an element occurs in constant time.

3. PROBLEM DEFINITION

3.1 Assumptions

We assume that symmetric keys are already established between each sender and its receivers. We recommend a different key for each sender, but our protocol is by no means restricted to such a setup. We also assume a secure routing protocol that can successfully route packets to the intended destination with non-zero probability.

A list of sensor network keying and routing protocols can be found in Section 9. The goal of MiniSec is to leverage such existing primitives to provide for secure node-to-node communication at low energy cost.

3.2 Attacker Model

Sensor nodes rely on radio broadcasts for communication. We thus adopt the Dolev-Yao attacker model, where an attacker can overhear, intercept, alter, or inject any messages into the radio communication channel.

We do not restrict attackers to computationally bounded motes. By using sufficiently long symmetric keys (i.e., 80 bits), we can defend against brute force attacks by a powerful adversary [13].

3.3 Desired Properties

We now present the desired properties of a secure sensor network communication architecture.

Data Authentication. Data authentication empowers legitimate nodes to verify whether a message indeed originated from another legitimate node (i.e., a node with which it shares a secret key) and was unchanged during transmission. As a result, illegitimate nodes

[1] http://www.ece.cmu.edu/~mluk

should not be able to participate in the network, either by injecting their own messages or by modifying legitimate messages.

Data authentication is one of the basic building blocks of a secure system. For example, nodes need to verify commands from the base station, and a base station needs to authenticate whether the data readings indeed originate from valid nodes.

Typically, data authentication is achieved by the sender computing a message-authentication code (MAC) over the payload and appending that to the message. Upon reception, the packet is considered to be valid if the receiver recomputes the MAC and it matches with the received MAC. ZigBee, TinySec and SNEP provide data authentication by using the CBC-MAC function, using Skipjack or RC5 as the block cipher.

Data Secrecy. Data secrecy, another basic requirement of any secure communication system, prevents unauthorized parties from discovering the plaintext. It is typically accomplished by setting up an encrypted communication channel.

Encryption schemes or modes can be evaluated based on different criteria. A strong level of security is the notion of *semantic security* [3, 9]. The Handbook of Applied Cryptography defines semantic security such that "a passive adversary with polynomially bounded computational resources can learn nothing about the plaintext from the cipher text" [1]. Semantic security implies that an eavesdropper cannot gain any information about the plaintext, even after observing many encryptions of the same plaintext.

In secure communication protocols, data secrecy is provided by a cryptographic encryption scheme. To guarantee semantic security, we typically require a probabilistic encryption scheme and an unique initialization vector (or IV) for each encryption to add variation to the ciphertext.

TinySec uses CBC-encryption, while SNEP and ZigBee employ counter-mode encryption. MiniSec provides both authentication and secrecy using OCB-encryption with a non-repeating counter.

Replay Protection. A replay attack is when attackers record entire packets and replay them at a later time. TinySec is not resilient to such an attack, while SNEP provides protection using a counter. Sections 5 and 6 demonstrate how MiniSec provides replay protection in the unicast and broadcast setting, respectively.

Freshness. Since sensor nodes often stream time-varying measurements, providing guarantee of message freshness is an important property. There are two types of freshness: strong freshness and weak freshness. MiniSec provides a mechanism to guarantee weak freshness, where a receiver can determine a partial ordering over received messages without a local reference time point. Note that both ZigBee and SNEP provide weak freshness, while TinySec does not provide any form of freshness.

Low Energy Overhead. Energy is an extremely scarce resource in sensor nodes. Thus, it is of paramount importance for the security protocol to retain a low energy overhead. In particular, radio communication consumes the most amount of energy. On the Telos platform, sending a single byte is equivalent to executing about 4720 instructions. Thus, to reduce energy consumption, it is imperative to minimize communication overhead.

Although public key cryptography had enjoyed major advancement recently, it is still 3–4 orders of magnitude more expensive than symmetric cryptography in typical sensor nodes. Because it requires less energy consumed by the processor, security protocols that only employ symmetric cryptography are preferred in sensor network applications.

Resilient to Lost Messages. The relatively high occurrence of dropped packets in wireless sensor networks requires a design that can tolerate high message loss rates.

4. MINISEC OVERVIEW

We present MiniSec, a secure network layer that satisfies all the security properties outlined in Section 3. MiniSec has two operating modes: unicast and broadcast, henceforth known as MiniSec-U and MiniSec-B. Both schemes employ the OCB-encryption scheme to provide for data secrecy and authentication (see Section 2), while using a counter as a nonce. The two modes differ in the way they manage the counters. In MiniSec-U, we employ synchronized counters, which require the receiver to keep a local counter for each sender. MiniSec-B has no such requirement for per-sender state. Instead, meta-data to prevent replay attacks is stored in a Bloom Filter. Both schemes will be explained in detail in Sections 5 and 6, respectively.

Notation. We use the following notation to describe our protocol and cryptographic operations:

A, B	Communicating nodes.
K_{AB}	OCB Encryption key used for communication channel from A to B. Note that key K_{BA} would be used to encrypt data from B to A.
C_{AB}	Monotonically increasing counter corresponding to K_{AB}.
$(C, tag) = OCB_K(N, M, H)$	Authenticated encryption under OCB, where M is the plaintext message, H is an optimal message header that only needs to be authenticated, N is a 64-bit nonce, and K is the OCB encryption key.
N_A	A nonce generated by device A.

5. MINISEC-U: UNICAST COMMUNICATION

5.1 Motivation

Both TinySec and SNEP have developed solutions for providing secure communication in the unicast setting, where we have one sender A and one receiver B. Although both protocols attempt to minimize energy consumption, there are aspects of both that demonstrate inefficient energy usage. TinySec uses an encrypted counter as its IV. This counter is appended to each message, resulting in a 2-byte overhead per packet. SNEP also uses a counter as the IV. However, SNEP conserves radio energy consumption by not sending the counter with each packet. Instead, the counter is kept as internal state by both sender A and receiver B. However, dropped packets would cause the counters to become inconsistent, in which case both parties need to participate in an expensive counter resynchronization protocol.

TinySec and SNEP represent two extremes: sending the entire counter as opposed to not sending the counter at all. The key insight behind MiniSec-U is that optimal energy usage can be achieved by combining the best of both approaches. Similar to SNEP, MiniSec-U maintains a synchronized monotonically increasing counter C_{AB} between a sender A and a receiver B as the IV. However, MiniSec-U includes the last x bits of the counter along with each packet. We call this the *LB* (Last Bits) *Optimization*, and the last x bits of the counter is called the *LB value*. By keeping x low, the radio's energy consumption is almost as low as not sending the counter at all.

The LB optimization addresses one of the main drawbacks of SNEP, which is running the expensive counter resynchronization protocol when packets are dropped. Instead, the LB optimization allows resynchronization to occur "implicitly." Since sender A sends the last x bits of the counter, receiver B can compare the last x bits of its local counter C_{AB} to the LB value appended to the packet. As long as there are fewer then 2^x dropped packets since

the last successfully received packet, the receiver can immediately increment his counter such that the final x bits match the LB value. For example, let x be 3, and the counter C_{AB} be synchronized on both parties at 0. Sender A sends six packets, but the first five packets were dropped. Receiver B successfully receives the 6^{th} packet, which was sent using C_{AB} of 5. B would thus immediately increment hits counter C_{AB} to 5 and attempt decryption.

The LB optimization is useful even if more than 2^x packets were dropped, since the receiver could simply continue to increment C_{AB} by 2^x and reattempt decryption. In practice, the receiver B would set a maximum such that after y consecutive failed decryptions, B would run the expensive counter resynchronization protocol.

Lastly, by specifying the parameter x, we could arbitrary lower the probability of reverting to the resynchronization protocol. By monitoring the quality of the channel, it is possible to analytically solve for the optimal values of x and y such that energy consumption is minimized. We demonstrate this in Section 5.3.

5.2 Protocol Description

In this section, we describe the mechanics of MiniSec-U, in the context of sender A and receiver B. In MiniSec-U, each pair of sender and receiver share two keys, K_{AB} to protect communication from A to B, and K_{BA} to protect communication from B to A. Furthermore, a monotonically increasing counter is assigned to each key as the IV (C_{AB} used to for key K_{AB}), and is kept as internal state by both sender and receiver.

We employ OCB-encryption with the packet payload as M, packet header as H, counter C_{AB} as the nonce, and K_{AB} as the encryption key. We selected Skipjack to be the underlying block cipher with a blocksize of 64 bits. Since OCB requires the nonce to be the same length as the block size, counter C_{AB} can also be 64 bits. Alternatively, the counter could be of shorter length, and be padded out to 64-bits when requested by the OCB encryption function. The second parameter of OCB is the tag length, which we set to 32 bits, a length suitable for security in retail banking [20]. Finally, receiver B needs to maintain counter C_{AB}. Upon receiving and decrypting a valid packet, B would increment its local copy of C_{AB} accordingly so that it remains consistent with A.

Since OCB is probabilistic, a strictly monotonically increasing counter C_{AB} guarantees semantic security. Even if the sender A repeatedly sends the same message, each ciphertext is different since a different counter value is used. Also, once a receiver observes a counter value C_{AB}, it rejects packets with an equal or smaller counter value. Therefore, an attacker cannot replay any packet that the receiver has previously received.

There are four interesting issues about using counter C_{AB} as the IV. First, because of the nature of OCB encryption, the counter itself is not a secret. Even if an attacker knows the counter, the specified security properties are not compromised. This contrasts to CBC encryption used by TinySec, which requires a random and unpredictable nonce as the IV. Second, in order to provide semantic security, the counter cannot wrap around. A longer counter achieves higher level of security, yet adds additional overhead to each packet. Since we do not append the entire counter to each packet, MiniSec enjoys the security benefits of a longer counter without paying the communication cost. This allows us to use a 32-bit or longer counter while TinySec uses a 16-bit counter. Third, despite the LB optimization, large number of dropped packets could still cause the counters to be desynchronized. Appendix A presents a counter resynchronization protocol similar to one used in SNEP. Finally, a different counter is needed to be instantiated per key. Thus, if we rekey, the counter can be reset to zero.

5.3 Energy Analysis

Let x be the number of lowest counter bits to append to the packet, and y be the maximum number of trial decryptions the receiver performs as explained above. Let random variable I represent the time at which the node is able to decrypt the message (if the node successfully decrypts the message), where $1 \leq I \leq y$. The goal of this section is to determine the parameters x and y.

Generally, a receiver would perform i decryptions, if there were at least $(i-1)2^x$ and at most $i2^x - 1$ dropped packets. Thus, the probability that exactly i decryptions occur is (where P_{gar} is the probability that a message is garbled in transmission, and p_r is the probability that the message is received):

$$\begin{aligned} P(I=i) &= (1-P_{\text{gar}}) \sum_{k=(i-1)2^x}^{i2^x-1} (1-p_r)^k p_r \\ &= (1-P_{\text{gar}})(1-p_r)^{(i-1)2^x}\left[1-(1-p_r)^{2^x}\right] \end{aligned}$$

Thus, the expectation of I is:

$$\begin{aligned} E(I) &= \sum_{i=1}^{y} iP(I=i) \\ &= \left[1-(1-p_r)^{2^x}\right] \sum_{i=1}^{y} i\left[(1-p_r)^{2^x}\right]^{i-1} \\ &= \left[1-(1-p_r)^{2^x}\right] \frac{1-(y+1)(1-p_r)^{y2^x}+y(1-p_r)^{(y+1)2^x}}{\left[1-(1-p_r)^{2^x}\right]^2} \\ &= \frac{1+(1-p_r)^{y2^x}\left[1+y\left[1-(1-p_r)^{2^x}\right]\right]}{1-(1-p_r)^{2^x}} \end{aligned}$$

The expected energy consumed is the sum of the expected energy spent when an event happens times the probability of this event to occur plus the energy required for receiving x bits. At the reception of a message, two scenarios are possible for the receiver: (1) the receiver is able to decrypt the message at the i-th trial decryption. This event occurs with probability $P(I=i)$ and its cost is $E(i * E_{\text{OCB}}) = E(I)E_{\text{OCB}}$, where E_{OCB} represents the cost of an OCB decryption. (2) The receiver is unable to decrypt the message even after trying y times. This happens either because more than $y2^x$ packets were lost consecutively, or because the packet that was just received is garbled due to a transmission error (an unlikely event).

Thus, let E_{rec} be the energy for receiving one bit, E_{resync} be the energy required to execute the counter resynchronization protocol, and $E(I)$ be as described above, then the expected energy consumption per packet E_{Energy} is:

$$\begin{aligned} E_{\text{Energy}} &= E(I)E_{\text{OCB}}(1-P_{\text{gar}}) + \\ &\quad \left[yE_{\text{OCB}} + E_{\text{resync}}\right]\left[P_{\text{gar}} + (1-P_{\text{gar}})(1-P_r)^{y2^x}\right] + xE_{\text{rec}} \end{aligned}$$

We need to find the ideal x and y for a given environment. A lossy channel with high packet loss requires larger values for x and y. In Section 8 we discuss how to select x and y in practice.

6. MINISEC-B: BROADCAST COMMUNICATION

MiniSec-U cannot be directly used to secure broadcast communication. First, it would be too expensive to run the counter resynchronization protocol among many receivers. Also, if a node B were to simultaneously receive packets from a large group of sending nodes $(A_1, A_2, \ldots, A_n)$, B would need to maintain a counter for each sender, resulting in high memory overhead.

Similar to MiniSec-U, MiniSec-B uses OCB encryption to secure broadcast communication. Simply encrypting each packet

with OCB provides secrecy and authenticity, while an increasing counter can still be used as the IV to provide for partial ordering of messages. However, without synchronized counters, there is no defense against replay attacks. In fact, defending against replay attacks in a broadcast setting without per-sender state is currently an open challenge in the sensor network community.

This section describes two related mechanisms used in MiniSec-B to provide replay protection. First, we present a sliding-windows approach that defends against replay attacks with a certain vulnerability window; replayed packets outside the sliding-window are always dropped. Next, we discuss a Bloom Filter-based approach which defends against attacks within the window.

6.1 Sliding-windows Approach

We define an epoch to be a finite time t_e, and we segment time into a series of t_e-length epochs $\{E_1, E_2, E_3, \ldots\}$. Leveraging loose time synchronization [8,21], each node agrees on the current epoch E_i. The maximum time synchronization error between any pair of nodes is Δ_T. Finally, let Δ_N be the maximum network latency.

Simply using the current epoch number as the nonce for OCB-encryption defends against replay attacks from older epochs. Unfortunately, because of time synchronization error and network latency, such a scheme experiences high false positives at epoch transitions, as legitimate packets sent from the previous epoch will be discarded.

The solution is to simply perform decryption with two possible candidate epoch values for the nonce. First, we define epoch length t_e to be $2\Delta_T + \Delta_N$. Let E_i be the current epoch number, and t_i be the time at the start of E_i. If one receives a packet within (t_i, $t_i + \Delta_T + \Delta_N$), the two candidate values for the nonce are E_i and E_{i-1}. If one receives a packet within ($t_i + \Delta_T + \Delta_N$, t_{i+1}), the two candidate values are E_i and E_{i+1}. This threshold is depicted in Figure 1 as the dotted line.

The intuition is as follows. First, we define t_{s_i} to be the local time recorded by the sender when packet i is sent, and t_{r_i} to be the local time experienced by receiver when packet i is received. Further, let δ_n be the actual network latency such that $0 < \delta_n < \Delta_N$, and δ_t be actual time synchronization error such that $-\Delta_T < \delta_t < \Delta_T$. Note that a positive δ_t indicates that the sender's clock is behind the receiver's clock, while a negative δ_t indicates the opposite. Thus, we have

$$t_{s_i} + \delta_n + \delta_t = t_{r_i}$$

In the first case, as illustrated in Figure 1, we deal with packets that are received within the range of t_{r_1} and t_{r_2}, where $t_{r_1} = t_i + \varepsilon$, and $t_{r_2} = t_i + \Delta_T + \Delta_N - \varepsilon$. To show that such traffic must originate from either epoch E_{i-1} or E_i, the receiver needs to determine the earliest possible value of t_{s_1} and the latest possible value of t_{s_2}. Since $t_{s_1} = t_{r_1} - \delta_t - \delta_n$, the earliest value of t_{s_1} is $t_{r_1} - \Delta_T - \Delta_N$, which falls within E_{i-1}. Similarly, since $t_{s_2} = t_{r_2} - \delta_t - \delta_n$, the latest value of t_{s_2} is $t_{r_2} + \Delta_T$, which falls within epoch E_i.

In the second case (not illustrated), we deal with packets received within the range of t_{r_3} and t_{r_4}, where $t_{r_3} = t_i + \Delta_T + \Delta_N + \varepsilon$ and $t_{r_4} = t_{i+1} - \varepsilon$. Again, the receiver needs to determine the earliest possible value of t_{s_3} and latest possible value of t_{s_4}. Using similar logic, we find that earliest value of t_{s_3} is $t_{r_3} - \Delta_T - \Delta_N$, which falls within epoch E_i. Latest value of t_{s_4} is $t_{r_4} + \Delta_T$, which is within E_{i+1}.

Based on this technique, there are only two candidate epochs for any incoming packet. However, if a valid packet had been sent at the beginning of an epoch, an attacker can replay that packet for the remainder of the epoch as well as $\delta_t + \delta_n$ into the next epoch. Thus, the maximum window of vulnerability for replay is $3\delta_t + 2\delta_n$.

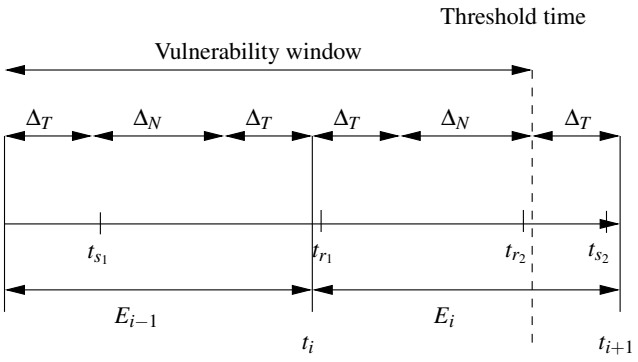

Figure 1: Timeline. Δ_T is the maximum time synchronization error between sensor nodes. Δ_N is the maximum network latency. E_i is the current epoch. This figure shows that a packet received at t_{r_1} could not have been sent earlier than t_{s_2}, and a packet received at t_{r_2} could have been sent after t_{s_2}.

6.2 Bloom Filter-based Approach

As stated above, the sliding-windows based approach has the drawback of a window of vulnerability for replay packets. This problem could be easily avoided by simply storing all packets received within this window. Unfortunately, this solution is not practical for sensor nodes because of limited storage. Thus, we augment the sliding-windows approach with a counter maintained as internal state by the sender, and two alternating Bloom Filters stored by all nodes. In concert, they provide replay protection within the vulnerability window using constant storage.

Similar to the counter used in MiniSec-U, counter C_A is kept as internal state by sender A and is incremented each time it sends a packet. The only difference is that the sender resets the counter at the beginning of each epoch, so that the counter itself can be short enough to be transmitted with each packet. If we can bound the number of broadcast messages sent by a node in each epoch to k, the length of the counter can be bounded to $\log_2(k)$.

Unlike MiniSec-U where simply the counter is used as the nonce for OCB encryption, the nonce in this case is the concatenation of the sender's nodeID A, C_A, and current epoch number E_i. Thus, the cipher-text sent is $(C, tag) = OCB_{K_A}(nodeID||C_A||E_i, M, H))$. Note that this nonce is never reused for a given key epoch numbers do not wrap around during a sender's lifetime.

Receiver B maintains its Bloom Filters in the following manner. A Bloom Filter BF_i is assigned to each epoch E_i. All valid packets corresponding to epoch E_i are stored in BF_i, using $(C_A||sourceID)$ as the Bloom Filter key. At all times, each node keeps a Bloom Filter BF_i for the current epoch E_i, and either BF_{i-1} for the previous epoch E_{i-1}, or BF_{i+1} for the next epoch E_{i+1}. For example, starting at the beginning of epoch E_i until $\Delta_T + \Delta_N$ into the epoch, the node maintains Bloom Filters BF_i and BF_{i-1}. After $\delta_t + \delta_n$, all packets from the epoch E_{i-1} will be dropped, while packets from the next epoch E_{i+1} will be accepted. Thus, BF_{i-1} is reset and reused as BF_{i+1}.

In the previous section, we concluded that only two candidate epochs exist for any incoming packet. Since we have one Bloom filter assigned to each epoch, a valid packet must correspond to one of the active two Bloom filters.

When receiver B receives a packet, it needs to perform two checks: 1) determine whether the packet is a valid packet encrypted under OCB, and 2) determine whether the packet has been replayed. To verify if it is a valid packet, the receiver performs OCB decryp-

tion. Since the counter and the source address are included in the packet as plaintext, the only portion of the nonce that is missing is the epoch number. The receiver can recover the epoch number easily, since there only exists two candidate epoch number for any received packet. Hence, the receiver attempts OCB decryption with both. If both decryption fail, the packet is dropped.

Next, the receiver needs to determine whether the packet has been replayed. Since the epoch number is known, B queries the corresponding Bloom Filter for this packet. If the query returns true, the packet is considered to be a replay and is consequently dropped. If the Bloom Filter declares that this packet is fresh, the packet is considered to be a non-replayed packet. It is consequently accepted and added into the Bloom Filter.

Using such a counter provides for semantic security since sending the same message never results in the same cipher-text. Also, this scheme provides for replay protection. If receiver B were to receive a replay packet, hashing the source ID and counter C_A, would result in a match in all the corresponding bits in the Bloom Filter. Therefore, B would suspect a replay attack and reject the packet.

Note that such a replay policy would detect all replayed attacks, resulting a 0% false negative rate. However, because Bloom Filters may cause false positives; a fresh packet may be deemed to be a replayed packet. There are various trivial solutions to lowering false positives. For example, by selecting the size of the Bloom Filter m and duration of an epoch, the probability of false positives can be lowered arbitrarily [4].

7. SECURITY ANALYSIS

In this section, we provide an analysis on the level of security promised by both MiniSec-U and MiniSec-B. First, we discuss properties that are common across both protocols. Next, we discuss how these protocols are different.

Authentication. Both MiniSec-U and MiniSec-B use OCB encryption to provide for data authentication over the payload and packet header. The security of OCB's authentication scheme is directly related to τ, the length of the *tag*. By setting τ to be 32 bits, an adversary has a 1 in 2^{32} chance of forging a correct *tag* for a particular message. This suffices for the majority of practical applications.

Secrecy and Semantic Security. Semantic security requires that nonces do not repeat. (Also note that since the sender uses a strictly monotonic counter as the nonce, each ciphertext would be different even if the plaintext were the same.) Avoiding repetition of nonce is easy. In MiniSec-U, the counter is kept as internal state, and thus can be made arbitrarily long. We choose 8 bytes, which means that the nonce would not repeat until after sending 2^{64} messages. In MiniSec-B, the nonce is the concatenation of sourceID, epoch number e, and a counter. By using all three variables, two messages in the network would never share the same nonce.

Weak Freshness. In MiniSec-U, the receiver can arrive at the counter value used for each packet by verifying the validity of OCB decryption. While in MiniSec-B, the counter value is included in the packet as plaintext. In both cases, the receiver can use the counter value of two messages to enforce message ordering, thus providing weak freshness.

MiniSec-U and MiniSec-B exhibit different behavior in replay protection.

Replay Protection in MiniSec-U. Each sender and receiver keeps a synchronized counter that is used as the nonce in OCB encryption. The receiver would only accept messages with higher counter values than the those maintained in the node state. Thus, replayed packets will all be rejected.

Replay Protection in MiniSec-B. The entire network lifetime is segmented into epochs. MiniSec-B leverages loose time synchronization to prevent replayed packets from previous epochs. Next, each receiver uses a Bloom Filter to track packet history for each epoch. Thus, MiniSec-B achieves protection against replay attacks.

8. IMPLEMENTATION

8.1 System Architecture

We have developed MiniSec for the Moteiv Telos motes - a popular architecture in the sensor network research community. It features the 8MHz TI MSP430 micro-controller, a 16-bit RISC processor that is well known for its low energy consumption. Although we implemented MiniSec on the Telos motes, our design principles are general enough such that porting MiniSec to different sensor platforms should yield similar performance results.

To implement MiniSec, we rewrote part of the TinyOS network stack. Specifically, we created `GenericCommMiniSec` based on `GenericComm`, a "generic" TinyOS network stack. Instead of using the interface `AMStandard` for Active Message transmission, `GenericCommMiniSec` directs all messages to `AMStandardMiniSec`, a custom-made ActiveMessage layer that encrypts outgoing messages and decrypts received packets. To use MiniSec, a developer simply needs to replace `GenericComm` with `GenericCommMiniSec` in the application's configuration file.

Packet Format. We base MiniSec's packet format on the current TinyOS packet header for Telos mote's CC2420 radio. The Telos mote is one of the first wireless sensor devices to be equipped with an IEEE 802.15.4 radio. Figure 2 shows the packet format for plain TinyOS, TinySec, and MiniSec.

The fields that MiniSec share with original TinyOS are: length, Frame Control Field, data sequence number, destination PAN address, destination address, and the AM number. MiniSec replaces the 2-byte CRC with a 4-byte tag, since the tag already protects the packet from tampering. In the original TinyOS, the 1-byte groupID serves as a crude form of access control. Each set of communicating nodes share a different group-ID, and messages with foreign group-IDs are dropped. This field is no longer necessary in MiniSec because access control is achieved through the use of different cryptographic keys. Finally, similar to TinySec, MiniSec requires a 2-byte source address, which is absent in a standard TinyOS packet. The net overhead of a MiniSec packet is 3-byte increase over a standard TinyOS packet.

8.2 MiniSec-U Implementation

MiniSec-U uses two security primitives: OCB encryption and Skipjack. We selected Skipjack to be the block-cipher because of efficient computation and low memory footprint [12]. To make encryption as flexible as possible, we set Skipjack's block size to 64 bits. Furthermore, we use 80-bit symmetric keys, since Lenstra and Verheul recommended that such keys are considered to be secure until 2012, even against resourceful adversaries [13]. When 80-bit keys become insecure, we would use 128-bit AES keys, which is secure for at least the next 20 years.

OCB implementation is ported from Rogaway's original OCB library[2] to the nesC interface. Similarly, the Skipjack implementation is ported to nesC from Yee Wei Law's implementation [12]. In total, MiniSec-U requires about 4000 lines of nesC code, and

[2]http://www.cs.ucdavis.edu/rogaway/ocb/code-2.0.htm, accessed June 26, 2006

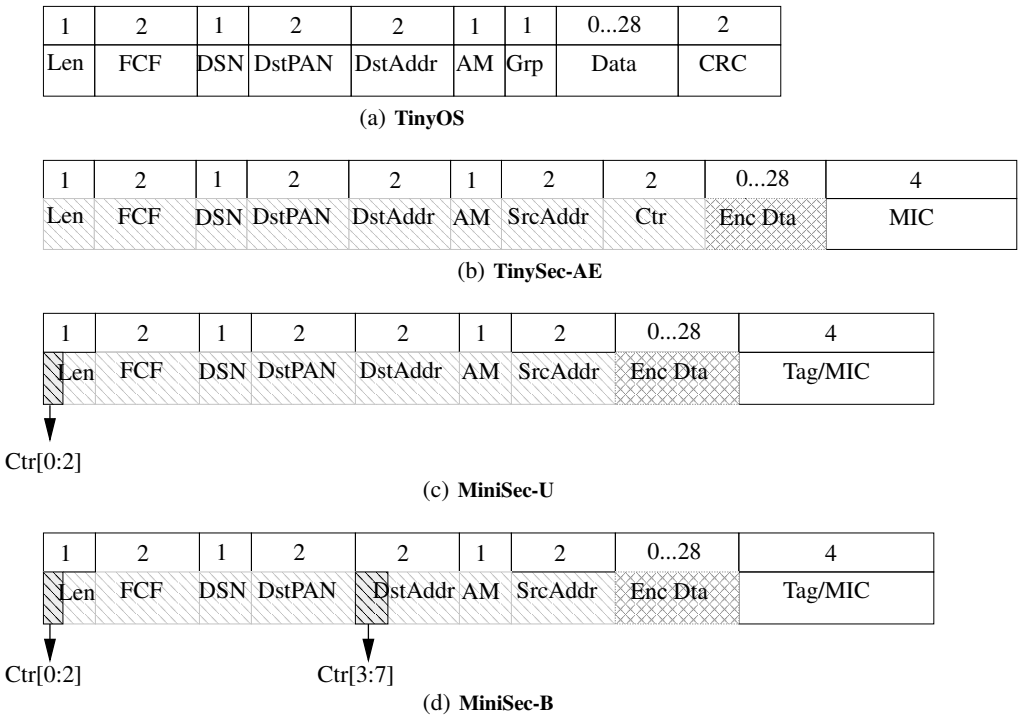

Figure 2: Packet Format. The size of each field is indicated in bytes. The packet format is based on the TinyOS packet for CC2420 radio. The diagonally hashed regions are authenticated; the checkered regions are encrypted; the gray region represents where we overload the corresponding field with the counter.

consumes 874 bytes of RAM and 16 KB of code memory.

Packet Format. MiniSec-U uses the LB optimization by sending the last x bits of the sender's counter along with each packet. Since TinyOS payloads are never greater than 29 bytes, we can safely overload the first 3 bits of the length field to store these bits, as shown in Figure 2(c). This is a significant advantage since we do not suffer any communication overhead for sending the last x bits of the counter.

Our empirical results show that by using the last 3 bits of the counter, even under high packet drop rate, the counter resynchronization protocol was rarely executed. Because of the above reasons, we use the default value of $x = 3$ for the remainder of the paper.

8.3 MiniSec-B Implementation

In addition to the security primitives in MiniSec-U, MiniSec-B utilizes loose time synchronization and Bloom filters. Here, we discuss practical issues in selecting epoch duration t_e and Bloom filter configurations.

Time Synchronization. As discussed in Section 6, epoch length t_e must be at least $2\delta_t + \delta_n$. Recent advancement in secure sensor network time synchronization [8, 21] enables pairwise time synchronization with error of mere μs. Transmission delay between neighboring nodes are on the order of ms. Even under extreme pessimistic conditions, epoch length of 1 second is longer than necessary according to the needs of MiniSec-B. For the remainder of our analysis, we will use epoch duration $t_e = 1s$.

Bloom filter configurations. A Bloom filter is defined by two configurations: size of the Bloom filter m, and number of hash functions h. We will show that under rather pessimistic assumptions of the hardware and network activity, we can achieve a 1% false positive rate by using $t_e = 1s$, $m = 18$ bytes, and $h = 8$ hash functions. We also show how the sensor network administrator can calculate custom values of m and h appropriate for a particular network parameterized on the network activity and underlying hardware.

The false positive rate of a Bloom filter can be calculated based on number of stored items. Thus, we first upper bound p_μ, the average number of packets received in one epoch of length t_e. If this is known a priori (e.g., regular heartbeats), the sensor network administrator can directly configure the Bloom filter accordingly. Else, we make the following argument.

Let t_l be a realistic lower bound of a node's lifetime, E_c be energy capacity of the node's battery, and E_p be energy consumption for receiving one packet. In the worse case, all energy provided by the battery will be used for packet reception. Hence, maximum possible packets received over the entire lifetime of the node is E_c/E_p. The number of packets received in one epoch, p_μ, can be calculated as follows.

$$p_\mu = \frac{E_c t_e}{E_p t_l}$$

In our calculations, we set t_l to 12 months, E_c to 2850 mAH (AA Duracell Coppertop alkaline battery[3]), and E_p to 0.0441mAS (receiving a TinyOS packet on CC2420 radio [18]). Average packet reception rate p_u is 7.48 packets per second.

Each packet adds one item into our Bloom filter. In traditional networking fashion, we model packet reception as a Poisson process. Thus, the number of packets received within an epoch can be approximated by a Poisson distribution with mean of p_μ. This model allows us to bound the maximum number of received packets in an epoch with high probability. The cumulative distribution function of a Poisson process is $\frac{\Gamma(k+1,\lambda)}{k!}$, where k is number of occurrences, λ is the Poisson mean p_μ, and Γ is the incomplete

[3]http://www.duracell.com/oem/Pdf/others/ATB-full.pdf

Table 1: Overhead. Table comparing packet size, communication overhead, and total energy spent in transmitting one TinyOS packet. Our of all three security protocols, MiniSec achieves the lowest communication overhead with respect to a standard TinyOS network stack.

	Payload (B)	Packet Overhead (B)	Security Overhead (B)	Total Size (B)	Energy (mAs)	Increase over TinyOS
TinyOS	24	12	–	36	0.034	–
TinySec	24	17	5	41	0.0387	13.9%
SNEP	24	20	8	44	0.0415	22.2%
MiniSec	24	15	3	39	0.0368	8.3%

gamma function. By setting the CDF of the Poisson distribution to an arbitrarily high probability p and solving for k, we conclude that one would not receive more than k packets in one epoch with probability p. In our case, we set p to 0.99, and arrived at $k = 14$. In other words, with probability of 0.99, we would never add more than 14 items to our Bloom filter during an epoch.

Given this information, we can set a particular false positive rate and solve for appropriate configurations for the Bloom filter size m and number of hash functions h. This problem had been previously studied by Almeida et al.'s work on Summary Cache, where they evaluated the statistics behind Bloom filters [11]. The probability of a false positive after inserting n elements is

$$\left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k$$

Thus, with the worst case of of $n = 14$ elements, we can achieve a 1% false positive rate with $m = 18$ bytes and $h = 8$ hash functions.

Note that this bound is extremely pessimistic, as the false positive rate should be significantly lower in practice. There are several factors:

- False positive rate increases as more packets are added into the Bloom filter. However, our calculation is based on the false positive rate at the *end* of the epoch, which corresponds to the highest possible false positive rate. For example, we stated that we achieve a 1% false positive rate with our Bloom filter configuration. However, if we were in the middle of an epoch, our calculation actually shows a 0.014% false positive rate.
- We make the pessimistic assumption that all energy is consumed by the radio for packet reception. In practice, energy is consumed for sending packets, controlling other devices (LEDs, sensors), and computation.
- We model the packet reception as a Poisson process. However, we don't use the mean packet arrival rate for false positive calculation. Instead, we use the 99^{th} percentile based on the CDF of the Poisson distribution. Thus, with probability of 0.99, we would have a false positive of at most 1%. As a matter of fact, using the same Bloom filter configuration, false positive rate would not exceed 0.0001% with probability of 0.5.
- We assume ideal conditions for batteries (constant current draw, constant temperature, no self-discharge prior to deployment). In practical settings, such ideal conditions are impossible to achieve. Thus, battery capacity and number of received packets are significantly lower.

Packet Format Figure 2(d) shows the packet header for MiniSec-B where the sender sends the entire counter with each packet. The default counter is 8 bits long, which we claim to be sufficient in most networks based on the following reasoning. First, since the counter is reset at the beginning of each epoch, the length of the counter can be bounded by $\log_2(k)$, where k is the maximum numbers of packets sent in each epoch. Next, it is unlikely for k to be large, since such resource-constrained nodes are unlikely to continually broadcast large amounts of data. An 8-bit counter is already extremely generous since it allows for 255 packets per epoch of one second.

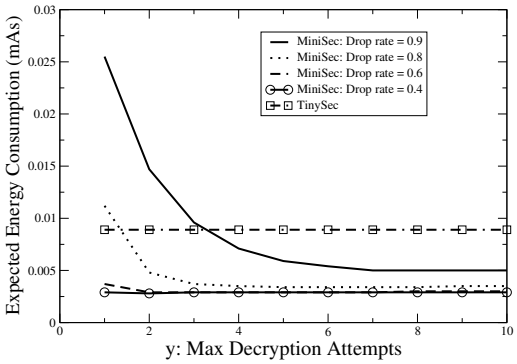

Figure 3: Optimal y. Max number of decryption attempts before counter resynchronization. Also shows aggregate energy consumption of all security related features.

This 8-bit counter could be declared as an addition field. However, based on the TinyOS packet format, we propose an optimization that overloads this counter with existing headers. First, 3 bits of this counter may be overloaded in the length field. Next, the remaining 5 bits of the counter may be embedded in the destination address, since the destination field is 2 bytes and it is unlikely for a network to have more than 2048 broadcasting participants. Furthermore, unlike unicasts, the destination address is generally not needed for routing in broadcasts.

8.4 Evaluation

To analytically evaluate the cost of security, we consider both the communication overhead on each packet as well as computational overhead from packet processing. Longer packets are extremely costly because of the extra energy consumption by the radio. Even sending one additional byte per packet would require significant amount of energy. As shown in Table 1, MiniSec was able to decrease a security overhead of 5 bytes by TinySec to 3 bytes. In addition, MiniSec employs OCB, which provides for authentication and secrecy with fewer block cipher calls than its cryptographic counterpart in TinySec.

MiniSec-U. MiniSec-U was able to save on packet header size by using synchronized counters between sender and receiver. First, as specified above, the LB optimization has the default value of $x = 3$. Using the expected energy equation from Section 5.3, it is possible to solve for the the optimal y, or max number of decryption attempts before counter resynchronization. Consequently, given the optimal

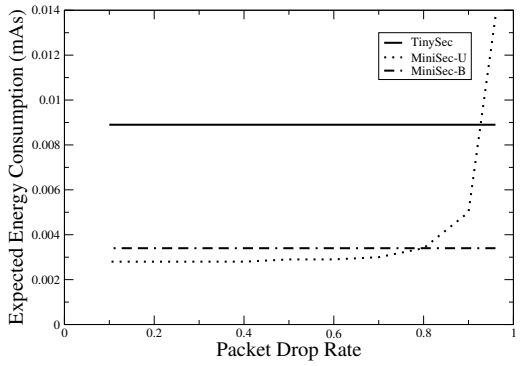

Figure 4: Drop Rate. Expected energy of security with respect to packet drop rate.

y, one can solve for the expected energy consumed by all security related features.

Figure 3 illustrates our findings given different packet drop rates based on analysis in Section 5.3. The aggregate energy consumed by TinySec is constant, as the energy overhead of security is always the amount of energy consumed by receiving 5 extra bytes in the header, computing CBC-MAC, and decrypting the packet. MiniSec-U, on the other hand, behaves differently based on environmental conditions. Given a packet drop rate, a higher *y* value exhibits lower energy consumption, since it reduces the probability of running the counter resynchronization protocol. However, increasing *y* exhibits diminishing marginal benefits, since the energy consumption of counter resynchronization becomes less and less of a factor in total energy consumption.

MiniSec-B. Figure 4 illustrates energy consumption between TinySec, MiniSec-B, and MiniSec-U using an optimal value *y*, while varying packet drop rate. The energy consumption of MiniSec-U is computed by first solving for the optimal *y*. Using this value, we were able to calculate the expected energy of security. MiniSec-B consumes a constant amount of energy, as its performance is not effected by lossiness of the communication channel. By leveraging an 8-bit counter and loose time synchronization, the receiver never needs to run any counter resynchronization protocol. Once the receiver successfully receives a packet, only two OCB decryptions and 8 hash functions needs to be performed.

Under normal circumstances, MiniSec consumes about $\frac{1}{3}$ amount of energy of TinySec. As packet drop rate increases beyond 0.9, TinySec is more efficient than MiniSec-U because of the high number of counter resynchronizations. Such a scenario represents an extremely rare case. On the other hand, since MiniSec-B's performance does not depend on the drop rate, MiniSec-B always outperforms TinySec.

Finally, we note that nothing prevents the use of MiniSec-B in unicasts. Since the energy consumption of MiniSec-B and MiniSec-U are comparable under normal conditions, while MiniSec-B far outperforms MiniSec-U under high packet loss, MiniSec-B is a much more robust protocol. If loose time synchronization is available, simply running MiniSec-B for all communication is an attractive solution.

9. RELATED WORK

Key establishment and management are considered to be prerequisites in secure sensor network communication, and have been extensively studied in the research community. There are numerous candidate solutions, such as Random Key Predistribution [5–7, 14, 19], Key Infection [2], and LEAP [23]. Asymmetric schemes based on elliptic curve cryptography [15] and Diffie-Hellman [22]. have also been proposed

Secure routing is another requirement for secure communication, such that packets would be successfully routed with non-zero probability as long as a path of non-compromised exists between the sender and receiver. In practice, a sensor network deployer can use one of the following routing techniques, such as INSENS [**?**, **?**], ARRIVE [**?**], JAM [**?**], and Secure Sensor Routing: A Clean Slate Approach [16].

The body of work most closely related to MiniSec consists of other secure communication protocols such as TinySec [10], ZigBee [24], and SNEP [17]. SNEP, part of the SPINS protocol suite, is one of the first attempts at a secure link layer protocol. It achieves low energy consumption by keeping a consistent counter between the sender and receiver, such that an IV is not required to be appended to each packet. However, packet loss would cause the counters to become inconsistent. Consequently, SNEP would have to execute a counter resynchronization protocol that is slow and expensive in terms of energy consumption. Inspired by SNEP, MiniSec makes various improvements to lower energy consumption. One such optimization is to reduce the probability that the resynchronization protocol needs to be executed.

TinySec is a link layer protocol designed by Karlof et al. [10]. It achieves low energy consumption by reusing part of the packet header as the IV. Thus, they were able to arrive at an 8-byte IV, but only adding a 2-byte counter overhead per packet. However, more serious drawbacks of TinySec are that (1) it only provides for authentication and data secrecy, but not replay protection; (2) it uses a single network-wide key, which is vulnerable to single node compromise.

ZigBee is a set of security standards proposed by a consortium interested in promoting embedded wireless technology. The security protocol is very similar to SNEP. However, the entire 8-byte counter is sent in the clear instead of being kept as consistent state between sender and receiver. Thus, ZigBee consumes significantly more energy than the other two protocols.

10. CONCLUSIONS

Battery energy is the main resource to conserve in current wireless sensor networks. Researchers have proposed several approaches for securing communication that optimize either for high level of security or for low energy utilization. Our secure sensor network communication package, MiniSec, offers a high level of security while requiring much less energy than previous approaches. We have implemented MiniSec on Telos motes and the source code is freely distributed under an open-source license.

11. REFERENCES

[1] *Handbook of Applied Cryptography*. CRC Press, 1997.

[2] Ross Anderson, Haowen Chan, and Adrian Perrig. Key infection: Smart trust for smart dust. In *Proceedings of IEEE International Conference on Network Protocols (ICNP 2004)*, October 2004.

[3] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption: Analysis of the des modes of operation. In *Proceedings of FOCS 97*, pages 394–403, 1997.

[4] B. Bloom. Space/time trade-offs in hash coding with allowable errors. In *Communications of the ACM*, July 1970.

[5] Haowen Chan, Adrian Perrig, and Dawn Song. Random key predistribution schemes for sensor networks. In *IEEE Symposium on Security and Privacy*, May 2003.

[6] W. Du, J. Deng, Y. Han, and P. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. In *Proceedings of the Tenth ACM Conference on Computer and Communications Security (CCS 2003)*, pages 42–51, October 2003.

[7] L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks. In *ACM CCS*, November 2002.

[8] S. Ganeriwal, S. Capkun, C. Han, and M. B. Srivastava. Secure time synchronization service for sensor networks. In *WiSe*, 2005.

[9] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer Security*, 28:270–299, 1984.

[10] Chris Karlof, Naveen Sastry, and David Wagner. TinySec: A link layer security architecture for wireless sensor networks. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys 2004)*, November 2004.

[11] J. Almeida L. Fan, P. Cao and A. Broder. Summary cache: A scalable wide-area web cache sharing protocol. In *ACM SIGCOMM 98*, 1998.

[12] Yee Wei Law, Jeroen Doumen, and Pieter Hartel. Survey and benchmark of block ciphers for wireless sensor networks. *ACM Transactions on Sensor Networks*, 2(1):65–93, February 2006.

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

[14] D. Liu, P. Ning, and W. Du. Group-based key pre-distribution in wireless sensor networks. In *WiSe*, April 2005.

[15] D. Malan, M. Welsh, and M. Smith. A public-key infrastructure for key distribution in TinyOS based on elliptic curve cryptography. In *IEEE SECON*, October 2004.

[16] Bryan Parno, Mark Luk, Evan Gaustad, and Adrian Perrig. Secure sensor network routing: A clean-slate approach. In *Proceedings of Conference on Future Networking Technologies (CoNEXT)*, December 2006.

[17] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. In *Seventh Annual International Conference on Mobile Computing and Networks (MobiCom 2001)*, pages 189–199, Rome, Italy, July 2001.

[18] Joseph Polastre, Robert Szewczyk, and David Culler. Telos: Enabling ultra-low power wireless research. In *IPSN/SPOTS*, 2005.

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

[20] P. Rogaway, M. Bellare, and J. Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. In *ACM TISSEC*, November 2001.

[21] K. Sun, P. Ning, C. Wang, A. Liu, and Y. Zhou. TinySeRSync: Secure and resilient time synchronization in wireless sensor networks. In *ACM CCS*, November 2006.

[22] R. Watro, D. Kong, S.-F. Cuti, C. Gardiner, C. Lynn, and P. Kruus. TinyPK: Securing sensor networks with public key technology. In *SASN*, October 2004.

[23] S. Zhu, S. Setia, and S. Jajodia. LEAP: Efficient security mechanisms for large-scale distributed sensor networks. In *Proceedings of the Tenth ACM Conference on Computer and Communications Security (CCS 2003)*, pages 62–72, October 2003.

[24] ZigBee Alliance. Zigbee specification. Technical Report Document 053474r06, Version 1.0, ZigBee Alliance, June 2005.

APPENDIX

A. MINISEC-U COUNTER RESYNCHRONIZATION

Reliable Unicast. If the message type is a reliable unicast, this implies there exists some type of acknowledgment protocol. Thus, the sender could use the presence of ACKs to determine if the packets had been received and authenticated successfully, which implies whether the counter is consistent between sender and receiver.

However, MiniSec is intended to be a network layer protocol. Since reliable messaging is typically implemented at a higher layer, this approach might not be appropriate. Nevertheless, we note that if reliable messaging is used, cooperation between the transport layer and MiniSec would be a viable solution.

Best Effort Unicast. Without support for reliable message delivery, TinyOS packets are best effort. In the case of unicast messages, where there only exists one receiver B, B can directly query sender A for the counter. Note that we use a nonce N_B to guarantee strong freshness.

$$\begin{aligned} B \rightarrow A: &\quad \langle N_B,\, \mathrm{MAC}_{K'_{AB}}(N_B)\rangle \\ A \rightarrow B: &\quad \langle C_A,\, \mathrm{MAC}_{K'_{AB}}(C_A||N_B)\rangle \end{aligned} \tag{1}$$

Since neither the nonce N_B nor the counter C_A are secrets, they can be sent in the clear. However, N_B needs to be authenticated to prevent a DOS attack, and C_A needs to be authenticated to prevent an attacker from injecting incorrect counter values. Any generic MAC function would be sufficient. Furthermore, we assume another pre-existing secret K'_{AB}, and there are numerous methods of bootstrapping this secret from the OCB encryption key.