SPINS: Security Protocols for Sensor Networks*

Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, J. D. Tygar

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

{perrig, szewczyk, vwen, culler, tygar}@cs.berkeley.edu

ABSTRACT

As sensor networks edge closer towards wide-spread deployment, security issues become a central concern. So far, the main research focus has been on making sensor networks feasible and useful, and less emphasis was placed on security.

We design a suite of security building blocks that are optimized for resource-constrained environments and wireless communication. SPINS has two secure building blocks: SNEP and μTESLA. SNEP provides the following important baseline security primitives: Data con£dentiality, two-party data authentication, and data freshness. A particularly hard problem is to provide ef£cient broadcast authentication, which is an important mechanism for sensor networks. μTESLA is a new protocol which provides authenticated broadcast for severely resource-constrained environments. We implemented the above protocols, and show that they are practical even on minimalistic hardware: The performance of the protocol suite easily matches the data rate of our network. Additionally, we demonstrate that the suite can be used for building higher level protocols.

1. INTRODUCTION

We envision a future where thousands to millions of small sensors form self-organizing wireless networks. How can we provide security for these sensor networks? Security is not easy; compared with conventional desktop computers, severe challenges exist — these sensors will have limited processing, storage, bandwidth, and energy.

Despite the challenges, security is important for these devices. As we describe below, we are deploying prototype wireless network sensors at UC Berkeley. These sensors measure environmental parameters and we are experimenting with having them control air conditioning and lighting systems. Serious privacy questions arise if third parties can read or tamper with sensor data. In the future, we envision wireless sensor networks being used for emergency and life-critical systems – and here the questions of security are foremost.

This paper presents a set of *Security Protocols for Sensor Networks*, SPINS. The chief contributions of this paper are:

Our main contributions include:

- Exploring the challenges for security in sensor networks.
- Designing and developing μTESLA (the "micro" version of the *Timed, Ef£cient, Streaming, Loss-tolerant Authentication Protocol*), providing authenticated streaming broadcast.
- Designing and developing SNEP (*Secure Network Encryption Protocol*) providing data con£dentiality, two-party data authentication, and data freshness, with low overhead.
- Designing and developing an authenticated routing protocol using SPINS building blocks

1.1 Sensor Hardware

At UC Berkeley, we are building prototype networks of small sensor devices under the SmartDust program [32]. We've deployed these in one of our EECS buildings, Cory Hall (see Figure 1). By design, these sensors are inexpensive, low-power devices. As a result, they have limited computational and communication resources. The sensors form a self-organizing wireless network and form a multihop routing topology. Typical applications may periodically transmit sensor readings for processing.

Our current prototype consists of *nodes*, small battery powered devices, that communicate with a more powerful *base station*, which in turn is connected to an outside network. As described below, the sensors form a self-organizing network (see Figure 1). Table 1 summarizes the performance characteristics of these devices. At 4MHz, they are slow and underpowered (the CPU has good support for bit and byte level I/O operations, but lacks support for many arithmetic and logic operations). They are only 8-bit processors (note that according to [40], 80% of all microprocessors shipped in 2000 were 4 bit or 8 bit devices). Communication is slow at 10 Kbps.

The operating system is particularly interesting for these devices. We use TinyOS [16]. This small, event-driven operating system consumes ab operating system consumes almost half of 8KB of instruction ¤ash memory, leaving just 4500 bytes for security and the application.

*We gratefully acknowledge funding support for this research. This research was sponsored in part the United States Postal Service (contract USPS 102592-01-Z-0236), by the United States Defense Advanced Research Projects Agency (contracts DABT63-98-C-0038, "Ninja", N66001-99-2-8913, "Endeavour", and DABT63-96-C-0056, "IRAM"), by the United States National Science Foundation (grants FD99-79852 and RI EIA-9802069) and from gifts and grants from the California MICRO program, Intel Corporation, IBM, Sun Microsystems, and Philips Electronics. DARPA Contract N66001-99-2-8913 is under the supervision of the Space and Naval Warfare Systems Center, San Diego. This paper represents the opinions of the authors and do not necessarily represent the opinions or policies, either expressed or implied, of the United States government, of DARPA, NSF, USPS, or any other of its agencies, or any of the other funding sponsors.

Mobile Computing and Networking 2001 Rome, Italy

CPU	8-bit, 4MHz
Storage	8KB instruction ¤ash
	512 bytes RAM
	512 bytes EEPROM
Communication	916 MHz radio
Bandwidth	10Kilobits per second
Operating System	TinyOS
OS code space	3500 bytes
Available code space	4500 bytes

Table 1: Characteristics of prototype SmartDust Nodes

It is hard to imagine how signi£cantly more powerful devices could be used without consuming large amounts of power. The energy source on our devices is a small battery, so we are stuck with relatively limited computational devices. Similarly, since communication over radio will be the most energy-consuming function performed by these devices, we need to minimize communications overhead. The limited energy supplies creates tensions for security: on the one hand, security needs to limit its consumption of processor power; on the other hand, limited power supply limits key lifetime (battery replacement is designed to reinitialize devices and zero out keys.) [1]

1.2 Is security on sensors possible?

These constraints make it impractical to use the majority of the current secure algorithms, which were designed for powerful workstations. For example, the working memory of a sensor node is insuf£cient to even hold the variables (of suf£cient length to ensure security) that are required in asymmetric cryptographic algorithms (*e.g.* RSA [35], Dif£e-Hellman [8]), let alone perform operations with them.

A particular challenge is broadcasting authenticated data to the entire sensor network. Current proposals for authenticated broadcast are impractical for sensor networks. First, most proposals rely on asymmetric digital signatures for the authentication, which are impractical for multiple reasons (*e.g.* long signatures with high communication overhead of 50-1000 bytes per packet, very high overhead to create and verify the signature).

Broadcast authentication is another problem. Even previously proposed purely symmetric solutions for broadcast authentication are impractical: Gennaro and Rohatgi's initial work required over 1 Kbyte of authentication information per packet [11], and Rohatgi's improved k-time signature scheme requires over 300 bytes per packet [36]. Some of the authors have also proposed the authenticated streaming broadcast *TESLA* protocol [31], and TESLA is ef£cient for the Internet with regular desktop workstations, but does not scale down to our resource-starved sensor nodes. In this paper, we extend and adapt TESLA such that it becomes practical for broadcast authentication for sensor networks. We call our new protocol μTESLA.

We've implemented all of these primitives, Our measurements show that adding security to a highly resource-constrained sensor network is feasible. The paper studies an authenticated routing protocol and a two-party key agreement protocol, and demonstrates that our security building blocks greatly facilitate the implementation of a complete security solution for a sensor network.

A common characteristic of sensor networks is their severely limited energy supply. Ultimately, the available energy determines the amount of computation, sensing, and communication a node can perform in its lifetime. Alternatively, the power harvested from the environment sets a bound on computation and communication per unit of time. In order to minimize the energy usage, a security subsystem should place minimal requirements on the processor, and add minimal information to each message transmitted. On the other hand, the limited lifespan of each node limits the life time of usable keys; we think of the battery replacement process as a rebirth.

Given the severe hardware and energy constraints, we must be careful in the choice of cryptographic primitives and the security protocols in the sensor networks.

[1]Note that base stations differ from nodes in having longer-lived energy supplies and having additional communications connections to outside networks.

2. SYSTEM ASSUMPTIONS

Before we outline the security requirements and present our security infrastructure, we need to de£ne our system architecture and the trust setup. The goal of this work is to propose a general security infrastructure that is applicable to a variety of sensor networks. Hence we chose a minimal hardware infrastructure as a basis for our design, such that SPINS can scale up to arbitrary sensor networks.

Sensor Hardware

The sensor nodes used in this design have the computational power and storage capacity comparable to that of the earliest PCs. The CPU is a RISC-like, 8-bit processor with 32 general purpose registers. This processor runs at a speed to 4MHz with the CPI of 2. The instruction set architecture is quite limited: it has a good support for bit- and byte-level I/O operations, but lacks support for many arithmetic and logic operations. The total amount of storage onboard is 8KB of instruction ¤ash, 512 bytes of data RAM and 512 bytes of EEPROM. Every node is equipped with a short-range, 916MHz ISM band radio with 10Kbps of bandwidth. Each node is running a small event-driven operating system called TinyOS [16]. A typical sensor network application establishes a multihop routing topology and periodically transmits unprocessed sensor readings. Such an application uses about 3500 bytes of code space for TinyOS, which leaves at most 4500 bytes for security and the application. As technology improves, we expect that sensor networks have devices with similar capability, but in a smaller form factor.

A common characteristic of sensor networks is their severely limited energy supply. Ultimately, the available energy determines the amount of computation, sensing, and communication a node can perform in its lifetime. Alternatively, the power harvested from the environment sets a bound on computation and communication per unit of time. In order to minimize the energy usage, a security subsystem should place minimal requirements on the processor, and add minimal information to each message transmitted. On the other hand, the limited lifespan of each node limits the life time of usable keys; we think of the battery replacement process as a rebirth.

Given the severe hardware and energy constraints, we must be careful in the choice of cryptographic primitives and the security protocols in the sensor networks.

Communication architecture

Generally, the sensor nodes communicate using RF, so broadcast is the fundamental communication primitive. The baseline protocols account for this property: on one hand it affects the trust assumptions, and on the other it is exploited to minimize the energy usage.

Figure 1 shows the organization of a typical SmartDust sensor network. The network forms around one or more *base stations*,

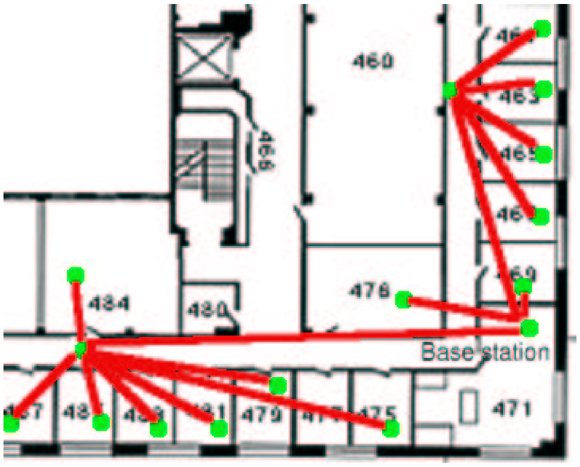

Figure 1: Communication organization within a sensor network. All messages are either destined for the base station or originate at the base station. The routes are discovered so that the number of hops is minimized and the reliability of each connection is maximized.

which interface the sensor network to the computing infrastructure. The sensor nodes establish a routing forest, with a base station at the root of every tree. Periodic transmission of beacons allows nodes to create a routing topology. Each node can forward a message towards a base station, recognize packets addressed to it, and handle message broadcasts. The base station accesses individual nodes using source routing. We assume that the base station has capabilities similar to the network nodes, except that it has enough battery power to surpass the lifetime of all sensor nodes, suf£cient memory to store cryptographic keys, and means for communicating with outside networks.

In the sensor applications developed so far, there has been limited local exchange and data processing. The communication patterns within our network fall into three categories:

- node to base station communication, *e.g.* sensor readings
- base station to node communication, *e.g.* speci£c requests
- base station to all nodes, *e.g.* routing beacons, queries or reprogramming of the entire network.

Our security goal is to address primarily these communication patterns, though we do show how to adapt our baseline protocols to other communication patterns, i.e. node to node or node broadcast.

Trust Setup

Generally, the sensor networks may be deployed in untrusted locations. While it is conceivable to guarantee the integrity of the each node through dedicated secure microcontrollers (*e.g.* [1] or [7]), we feel that such an architecture is too restrictive and does not generalize to the majority of sensor networks. Instead, we assume that individual sensors are untrusted. Our goal is to design the SPINS key setup such that a compromised sensor only compromises that sensor, and no other sensors of the network.

Wireless communication is fundamentally untrusted. Because of its broadcast nature any adversary can eavesdrop on the traf£c, and inject new messages or replay and change old messages. Hence, SPINS does not place any trust assumptions on the communication infrastructure, except that messages are delivered to the destination with non-zero probability.

Since the base station is the gateway for the nodes to communicate with the outside world, compromising the base station could render the entire sensor network useless. Thus the base stations are a necessary part of our trusted computing base. Our trust setup mimics this and so all sensor nodes intimately trust the base station: at creation time, each node is given a *master key* which is shared with the base station. All other keys are derived from this key.

Finally, each node trusts itself and its sensors. This assumption seems necessary to make any forward progress. In particular, we trust the local clock to be accurate, *i.e.* to have a small drift. This assumption is necessary for the authenticated broadcast protocol described below in Section 5.2.

Design guidelines

With the limited computation resources available on our platform, we cannot afford to use asymmetric cryptography and hence we use purely symmetric cryptographic primitives to construct the SPINS protocols. Due to the limited program store, we construct all cryptographic primitives (i.e. encryption, message authentication code (MAC), hash, random number generator) out of a single block cipher for code reuse. To reduce communication overhead we exploit common state between the communicating parties.

3. REQUIREMENTS FOR SENSOR NETWORK SECURITY

In this section, we formalize the security properties required by sensor networks, and show how they are directly applicable in a sample network deployed within a typical building.

Data Con£dentiality

A sensor network within an apartment should not leak sensor readings to the neighboring networks. In many applications (*e.g.* key distribution) the nodes communicate highly sensitive data. The standard solution to keep sensitive data secret is to encrypt the data with a secret key that only the intended receivers possess, hence achieving con£dentiality. Given the observed communication patterns, we use initially set up secure channels between nodes and base stations to bootstrap other secure channels, if necessary.

Data Authentication

Message authentication is of paramount importance for many applications in sensor networks. Within the building sensor network, authentication is necessary for many administrative tasks (*e.g.* network reprogramming or controlling sensor node duty cycle). At the same time, an adversary can easily inject messages, so the receiver needs to make sure that the data used in any decision-making process originates from the coerect source. Informally, *data authentication* allows the receiver to verify that the data really was sent by the claimed sender.

In the two-party communication case, data authentication can be achieved through a purely symmetric mechanism: The sender and the receiver share a secret key to compute a message authentication code (MAC) of all communicated data. When a message with a correct MAC arrives, the receiver knows that it must have been sent by the sender.

This style of authentication cannot be applied to a broadcast setting, without placing much stronger trust assumptions on the network nodes. If one sender wants to send authentic data to mutually untrusted receivers using a symmetric MAC is insecure: Any one of the receivers knows the MAC key, and hence could impersonate the sender and forge messages to other receivers. Hence, we need an asymmetric mechanism to achieve authenticated broadcast. Our contribution is to construct authenticated broadcast from symmetric primitives only, and introduce asymmetry with delayed key disclosure and one-way function key chains.

Data Integrity

In communication, *data integrity* ensures the receiver that the received data is not altered in transit by an adversary. In SPINS, we achieve data integrity through data authentication, which is a stronger property.

Data Freshness

Given that all sensor networks stream some forms of time varying measurements, it is not enough to guarantee con£dentiality and authentication; we also must ensure each message is *fresh*. Informally, data freshness implies that the data is recent, and it ensures that no adversary replayed old messages. We identify two types of freshness: weak freshness, which provides partial message ordering, but carries no delay information, and strong freshness, which provides a total order on a request-response pair, and allows for delay estimation. Weak freshness is required by sensor measurements, while strong freshness is useful for time synchronization within the network.

4. NOTATION

We use the following notation to describe security protocols and cryptographic operations in this paper.

A, B are principals, such as communicating nodes

N_A is a nonce generated by A (a nonce is an unpredictable bitstring, usually used to achieve freshness).

$M_1 \mid M_2$ denotes the concatenation of messages M_1 and M_2

K_{AB} denotes the secret (symmetric) key which is shared between A and B

$\{M\}_{K_{AB}}$ is the encryption of message M with the symmetric key shared by A and B.

$\{M\}_{\langle K_{AB},IV\rangle}$ denotes the encryption of message M, with key K_{AB}, and the initialization vector IV which is used in encryption modes such as cipher-block chaining (CBC), output feedback mode (OFB), or counter mode (CTR) [9, 21, 22].

In the following, when we refer to a **secure channel**, we mean a channel that offers con£dentiality, authenticity, integrity, and freshness.

5. SPINS SECURITY BUILDING BLOCKS

To achieve the security requirements we established in Section 3 we have designed and implemented two security building blocks: SNEP and μTESLA. SNEP provides data con£dentiality, two-party data authentication, integrity, and freshness. μTESLA provides authentication for data broadcast. We bootstrap the security for both mechanisms with a shared secret key between each node and the base station (see Section 2). We demonstrate in Section 8 how we can extend the trust to node-to-node interactions from the node-to-base-station trust.

5.1 SNEP: Data Con£dentiality, Authentication, Integrity, and Freshness

Our Sensor Network Encryption Protocol (SNEP) provides a number of unique advantages. First, it has low communication overhead since it only adds 8 bytes per message. Second, like many cryptographic protocols it uses a counter, but we avoid transmitting the counter value by keeping state at both end points. Third, SNEP achieves even semantic security, a strong security property which prevents eavesdroppers from inferring the message content from the encrypted message. Finally, the same simple and ef£cient protocol also gives us data authentication, replay protection, and weak message freshness.

Data con£dentiality is one of the most basic security primitives and it is used in almost every security protocol. A simple form of con£dentiality can be achieved through encryption, but pure encryption is not suf£cient. Another important security property is *semantic security*, which ensures that an eavesdropper has no information about the plaintext, even if it sees multiple encryptions of the same plaintext [12]. For example, even if an attacker has an encryption of a 0 bit and an encryption of a 1 bit, it will not help it distinguish whether a new encryption is an encryption of 0 or 1. The basic technique to achieve this is randomization: Before encrypting the message with a chaining encryption function (i.e. DES-CBC), the sender precedes the message with a random bit string. This prevents the attacker from inferring the plaintext of encrypted messages if it knows plaintext-ciphertext pairs encrypted with the same key.

In our resource-constrained environment, however, sending the randomized data over the RF channel requires more energy. So we construct another cryptographic mechanism that achieves semantic security with no additional sending overhead. Instead, we rely on a shared counter between the sender and the receiver which we use as an initialization vector (IV) for the block cipher in counter mode (CTR) discussed in Section 6. Since the communicating parties share the counter and increment it after each block, the counter does not need to be sent with the message. To achieve two-party authenticity and data integrity, we use a message authentication code (MAC).

The combination of these mechanisms form our Sensor Network Encryption Protocol SNEP. The encrypted data has the following format: $E = \{D\}_{\langle \mathcal{K}_{encr},C\rangle}$, where D is the data, the encryption key is $\mathcal{K}_{encr}$, and the counter C is the initialization vector (IV). The MAC is $M = \text{MAC}(\mathcal{K}_{mac}, C|E)$. We derive the keys $\mathcal{K}_{encr}$ and $\mathcal{K}_{mac}$ from the master secret key $\mathcal{K}$ as we show in Section 6. The complete message that A sends to B is:

$$A \rightarrow B : \{D\}_{\langle \mathcal{K}_{encr},C\rangle},\ \text{MAC}(\mathcal{K}_{mac}, C|\{D\}_{\langle \mathcal{K}_{encr},C\rangle})$$

SNEP offers the following nice properties:

- *Semantic security*: Since the counter value is incremented after each message, the same message is encrypted differently each time. The counter value is long enough that it never repeats within the lifetime of the node.
- *Data authentication*: If the MAC veri£es correctly, the receiver can be assured that the message originated from the claimed sender.
- *Replay protection*: The counter value in the MAC prevents replaying old messages. Note that if the counter were not present in the MAC, an adversary could easily replay messages.
- *Weak freshness*: If the message veri£ed correctly, the receiver knows that the message must have been sent after

the previous message it received correctly (that had a lower counter value). This enforces a message ordering and yields weak freshness.

- *Low communication overhead*: The counter state is kept at each end point and does not need to be sent in each message.[2]

Plain SNEP provides weak data freshness only, because it only enforces a sending order on the messages within node B, but no absolute assurance to node A that a message was created by B in response to an event in node A.

Node A achieves strong data freshness for a response from node B through a nonce N_A (which is a random number suf£ciently long such that it is unpredictable). Node A generates N_A randomly and sends it along with a request message R_A to node B. The simplest way to achieve strong freshness is for B to return the nonce with the response message R_B in an authenticated protocol. However, instead of returning the nonce to the sender, we can optimize the process by using the nonce implicitly in the MAC computation. The entire SNEP protocol providing strong freshness for B's response is:

$$A \rightarrow B : N_A, R_A$$
$$B \rightarrow A : \{R_B\}_{\langle \mathcal{K}_{encr}, C \rangle},\ \text{MAC}(\mathcal{K}_{mac}, N_A | C | \{R_B\}_{\langle \mathcal{K}_{encr}, C \rangle})$$

If the MAC veri£es correctly, node A knows that node B generated the response after it sent the request. The £rst message can also use plain SNEP if con£dentiality and authenticity are needed.

5.2 μTESLA: Authenticated Broadcast

Current proposals for authenticated broadcast are impractical for sensor networks. First, most proposals rely on asymmetric digital signatures for the authentication, which are impractical for multiple reasons. They require long signatures with high communication overhead of 50-1000 bytes per packet, very high overhead to create and verify the signature. Even previously proposed one-time signature schemes that are based on symmetric cryptography (one-way functions without trapdoors) have a high overhead: Gennaro and Rohatgi's broadcast signature based on Lamport's one-time signature [20] requires over 1 Kbyte of authentication information per packet [11], and Rohatgi's improved k-time signature scheme requires over 300 bytes per packet [36].

The recently proposed TESLA protocol provides ef£cient authenticated broadcast [31, 30]. However, TESLA is not designed for such limited computing environments as we encounter in sensor networks for three reasons.

First, TESLA authenticates the initial packet with a digital signature. Clearly, digital signatures are too expensive to compute on our sensor nodes, since even £tting the code into the memory is a major challenge. For the same reason as we mention above, one-time signatures are a challenge to use on our nodes.

Standard TESLA has an overhead of approximately 24 bytes per packet. For networks connecting workstations this is usually not signi£cant. Sensor nodes, however, send very small messages that are around 30 bytes long. It is simply impractical to disclose the TESLA key for the previous intervals with every packet: with 64 bit keys and MACs, the TESLA-related part of the packet would be constitute over 50% of the packet.

[2] In case the MAC does not match, the receiver can try out a £xed, small number of counter increments to recover from message loss. In case the optimistic re-synchronization fails, the two parties engage in a counter exchange protocol, which uses the strong freshness protocol described below.

Finally, the one-way key chain does not £t into the memory of our sensor node. So pure TESLA is not practical for a node to broadcast authenticated data.

We design μTESLA to solve the following inadequacies of TESLA in sensor networks:

- TESLA authenticates the initial packet with a digital signature, which is too expensive for our sensor nodes. μTESLA uses only symmetric mechanisms.
- Disclosing a key in each packet requires too much energy for sending and receiving. μTESLA discloses the key once per epoch.
- It is expensive to store a one-way key chain in a sensor node. μTESLA restricts the number of authenticated senders.

μTESLA Overview

In this subsection, we give a brief overview of μTESLA, the details are explained in the next subsection.

As we discussed in Section 3, authenticated broadcast requires an asymmetric mechanism, otherwise any compromised receiver could forge messages from the sender. Unfortunately, asymmetric cryptographic mechanisms have high computation, communication, and storage overhead, which makes their usage on resource-constrained devices impractical. μTESLA overcomes this problem by introducing asymmetry through a delayed disclosure of symmetric keys, which results in an ef£cient broadcast authentication scheme.

For simplicity, we explain μTESLA for the case where the base station broadcasts authenticated information to the nodes, and we discuss the case where the nodes are the sender at the end of this section.

μTESLA requires that the base station and the nodes are loosely time synchronized, and each node needs to know an upper bound on the maximum synchronization error. To send an authenticated packet, the base station simply computes a MAC on the packet with a key that is secret at that point in time. When the node gets the packet, it can verify that the corresponding MAC key was not yet disclosed by the base station (based on its loosely synchronized clock and its maximum synchronization error and because it knows the time schedule at which keys are disclosed). Since the receiving node is assured that the MAC key is only known by the base station, it is assured that no adversary could have altered the packet in transit. So the node stores the packet in a buffer. At the time of key disclosure, the base station broadcasts the veri£cation key to all receivers. When the node receives the disclosed key, it can easily verify the authenticity of the key (which we explain below). If the key is authentic, the node can now use it to authenticate the packet stored in its buffer.

Each MAC key is a key of a key chain, generated by a publically known one-way function F. To generate the one-way key chain, the sender chooses the last key K_n of the chain randomly, and repeatedly applies F to compute all other keys: $K_i = F(K_{i+1})$. Each node can easily perform time synchronization and retrieve an authenticated key of the key chain for the commitment in a secure and authenticated manner, using the SNEP building block. (We explain more details in the next subsection).

Example Figure 2 shows an example of μTESLA. Each key of the key chain corresponds to a time interval and all packets sent within one time interval are authenticated with the same key. The time until keys of a particular interval are disclosed is 2 time intervals in this example. We assume that the receiver node is loosely time synchronized and knows K_0 (a commitment to the key chain)

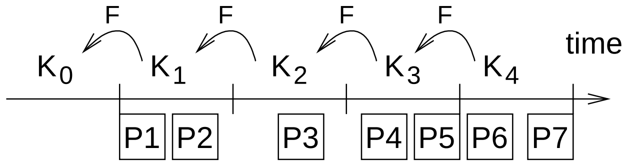

Figure 2: Using a Time-Released Key Chain for Source Authentication.

in an authenticated way. Packets P_1 and P_2 sent in interval 1 contain a MAC with key K_1. Packet P_3 has a MAC using key K_2. So far, the receiver cannot authenticate any packets yet. Let us assume that packets P_4, P_5, and P_6 are all lost, as well as the packet that discloses key K_1, so the receiver can still not authenticate P_1, P_2, or P_3. In interval 4 the base station broadcasts key K_2, which the node authenticates by verifying $K_0 = F(F(K_2))$, and hence knows also $K_1 = F(K_2)$, so it can authenticate packets P_1, P_2 with K_1, and P_3 with K_2.

Instead of adding a disclosed key to each data packet, the key disclosure is independent from the packets broadcast, and is tied to time intervals. Within the context of μTESLA, the sender broadcasts the current key periodically in a special packet.

μTESLA Detailed Description

μTESLA has multiple phases: Sender setup, sending authenticated packets, bootstrapping a new receiver, and authenticating packets. For simplicity, we explain μTESLA for the case where the base station broadcasts authenticated information, and we discuss the case where nodes send authenticated broadcasts at the end of this section.

Sender setup The sender £rst generates a sequence of secret keys (or key chain). To generate the one-way key chain of length n, the sender chooses the last key K_n randomly, and generates the remaining values by successively applying a one-way function F (*e.g.* a cryptographic hash function such as MD5 [34]): $K_j = F(K_{j+1})$. Because F is a one-way function, anybody can compute forward, *e.g.* compute $K_0, \dots, K_j$ given K_{j+1}, but nobody can compute backward, *e.g.* compute K_{j+1} given only $K_0, \dots, K_j$, due to the one-way generator function. This is similar to the S/Key one-time password system [14].

Broadcasting authenticated packets The time is divided into time intervals and the sender associates each key of the one-way key chain with one time interval. In time interval t, the sender uses the key of the current interval, K_t, to compute the message authentication code (MAC) of packets in that interval. The sender will then reveal the key K_t after a delay of δ intervals after the end of the time interval t. The key disclosure time delay δ is on the order of a few time intervals, as long as it is greater than any reasonable round trip time between the sender and the receivers.

Bootstrapping a new receiver The important property of the one-way key chain is that once the receiver has an authenticated key of the chain, subsequent keys of the chain are self-authenticating, which means that the receiver can easily and ef£ciently authenticate subsequent keys of the one-way key chain using the one authenticated key. For example, if the receiver has an authenticated value K_i of the key chain, the receiver can easily authenticate K_{i+1}, by verifying $K_i = F(K_{i+1})$. Therefore to bootstrap μTESLA, each receiver needs to have one *authentic key of the one-way key chain* as a commitment to the entire chain. Another requirement of μTESLA is that the sender and receiver are *loosely time synchronized*, and that the receiver knows the *key disclosure schedule* of the keys of the one-way key chain. Both the loose time synchronization as well as the authenticated key chain commitment can be established with a mechanism that provides strong freshness and point-to-point authentication. The receiver sends a nonce in the request message to the sender. The sender replies with a message containing its current time T_S (for time synchronization), a key K_i of the one-way key chain used in a past interval i (the commitment to the key chain), and the starting time T_i of interval i, the duration T_{int} of a time interval, and the disclosure delay δ (the last three values describe the key disclosure schedule).

$$M \to S : N_M$$
$$S \to M : T_S \mid K_i \mid T_i \mid T_{\text{int}} \mid \delta$$
$$\text{MAC}(K_{MS}, N_M \mid T_S \mid K_i \mid T_i \mid T_{\text{int}} \mid \delta)$$

Since we do not need con£dentiality, the sender does not need to encrypt the data. The MAC uses the secret key shared by the node and base station to authenticate the data, the nonce N_M allows the node to verify freshness. Instead of using a digital signature scheme as in TESLA, we use the node-to-base-station authenticated channel to bootstrap the authenticated broadcast.

Authenticating broadcast packets When the receiver receives the packets with the MAC, it needs to ensure that the packet could not have been spoofed by an adversary. The threat is that the adversary already knows the disclosed key of a time interval and so it could forge the packet since it knows the key used to compute the MAC. Hence the receiver needs to be sure that the sender did not disclose the key yet which corresponds to an incoming packet, which implies that no adversary could have forged the contents. This is called the *security condition*, which the receiver checks for all incoming packets. Therefore the sender and the receiver need to be loosely time synchronized and the receiver needs to know the key disclosure schedule. If the incoming packet satis£es the security condition the receiver stores the packet (it can verify it only once the corresponding key is disclosed). If the security condition is violated (the packet had an unusually long delay), the receiver needs to drop the packet, since an adversary might have altered it.

As soon as the node receives a key K_j of a previous time interval, it authenticates the key by checking that it matches the last authentic key it knows K_i, using a small number of applications of the one-way function F: $K_i = F^{j-i}(K_j)$. If the check is successful, the new key K_j is authentic and the receiver can authenticate all packets that were sent within the time intervals i to j. The receiver also replaces the stored K_i with K_j.

Nodes broadcast authenticated data New challenges arise if a node broadcasts authenticated data. Since the node is severely memory limited, it could not store the keys of a one-way key chain. Moreover, re-computing each key from the initial generating key K_n is computationally expensive. Another issue is that the node might not share a key with each receiver, hence sending out the authenticated commitment to the key chain would involve an expensive node-to-node key agreement, as we describe in Section 8. Finally, broadcasting the disclosed keys to all receivers can also be expensive on the node and drain precious battery energy.

Here are two viable approaches to deal with this problem:

- The node broadcasts the data through the base station. It uses SNEP to send the data in an authenticated way to the base station, which subsequently broadcasts it.

- The node broadcasts the data. However, the base station keeps the one-way key chain and sends keys to the broadcasting node as needed. To conserve energy for the broadcasting node, the base station can also broadcast the disclosed keys, and/or perform the initial bootstrapping procedure for new receivers.

6. IMPLEMENTATION

Due to the stringent resource constraints of the sensor nodes, the implementation of the cryptographic primitives is a major challenge. Usually for the sake of feasibility and ef£ciency, security is sacri£ced. Our belief, however, is that strong cryptography is necessary for trustworthy devices. Hence, one of our main goals is to provide strong cryptography despite the severe hardware restrictions.

A hard constraint is the memory size: Our sensor nodes have 8 KBytes of read-only program memory, and 512 bytes of RAM. The program memory is used for TinyOS, our security infrastructure, and the actual sensor net application. To save program memory we implement all cryptographic primitives from one single block cipher [22, 38].

Block Cipher. We evaluated several algorithms for use as a block cipher. An initial choice is the AES algorithm Rijndael [6], however, after closer inspection, we sought alternatives with smaller code size and higher speed. The baseline version of Rijndael uses over 800 bytes of lookup tables which was judged excessive for the constraints of our environment. An optimized version of that algorithm which runs about a 100 times faster, uses over 10 Kbytes of lookup tables. Similarly, we rejected the DES block cipher which requires a 512-entry SBox table, and a 256-entry table for various permutations [42]. We defer using other small encryption algorithms such as TEA [43] or TREYFER [44] until they matured after thorough scrutiny of cryptanalysts. We chose to use RC5 [33] because of the small code size and high ef£ciency. RC5 does not rely on multiplication, and does not require large tables. However, RC5 does use 32-bit data-dependent rotates, and our Atmel processor only has an 8-bit single bit rotate, which makes this operation expensive.

Even though the RC5 algorithm can be expressed very succinctly, the common RC5 libraries were signi£cantly too large to £t on our platform. It is apparent that in the regime of network sensors, compactness it often preferable to generality and ¤exibility. With a judicious selection of functionality, we were able to use a subset of RC5 from OpenSSL, and after further tuning of the code we achieve an additional 40% reduction in code size.

Encryption Function. To save code space, we use the same function both for encryption and decryption. The counter (CTR) mode of block ciphers, shown in Figure 3 has this property. Another property of the CTR mode is that it is a stream cipher in nature. Therefore the size of the ciphertext is exactly the size of the plaintext and not a multiple of the block size.[3] This property is particularly desirable in our environment. Message sending and receiving is very expensive in terms of energy. Also, longer messages have a higher probability of data corruption. Therefore, message expansion by the block cipher is undesirable. CTR mode requires a counter for proper operation. Reusing a counter value severely degrades security. In addition, CTR-mode offers us semantic security, since the same plaintext sent at different times is encrypted into different cyphertext. To an adversary that does not know the key, these messages will appear as two different, unrelated, random strings. This approach allows us to omit the explicit counter from the message, when the sender and receiver share the same counter value. If the two nodes lose the synchronization of the counter, they can simply transmit the counter explicitly to resynchronize using SNEP with strong freshness.

[3]The same property can also be achieved with a block cipher and the "ciphertext-stealing" method described by Schneier [38]. The downside is that this approach requires both encryption and decryption functions.

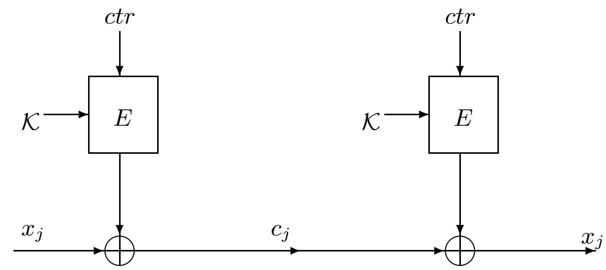

Figure 3: Counter mode encryption and decryption. The encryption function is applied to a monotonically increasing counter to generate a one time pad. This pad is then XORed with the plaintext. The decryption operation is identical.

Freshness Weak freshness is automatically provided by the CTR encryption. Since the sender increments the counter after each message, the receiver veri£es weak freshness by verifying that received messages have a monotonically increasing counter. For applications that require strong freshness, the node creates a random nonce N_M (a 64-bit value that is unpredictable) and sends in the request message to the receiver. The receiver generates the response message and includes the nonce in the MAC computation as we describe in Section 5.1. If the MAC of the response veri£es successfully, the node knows that the response was generated after it sent out the request message and hence achieves strong freshness.

Random-number generation. Although the node has its own sensors, radio receiver, and scheduling process, from which we could derive random digits, we chose to take the route of least power requirement and most ef£cient random number generation. We use a MAC function as our pseudo-random number generator (PRG), with the secret pseudo-random number generator key $\mathcal{K}_{\textsf{rand}}$. We also keep a counter $\mathcal{C}$ that we increment after each pseudo-random block we generate. We compute the $\mathcal{C}$-th pseudo-random output block as MAC$(\mathcal{K}_{\textsf{rand}}, \mathcal{C})$. If $\mathcal{C}$ wraps around (which should never happen because the node will exhaust its energy before then), we derive a new PRG key from the master secret key and the current PRG key using our MAC as a pseudo-random function (PRF): $\mathcal{K}'_{\textsf{rand}} =$ MAC$(\mathcal{K}, \mathcal{K}_{\textsf{rand}})$.

Message authentication. We also need a secure message authentication code. Because we intend to re-use our block cipher, we use the well-known CBC-MAC [41]. A block diagram for computing CBC MAC is shown in Figure 4.

To achieve authentication and message integrity we use the following standard approach. Assuming a message M, an encryption key $\mathcal{K}_{encr}$, and a MAC key $\mathcal{K}_{mac}$, we use the following construction: $\{M\}_{\mathcal{K}_{encr}},$ MAC$(\mathcal{K}_{mac}, \{M\}_{K_E})$. This construction prevents the nodes from decrypting erroneous ciphertext, which is a potential security risk.

In our implementation, we decided to compute a MAC per packet. This approach £ts well with the lossy nature of communications within this environment. Furthermore, at this granularity, MAC is used to check both authentication and integrity of messages, eliminating the need for mechanisms like CRC.

Key setup Recall that our key setup depends on a secret master key, initially shared by the base station and the node. We denote that key with $\mathcal{K}_i$ for node M_i. All keys subsequently needed are bootstrapped from the initial master secret key. Figure 5 shows

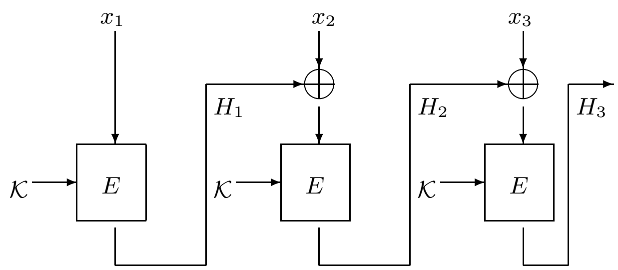

Figure 4: CBC MAC. The output of the last stage serves as the authentication code.

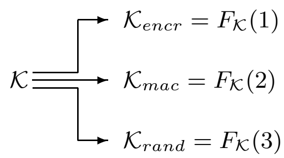

Figure 5: Deriving internal keys from the master secret key

our key derivation procedure. We use the pseudo-random function (PRF[4]) F to derive the keys, which we implement as $F_K(x) = \text{MAC}(K, x)$. Again, this allows for more code reuse. Since MAC has strong one-way properties, all keys derived in this manner are computationally independent. Even if the attacker could break one of the keys, the knowledge of that key would not help it to determine the master secret or any other key. Additionally, if we detect that a key has been compromised, both parties can derive a new key without transmitting any con£dential information.

7. EVALUATION

We evaluate the implementation of our protocols in terms of code size, RAM size, and processor and communication overheads.

Code size Table 2 shows the code size of three implementations of crypto routines in TinyOS. The smallest version of the crypto routines occupies about 20% of the available code space. Additionally, the implementation of μTESLA protocol uses another 574 bytes. Together, the crypto library and the protocol implementation consume about 2 kbytes of program memory, which is quite acceptable in most applications.

While optimizing the crypto library, it became apparent that at these scales it is not only important to identify the reusable routines, but also to identify these routines in such a way as to minimize the call setup costs. For example, OpenSSL implements the RC5 encryption routine as a function. In the case of a sensor network it became clear that the costs of call setup and return outweigh the costs of the RC5 itself. Thus, we made the decision to implement RC5 encrypt as a macro, and only expose interfaces to the MAC and CTR-ENCRYPT functions.

[4]If the adversary does not know the secret key of the function, it cannot predict a single bit of the input given an output of the function.

Version	Total Size	MAC	Encrypt	Key Setup
Smallest	1594	480	392	622
Fastest	1826	596	508	622
Original	2674	1210	802	686

Table 2: Code size breakdown (in bytes) for the security modules.

Operation	No. of Instructions
MAC (16 byte message)	600
Encrypt (16 byte message)	120
Key setup	8000

Table 3: Performance of security primitives in TinyOS.

Module	RAM size (bytes)
RC5	80
TESLA	120
Encrypt/MAC	20

Table 4: RAM requirements of the security modules.

Performance The performance of the cryptographic primitives is adequate for the bandwidth supported by the current generation of network sensors. Our sensors currently support a maximum throughput of twenty 30-byte messages per second, with the microcontroller being idle for about 50% of the time [16]. Assuming a single key setup, one MAC operation, and one encryption operation, our code is still able to encrypt and sign every message.

We infer the time required for μTESLA based on static analysis of the protocol. As stated in the previous section, μTESLA has a disclosure interval of 2. The stringent buffering requirements also dictate that the we cannot drop more that one key disclosure beacon. Thus, we require a maximum of two key setup operations and two CTR encryptions to check the validity of a disclosed TESLA key. Additionally, we perform up to two key setup operations, two CTR encryptions, and up to four MAC operation to check an integrity of a TESLA message.[5] That gives an upper bound of 17800 μs for checking the buffered messages. This amount of work is easily performed on our processor. In fact, the limiting factor on the bandwidth of authenticated broadcast traf£c is the amount of buffering we can dedicate on individual sensor nodes. Table 4 shows the amount of RAM that the security modules require. We con£gure the μTESLA protocol with 4 messages: the disclosure interval dictates a buffer space of 3 messages just for key disclosure, and we need an additional buffer to use this primitive in a more ¤exible way. Despite allocating minimal amounts of memory to μTESLA, the protocols we implement consume nearly half of the available RAM, and we do not feel that we can afford to dedicate any more RAM to security related tasks.

Energy Costs Finally we examine the energy costs of security mechanisms. Most of the energy costs will come from extra transmissions required by the protocols. Since we use a stream cipher for encryption, the size of encrypted message is the same as the size of the plaintext. The MAC uses 8 bytes of every 30 byte message, however, the MAC also achieves integrity so we do not need to use other message integrity mechanisms (e.g. a 16-bit CRC). Thus, encrypting and signing messages imposes an overhead of 6 bytes per message over an unecrypted message with integrity checking, or about 20 %. Figure 6 expresses the costs of computation and

[5]Key setup operations are dependent on the minimal and maximal disclosure interval, whereas the number of MAC operations depends on the number of buffered messages.

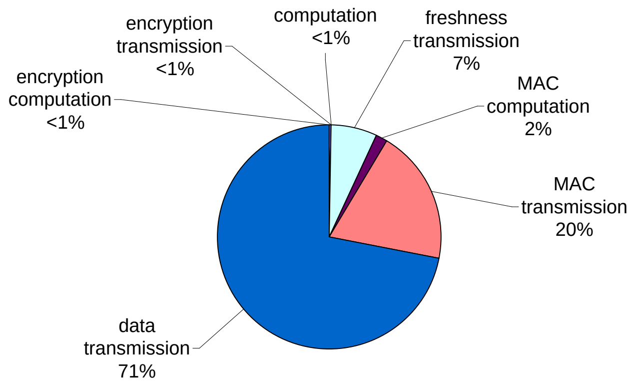

Figure 6: Energy costs of adding security protocols to the sensor network. Most of the overhead arises from the transmission of extra data rather than from any computational costs.

communication in terms of energy required for the SNEP protocol.

The messages broadcast using μTESLA have the same costs of authentication per message. Additionally, μTESLA requires a periodic key disclosure, but these messages are grafted onto routing updates (see Section 8). We can take two different views regarding the costs of these messages. If we accept that the routing beacons are necessary, then μTESLA key disclosure is nearly free, because energy of transmitting or receiving dominate the computational costs of our protocols. On the other hand, one might claim that the routing beacons are not necessary and that it is possible to construct an *ad hoc* multihop network implicitly. In that case the overhead of key disclosure would be 1 message per time interval, regardless of the traf£c pattern within the network. We believe that the bene£t of authenticated routing justi£es the costs of explicit beacons.

Remaining Security Issues Although this protocol suite addresses many security related problems, there remain many additional issues. First, we do not address the problem of information leakage due to covert channels. An attacker could gain information by eavesdropping encrypted messages if the communication protocol is poorly designed. Second, we do not deal completely with compromised sensors, we merely ensure that compromising a single sensor doesn't not reveal the keys of all the sensors in the network. It is an interesting research problem on how to design ef£cient protocols that scale down to sensor networks which are robust to compromised sensors. Third, we do not deal with denial-of-service (DoS) attacks in this work. Since we operate on a wireless network, an adversary can always perform a DoS attack by jamming the radio channel with a strong signal. Finally, due to our hardware limitations, we cannot provide Dif£e-Hellman style key agreement or use digital signatures to achieve non-repudiation. We believe that for the majority of sensor network applications, authentication is suf£cient.

8. APPLICATIONS

In this section we demonstrate how we can build secure protocols out of the SPINS secure building blocks. First, we build an authenticated routing application, and second, a two-party key agreement protocol.

8.1 Authenticated Routing

Using the μTESLA protocol, we developed a lightweight, authenticated ad-hoc routing protocol that builds an authenticated routing topology. Ad hoc routing has been an active area of research [5, 13, 17, 18, 26, 29, 28, 37]. However, none of these solutions offer authenticated routing messages. Hence it is potentially easy for a malicious user to take over the network by injecting erroneous, replaying old, or advertise incorrect routing information. The authenticated routing scheme we developed mitigates these problems.

The routing scheme within our prototype network assumes bidirectional communication channels, i.e. if node A hears node B, then node B hears node A. The route discovery depends on periodic broadcast of beacons. Every node, upon reception of a beacon packet, checks whether it has already received a beacon (which is a normal packet with a globally unique sender ID and current time at basestation, protected by MAC to ensure authenticity) in the current epoch [6]. If a node hears the beacon within the epoch, it does not take any further action. Otherwise, the node accepts the sender of the beacon as its parent to route towards the base station. Additionally, the node would repeat the beacon with the sender ID changed to itself. This route discovery resembles a distributed, breadth £rst search algorithm, and produces a routing topology similar to Figure 1 (see [16] for details).

However, in the above algorithm, the route discovery depends only on the receipt of route packet, not on its contents. It is easy for any node to claim to be a valid base station. We note that the μTESLA key disclosure packets can easily function as routing beacons. We accept only the sources of authenticated beacons as valid parents. Reception of a μTESLA packet guarantees that that packet originated at the base station, and that it is fresh. For each time interval, we accept as the parent the £rst node that sends a packet that is later successfully authenticated. Combining μTESLA key disclosure with the distribution of routing beacons allows us to charge the costs of the transmission of the keys to network maintenance, rather than the encryption system.

This scheme leads to a lightweight authenticated routing protocol. Since each node accepts only the £rst authenticated packet as the one to use in routing, it is impossible for an attacker to reroute arbitrary links within the sensor network. Furthermore, each node can easily verify whether the parent forwarded the message: by our assumption of bidirectional connectivity, if the parent of a node forwarded the message, the node must have heard that.

The authenticated routing scheme above is just one way to build authenticated ad hoc routing protocol using μTESLA. In protocols where base stations are not involved in route construction, μTESLA can still be used for security. In these cases, the initiating node will temporarily act as base station and beacons authenticated route updates [7].

8.2 Node-to-Node Key Agreement

A convenient method to bootstrap secure connections is to use public-key cryptography protocols for symmetric-key setup [2, 15]. Unfortunately, our resource-constrained sensor nodes prevent us from using computationally expensive public-key cryptography. Therefore, we need to construct our protocols solely from symmetric-key algorithms. Hence we design a symmetric protocol that uses the base station as a trusted agent for key setup.

Assume that the node A wants to establish a shared secret session key SK_{AB} with node B. Since A and B do not share any secrets, they need to use a trusted third party S, which is the base

[6] By epoch, we meant the interval of a routing updates.

[7] However, the node here will need to have signi£cantly more memory resource than the sensor nodes we explored here in order to store the key chain.

station in our case. In our trust setup, both A and B share a secret key with the base station, K_{AS} and K_{BS}, respectively. The following protocol achieves secure key agreement as well as strong key freshness:

$$A \rightarrow B : N_A, A$$
$$B \rightarrow S : N_A, N_B, A, B, \text{MAC}(K_{BS}, N_A|N_B|A|B)$$
$$S \rightarrow A : \{SK_{AB}\}_{K_{AS}}, \text{MAC}(K_{AS}, N_A|B|\{SK_{AB}\}_{K_{AS}})$$
$$S \rightarrow B : \{SK_{AB}\}_{K_{BS}}, \text{MAC}(K_{BS}, N_B|A|\{SK_{AB}\}_{K_{BS}})$$

The protocol uses our SNEP protocol with strong freshness. The nonces N_A and N_B ensure strong key freshness to both A and B. The SNEP protocol is responsible to ensure con£dentiality of the established session key SK_{AB}, as well as message authenticity to make sure that the key was really generated by the base station. Note that the MAC in the second protocol message helps defend the base station from denial-of-service attacks (DoS), so the base station only sends two messages to A and B if it received a legitimate request from one of the nodes.

A nice feature of the above protocol is that the base station performs most of the transmission work. Other protocols usually involve a ticket that the server sends to one of the parties which forwards it to the other node, which requires more energy for the nodes to forward the message.

The Kerberos key agreement protocol achieves similar properties, except that it does not provide strong key freshness [19, 23]. However, it would be straightforward to implement it with strong key freshness by using SNEP with strong freshness.

9. RELATED WORK

We review related work that deals with security issues in a ubiquitous computing environment. We also review work on cryptographic protocols for low-end devices.

Fox and Gribble present a security protocol that provides secure access to application-level proxied services [10]. Their protocol is designed to interact with a proxy to Kerberos and to facilitate porting services that rely on Kerberos to wireless devices. The work of Patel and Crowcroft focuses on security solutions for mobile user devices [27]. Unfortunately, their work uses asymmetric cryptography and is hence too expensive for the environments we envision. The work of Czerwinski et al. also relies on asymmetric cryptography for authentication [4]. Stajano and Anderson discuss the issues of bootstrapping security devices [39]. Their solution requires physical contact of the new device with a master device to imprint the trusted and secret information. Zhou and Hass propose to secure ad-hoc networks using asymmetric cryptography [45]. Carman, Kruus, and Matt analyze a wide variety of approaches for key agreement and key distribution in sensor networks [3]. They analyze the overhead of these protocols on a variety of hardware platforms.

A number of researchers investigate the problem to provide cryptographic services in low-end devices. We £rst discuss the hardware efforts, followed by the algorithmic work on cryptography. Several systems integrate cryptographic primitives with low cost microcontrollers. Examples of such systems are secure AVR controllers [1], the Fortezza government standard, and the Dallas iButton [7]. These systems support primitives for public key encryption, with instructions for modular exponentiation, and attempt to zeroize their memory if tampering is detected. Using such microcontrollers would make sensor networks more secure, but it would also introduce several dif£culties. First, the high price may be a deterrent for wide-spread adoption. Also, despite the hardware acceleration for asymmetric cryptography, these operations are still computationally and energy intensive.

On the cryptographic algorithm front for low-end devices the majority of research focuses on symmetric cryptography. A notable exception is the work of Modadugu, Boneh, and Kim which of¤oad the heavy computation for £nding an RSA keypair to untrusted servers [24].

Symmetric encryption algorithms seem to be inherently well suited for low-end devices, due to their relatively low overhead. In practice, however, low-end microprocessors are only 4-bit or 8-bit, and do not provide (ef£cient) multiplication or variable rotate/shift instructions. Hence many symmetric ciphers were too expensive to implement on our target platform. Even though one of the goals for the Advanced Encryption Standard (AES) [25] was ef£ciency and small code size on low-end processors, the chosen Rijndael block cipher [6] is nevertheless too expensive for our platform, either in terms of speed or code size. Due to our severely limited code size, we chose to use RC5 by Ron Rivest [33]. Algorithms such as TEA by Wheeler and Needham [43] or TREYFER by Yuval [44] would be smaller alternatives, but we still choose RC5 to attain high security because the security of these other ciphers is not yet thoroughly analyzed.

10. CONCLUSION

We have successfully demonstrated the feasibility of implementing a security subsystem for an extremely limited sensor network platform. We have identi£ed and implemented useful security protocols for sensor networks: Authenticated and con£dential communication, and authenticated broadcast. To illustrate the utility of our security building blocks, we implemented an authenticated routing scheme and a secure node-to-node key agreement protocol.

Many elements of our design are universal and apply easily to other sensor networks. Since our primitives are solely based on fast symmetric cryptography, and use no asymmetric algorithms, our building blocks are applicable to a wide variety of device con£gurations. The computation costs of symmetric cryptography are low. Even on our limited platform the energy spent for security is negligible compared with the energy cost of sending or receiving messages. In the absence of other constraints, it should be possible to encrypt and authenticate all sensor readings.

The communication costs are also small. Since the authentication, freshness, and con£dentiality require transmitting a mere 8 bytes per unit, it is feasible to guarantee these properties on a per packet basis, even with small 30 byte packets. It is dif£cult to improve on this scheme, as transmitting a MAC is fundamental to guaranteeing authenticity.

Certain elements of the design were in¤uenced by the available experimental platform. The choice of RC5 as our cryptographic primitive falls into this category; on a more powerful platform we could use any number of shared key algorithms with equal success. The extreme emphasis on code reuse is another property forced by our platform. A more powerful device would also allow for more basic modes of authentication. The main limitation of our platform was the amount of available memory. In particular, the buffering restrictions limited the effective bandwidth of authenticated broadcast.

Despite the shortcomings of our target platform, we were able to demonstrate a security subsystem for the prototype sensor network. With our techniques, we believe that security systems can become an integral part of practical sensor networks.

11. ACKNOWLEDGMENTS

We would like to thank Dawn Song and David Wagner for helpful discussions and comments. We would also like to thank the anonymous referees for their comments.

12. REFERENCES

[1] Secure Microcontrollers for SmartCards. `http://www.atmel.com/atmel/acrobat/1065s.pdf`.

[2] Steven Bellovin and Michael Merrit. Augmented encrypted key exchange: a password-based protocol secure against dictionary atttacks and password £le compromise. In *First ACM Conference on Computer and Communications Security CCS-1*, pages 244–250, 1993.

[3] David W. Carman, Peter S. Kruus, and Brian J. Matt. Constraints and approaches for distributed sensor network security. *NAI Labs Technical Report #00-010*, September 2000.

[4] Steven E. Czerwinski, Ben Y. Zhao, Todd D. Hodes, Anthony D. Joseph, and Randy H. Katz. An architecture for a secure service discovery service. In *Fifth annual ACM/IEEE international conference on Mobile computing and networking*, pages 24 – 35, Seattle, WA USA, August 1999.

[5] D. Johnson and D.A. Maltz and J. Broch. The dynamic source routing protocol for mobile ad hoc networks (internet-draft). In *Mobile Ad-hoc Network (MANET) Working Group, IETF*, October 1999.

[6] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael, March 1999.

[7] iButton: A Java-Powered Cryptographic iButton. `http://www.ibutton.com/ibuttons/java.html`.

[8] W. Dif£e and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22:644–654, November 1976.

[9] Whit£eld Dif£e and Martin E. Hellman. Privacy and authentication: An introduction to cryptography. *Proceedings of the IEEE*, 67(3):397–427, March 1979.

[10] Armando Fox and Steven D. Gribble. Security on the move: indirect authentication using kerberos. In *Second annual international conference on Mobile computing and networking (MOBICOM 1996)*, pages 155–164, White Plains, NY USA, November 1996.

[11] R. Gennaro and P. Rohatgi. How to sign digital streams. In Burt Kaliski, editor, *Advances in Cryptology - Crypto '97*, pages 180–197, Berlin, 1997. Springer-Verlag. Lecture Notes in Computer Science Volume 1294.

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

[13] Z.J. Haas and M. Perlman. The zone routing protocol (zrp) for ad hoc networks (internet-draft). 1998.

[14] Neil M. Haller. The S/KEY one-time password system. In *ISOC*, 1994.

[15] D. Harkins and D. Carrel. The internet key exchange (IKE). Request for Comments 2409, Information Sciences Institute, University of Southern California, November 1998.

[16] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, November 2000.

[17] D.B. Johnson and D.A. Maltz. Dynamic source routing in ad-hoc wireless networks. In *Mobile Computing*, 1996.

[18] Young-Bae Ko and Nitin Vaidya. Location-aided routing (lar) in mobile ad hoc networks. In *Proceedings of the Fourth International Conference on Mobile Computing and Networking (MobiCom'98)*, October 1998.

[19] J. Kohl and C. Neuman. RFC 1510: The Kerberos Network Authentication Service (V5), September 1993. Status: PROPOSED STANDARD.

[20] L. Lamport. Constructing digital signatures from a one-way function. Technical Report CSL-98, SRI International, October 1979.

[21] H. Lipmaa, P. Rogaway, and D. Wagner. Counter mode encryption. `http://csrc.nist.gov/encryption/modes/`.

[22] Alfred J. Menezes, Paul van Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[23] S. P. Miller, C. Neuman, J. I. Schiller, and J. H. Saltzer. Kerberos authentication and authorization system. In *Project Athena Technical Plan*, page section E.2.1, 1987.

[24] N. Modadugu, D. Boneh, and M. Kim. Generating rsa keys on a handheld using an untrusted server. In *RSA 2000*, 2000.

[25] NIST. Advanced encryption standard (aes) development effort. `http://csrc.nist.gov/encryption/aes/`, October 2000.

[26] V.D. Park and M.S. Corson. A highly adaptable distributed routing algorithm for mobile wireless networks. In *IEEE INFOCOMM'97*, 1997.

[27] Bhrat Patel and Jon Crowcroft. Ticket based service access for the mobile user. In *Third annual ACM/IEEE international conference on Mobile computing and networking*, pages 223–233, Budapest Hungary, September 1997.

[28] C.E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *ACM SIGCOMM Symposium on Communication, Architectures and Applications*, 1994.

[29] C.E. Perkins and E.M. Royer. Ad hoc on-demand distance vector routing. In *IEEE WMCSA'99*, February 1999.

[30] Adrian Perrig, Ran Canetti, Dawn Song, and Doug Tygar. Ef£cient and secure source authentication for multicast. In *Network and Distributed System Security Symposium, NDSS '01*, February 2001.

[31] Adrian Perrig, Ran Canetti, J.D. Tygar, and Dawn Xiaodong Song. Ef£cient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, May 2000.

[32] K. S. J. Pister, J. M. Kahn, and B. E. Boser. Smart dust: Wireless networks of millimeter-scale sensor nodes, 1999.

[33] R. L. Rivest. The RC5 encryption algorithm. *Proc. 1st Workshop on Fast Software Encryption*, pages 86–96, 1995.

[34] Ronald L. Rivest. The MD5 message-digest algorithm. Internet Request for Comments, April 1992. RFC 1321.

[35] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[36] Pankaj Rohatgi. A compact and fast hybrid signature scheme for multicast packet authentication. In *6th ACM Conference on Computer and Communications Security*, November 1999.

[37] S. Marti and T. Giuli and K. Lai and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of Mobicom 2000*, August 2000.

[38] Bruce Schneier. *Applied Cryptography (Second Edition)*. John Wiley & Sons, 1996.

[39] Frank Stajano and Ross Anderson. The resurrecting duckling: Security issues for ad-hoc wireless networks. In

B. Christianson, B. Crispo, and M. Roe, editors, *Security Protocols, 7th International Workshop*. Springer Verlag Berlin Heidelberg, 1999.

[40] David Tennenhouse. Embedding the Internet: proactive computing. *Communications of the ACM*, 43(5):43–43, 2000.

[41] U. S. National Institute of Standards and Technology (NIST). DES model of operation. Federal Information Processing Standards Publication 81 (FIPS PUB 81).

[42] U. S. National Institute of Standards and Technology (NIST). Data Encryption Standard (DES). Draft Federal Information Processing Standards Publication 46-3 (FIPS PUB 46-3), January 1999.

[43] David Wheeler and Roger Needham. Tea, a tiny encryption algorithm. `http://www.ftp.cl.cam.ac.uk/ftp/papers/djw-rmn/djw-rmn-tea.html`, November 1994.

[44] Gideon Yuval. Reinventing the travois: encryption/mac in 30 rom bytes. In *Proc. 4th Workshop on Fast Software Encryption*, 1997.

[45] L. Zhou and Z.J. Hass. Securing ad hoc networks. 13(6), November/December 1999.