Wireless Networks 8, 521–534, 2002

SPINS: Security Protocols for Sensor Networks

ADRIAN PERRIG, ROBERT SZEWCZYK, J.D. TYGAR, VICTOR WEN and DAVID E. CULLER
Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, 387 Soda Hall, Berkeley, CA 94720, USA

Abstract. Wireless sensor networks will be widely deployed in the near future. While much research has focused on making these networks feasible and useful, security has received little attention. We present a suite of security protocols optimized for sensor networks: SPINS. SPINS has two secure building blocks: SNEP and μTESLA. SNEP includes: data confidentiality, two-party data authentication, and evidence of data freshness. μTESLA provides authenticated broadcast for severely resource-constrained environments. We implemented the above protocols, and show that they are practical even on minimal 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.

Keywords: secure communication protocols, sensor networks, mobile ad hoc networks, MANET, authentication of wireless communication, secrecy and confidentiality, cryptography

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 power, storage, bandwidth, and energy.

We need to surmount these challenges, because security is so important. Sensor networks will expand to fill all aspects of our lives. Here are some typical applications:

- *Emergency response information*: sensor networks will collect information about the status of buildings, people, and transportation pathways. Sensor information must be collected and passed on in meaningful, secure ways to emergency response personnel.
- *Energy management*: in 2001 power blackouts plagued California. Energy distribution will be better managed when we begin to use remote sensors. For example, the power load that can be carried on an electrical line depends on ambient temperature and the immediate temperature on the wire. If these were monitored by remote sensors and the remote sensors received information about desired load and current load, it would be possible to distribute load better. This would avoid circumstances where Californians cannot receive electricity while surplus electricity exists in other parts of the country.
- *Medical monitoring*: we envision a future where individuals with some types of medical conditions receive constant monitoring through sensors that monitor health conditions. For some types of medical conditions, remote sensors may apply remedies (such as instant release of emergency medication to the bloodstream).
- *Logistics and inventory management*: commerce in America is based on moving goods, including commodities from locations where surpluses exist to locations where needs exist. Using remote sensors can substantially improve these mechanisms. These mechanisms will vary in scale – ranging from worldwide distribution of goods through transportation and pipeline networks to inventory management within a single retail store.
- *Battlefield management*: remote sensors can help eliminate some of the confusion associated with combat. They can allow accurate collection of information about current battlefield conditions as well as giving appropriate information to soldiers, weapons, and vehicles in the battlefield.

At UC Berkeley, we think these systems are important, and we are starting a major initiative to explore the use of wireless sensor networks. (More information on this new initiative, CITRIS, can be found at `www.citris.berkeley.edu`.) Serious security and privacy questions arise if third parties can read or tamper with sensor data. We envision wireless sensor networks being widely used – including for emergency and life-critical systems – and here the questions of security are foremost.

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

- Exploring the challenges for security in sensor networks.
- Designing and developing μTESLA (the "micro" version of TESLA), providing authenticated streaming broadcast.
- Designing and developing SNEP (*Secure Network Encryption Protocol*) providing data confidentiality, two-party data authentication, and data freshness, with low overhead.
- Designing and developing an authenticated routing protocol using our building blocks.

1.1. Sensor hardware

At UC Berkeley, we are building prototype networks of small sensor devices under the SmartDust program [45], one of the components of CITRIS. We have deployed these in one of

Table 1
Characteristics of prototype SmartDust nodes.

CPU	8-bit, 4 MHz
Storage	8 Kbytes instruction flash
	512 bytes RAM
	512 bytes EEPROM
Communication	916 MHz radio
Bandwidth	10 Kbps
Operating system	TinyOS
OS code space	3500 bytes
Available code space	4500 bytes

our EECS buildings, Cory Hall. We are currently using these for a very simple application – heating and air-conditioning control in the building. However, the same mechanisms that we describe in this paper can be modified to support sensor that handle emergency system such as fire, earthquake, and hazardous material response.

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. Table 1 summarizes the performance characteristics of these devices. At 4 MHz, 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 some logic operations). They are only 8-bit processors (note that according to [53], 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 [23]. This small, event-driven operating system consumes almost half of 8 Kbytes of instruction flash memory, leaving just 4500 bytes for security and the application.

It is hard to imagine how significantly 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. Wireless communication is the most energy-consuming function performed by these devices, so we need to minimize communications overhead. The limited energy supplies create tensions for security: on the one hand, security needs to limit its consumption of processor power; on the other hand, limited power supply limits the lifetime of keys (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 most current secure algorithms, since they were designed for powerful processors. For example, the working memory of a sensor node is not sufficient to even hold the variables for asymmetric cryptographic algorithms (e.g., RSA [48] with 1024 bits), 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. 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). Furthermore, 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 [17], and Rohatgi's improved k-time signature scheme requires over 300 bytes per packet [49]. Some of the authors of this article have also proposed the authenticated streaming broadcast *TESLA* protocol [43]. TESLA works well on regular desktop workstations, but uses too much communication and memory on our resource-starved sensor nodes. This article extends and adapts TESLA to make it practical for broadcast authentication for sensor networks. We call our new protocol μTESLA.

We have implemented all of these primitives. Our measurements show that adding security to a highly resource-constrained sensor network is feasible.

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.

2. System assumptions

Before we outline the security requirements and present our security infrastructure, we need to define the system architecture and the trust requirements. The goal of this work is to propose a general security infrastructure that is applicable to a variety of sensor networks.

2.1. Communication architecture

Generally, the sensor nodes communicate over a wireless network, so broadcast is the fundamental communication primitive. The baseline protocols account for this property: on one hand they affect the trust assumptions, and on the other they minimize energy usage.

A typical SmartDust sensor network forms around one or more *base stations*, which interface the sensor network to the outside network. 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 sufficient battery power to surpass the lifetime of all sensor nodes, sufficient memory to store cryptographic keys, and means for communicating with outside networks.

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

We do have an advantage with sensor networks, because most communication involves the base station and is not between two local nodes. 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., specific requests.
- Base station to all nodes, e.g., routing beacons, queries or reprogramming of the entire network.

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

2.2. Trust requirements

Generally, the sensor networks may be deployed in untrusted locations. While it may be possible to guarantee the integrity of the each node through dedicated secure microcontrollers (e.g., [1] or [13]), 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 so a compromise of a node does not spread to other nodes.

Basic wireless communication is not secure. Because it is broadcast, any adversary can eavesdrop on traffic, inject new messages, and replay old messages. Hence, our protocols do 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 can render the entire sensor network useless. Thus the base stations are a necessary part of our trusted computing base. Our trust setup reflects this and so all sensor nodes intimately trust the base station: at creation time, each node gets a *master secret key* $\mathcal{X}$ which it shares with the base station. All other keys are derived from this key, as we show in section 6.

Finally, each node trusts itself. This assumption seems necessary to make any forward progress. In particular, we trust the local clock to be accurate, i.e. to have small drift. This is necessary for the authenticated broadcast protocol we describe in section 5.

2.3. Design guidelines

With the limited computation resources available on our platform, we cannot afford to use asymmetric cryptography and so we use 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

This section formalizes the security properties required by sensor networks, and shows how they are directly applicable in a typical sensor network.

3.1. Data confidentiality

A sensor network should not leak sensor readings to neighboring networks. In many applications (e.g., key distribution) nodes communicate highly sensitive data. The standard approach for keeping sensitive data secret is to encrypt the data with a secret key that only intended receivers possess, hence achieving confidentiality. Given the observed communication patterns, we set up secure channels between nodes and base stations and later bootstrap other secure channels as necessary.

3.2. Data authentication

Message authentication is important for many applications in sensor networks (including administrative tasks such as network reprogramming or controlling sensor node duty cycle). Since an adversary can easily inject messages, the receiver needs to ensure that data used in any decision-making process originates from a trusted source. Informally, *data authentication* allows a receiver to verify that the data really was sent by the claimed sender. Informally, *data authentication* allows a 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. One of our contributions is to construct authenticated broadcast from symmetric primitives only, and introduce asymmetry with delayed key disclosure and one-way function key chains.

3.3. 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.

3.4. Data freshness

Sensor networks send measurements over time, so it is not enough to guarantee confidentiality 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 useful for 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 article:

- A, B are principals, such as communicating nodes.
- N_A is a nonce generated by A (a nonce is an unpredictable bit string, usually used to achieve freshness).
- $\mathcal{X}_{AB}$ denotes the master secret (symmetric) key which is shared between A and B. No direction information is stored in this key, so we have $\mathcal{X}_{AB} = \mathcal{X}_{BA}$.
- K_{AB} and K_{BA} denote the secret encryption keys shared between A and B. A and B derive the encryption key from the master secret key $\mathcal{X}_{AB}$ based on the direction of the communication: $K_{AB} = F_{\mathcal{X}_{AB}}(1)$ and $K_{BA} = F_{\mathcal{X}_{AB}}(3)$, where F is a Pseudo-Random Function (PRF) [18].[2] We describe the details of key derivation in further detail in section 6.
- K'_{AB} and K'_{BA} denote the secret MAC keys shared between A and B. A and B derive the encryption key from the master secret key $\mathcal{X}_{AB}$ based on the direction of the communication: $K'_{AB} = F_{\mathcal{X}_{AB}}(2)$ and $K'_{BA} = F_{\mathcal{X}_{AB}}(4)$, where F is a pseudo-random function.
- $\{M\}_{K_{AB}}$ is the encryption of message M with the encryption key K_{AB}.
- $\{M\}_{\langle K_{AB}, IV\rangle}$ 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) [3, 14,29].
- $\mathsf{MAC}(K'_{AB}, M)$ denotes the computation of the message authentication code (MAC) of message M, with MAC key K'_{AB}.

By a *secure channel*, we mean a channel that offers confidentiality, data authentication, integrity, and freshness.

5. SPINS security building blocks

To achieve the security requirements we established in section 3 we design two security building blocks: SNEP and μTESLA. SNEP provides data confidentiality, 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 confidentiality, authentication, integrity, and freshness

SNEP provides a number of unique advantages. First, it has low communication overhead; 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 semantic security, a strong security property which prevents eavesdroppers from inferring the message content from the encrypted message (see discussion below). Finally, the same simple and efficient protocol also gives us data authentication, replay protection, and weak message freshness.

Data confidentiality is one of the most basic security primitives and it is used in almost every security protocol. A simple form of confidentiality can be achieved through encryption, but pure encryption is not sufficient. 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 [19]. 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. A 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.

Sending the randomized data over a wireless channel, however, requires more energy. So we construct another cryptographic mechanism that achieves semantic security with no additional transmission overhead. We use two counters shared by the parties (one for each direction of communication) for the block cipher in counter mode (CTR) (as we discuss in section 6). A traditional approach to manage the counters is to send the counter along with each message. But since we are using sensors and the communicating parties share the counter and increment it after each block, the sender can save energy by sending the message without the counter. At the end of this section we describe a counter exchange protocol, which the communicating parties use to synchronize (or resynchronize) their counter values. To achieve two-party authentication and data integrity, we use a message authentication code (MAC).

A good security design practice is not to reuse the same cryptographic key for different cryptographic primitives; this prevents any potential interaction between the primitives that might introduce a weakness. Therefore we derive independent keys for our encryption and MAC operations. The two

[2] To uniquely define K_{AB} and K_{BA}, the identifiers A and B of $\mathcal{X}_{AB}$ are lexicographically sorted.

communicating parties A and B share a master secret key $\mathcal{X}_{AB}$, and they derive independent keys using the pseudo-random function F: encryption keys $K_{AB} = F_{\mathcal{X}}(1)$ and $K_{BA} = F_{\mathcal{X}}(3)$ for each direction of communication, and MAC keys $K'_{AB} = F_{\mathcal{X}}(2)$ and $K'_{BA} = F_{\mathcal{X}}(4)$ for each direction of communication. Section 6 gives more details on key derivation.

The combination of these mechanisms form our Sensor Network Encryption Protocol SNEP. The encrypted data has the following format: $E = \{D\}_{\langle K,C\rangle}$, where D is the data, the encryption key is K, and the counter is C. The MAC is $M = \mathsf{MAC}(K', C||E)$. The complete message that A sends to B is

$$A \to B: \quad \{D\}_{\langle K_{AB},C_A\rangle},\ \mathsf{MAC}\big(K'_{AB}C_A \,||\, \{D\}_{\langle K_{AB},C_A\rangle}\big). \quad (1)$$

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 sufficiently long enough to never repeat within the lifetime of the node.
- *Data authentication*. If the MAC verifies correctly, a receiver knows that the message originated from the claimed sender.
- *Replay protection*. The counter value in the MAC prevents replay of old messages. Note that if the counter were not present in the MAC, an adversary could easily replay messages.
- *Weak freshness*. If the message verifies correctly, a 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.[3]

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 so long that exhaustive search of all possible nonces is not feasible). 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

$$\begin{aligned} &A \to B: \quad N_A, R_A, \quad (2)\\ &B \to A:\\ &\quad \{R_B\}_{\langle K_{BA},C_B\rangle},\ \mathsf{MAC}\big(K'_{BA}, N_A \,||\, C_B \,||\, \{R_B\}_{\langle K_{BA},C_B\rangle}\big). \end{aligned}$$

If the MAC verifies correctly, node A knows that node B generated the response after it sent the request. The first message can also use plain SNEP (as described in equation (1)) if confidentiality and data authentication are needed.

5.2. Counter exchange protocol

To achieve small SNEP messages, we assume that the communicating parties A and B know each other's counter values C_A and C_B and so the counter does not need to be added to each encrypted message. In practice, however, messages might get lost and the shared counter state can become inconsistent. We now present protocols to synchronize the counter state. To bootstrap the counter values initially, we use the following protocol:

$$\begin{aligned} &A \to B: \quad C_A,\\ &B \to A: \quad C_B,\ \mathsf{MAC}\big(K'_{BA}C_A \,||\, C_B\big),\\ &A \to B: \quad \mathsf{MAC}\big(K'_{AB}, C_A \,||\, C_B\big). \end{aligned}$$

Note that the counter values are not secret, so we do not need encryption. However, this protocol needs strong freshness, so both parties use their counters as a nonce (assuming that the protocol never runs twice with the same counter values, hence incrementing the counters if necessary). Also note that the MAC does not need to include the names of A or B, since the MAC keys K'_{AB} and K'_{BA} implicitly bind the message to the parties, and ensure the direction of the message.

If party A realizes that the counter C_B of party B is not synchronized any more, A can request the current counter of B using a nonce N_A to ensure strong freshness of the reply:

$$\begin{aligned} &A \to B: \quad N_A,\\ &B \to A: \quad C_B,\ \mathsf{MAC}(K'_{BA}, N_A \,||\, C_B). \end{aligned}$$

To prevent a potential denial-of-service (DoS) attack, where an attacker keeps sending bogus messages to lure the nodes into performing counter synchronization, the nodes can switch to sending the counter with each encrypted message they send. Another approach to detect such a DoS attack is to attach another short MAC to the message that does not depend on the counter.

5.3. μTESLA: *Authenticated broadcast*

Previous proposals for authenticated broadcast are impractical for sensor networks. First, most proposals rely on asymmetric digital signatures for authentication, which are impractical for multiple reasons, which we describe in section 1.

The recently proposed TESLA protocol provides efficient authenticated broadcast [42,43]. However, TESLA is not designed for the limited computing environments we encounter in sensor networks for the following three reasons:

[3] If the MAC does not match, the receiver can try a fixed, small number of counter increments to recover from message loss. If this still fails, the two parties engage in the counter exchange protocol we describe below.

TESLA authenticates the initial packet with a digital signature. Clearly, digital signatures are too expensive to compute on our sensor nodes, since even fitting 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 significant. 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.

Finally, the one-way key chain does not fit 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.

5.4. μTESLA *overview*

We give a brief overview of μTESLA, followed by a detailed description.

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, making 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 efficient broadcast authentication scheme.

We first explain μTESLA for the case where the base station broadcasts authenticated information to the nodes. Later we discuss the case where the nodes are the sender.

μTESLA requires that the base station and nodes be loosely time synchronized, and each node knows an upper bound on the maximum synchronization error. To send an authenticated packet, the base station computes a MAC on the packet with a key that is secret at that point in time. When a node gets a packet, it can verify that the corresponding MAC key was not yet disclosed by the base station (based on its loosely synchronized clock, its maximum synchronization error, and the time schedule at which keys are disclosed). Since a receiving node is assured that the MAC key is known only by the base station, the receiving node is assured that no adversary could have altered the packet in transit. The node stores the packet in a buffer. At the time of key disclosure, the base station broadcasts the verification key to all receivers. When a node receives the disclosed key, it can verify the correctness of the key (which we explain below). If the key is correct, 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 public 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 1 shows the μTESLA one-way key chain derivation, the time intervals, and some sample packets that the sender broadcasts. 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. In this example, the sender discloses keys two time intervals after it uses them to compute MACs. We assume that the receiver node is loosely time synchronized and knows K_0 (a commitment to the key chain). 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. 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 $K_0 = F(F(K_2))$. The node derives $K_1 = F(K_2)$, so it can authenticate packets P_1, P_2 with K_1, and P_3 with K_2.

Key disclosure is independent from the packets broadcast, and is tied to time intervals. In μTESLA, the sender broadcasts the current key periodically in a special packet.

5.5. μTESLA *detailed description*

μTESLA has multiple phases: sender setup, sending authenticated packets, bootstrapping new receivers, and authenticating packets. We first explain how μTESLA allows the base station to broadcast authenticated information to the nodes,

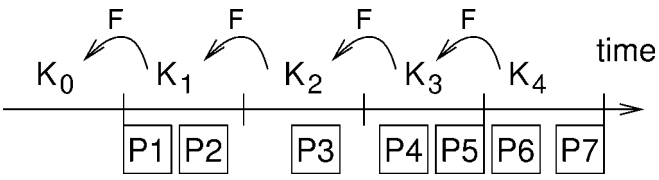

Figure 1. The μTESLA one-way key chain. The sender generates the one-way key chain right-to-left by repeatedly applying the one-way function F. The sender associates each key of the one-way key chain with a time interval. Time runs left-to-right, so the sender uses the keys of the key chain in reverse order, and computes the MAC of the packets of a time interval with the key of that time interval.

and we then explain how TESLA allows nodes to broadcast authenticated messages.

Sender setup. The sender first generates a sequence of secret keys (a one-way key chain). To generate a 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 [46]): $K_j = F(K_{j+1})$. Because F is a one-way function, anybody can compute forward, e.g., compute $K_0, \ldots, K_j$ given K_{j+1}. On the other hand, nobody can compute backward, e.g., compute K_{j+1} given only $K_0, \ldots, K_j$, because the generator function is one-way. The S/Key one-time password system uses a similar approach [21].

Broadcasting authenticated packets. Time is divided into uniform time intervals, and the sender associates each key of the one-way key chain with one time interval. In time interval i, the sender uses the key of the current interval, K_i, to compute the message authentication code (MAC) of packets in that interval. In time interval $(i + \delta)$, the sender reveals key K_i. The key disclosure time delay 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. In a one-way key chain, keys are self-authenticating. The receiver can easily and efficiently authenticate subsequent keys of the one-way key chain using one authenticated key. For example, if a receiver has an authenticated value K_i of the key chain, it can easily authenticate K_{i+1}, by verifying $K_i = F(K_{i+1})$. To bootstrap μTESLA, each receiver needs to have one authentic key of the one-way key chain as a commitment to the entire chain. Other requirements are that the sender and receiver be 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 and the authenticated key chain commitment can be established with a mechanism providing strong freshness and point-to-point authentication. A receiver R sends a nonce N_R in the request message to the sender S. The sender S replies with a message containing its current time T_S, a key K_i of the one-way key chain used in a past interval i (the commitment to the key chain), 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):

$$\begin{aligned} M \to S: &\quad N_M \\ S \to M: &\quad T_S \mid K_i \mid T_i \mid T_{\text{int}} \mid \delta \\ &\quad \mathsf{MAC}(K_{MS}, N_M \mid T_S \mid K_i \mid T_i \mid T_{\text{int}} \mid \delta). \end{aligned}$$

Since we do not need confidentiality, 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 a receiver receives the packets with the MAC, it needs to ensure that the packet is not a spoof from an adversary. The adversary already knows the disclosed key of a time interval, so it could forge the packet since it knows the key used to compute the MAC. We say that the receiver needs to be sure that the packet is *safe* – i.e. that the sender did not yet disclose the key that was used to compute the MAC of an incoming packet. As stated above, the sender and receivers need to be loosely time synchronized and the receivers need to know the key disclosure schedule. If the incoming packet is safe, the receiver stores the packet (it can verify it only once the corresponding key is disclosed). If the incoming packet is not safe (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 new key K_i, it authenticates the key by checking that it matches the last authentic key it knows K_v, using a small number of applications of the one-way function F: $K_v = F^{i-v}(K_i)$. If the check is successful, the new key K_i is authentic and the receiver can authenticate all packets that were sent within the time intervals v to i. The receiver also replaces the stored K_v with K_i.

Nodes broadcasting authenticated data. New challenges arise if a node broadcasts authenticated data. Since the node is memory limited, it cannot store the keys of a one-way key chain. Moreover, re-computing each key from the initial generating key K_n is computationally expensive. Also, the node might not share a key with each receiver, so sending out the authenticated commitment to the key chain would involve an expensive node-to-node key agreement. Finally, broadcasting the disclosed keys to all receivers is expensive for the node and drains precious battery energy.

Here are two solutions to the 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

Because of stringent resource constraints on the sensor nodes, implementation of the cryptographic primitives is a major challenge. We can sacrifice some security to achieve feasibility and efficiency, but we still need a core level of strong cryptography. Below we discuss how we provide strong cryptography despite restricted resources.

Memory size is a constraint: 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 [29,50].

Block cipher. We evaluated several algorithms for use as a block cipher. An initial choice was the AES algorithm Rijndael [12]; however, after further inspection, we sought alternatives with smaller code size and higher speed. The baseline version of Rijndael uses over 800 bytes of lookup tables which is too large for our memory-deprived nodes. An optimized version of that algorithm (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 [32]. A small encryption algorithm such as TEA [54] is a possibility, but is has not yet been subject to cryptanalytic scrutiny.[4] We use RC5 [47] because of its small code size and high efficiency. RC5 does not rely on multiplication and does not require large tables. However, RC5 does use 32-bit data-dependent rotates, which are expensive on our Atmel processor (it only supports an 8-bit single bit rotate operation).

Even though the RC5 algorithm can be expressed succinctly, the common RC5 libraries are too large to fit on our platform. With a judicious selection of functionality, we 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 for both encryption and decryption. The counter (CTR) mode of block ciphers (figure 2) has this property. CTR mode is a stream cipher. Therefore, the size of the ciphertext is exactly the size of the plaintext and not a multiple of the block size.[5] This property is particularly desirable in our environment. Message sending and receiving consume a lot of energy. Also, longer messages have a higher probability of data corruption. Therefore, block cipher message expansion is undesirable. CTR mode requires a counter for proper operation. Reusing a counter value severely degrades security. In addition, CTR-mode offers semantic security: the same plaintext sent at different times is encrypted into different ciphertext since the encryption pads are generated from different counters. To an adversary who does not know the key, these messages will appear as two unrelated random strings. Since the sender and the receiver share the counter, we do not need to include it in the message. If the two nodes lose the synchronization of the counter, they can simply transmit the counter explicitly to resynchronize using SNEP with strong freshness.

Freshness. Weak freshness is automatically provided by the CTR encryption. Since the sender increments the counter after each message, the receiver verifies weak freshness by verifying that received messages have a monotonically increasing counter. For applications requiring strong freshness, the sender creates a random nonce N_M (an unpredictable 64-bit value) and includes it in the request message to the receiver. The receiver generates the response message and includes the nonce in the MAC computation (see section 5). If the MAC of the response verifies successfully, the node knows that the response was generated after it sent the request message and hence achieves strong freshness.

[4] TREYFER [56] by Yuval is a small and efficient cipher, but Biryukov and Wagner describe an attack on it [7].

[5] The same property can be achieved with a block cipher and the *ciphertext-stealing* method described by Schneier [50]. The downside is that Schneier's approach requires both encryption and decryption functions.

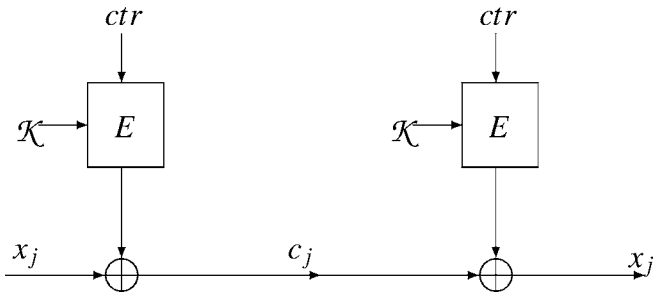

Figure 2. Counter mode encryption and decryption. The encryption function is applied to a monotonically increasing counter to generate a one time pad. This pad is then XORed with the plaintext. The decryption operation is identical.

Random-number generation. The node has its own sensors, wireless receiver, and scheduling process, from which we could derive random digits. But to minimize power requirements, we use a MAC function as our pseudo-random number generator (PRG), with the secret pseudo-random number generator key $\mathcal{X}_{\text{rand}}$. 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 $\mathsf{MAC}(\mathcal{X}_{\text{rand}}, \mathcal{C})$. If $\mathcal{C}$ wraps around (which should never happen because the node will run out of energy first), we can derive a new PRG key from the master secret key and the current PRG key using our MAC as a pseudo-random function (PRF): $\mathcal{X}_{\text{rand}} = \mathsf{MAC}(\mathcal{X}, \mathcal{X}_{\text{rand}})$.

Message authentication. We also need a secure message authentication code. Because we intend to reuse our block cipher, we use the well-known CBC-MAC [33]. A block diagram for computing CBC MAC is shown in figure 3.

To achieve authentication and message integrity we use the following standard approach. Assuming a message M, an encryption key $\mathcal{K}$, and a MAC key $\mathcal{K}'$, we use the following construction: $\{M\}_{\mathcal{K}}, \mathsf{MAC}(\mathcal{K}', \{M\}_{\mathcal{K}})$. 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 fits well with the lossy nature of communications within this environment. Furthermore, at this granularity, the MAC is used to check both authentication and integrity of messages, eliminating the need for mechanisms such as CRC.

Key setup. Recall that our key setup depends on a secret master key, initially shared by the base station and the node. We call that shared key $\mathcal{X}_{AS}$ for node A and base station S. All other keys are bootstrapped from the initial master secret key. Figure 4 shows our key derivation procedure. We use the

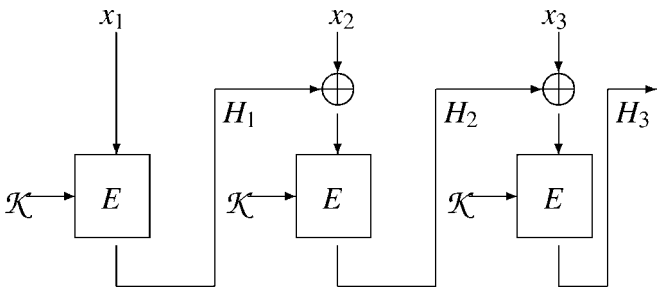

Figure 3. CBC MAC. The output of the last stage serves as the authentication code.

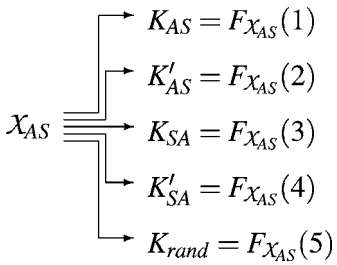

Figure 4. Deriving internal keys from the master secret key.

pseudo-random function (PRF) F to derive the keys, which we implement as $F_K(x) = \mathsf{MAC}(K, x)$. Again, this allows for more code reuse. Because of cryptographic properties of the MAC, it must also be a good pseudo-random function. 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 find 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 confidential information.

7. Evaluation

We evaluate the implementation of our protocols by code size, RAM size, and processor and communication overhead.

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. The difference between the fastest and the smallest implementation stems from two different implementations of the variable rotate function. The μTESLA protocol uses another 574 bytes. Together, the crypto library and the protocol implementation consume about 2 Kbytes of program memory, which is acceptable in most applications.

It is important to identify reusable routines to minimize call setup costs. For example, OpenSSL implements RC5 encryption as a function. On our sensor hardware, the code size of call setup and return outweigh the code size of the body of the RC5 function. We implement RC5 as a macro and only expose interfaces to the MAC and CTR-ENCRYPT functions.

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

Version	Total size	MAC	Encrypt	Key setup
Smallest	1580	580	402	598
Fastest	1844	728	518	598
Original	2674	1210	802	686

Table 3
Performance of security primitives in TinyOS.

Operation	Time in ms Fast implementation	Time in ms Small implementation
Encrypt (16 bytes)	1.10	1.69
MAC (16 bytes)	1.28	1.63
Key setup	3.92	3.92

Performance. The performance of the cryptographic primitives is adequate for the bandwidth supported by the current generation of network sensors. Key setup is relatively expensive (4 ms). In contrast, the fast version of the code uses less than 2.5 ms to encrypt a 16 byte message and to compute the MAC (the smaller but slower version takes less than 3.5 ms). Let us compare these time figures against the speed of our network. Our radio operates at 10 kbps at the physical layer. If we assume that we communicate at this rate, we can perform a key setup, an encryption, and a MAC for every message we send out.[6]

In our implementation, μTESLA discloses the key after two intervals ($\delta = 2$). The stringent buffering requirements also dictate that we cannot drop more than one key disclosure beacon. 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 the integrity of a TESLA message.[7] That gives an upper bound of 17.8 ms 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 traffic is the amount of buffering we can dedicate on individual sensor nodes. Table 4 shows the memory size required by the security modules. We configure the μTESLA protocol with four messages: the disclosure interval dictates a buffer space of three messages just for key disclosure, and we need an additional buffer to use this primitive in a more flexible way. Despite allocating minimal amounts of memory to μTESLA, the protocols we implement consume half of the available memory, and we cannot afford any more memory.

Energy costs. We examine the energy costs of security mechanisms. Most energy costs will come from extra transmissions required by the protocols.

[6] The data rate available to the application is significantly smaller, due to physical layer encoding, forward error correction, media access protocols, and packet format overheads.

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

Table 4
RAM requirements of the security modules.

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

Table 5
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.

71%	Data transmission
20%	MAC transmission
7%	Nonce transmission (for freshness)
2%	MAC and encryption computation

Table 5 lists the energy costs of computation and communication for the SNEP protocol. The energy costs are computed for 30 byte packets. The energy overhead for the transmission dominates energy overhead for computation. Since we use a stream cipher for encryption, the size of encrypted message is the same as the size of the plaintext. The MAC adds 8 bytes to a message. But, because the MAC gives us integrity guarantees, we do not need an extra 2 bytes of CRC, so the net overhead is only 6 bytes. The transmission of these 6 bytes requires 20% of the total energy for a 30 byte packet, as table 5 shows.

Messages broadcast using μTESLA have the same costs of authentication per message. Additionally, μTESLA requires a periodic key disclosure, but these messages are combined with routing updates. We can take two 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 one message per time interval, regardless of the traffic pattern within the network. We believe that the benefits of authenticated routing justify 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 through covert channels. Second, we do not deal completely with compromised sensors, we merely ensure that compromising a single sensor does not reveal the keys of all the sensors in the network. 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 wireless channel with a strong signal. Finally, due to our hardware limitations, we cannot provide Diffie-Hellman style key agreement or use digital signatures to achieve non-repudiation. For the majority of sensor network applications, authentication is sufficient.

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 [11,20,25,26,38,40,41]. Marti et al. discuss a mechanism to protect an ad hoc network against misbehaving nodes that fail to forward packets correctly [28]. They describe two mechanisms: a watchdog to detect misbehaving neighboring nodes, and a pathrater to keep state about the goodness of other nodes. They propose running these mechanisms on each node. However, we are not aware of a routing protocol that uses authenticated routing messages. It is possible 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 base station, protected by a MAC to ensure integrity and that the data is authentic) in the current epoch.[8] 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 first search algorithm, and produces a routing topology (see [23] for details).

However, in the above algorithm, 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. In contrast, 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 first node sending a successfully authenticated packet. Combining μTESLA key disclosure with distribution of routing beacons allows us to combine transmission of the keys with network maintenance.

We have outlined a scheme leading to a lightweight authenticated routing protocol for sensor networks. Since each node accepts only the first authenticated packet as the one to use in routing, it is impossible for an attacker to reroute arbitrary links within the sensor network. Each node verifies the

[8] Epoch means the interval between routing updates.

behavior of the parent by implementing functionality similar to watchdogs described in [28].

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.[9]

8.2. Node-to-node key agreement

A convenient technology for bootstrapping secure connections is to use public key cryptography protocols for symmetric key setup [5,22]. Unfortunately, our resource constrained sensor nodes prevent us from using computationally expensive public key cryptography. We need to construct our protocols solely from symmetric key algorithms. 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 $\mathcal{SK}_{AB}$ 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 station in our case. In our trust setup, both A and B share a master secret key with the base station, $\mathcal{X}_{AS}$ and $\mathcal{X}_{BS}$, respectively. The following protocol achieves secure key agreement as well as strong key freshness:

$$
\begin{aligned}
A \to B: &\quad N_A, A,\\
B \to S: &\quad N_A, N_B, A, B, \mathsf{MAC}\big(K'_{BS}, N_A|N_B|A|B\big),\\
S \to A: &\quad \{\mathcal{SK}_{AB}\}_{K_{SA}}, \mathsf{MAC}\big(K'_{SA}, N_A|B|\{\mathcal{SK}_{AB}\}_{K_{SA}}\big),\\
S \to B: &\quad \{\mathcal{SK}_{AB}\}_{K_{SB}}, \mathsf{MAC}\big(K'_{SB}, N_A|B|\{\mathcal{SK}_{AB}\}_{K_{SB}}\big).
\end{aligned}
$$

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 ensures confidentiality (through encryption with the keys K_{AS} and K_{BS}) of the established session key $\mathcal{SK}_{AB}$, as well as message authentication (through the MAC using keys K'_{AS} and K'_{BS}), so we are 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, and 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. Many other protocols 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, but it does not provide strong key freshness [27,30]. If Kerberos used SNEP with strong freshness, then Kerberos would have greater security.

[9] The node needs significantly more memory resources than our current sensor nodes to store the key chain.

9. Related work

Tatebayashi et al. consider key distribution for resource-starved devices in a mobile environment [52]. Park et al. [37] point out weaknesses and improvements. Beller and Yacobi further develop key agreement and authentication protocols [4]. Boyd and Mathuria survey the previous work on key distribution and authentication for resource-starved devices in mobile environments [8]. The majority of these approaches rely on asymmetric cryptography. Bergstrom et al. consider the problem of secure remote control of resource-starved devices in a home [6].

Fox and Gribble present a security protocol providing secure access to application level proxy services [16]. Their protocol is designed to interact with a proxy to Kerberos and to facilitate porting services relying on Kerberos to wireless devices.

The work of Patel and Crowcroft focuses on security solutions for mobile user devices [39]. 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 [10].

Stajano and Anderson discuss the issues of bootstrapping security devices [51]. Their solution requires physical contact of the new device with a master device to imprint the trusted and secret information.

Zhou and Haas propose to secure ad hoc networks using asymmetric cryptography [57]. Recently, Basagni et al. proposed to use a network-wide symmetric key to secure an ad hoc routing protocol [2]. While this approach is efficient, it does not resist compromise of a single node.

Carman et al. analyze a wide variety of approaches for key agreement and key distribution in sensor networks [9]. They analyze the overhead of these protocols on a variety of hardware platforms.

Marti et al. discuss a mechanism to protect an ad hoc network against misbehaving nodes that fail to forward packets correctly [28]. They propose that each node runs a watchdog (to detect misbehaving neighboring nodes) and a pathrater (to keep state about the goodness of other nodes); their solution, however, is better suited for traditional networks, with emphasis on reliable point-to-point communication, than to sensor networks.

Hubaux et al. present a system for ad hoc peer-to-peer authentication based on public key certificates [24]. They consider an ad hoc network with nodes powerful enough for performing asymmetric cryptographic operations.

A number of researchers investigate the problem to provide cryptographic services in low-end devices. We first 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 [15], the Dallas iButton [13], and the Dyad system [55]. These systems support primitives for cryptography, and attempt to zeroize their memory if tampering is de-

tected (as per the FIPS 140 standard [34,35]). However, these devices were designed for different applications, and are not meant as low-power devices.

Modadugu et al. describe an asymmetric crypto system for low-end devices, which offloads the heavy computation for finding an RSA key pair to untrusted servers [31].

Symmetric encryption algorithms seem to be inherently well suited to low-end devices, because they have relatively low overhead. In practice, however, many low-end microprocessors are only 4-bit or 8-bit, and do not provide (efficient) multiplication or variable rotate/shift instructions. Hence many symmetric ciphers are too expensive to implement on our target platform. The Advanced Encryption Standard (AES) [36] Rijndael block cipher [12] is too expensive for our platform. Depending on the implementation, AES was either too big or too slow for our application. Due to our severe limitation on our maximum code size, we chose to use RC5 by Ron Rivest [47]. Algorithms such as TEA by Wheeler and Needham [54] or TREYFER by Yuval [56] would be smaller alternatives, but those other ciphers have not yet been thoroughly analyzed.

10. Conclusion

We designed and built a security subsystem for an extremely limited sensor network platform. We have identified and implemented useful security protocols for sensor networks: authenticated and confidential communication, and authenticated broadcast. We have implemented applications including an authenticated routing scheme and a secure node-to-node key agreement protocol.

Most of our design is universal and applicable to other networks of low-end devices. Our primitives only depend on fast symmetric cryptography, and apply to a wide variety of device configurations. On our limited platform energy spent for security is negligible compared with to energy spent on sending or receiving messages. It is possible to encrypt and authenticate all sensor readings.

The communication costs are also small. Data authentication, freshness, and confidentiality properties use up a net 6 bytes out of 30 byte packets. So, it is feasible to guarantee these properties on a per packet basis. It is difficult to improve on this scheme, as transmitting a MAC is fundamental to guaranteeing data authentication.

Certain elements of the design were influenced by the available experimental platform. If we had a more powerful platform, we could have used block ciphers other than RC5. The emphasis on code reuse is another property forced by our platform. A more powerful device would allow more modes of authentication. In particular, memory restrictions on buffering limit the effective bandwidth of authenticated broadcast.

Despite the shortcomings of our target platform, we built a system that is secure and works. With our techniques, we believe security systems can become an integral part of practical sensor networks.

Acknowledgements

We gratefully acknowledge funding support for this research. This research was sponsored in part by 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 F33615-01-C-1895, “NEST”), 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.

We thank Jean-Pierre Hubaux, Dawn Song and David Wagner for helpful discussions and comments. An earlier version of this work appeared as [44].

References

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

[2] S. Basagni, K. Herrin, E. Rosti and D. Bruschi, Secure Pebblenets, in: *ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2001)* (2001) pp. 156–163.

[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: *Symposium on Foundations of Computer Science (FOCS)* (1997).

[4] M. Beller and Y. Yacobi, Fully-fledged two-way public key authentication and key agreement for low-cost terminals, Electronics Letters 29(11) (1993) 999–1001.

[5] S. Bellovin and M. Merrit, Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and password file compromise, in: *ACM Conference on Computer and Communications Security CCS-1* (1993) pp. 244–250.

[6] P. Bergstrom, K. Driscoll and J. Kimball, Making home automation communications secure, IEEE Computer 34(10) (2001) 50–56.

[7] A. Biryukov and D. Wagner, Slide attacks, in: *International Workshop on Fast Software Encryption* (1999).

[8] C. Boyd and A. Mathuria, Key establishment protocols for secure mobile communications: A selective survey, in: *Australasian Conference on Information Security and Privacy* (1998) pp. 344–355.

[9] D.W. Carman, P.S. Kruus and B.J. Matt, Constraints and approaches for distributed sensor network security, NAI Labs Technical Report No. 00-010 (2002).

[10] S.E. Czerwinski, B.Y. Zhao, T.D. Hodes, A.D. Joseph and R.H. Katz, An architecture for a secure service discovery service, in: *ACM International Conference on Mobile Computing and Networking (MobiCom'99)* (1999) pp. 24–35.

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

[12] J. Daemen and V. Rijmen, AES proposal: Rijndael (1999).

[13] Dallas, iButton: A Java-powered cryptographic iButton, `http://www.ibutton.com/ibuttons/java.html`

[14] W. Diffie and M.E. Hellman, Privacy and authentication: An introduction to cryptography, Proceedings of the IEEE 67(3) (1979) 397–427.

[15] Fortezza, Fortezza: Application implementers guide (1995).

[16] A. Fox and S.D. Gribble, Security on the move: Indirect authentication using Kerberos, in: *International Conference on Mobile Computing and Networking (MobiCom'96)* (1996) pp. 155–164.
[17] R. Gennaro and P. Rohatgi, How to sign digital streams, in: *Advances in Cryptology – Crypto'97*, Lecture Notes in Computer Science, Vol. 1294 (1997) pp. 180–197.
[18] O. Goldreich, S. Goldwasser and S. Micali, How to construct random functions, Journal of the ACM 33(4) (1986) 792–807.
[19] S. Goldwasser and S. Micali, Probabilistic encryption, Journal of Computer Security 28 (1984) 270–299.
[20] Z. Haas and M. Perlman, The Zone Routing Protocol (ZRP) for ad hoc networks, Internet draft, Mobile Ad-Hoc Network (MANET) Working Group, IETF (1998).
[21] N.M. Haller, The S/KEY one-time password system, in: *Symposium on Network and Distributed Systems Security* (1994).
[22] D. Harkins and D. Carrel, The Internet key exchange (IKE), RFC 2409, Information Sciences Institute, University of Southern California (1998).
[23] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler and K. Pister, System architecture directions for networked sensors, in: *International Conference on Architectural Support for Programming Languages and Operating Systems*.
[24] J.-P. Hubaux, L. Buttyán and S. Čapkun, The quest for security in mobile ad hoc networks, in: *ACM Symposium on Mobile Ad Hoc Networking and Computing* (2001).
[25] D.B. Johnson and D.A. Maltz, Dynamic source routing in ad hoc wireless networks, in: *Mobile Computing* (Kluwer Academic, 1996) chapter 5, pp. 153–181.
[26] Y.-B. Ko and N. Vaidya, Location-Aided Routing (LAR) in mobile ad hoc networks, in: *International Conference on Mobile Computing and Networking (MobiCom'98)* (1998).
[27] J. Kohl and C. Neuman, The Kerberos network authentication service (V5), RFC 1510 (1993).
[28] S. Marti, T. Giuli, K. Lai and M. Baker, Mitigating routing misbehaviour in mobile ad hoc networks, in: *International Conference on Mobile Computing and Networking (MobiCom 2000)* (2000) pp. 255–265.
[29] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone, *Handbook of Applied Cryptography* (CRC Press, 1997).
[30] S.P. Miller, C. Neuman, J.I. Schiller and J.H. Saltzer, Kerberos authentication and authorization system, Project Athena Technical Plan (1987).
[31] N. Modadugu, D. Boneh and M. Kim, Generating RSA keys on a handheld using an untrusted server, RSA 2000 (2000).
[32] National Bureau of Standards (NBS), Specification for the data encryption standard, Federal Information Processing Standards (FIPS) Publication 46 (1977).
[33] National Institute of Standards and Technology (NIST), DES model of operation, Federal Information Processing Standards Publication 81 (FIPS PUB 81) (1981).
[34] National Institute of Standards and Technology (NIST), Security requirements for cryptographic modules, Federal Information Processing Standards (FIPS) Publication 140-1 (1994).
[35] National Institute of Standards and Technology (NIST), Security requirements for cryptographic modules, Federal Information Processing Standards (FIPS) Publication 140-2 (1999).
[36] National Institute of Standards and Technology (NIST), Advanced encryption standard (AES) development effort (2000) `http://csrc.nist.gov/encryption/aes/`
[37] C. Park, K. Kurosawa, T. Okamoto and S. Tsujii, On key distribution and authentication in mobile radio networks, in: *Advances in Cryptology – EuroCrypt'93*, Lecture Notes in Computer Science, Vol. 765 (1993) pp. 461–465.
[38] V. Park and M. Corson, A highly adaptable distributed routing algorithm for mobile wireless networks, in: *IEEE INFOCOMM'97* (1997).
[39] B. Patel and J. Crowcroft, Ticket based service access for the mobile user, in: *International Conference on Mobile Computing and Networking (MobiCom'97)* (1997) pp. 223–233.
[40] C. 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).
[41] C. Perkins and E. Royer, Ad hoc on-demand distance vector routing, in: *IEEE WMCSA'99* (1999).
[42] A. Perrig, R. Canetti, D. Song and J.D. Tygar, Efficient and secure source authentication for multicast, in: *Network and Distributed System Security Symposium, NDSS'01* (2001).
[43] A. Perrig, R. Canetti, J. Tygar and D. Song, Efficient authentication and signing of multicast streams over lossy channels, in: *IEEE Symposium on Security and Privacy* (2000).
[44] A. Perrig, R. Szewczyk, V. Wen, D. Culler and J.D. Tygar, SPINS: Security protocols for sensor networks, in: *International Conference on Mobile Computing and Networking (MobiCom 2001)*, Rome, Italy (2001).
[45] K.S.J. Pister, J.M. Kahn and B.E. Boser, Smart dust: Wireless networks of millimeter-scale sensor nodes (1999).
[46] R. Rivest, The MD5 message-digest algorithm. RFC 1321, Internet Engineering Task Force (1992).
[47] R.L. Rivest, The RC5 encryption algorithm, in: *Workshop on Fast Software Encryption* (1995) pp. 86–96.
[48] R.L. Rivest, A. Shamir and L.M. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21(2) (1978) 120–126.
[49] P. Rohatgi, A compact and fast hybrid signature scheme for multicast packet authentication, in: *ACM Conference on Computer and Communications Security* (1999).
[50] B. Schneier, *Applied Cryptography*, 2nd ed. (Wiley, 1996).
[51] F. Stajano and R. Anderson, The resurrecting duckling: Security issues for ad-hoc wireless networks, in: *International Workshop on Security Protocols* (1999).
[52] M. Tatebayashi, N. Matsuzaki and D.B.J. Newman, Key distribution protocol for digital mobile communication systems, in: *Advances in Cryptology – Crypto'89*, Lecture Notes in Computer Science, Vol. 435 (1989) pp. 324–334.
[53] D. Tennenhouse, Embedding the Internet: Proactive computing, Communications of the ACM 43(5) (2000) 43.
[54] D. Wheeler and R. Needham, TEA, a Tiny Encryption Algorithm (1994) `http://www.ftp.cl.cam.ac.uk/ftp/papers/djw-rmn/djw-rmn-tea.html`
[55] B. Yee and J.D. Tygar, Secure coprocessors in electronic commerce applications, in: *USENIX Workshop on Electronic Commerce*, New York (1995).
[56] G. Yuval, Reinventing the Travois: Encryption/MAC in 30 ROM bytes, in: *Workshop on Fast Software Encryption* (1997).
[57] L. Zhou and Z. Haas, Securing ad hoc networks, IEEE Network Magazine 13(6) (1999).

Adrian Perrig received his Bachelors degree in computer science from the Swiss Federal Institute of Technology in Lausanne (EPFL) in 1997. He is now a graduate student at Carnegie Mellon University, and is currently completing his degree with his advisor Doug Tygar at the University of California, Berkeley. His research interests include cryptography, and designing security protocols for wireless and broadcast networks.
E-mail: perrig@cs.berkeley.edu

Robert Szewczyk received the B.S. *magna cum laude* from Cornell University in 1997. He is currently pursuing a Ph.D. at University of California, Berkeley, under the supervision of David Culler. His research interests include sensor networks and their applications.
E-mail: szewczyk@cs.berkeley.edu

J.D. Tygar is a Professor of Computer Science and Information Management at the University of California, Berkeley. He has worked widely in the fields of computer security and electronic commerce. He has received numerous awards and was selected as an NSF Presidential Young Investigator. He has designed and built systems for electronic commerce, mobile security, tamper-resistant systems, and electronic postage. He has co-written two books.
E-mail: tygar@cs.berkeley.edu

Victor Wen received a B.S. degree in electrical engineering and computer science from the University of California, Berkeley in 1999. He is currently a Ph.D. student at the University of California, Berkeley. His research interests include coding techniques for power reduction on chip, and wireless networking applications.
E-mail: vwen@cs.berkeley.edu

David E. Culler is a Professor of Computer Science at the University of California. He has been on the faculty at Berkeley since 1989 and has served as a Vice Chair for Computing and Networking. He received his Ph.D. from MIT in 1989. He was awarded the NSF Presidential Young Investigator in 1990 and the Presidential Faculty Fellowship in 1992. His research addresses parallel computer architecture, parallel programming languages, and high performance communication structures. He is well known for his work on Networks of Workstations (NOW), Active Messages, Split-C, the Threaded Abstract Machine (TAM), and dataflow systems. He has published widely in leading conferences and journals, obtained three patents, and recently completed a graduate text called *Parallel Computer Architecture: A Hardware/Software Approach* (Morgan-Kaufmann, publisher). He has served as a General Chair and Program Chair for Hot Interconnects, Program Chair for the ACM Symposium on Parallel Algorithms and Architectures and Operating Systems Design and Implementation, Technical Papers Chair for SC2001 and Co-Editor for special issues of IEEE Transactions on Parallel and Distributed Computing and IEEE Micro.
E-mail: culler@cs.berkeley.edu