Authentication for Bulk Data Dissemination in Sensor Networks Using Symmetric Keys [1]

Limin Wang
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824

Mahesh Arumugam
Cisco Systems, Inc.
170 West Tasman Dr.
San Jose, CA 95134

Sandeep S. Kulkarni
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824

Abstract

We focus on authentication of bulk data dissemination in sensor networks using symmetric keys. By bulk data, we mean that the data is large enough that sufficient RAM memory cannot be allocated to store the data (due to limited memory as well as memory requirements from other parts of the code in the sensor). Hence, the data must be stored in EEPROM. Due to cost of write to EEPROM, this data must be verified before storage.

We consider three scenarios: multihop coarse-grained pipelining, multihop fine-grained pipelining, and single hop. In the second and third scenarios, we show that the time to transmit 3-4 KB of data is less than (or, close to) sending even a single packet using public keys. For the first scenario, we argue that the use of symmetric keys would marginally improve the performance. For this scenario, we also propose additional techniques to reduce the cost of secure data dissemination by more than 70%.

Keywords: Sensor networks, Data dissemination, Authentication, Symmetric keys

1 Introduction

In this paper, we focus on the problem of securing the transmission of bulk data in a sensor network. This problem is crucial in the context of sensor networks where the amount of available RAM is significantly small (e.g., Mica2 sensors [4] have only 4KB RAM). Therefore, if large amount of data needs to be sent to the sensor nodes then this data must be stored on the EEPROM (i.e., external flash of sensors), which is much larger in size. One of the important problems with this however is that EEPROM writes are expensive in terms of energy (e.g., writing a 16-byte block to an EEPROM is approximately four times more expensive than transmitting a message or it is equivalent to execution of more than 2,300,000 instructions [28, 32]). Moreover, each EEPROM location can be successfully written only a finite number of times (typically about 10,000 operations [6]). In other words, after a certain number of writes, the EEPROM location cannot be changed subsequently. Thus, an adversary can launch a denial of service (DOS) attack by sending garbage data to a sensor. Even if the sensor later finds out that the data is invalid, it would have had spent significant energy in saving the data to EEPROM. This suggests that data must be authenticated *before* it is written to EEPROM. We define this problem as the problem of bulk data dissemination.

We consider the problem of bulk data dissemination in three scenarios. In one extreme, bulk data dissemination needs to be solved for reprogramming a sensor network where the base station sends the new program to the sensors. This program size is typically tens of kilobytes in typical applications. In the second scenario, the data is moderate in size. Such a scenario may occur if the base station is collecting statistics about the network performance. The base station may then send a summary of such data along with new commands, reconfigured values of different parameters, code for revised functionality, etc., to all the sensors in the network. It is expected that the size of such data would typically be smaller (1-4 KB) compared to the case of reprogramming where the old program is completely replaced by the new program. However, it can still be large enough that storing it entirely in memory may not be feasible, especially due to the fact that typical applications use static memory allocation for main memory. The third scenario occurs in cases where the network is divided into a collection of (possibly overlapping) clusters. In such a scenario, the cluster leader would need to communicate data to all sensors in its network. This scenario differs from the second scenario in that the communication within the cluster is expected to be single hop in the third scenario whereas the communication is expected to be multihop in the second scenario. A variation of the third scenario also occurs when reprogramming needs to be done in a laboratory environment where all sensors are close to the base station (i.e., within distance 1). Even in such a scenario, security is needed to ensure that two users trying to reprogram their respective sensors do not (accidentally or otherwise) interfere with each other. The goal of our work is to develop a mechanism that will allow sensors to authenticate the data they receive before they store it to the EEPROM.

Scheme for authenticating a data stream. One way to authenticate such data is to use the approach from [9] for sign-

[1] Email: wanglim1@cse.msu.edu, maarumug@cisco.com, sandeep@cse.msu.edu
Web: `http://www.cse.msu.edu/~{wanglim1,arumugam,sandeep}`.
This work was partially sponsored by NSF CAREER CCR-0092724, DARPA Grant OSURS01-C-1901, ONR Grant N00014-01-1-0744, NSF grant EIA-0130724, and a grant from Michigan State University.

ing digital streams (as done in [8]). In this approach, the data is divided into a collection of packets, say $x_1, x_2, ...x_n$. Subsequently, hash of x_n is computed and attached to x_{n-1}. Then, hash is computed on this modified packet (i.e., packet x_{n-1} and hash of x_n) and attached to x_{n-2}. This process is then repeated until we obtain x_1 and the hash of the modified x_2 (that contains x_2 and hash of *modified* x_3). Finally, the hash of the modified first packet is then *signed*. (We call this approach *the basic hash chain scheme*.) With this approach, when a node receives the first packet, it can use the signature to authenticate it. Additionally, it obtains the hash value for the (modified) second packet. Thus, when it receives the second packet, it can use the hash value to verify it, and so on. In this approach, the important issue is the approach used for signing the hash of the first packet. If the *cost* of this signature is reduced then it would assist in reducing the overall cost. The cost of the signature is especially important in the second and third scenarios in the previous paragraph, where the size of the data is comparatively smaller. Note that with this approach, if any packet is lost then a node cannot verify subsequent packets. Hence, in the context of our problem, such packets cannot be stored in EEPROM. This problem can be remedied if all the hash values are sent in advance (as done in [7]) or by signing intermediate packets in the packet stream; these signatures would allow a node to authenticate (and store to EEPROM) packets received after the signature.

Existing approaches [7, 8, 19] use public keys for authenticating the data used in reprogramming. The base station signs the hash of the first packet using its private key and each sensor decrypts it using the corresponding public key. However, creating and verifying the asymmetric digital signatures have very high computation overhead. Moreover, since the cryptographic operations are overlapped with the radio operations; if the encryption/decryption operations are not fast enough, we may encounter problems if the radio packets needed by the sensors are no longer available. Although recent work has shown that RSA and elliptic curve cryptography (ECC) are feasible on Mica/TelosB motes [11, 23, 33], they should still be avoided or used sparingly.

With this motivation, we focus on use of symmetric keys. For example, if we use Skipjack (or RC5) on Mica2 motes, the execution time for encryting/decrypting an 8-byte block is only 0.38ms (or 0.26ms in the case of RC5), which is less than the time for sending one byte data over radio [13]. This approach also has the potential to allow intermediate packets to be signed so that even if a sensor misses some packet, it can authenticate and store packets received after the subsequent signature. Inserting such intermediate signatures is feasible since the cost of computing these signatures is small. A simple approach is to use a single network-wide key shared by the base station and all the sensors [13]. The problem with this approach is that a sensor cannot verify if the data received is from the base station or another sensor in the network. Therefore, we must use symmetric keys in such a way that the nodes can verify that the data was indeed sent by the base station.

Contributions of the paper.

- We propose a symmetric key based protocol that authenticates the bulk data dissemination process in sensor networks. The algorithm requires only $O(\log n)$ keys to be maintained at each sensor. Thus, in our protocol, only a very small number of keys are maintained at every sensor.
- We illustrate the applicability of our approach in the three scenarios considered above. Regarding the first scenario, we focus on the reprogramming protocol MNP [18] and show that security can be added to it using our approach. Furthermore, we illustrate how adding redundancy to the transmitted data can further reduce the cost of adding security. Regarding the second scenario, we focus on Infuse [16]. We show that the time to transmit moderate amount of data is less than the time for transmitting even **a single packet** with public keys. Finally, we show that the same observation holds even for the third scenario.

Organization of the paper. In Section 2, we describe the threat model and security requirements of the secure bulk data dissemination problem. In Section 3, we introduce the secret instantiation algorithm that we use to sign the hash of the first packet. In Section 4, we present our authentication protocol in the three scenarios mentioned above. We add authentication to the existing dissemination protocols and evaluate the performance. In Section 5, we propose additional techniques to improve the performance for the first scenario. In Section 6, we evaluate the performance enhancement by applying these techniques. In Section 7, we compare with related work, and discuss issues on key distribution and updates. We conclude this paper in Section 8.

2 Threat Model and Security Requirements

We consider an adversary as one who tries to inject its own code into sensor nodes or launch denial of service attacks that aim to exhaust sensors' battery power. It can eavesdrop on any communication in the network. It is able to compromise a sensor node, and acquire all information inside it. It can also inject, change, delete packets. However, an adversary cannot compromise the base station, which is securely protected.

We focus on authentication only , i.e., we assume that confidentiality is not required, i.e., the data being transmitted are public and can be acquired by the adversary. Hence, the data are sent in plain text along with appropriate authentication.

The goals of the proposed protocol are as follows:

1. Authenticity. Each sensor must be able to verify that data are from a trusted source and have not been changed during transit. We consider the base station as a trusted source, and is protected against compromise.
2. Node-compromise resilience. It must not be possible that compromising a single sensor node will cause the other parts of the network insecure.
3. Authentication before storage. A sensor should verify the authenticity and integrity of a received packet before writing it to flash. This is to reduce the time and energy cost of receiving fake packets from an adversary in a denial of service attack.

3 Protocol For Signing The Hash of The First Packet

In order to sign the first hash in the hash chain, we can either use public key based signatures or symmetric key based signatures. As illustrated in Section 1, symmetric key based signatures have much lower cost compared to public key based signatures. Hence, we focus on use of symmetric keys. Specifically, we use the secret instantiation algorithm [10,17], which requires only $O(\log n)$ keys to be maintained at each sensor. We describe the secret instantiation algorithm below.

The base station has a collection of secrets. Initially, each sensor receives some subset of this collection. Whenever the base station sends a message, it separately signs it using all the secrets in its collection. Thus, message transmission is associated with a collection of signatures, one for each secret that the base station has. To sign message m, with secret s, the base station can use algorithms such as MD5. (Additionally, if the length of the signature needs to be small, then only a small part of this signature (e.g., last few bytes) may be used.) Whenever a sensor receives this communication, it verifies the signatures based on the collection of secrets it has. Of course, a sensor will only be able to verify a subset of the signatures, as it does not have all the secrets. It is required that if all these signature verifications are successful, the sensor can assume that the communication is truly from the base station (and not from an outsider or anther sensor pretending to be the base station).

To implement this algorithm, a 2-dimensional array of secrets with r rows (numbered $0..r-1$) and $\log_r n$ (numbered $1..\log_r n$) columns (where $2 \leq r \leq n$ and n is the number of sensors) is maintained. The base station knows all these secrets. Each sensor is assigned a unique ID that is a number with radix r. Observe that the ID is of length $log_r n$. (Leading 0s are added if necessary.) This ID identifies the secrets that a sensor should get. Specifically, if the first digit (most significant) of the ID is x then the sensor gets x^{th} secret in the first column. If the second digit of the ID is y then the sensor gets y^{th} secret in the second column, and so on.

To illustrate the algorithm, let us consider an example. Let the number of nodes be 16 and let r be 2. Then the base station contains 8 (i.e., 2 $\log_2 16$) secrets with 2 rows and 4 columns. Each sensor has 4 (i.e., $\log_2 16$) secrets. The set of secrets a sensor has are decided by its unique ID. For example, if a sensor's ID is 0011, then it has the secrets on the first row in the first two columns and the secrets on the second row in the next two columns.

Theorem 1. If sensor j receives a message and it verifies all the signatures based on the secrets it knows then that message must be sent by the base station.

Proof: See [10] for proof. □

Collusion. In the secret instantiation algorithm, compromising a single sensor node will not compromise the entire network. This is due to the facts that each sensor has only a subset of the secrets, and if an adversary attempts to pretend to be the base station, it needs to get all the secrets. However, colluding sensors may be able to obtain all the keys and, thereby, pretend to be the base station. By choosing an appropriate value for r, this key distribution provides a tradeoff between level of collusion resistance and number of keys at the base station.

Computation cost of signing/verifying the signatures and computing the hashes. The hash chain and signatures are computed only once at the base station. The sensors simply use/forward the signatures received from the base station. Moreover, since we use symmetric keys for signatures, the overhead is much lower than (about 0.0005 times) the cost of using the asymmetric keys. While hash computation is performed for every packet, it is very efficient (less than 10ms per packet). Hence, hash computation does not significantly increase the computation cost either.

4 Performance of Authentication Protocol In Three Scenarios

In this section, we show the effectiveness of our approach in the three scenarios discussed in Section 1. First, in Section 4.1, we discuss the third scenario where the sensors are within a single hop of the source. Subsequently, in Section 4.2, we discuss the second scenario where a moderate amount of data (1-4KB) is sent using a *fine-grained pipelining* (i.e., packet-level pipelining) protocol. In Section 4.3, we discuss the first scenario where large amount of data is sent using a *coarse-grained pipelining* (i.e., segment-level pipelining) protocol.

4.1 Secure Single-hop Dissemination

In this section, we focus on the scenario where the source node is disseminating a moderate amount of data to all the receivers within single hop. This typically happens in a small/indoor network, or within a cluster in a large network. The source node can be the base station or a cluster head depending on how the protocol is used. Any single-hop/multi-hop reprogramming protocol (e.g., [5, 12, 16, 18, 25]) can be used for dissemination in this scenario. We use a simple CSMA-based protocol (similar to that in [5]), which is described as follows.

The base station computes hash for each data packet. The hashes are put into one or a few packets (called hash packets). The base station then computes hashes for these hash packets, and accommodates them into one or a few higher-level hash packets. In this way, the hashes of packets are organized into a hash tree (this uses the approach in [7]). The base station signs the root of the hash tree using all the secrets it has (based on our symmetric key algorithm). The base station sends the signatures at first, followed by the hash tree, from higher level or lower level. After the hash tree has been sent, it sends the data packets. Because the hashes are sent before the data packets, when a sensor receives a data packet, it can authenticate this packet immediately using the hash value. After the base station has transmitted all the data packets, it will send query several times. If the base station receives requests from the receivers, it will retransmit the requested packets. This continues until all the receivers have received all the hashes and data packets. We call this approach *secure single-hop dissemination with hash tree*.

Alternatively, we can also use the basic hash chain approach, as described in Section 1. The base station sends the signatures of the hash of the first packet, the first packet contains the hash of the second packet, and so on. We call it *secure single-hop dissemination with hash chain*.

We run this simple protocol (with hash tree and hash chain variations) on a special purpose simulator. The base station sends the signatures and the hash packets twice for robustness. The payload size of a data packet is 70 bytes, including 6 bytes header and 64 bytes data. A hash is 4 bytes long. Hence, each packet can carry 16 hashes. The signatures fit in one packet.

The transmission interval of a data packet is 45ms. When a receiver receives a query packet, it will wait for a short random duration before sending a request. We vary the data size from 0.5KB to 4KB, and simulated the two approaches at different receiver set sizes. Specifically, the number of receivers is set to 5, 15, 50. Packet loss rate is 5%. We repeat the simulations three times, and use the average.

In Figure 1, we show the propagation time of this secure single hop protocol at different data sizes and different receiver set sizes. For comparison, we also show the time required to send **a single packet** through the network. According to [33], the time required for the base station to sign a packet with its private key is 21.5s. And, the time required to verify the packet at each sensor is 0.79s. Thus, the time to propagate one packet to a single hop network is at least 22.29s (21.5s+0.79s). From Figure 1, we can see that using our symmetric key based authentication with hash tree approach (respectively, with hash chain approach), the time to send 4KB data through the network (including request and retransmission time) is only 53-63% (respectively, is close to) the time to transmit **a single packet** to the network using public key based authentication. This shows that using symmetric key based authentication can significantly reduce the time (and energy) cost compared to the public key based authentication.

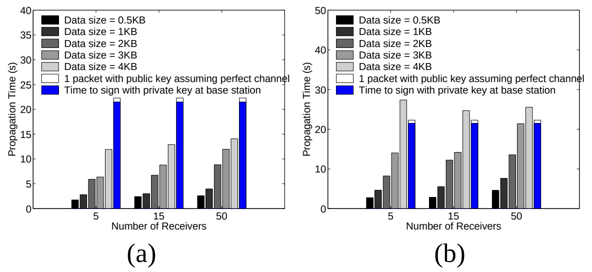

Figure 1. Propagation time in a single hop network. (a) hash tree approach (b) hash chain approach. (For approaches with public key schemes, only signing/verifi cation cost is considered. Communication cost, although possibly significant when the size of signature is large, is not considered. But for our protocol, both signing/verication cost and communication cost are considered.)

Memory requirement. The memory requirement of the protocol are listed as following. First, $\log_r n$ secrets are maintained at each sensor. For example, in a 10x10 network, the number of secrets maintained at each sensor is at most 7. Second, the signatures from the base station for each segment need to be stored either in memory or in flash. Third, the signing/verification process consumes some amount of memory. This amount of memory is much lower compared to the asymmetric key based approaches. Fourth, in the hash tree approach, each sensor caches all the received hashes in RAM (and use them to verify the data packets later). If the length of the data stream is 4KB (64 packets), the amount of memory for storing the hashes is 256 bytes. If the hash chain approach is used, only the hash contained in the last packet that has been authenticated needs to be cached in RAM. In this case, the RAM requirement for hash cache is only 4 bytes.

4.2 Secure Multihop Dissemination with Fine-Grained Pipelining

In this section, we focus on the scenario where the base station disseminates a moderate size data stream (of size 0.5-3KB) across the network in a *fine-grained pipelining* fashion. Such fine-grained pipelining service can be achieved using any TDMA based data dissemination protocol (e.g., Infuse [16], Sprinkler [25]). To illustrate secure dissemination in a multihop network with fine-grained pipelining, we use Infuse [16] to disseminate the data across the network. We note that the evaluation results presented in this section are applicable to other data dissemination protocols that use fine-grained pipelining.

Overview of Infuse. Infuse is a reliable TDMA based data dissemination protocol. Since we consider grid based networks in this paper, we present an overview of Infuse for such a network. The basic idea of Infuse is to assign time slots in such a way that no two nodes within distance two (in the grid) of each other should get the same slot. This ensures that whenever any node transmits data, nodes within distance 1 can always receive that message without collision. For the case where the communication range of a node exceeds the distance with the closest neighbor, nodes can be assigned different frequencies to prevent collision. Note that Mica motes can provide multiple channels (e.g., 54 channels in the 902-928 MHz frequency band) on which they can transmit [1]. However, they can listen to only one frequency at a time. To illustrate this, consider a line network a-b-c-d-e. Suppose that sensors can communicate/interfere with nodes upto distance 2 then, a and d should transmit on different frequencies. Depending on the communication/interference range of the sensors, the number of frequency channels required varies. The issue of number of frequency channels required is outside the scope of this paper. Although TDMA ensures collision freedom, messages can still be lost due to random channel errors. To deal with this, Infuse uses sliding window based recovery mechanism and *implicit acknowledgments* (by listening to the transmissions of the neighbors).

Evaluation with Infuse. We authenticate the data stream disseminated with Infuse using the basic hash chain approach discussed in Section 1. We call this protocol *Infuse+Auth*. We simulate the protocol in Prowler [30], a probabilistic wireless network simulator for Mica motes. The goal of the simulations is to illustrate that the propagation time with

Infuse+Auth is significantly less than that with public key scheme. We disseminate data of sizes 0.5-3KB across 10x1, 5x5, and 10x10 networks. The payload size of the messages transmitted by Infuse is 16 bytes. In our simulations, we use 4 packets as the sliding window size (for dealing with random channel errors). Since random channel errors can cause the link reliability to go down, we choose a conservative estimate of 95% link reliability in our simulations.

Figure 2 shows the propagation time of *Infuse+Auth*. As observed from the figure, the time to disseminate data with *Infuse+Auth* is significantly less than the time to propagate one packet across the network using public key mechanism. Based on [33], the analytical result on time required to sign and verify a packet across a 10x1 network (respectively, 5x5 and 10x10 networks) is 28.61s (respectively, 27.82s and 35.72s). On the other hand, with *Infuse+Auth*, the time to disseminate the data stream of size 3KB is approximately the same as the time required with the public key scheme for **a single packet**. Thus, the time to securely disseminate a moderate size data across a multihop network is significantly less with the use of symmetric keys. (Note that due to multiple paths in a 10x10 network, a sensor may recover a lost packet from a different neighbor. For this reason, the transmission time lower in a 10x10 network than in a 10x1 network. For details of such behavior, we refer the reader to [16], as this issue is not central to the topic of this paper.)

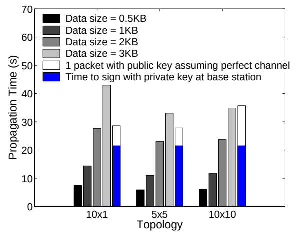

Figure 2. Propagation time with window size = 4. (For approaches with public key schemes, only signing/verifi cation cost is considered. Communication cost, although possibly signifi cant when the size of signature is large, is not considered. But for our protocol, both signing/verication cost and communication cost are considered.)

Memory requirement. Similar to the discussion in Section 4.1, *Infuse+Auth* also maintains $\log_r n$ secrets at each sensor. Additionally, since *Infuse+Auth* uses the basic hash chain approach, only the hash contained in the last authenticated packet needs to be kept in RAM. Finally, Infuse maintains a sliding window of 4 packets (=64 bytes, as the packet size in Infuse is 16 bytes) to deal with the problem of random message losses. However, this requirement is inherent to Infuse irrespective of whether authentication is enabled or not.

4.3 Secure Multihop Dissemination with Coarse-Grained Pipelining

Coarse-grained pipelining are implemented in protocols such as MNP [18] and Deluge [12]. In these protocols, the data stream is divided into segments and pipelining is provided at the segment level. Security of Deluge is considered in [8] and uses the scheme in [9] that is described in Section 1. Using our approach would be identical except that our approach reduces the cost of signing and verifying the hash of the first packet. Thus, using our approach would reduce the cost of data propagation by approximately 22 seconds than that in [8]. Likewise, if hash tree approach is used with symmetric keys then the data propagation time would improve by approximately 22 seconds over that in [7]. For this reason as well as the fact that the data propagation time in [7, 8] is in hundreds of seconds, we do not provide detailed simulation results for this scenario. Instead, in the next section, we present additional mechanisms that would reduce the cost of secure data dissemination in coarse-grained pipelining.

5 Performance Enhancement

In this section, we propose three techniques to improve the performance of our authentication protocol when it is used to disseminate a large amount of data in a multihop network with coarse-grained pipelining. We do a case study on MNP [18]. We note that the design and simulation results we present in this section and Section 6 are applicable to other data dissemination protocols as well.

In Section 5.1, we give an overview of MNP. In Section 5.2, we describe how we use the keys and hashes to sign the data stream. Specifically, we propose the *double connected hash chains*, combined with symmetric key signatures, to authenticate the data stream. We also propose two other schemes to further improve the efficiency: creating a cache on the receiver side (Section 5.3) and using forward error correction (FEC) (Section 5.4). We will evaluate the performance of our authentication protocol and show the effect of applying these schemes in Section 6.

5.1 A Brief Overview of MNP

MNP is a bulk data dissemination protocol, which provides a reliable and energy efficient service to propagate a large amount of data to all the sensors in the network over radio. Data are sent in *segments*. Each segment contains K packets. Sensors must receive the segments in order. A sensor (or the base station) advertises segment N only if all the packets in segments 1-N are available. Within a segment, the packets may be received out of order. When the neighbors receive the advertisements for a segment, if they have not received that segment completely, they will send requests to the advertiser. This request also specifies the packets that the requester wants. The sender then transmits the requested packets in the segment. This process continues until every packet from the base station is received by every sensor. Note that a sender can send either an entire segment or a partial segment (only includes the packets that are requested by at least one receiver). For simplicity of presentation, we present the protocol as if the sender only sends an entire segment. The extension for sending a partial segment is straightforward, and hence omitted.

To reduce the message collision problem, MNP uses a sender

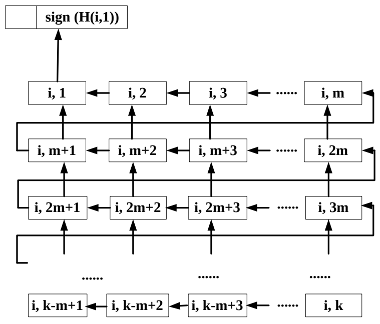

Figure 3. The double connected hash chain (segment i).

selection algorithm to try to guarantee that there is only one sender in a neighborhood at a time. In the sender selection algorithm, sensors compete with each other based on the number of requests (higher the better) they have received and the ID of the segment (lower the better) they are transmitting. If a sensor wins in the sender selection algorithm and becomes the sender, it transmits the corresponding segment it advertised earlier. The sensors that are not transmitting or receiving data are put to sleep state to save energy. The sender selection algorithm effectively reduces message collision. Hence, in MNP, packet loss is caused by random effect, rather than collisions.

5.2 Double Connected Hash Chain

In the basic hash chain scheme (cf. Section 1), the data packets have to be *verified* in order. Losing a single data packet can lead to all the succeeding packets to be thrown away. This leads to significant time/energy waste. To address this problem, we propose a double connected hash chain to enhance the inter-connection among the packets, as illustrated in Figure 3. Assume that the entire data stream has N ($N \geq 1$) segments and each segment contains K packets. We represent the j^{th} data packet of the i^{th} segment as $P(i, j)$, $i = 1..N$, $j = 1..K$. (We also refer to it as packet j for simplicity, as long as it does not cause confusion.) The hash of packet $P(i, j)$ is denoted as $H(i, j)$. As shown in Figure 3, a segment is further divided into *hash groups*. Each hash group contains m packets. m is an integer factor of K. A packet $P(i, j)$ contains a data part and two hashes: the hash of the next packet (with successive packet ID, e.g., $P(i, j + 1)$), and the hash of the corresponding packet in the next hash group (e.g., $P(i, j+m)$). In Figure 3, an arrow pointing from packet j to packet i indicates that packet i contains the hash of packet j. In this way, we construct multiple authentication paths for verifying a data packet. To illustrate how the double connected hash chain works, consider the scenario where a single packet (packet 2) is lost, while all other packets in the segment have been received. If we use the double connected hash chain, all the packets starting from packet m+1 can be authenticated because packet 1 contains the hash for packet m+1.

5.3 Caching

Data cache and hash cache. We use two caches: a data cache, for storing the data packets, and a hash cache, for storing the hashes. Assuming a hash is 4 bytes long and the size of a segment is 64 packets, we only need 256 bytes to store all the hashes for the entire segment. On the other hand, storing data packets requires much more space. To reduce the memory consumption, we divide a segment into *cache groups*. At any given time, each sensor caches data packets only from one cache group. The cache group that is currently kept in the data cache is called *the active cache group*. Note that when a sensor writes a data packet to its data cache, it writes the data part as well as the two hashes. By doing this, when it authenticates a packet, the hashes contained in the packet are authenticated at the same time. Only those hashes that have been authenticated are written to the hash cache.

When a receiver receives a data packet, if it needs the packet, it computes the cache group this packet belongs to. If the packet is in the active cache group, the sensor stores the packet to its assigned slot in the data cache. Otherwise, it changes the active cache group to the one that this data packet belongs to, and stores the received packet to the data cache. When a sensor changes its active cache group, if there are data packets in the data cache that are not yet authenticated, these packets are discarded. Moreover, when a sensor writes a packet to the data cache, it also checks if the hash cache contains the hash for this packet. If so, the packet is authenticated: the data part of the packet is written to EEPROM, and the hashes are written to the hash cache (if they are not already in the hash cache).

When a hash is written to the hash cache, the sensor checks the data cache to see if the newly added hash can be used to authenticate any data packet. If so, the packet is authenticated, and the two hashes contained in this packet can be added to the hash cache, which could in turn be used to authenticate more packets in the data cache. Hence, one reception of a data packet can possibly lead to several continuous writes to EEPROM. These writes to EEPROM must be queued and data are buffered when necessary so that the erasure of the data cache will not cause loss of data.

5.4 Forward Error Correction

MNP, as well as all other existing data dissemination protocols [12, 16, 25, 31], uses automatic repeat request (ARQ) scheme to recover the lost packets. In ARQ schemes, a receiver detects its own losses, and informs the sender of the missing packets, either by sending requests or acknowledgements. The sender retransmits the packets that are requested by the receivers. In the current problem, a single lost packet may cause a sensor to discard other (valid but not yet authenticated) packets. Hence, in order to reduce packet loss, in our protocol, we use forward error correction (FEC).

FEC provides reliability by transmitting redundant packets in a proactive manner. Due to computational limits on sensors, we use the simple XOR FEC scheme. For simple XOR code, each transmission group has only one parity packet, which is the XOR or all the source packets in the group. XOR code is very simple to implement, and it can repair a single packet loss in a transmission group.

The data cache provides required memory space for encoding and decoding XOR parity packets, hence, there is no additional overhead on memory consumption for employing FEC. In our approach, a sender transmits a parity packet after transmitting t data packets (t is the size of the transmission group). The parity packet is XOR of the t data packets that proceeding it. We require that t be not larger than the size of the data cache.

When a receiver receives a parity packet, it checks if exactly one packet is missing in the transmission group that this parity packet belongs to. If so, it uses the parity packet to recover the packet that is missing. When a receiver tries to fix a missing packet by decoding a parity packet, if all the received packets in this transmission group are cached in the data cache (i.e., they are in the active cache group), decoding the parity packet can be conducted directly. In the case that some data packets were received in earlier transmissions and are not in the data cache, they must be read from EEPROM to the data cache. As reading a packet from EEPROM is an optimized operation with low cost, employing simple XOR code in our protocol does not incur much time/energy overhead.

6 Evaluation of Enhancement

We integrate our protocol with MNP [18]. We refer to the integrated protocol as *MNP+Auth*. We simulate *MNP+Auth* using TOSSIM [20]. TOSSIM is a discrete event simulator for TinyOS wireless sensor networks. In TOSSIM, the network is modeled as a directed graph. Each vertex in the graph is a sensor node. Each edge has a bit-error rate, representing the probability with which a bit can be corrupted if it is sent along this link.

In our simulation, each segment has 64 data packets. The size of a hash group m is 8. The simulations are performed in a grid topology. The base station is at the corner of the network. The inter-node distance is 10 feet. We consider a 10x10 network, i.e., the number of sensors in the network is 100. We set r to be 2. In this case, the base station contains 14 (i.e., $2 \log_2 100$) secrets, and each sensor has 7 secrets. Due to the fact that the execution time of each simulation is of order of tens of hours, we do not provide confidence intervals.

We set the packet size to 70 bytes, among which, 6 bytes are used for the packet header (including source node ID, destination node ID, program ID, segment ID, packet type), the remaining 64 bytes are for the data and hashes. In *MNP+Auth*, each data packet carries 2 hashes. Each hash is 4 bytes long. Hence, excluding the hashes, the *effective* data payload is 56 bytes. Therefore, in every data packet, 8 out of 64 bytes of the payload is consumed in authentication.

We first analyze the memory requirement of *MNP+Auth* in Section 6.1. Then, we show the performance of *MNP+Auth* in Section 6.2. Due to limitation of space, we present performance only in terms of propagation time. We refer the reader to [34] for more detailed evaluation results (e.g., energy consumption, communication overhead). Finally, in Section 6.3, we investigate how our design decisions (i.e., double connected hash chain, caching and FEC techniques) affect the performance.

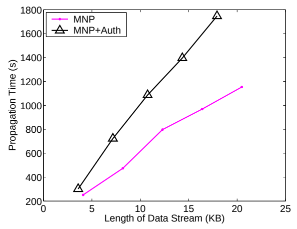

Figure 4. Propagation time of MNP and *MNP+Auth*.

6.1 Memory Requirement

The memory requirement of our authentication protocol with the enhancements in Section 5 is as follows. The amount of memory required for caching the secrets and the signatures, and that required during signing/verification process are the same as what we have discussed for the secure single-hop dissemination protocol (cf. Section 4.1). In addition, the data cache and the hash cache contribute to the major part of memory consumption. If the data cache contains 8 packets, each packet is 64 bytes long (including the data part and the hashes), plus 1-byte packet ID (local ID, used inside a segment) and a *valid* bit (to indicate if the packet is in the data cache), the data cache requires 521 bytes. As we discussed in Section 5.3, the hash cache needs 256 bytes memory. Moreover, a variable is needed to record the current active cache group. This only needs to be 1 byte long. And, as discussed in Section 5.4, since the data cache provides the memory required for FEC encoding/decoding, there is no extra memory overhead by applying FEC scheme.

6.2 Performance of *MNP+Auth*

We set the size of the data cache c to 8 packets. The size of FEC transmission group t is set to the same as the data cache size, which is also 8 packets. As we have pointed out earlier, 8 out of 64 bytes data payload in a data packet are used for hashes. Therefore, transmitting one segment (64 packets per segment) in MNP disseminates 4KB data, while transmitting one segment in *MNP+Auth* only disseminates 3.5KB data. In Figure 4, we show the propagation time of MNP and *MNP+Auth* at different data stream lengths. We can see that given a certain amount of data to transmit, the propagation time required by *MNP+Auth* is 37%-74% higher than that required by MNP. This cost is much lower than the case where we simply use the basic hash chain approach without optimization. In the latter case, the propagation time of the secured version of MNP is 6 or 7 times that of the original MNP. The same conclusion holds for Deluge [12], as shown in [7, 8]. Hence, the performance improvement by applying the techniques we proposed in Sections 5.2-5.4 is significant.

6.3 Analysis of Design Options

In this section, we analyze how the techniques we proposed in Sections 5.2-5.4 contribute to the overall performance. We

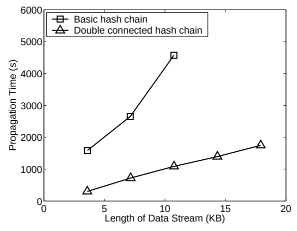

Figure 5. Comparison of the basic hash chain with the double connected hash chain.

evaluate each of them by disabling/replacing/varying it while fixing other parts of the protocol.

Comparing basic hash chain with double connected hash chain. To evaluate the effectiveness of the double connected hash chain, we compare it with the case where the basic hash chain is used (cf. Figure 5).

We can see that using the double connected hash chain, the performance is significantly improved compared to the basic hash chain. Replacing the basic hash chain with the double connected hash chain can reduce the propagation time by 73-81%. Specifically, the time to disseminate 10.7KB data using the basic hash chain scheme is 4570 seconds, which is 4.2 times the time for disseminating the same amount of data using the double connected hash chain (1087 seconds).

Varying the size of the data cache. To evaluate the effectiveness of data cache, we vary the size of the data cache from 8 packets to 64 packets. In the case that the data cache size is 64 packets, the entire segment can be stored in memory. In this case, even if the data packets arrive out of order, they can be temporarily saved in the data cache, waiting to be authenticated later. Hence, the basic hash chain works well in this scenario. In Figure 6 (a), we show the propagation time of *MNP+Auth* for disseminating different amounts of data, when the data cache size varies. For comparison, we also show the corresponding performance of MNP. In the case that the data cache size is 64 packets, we use the basic hash chain, instead of the double connected hash chain, to reduce the cost of distributing hashes. As shown in Figure 6 (a), when the data cache size is 8 packets, 16 packets, and 32 packets, the lines that represent the propagation time in Figure 6 (a) intersect with each other. In other words, there is no significant performance improvement when we increase the size of the data cache.

By increasing the cache size, we are able to cache more hash groups in memory. This helps to improve the performance only if some delayed packets in the earlier group(s) arrive later. However, this long delay of packets is uncommon (although short delays of packets within a cache group are common) for two reasons: the sensors send packets in order (with increasing packet IDs), and sensors communicate with their direct neighbors (i.e., there is no path delay). Moreover, the FEC encoding/decoding is done in the active cache group. Therefore, setting the data cache size to 8 packets (which is the same as the hash group size and FEC transmission group size) is optimal (for reducing memory consumption and achieving reasonable performance), unless memory is big enough to accommodate the entire segment.

If we increase the data cache size to 64 packets, the propagation time is reduced significantly. In this case, all the received packets are buffered in memory, the only cost is on distributing the hashes. Hence, when the data cache size is 64 packets, the propagation time is only a little higher than that of MNP.

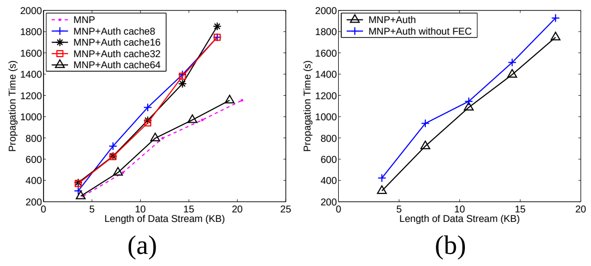

Figure 6. Analysis of design options. (a) varying the data cache size from 8 packets to 64 packets. (b) effect of using FEC. The data cache size is 8 packets.

Effect of using FEC. Forward error correction (FEC) reduces the number of requests and retransmits by sending extra parity packets. In this section, we investigate if the use of FEC is beneficial. The size of the FEC transmission group and the size of the data cache are both 8 packets. In Figure 6 (b), we compare the performance of *MNP+Auth* in the cases that FEC is enabled and disabled. We can see that by employing FEC, we can reduce the propagation time by 5-29%.

7 Discussion

In this section, we first compare our protocol with existing work in secure data dissemination (Section 7.1), then we discuss the issues of initial key distribution and key updates (Section 7.2).

7.1 Comparison With Related Work

Security of data dissemination in sensor networks is studied in [2, 7, 8, 14, 15, 19, 21, 22, 26, 27, 29]. Of these, [7, 8, 19] use asymmetric keys; the base station signs the (hash of) data using the private key and the sensors use the corresponding public key to authenticate the data. As discussed in Section 4, the cost of asymmetric keys is exorbitant when the data size is moderate (1-4KB). In particular, in Section 4, we showed that for several scenarios of moderate data dissemination, the cost of signing and sending one packet using asymmetric keys is close to the cost of sending 3-4 KB of data using symmetric keys. Finally, our approach is orthogonal to that in [7], i.e., as discussed in Section 4.3, our approach of using symmetric keys could be used in [7] to reduce the cost of those algorithms.

Our approach also differs from μTESLA [27] and its extensions [21, 22] in that in [21, 22, 27], *after-the-fact* authentication is provided. In particular, in these approaches, the sensors first receive a packet (set of packets) and subsequently receive the key to authenticate it. To ensure security of such a protocol, the time to reveal the key must be large enough so that all nodes receive the packet (respectively, set of packets) before the key is revealed (loose time synchronization is required in these protocols). In the context of bulk data transmission, this would require the sensors to buffer a large amount of data before it can be authenticated. With limited storage on sensors, this would require the data to be stored on EEPROM, thereby, opening up the possibility of denial of service attack. By contrast, in our approach, only authenticated packets are stored on EEPROM. Thus, the approaches in [21, 22, 27] are applicable in broadcasts of small data whereas our approach is also applicable in broadcasts of bulk data as well. Moreover, as discussed later in this section, it is possible to combine the features of our algorithm and that in [27].

One-time signatures commit a secret key via one-way functions, hence, have much lower signing and verification time compared to asymmetric primitives. One-time signatures have been used in many broadcast authentication protocols [2,14,15,26,29]. BiBa [26] performs authenticated broadcast via pre-computed hash collisions and chains. HORS [29] improves BiBa by reducing the signature generation time. BiBa and HORS are inappropriate for sensor networks due to its large public key size (e.g., a typical public key size is 20KB). Since the public key must be stored on all sensors, it is desirable to keep its size small. The approach in [2, 14, 15] decreases the public key size of HORS by constructing Merkle trees [24] on the secret keys, and use the roots of the Merkle trees as the public key. Even with this reduction, the public key size is still hundreds to thousands of bytes, which is much larger compared to the size of secrets (e.g., at most 7 secrets in a 10x10 network) in our protocol. Moreover, the signature size in [2, 14, 15] is also large. For example, in [15], the typical signature size is 690-2560 bytes. Hence, when the authors apply their one-time signature protocol to Deluge [12], the first few *pages* are used for sending the signature. By contrast, in our protocol, all the signatures are sent in one (or a few) message(s). Therefore, our protocol has much lower memory requirement and communication overhead.

7.2 Key Distribution and Updates

As discussed in Section 3, the initial keys are assigned to sensors at deployment. Alternate approaches are also possible for distribution. For example, at deployment, the sensors may include all the secrets and then depending upon their logical ID (either known at deployment or communicated thereafter), they can delete the secrets they are not supposed to have. This approach is based on the assumption (true in many sensor network deployments) that the initial communication among sensors is secure and that the sensors cannot be compromised for a certain duration after deployment.

As discussed in Section 3, the approach in [10, 17] allows us to provide a tradeoff between the level of security and the number of secrets maintained by the base station. The designer can choose an appropriate value of r to ensure that the effect of collusion is moderate. Increasing the value of r increases the overhead only marginally, as the cost of signing with a symmetric key is very small.

Also, our approach can be combined with the approach in [21, 22, 27]. In particular, instead of maintaining a single value for each secret, we could have a hash chain of values for each secret. (Optimizations from [3] allow only a little more than $\log_2 n$ entries from a hash chain to be maintained at the base station.) The last value in the hash chain is made available to the sensors at the time of deployment. However, periodically, the base station will reveal the previous value in the hash chain and this value would be encrypted with the current secret to ensure that only those sensors that have the old secret can obtain the new one.

8. Conclusion

In this paper, we showed how authentication could be achieved for bulk data dissemination in sensor networks. We used symmetric key distribution algorithms from [10, 17] to ensure that the base station can communicate securely with each sensor in the network. Based on the security of the key distribution, our protocol allows sensors to conclude that the data is truly transmitted by the base station.

We considered the secure data dissemination problem in three scenarios: multihop dissemination with coarse-grained (segment-level) pipelining, multihop dissemination with fine-grained (packet-level) pipelining, and single-hop dissemination. We showed that the use of symmetric keys can significantly reduce the cost of disseminating a moderate amount of data, especially in the second and third scenarios. In particular, the time to sign and verify a single packet using public key scheme is more than 22 seconds. Within the same amount of time, we can disseminate 3-4KB of data across the network using symmetric key scheme. For the first scenario, we proposed additional mechanisms to reduce the cost of secure data dissemination. We showed that the basic hash chain that is commonly used for authenticating data streams is not efficient in the presence of packet loss, as it requires that packets arrive in order. We proposed the double connected hash chain to strengthen the inter-connection among the packets so that loss of a few packets does not fail the authentication for the entire segment. To further improve the performance, we proposed a caching scheme and employed forward error correction (FEC) technique to try to minimize the effect of packet loss.

We showed that our algorithm can be easily applied to different types of data dissemination protocols: simple non-pipelined, fine-grained pipelined (e.g., Infuse [16], Sprinkler [25]), and coarse-grained pipelined (e.g., MNP [18], Deluge [12]). Hence, our protocol is applicable to various application scenarios. We analyzed/simulated the overhead of our protocol, and showed the effectiveness of our design in enhancing the performance.

As discussed in Section 3, the key distribution algorithm allows the designer to choose the appropriate parameter, r, to determine the desired level of collusion resistance. With the use of this parameter, the base station maintains $rlog_r n$ (n is the number of sensors) secrets and each sensor maintains $log_r n$ secrets. With increased value of r, the collusion resistance increases, and each sensor maintains fewer secrets. As a tradeoff, the base station maintains more secrets, and the cost of creating/verifying the signatures increases. For example, in a 10x10 network, if we increase r from 2 to 10, the number of secrets maintained at the base station increases from 14 to 20. However, due to the facts that the symmetric key operation is very fast (e.g., as discussed in Section 1, the execution time of encrypting/decrypting a 8-byte block using RC5 is only 0.26ms) and these secrets are used only a few times during data dissemination, this increase in the cost of signing/verification is negligible. Hence, the performance of data dissemination changes minimally when we increase the level of collusion resistance.

References

[1] Crossbow Technology, Inc. MPR-MIB Users Manual, Revision B, June 2006, PN: 7430-0021-07. Available at:`http://www.xbow.com/Support/Support_pdf_files/MPR-MIB_Series_Users_M%anual.pdf`.

[2] S. M. Chang, S. Shieh, W. W. Lin, and C. M. Hsieh. An effi cient broadcast authentication scheme in wireless sensor networks. *The 2006 ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, March 2006.

[3] D. Coppersmith and M. Jakobsson. Almost optimal hash sequence traversal. *The Fifth Conference on Financial Cryptography (FC)*, February 2002.

[4] Crossbow Technology, Inc. *MICA2 Datasheet*. `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datas%heet.pdf`.

[5] Crossbow Technology, Inc. *Mote In-Network Programming User Reference Version 20030315*, 2003. http://webs.cs.berkeley.edu/tos/tinyos-1.x/doc/Xnp.pdf.

[6] H. Dai, M. Neufeld, and R. Han. ELF: An effi cient log-structured fash fi le system for micro sensor nodes. *The 2nd ACM Conference on Embedded Networked Sensor Systems (Sensys)*, November 2004.

[7] J. Deng, R. Han, and S. Mishra. Secure code distribution in dynamically programmable wireless sensor networks. *The Fifth International Conference on Information Processing in Sensor Networks (IPSN)*, April 2006.

[8] P. K. Dutta, J. W. Hui, D. C. Chu, and D. E. Culler. Securing the deluge network programming system. *The Fifth International Conference on Information Processing in Sensor Networks (IPSN)*, April 2006.

[9] R. Gennaro and P. Rohatgi. How to sign digital streams. *Lecture Notes in Computer Science*, 1294:180+, 1997.

[10] M. Gouda, S. Kulkarni, and E. Elmallah. Logarithmic keying of communication networks. *The Eighth International Symposium on Stabilization, Safety, and Security of Distributed Systems*, November 2006.

[11] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. *The 6th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, August 2004.

[12] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the second International Conference on Embedded Networked Sensor Systems (SenSys 2004)*, Baltimore, Maryland, 2004.

[13] C. Karlof, N. Sastry, and D. Wagner. Tinysec: A link layer security architecture for wireless sensor networks. *The 2nd ACM Conference on Embedded Networked Sensor Systems (Sensys)*, November 2004.

[14] I. Krontiris and T. Dimitriou. Authenticated in-network programming for wireless sensor networks. *The 5th International Conference on AD-HOC Networks and Wireless (Adhoc-Now)*, 2006.

[15] I. Krontiris and T. Dimitriou. A practical authentication scheme for in-network programming in wireless sensor networks. *ACM Workshop on Real-World Wireless Sensor Networks (REALWSN)*, 2006.

[16] S. S. Kulkarni and M. Arumugam. Infuse: A tdma based data dissemination protocol for sensor networks. *International Journal on Distributed Sensor Networks (IJDSN)*, 2(1):55–78, 2006.

[17] S. S. Kulkarni and M. G. Gouda. A note on instantiating security in sensor networks. Available at `http://www.cse.msu.edu/~sandeep/securitydistribution/`.

[18] S. S. Kulkarni and L. Wang. MNP: Multihop network reprogramming service for sensor networks. *In Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS)*, pages 7–16, June 2005.

[19] P. E. Lanigan, R. Gandhi, and P. Narasimhan. Sluice: Secure dissemination of code updates in sensor networks. *the 26th International conference on distributed computing systems (ICDCS)*, July 2006.

[20] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, Los Angeles, CA, November 2003.

[21] D. Liu and P. Ning. Multi-level μtesla: Broadcast authentication for distributed sensor networks. *ACM Transaction in Embedded Computing Systems (TECS)*, 3(4):800–836, November 2004.

[22] D. Liu, P. Ning, S. Zhu, and S. Jajodia. Practical broadcast authentication in sensor networks. *The 2nd Annual International Conference on MObile and Ubiquitous Systems: Networking and Services (MobiQuitous)*, pages 118–129, July 2005.

[23] D. Malan, M. Welsh, and M. Smith. A public-key infrastructure for key distribution in tinyos based on elliptic curve cryptography. *The 1st IEEE International Conference on Sensor and Ad Hoc Communications and Networks*, 2004.

[24] R. C. Merkle. Protocols for public key cryptosystems. *IEEE Symposium on Research in Security and Privacy*, pages 122–134, April 1980.

[25] V. Naik, A. Arora, P. Sinha, and H. Zhang. Sprinkler: A reliable and energy effi cient data dissemination service for wireless embedded devices. *To appear in Proceedings of the 26th IEEE Real-Time Systems Symposium*, December 2005.

[26] A. Perrig. The biba one-time signature and broadcast authentication protocol. *Proceedings of the Eighth ACM Conference on Computer and Communication Security (CCS-8)*, November 2001.

[27] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. *Seventh Annual International Conference on Mobile Computing and Networks (MobiCOM)*, July 2001.

[28] S. B. Qaisar and H. Radha. Opera, anoptimal progressive error recovery algorithm for wireless sensor networks. *The Fourth IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, June 2007.

[29] L. Reyzin and N. Reyzin. Better than biba: Short one-time signatures with fast signing and verifying. *The 7th Australian Conference on Information Security and Privacy (ACISP)*, pages 144–153, July 2002.

[30] G. Simon, P. Volgyesi, M. Maroti, and A. Ledeczi. Simulation-based optimization of communication protocols for large-scale wireless sensor networks. *In Proceedings of The IEEE Aerospace Conference*, pages 1339–1346, March 2003.

[31] T. Stathopoulos, J. Heidemann, and D. Estrin. A remote code update mechanism for wireless sensor networks. Technical report, UCLA, 2003.

[32] A. S. Wander, N. Gura, H. Eberle, V. Gupta, and S. C. Shantz. Energy analysis of public-key cryptography for wireless sensor networks. *The Third IEEE International Conference on Pervasive Computing and Communications (Percom)*, March 2005.

[33] H. Wang and Q. Li. Effi cient implementation of public key cryptosystems on mote sensors (short paper). *International Conference on Information and Communication Security (ICICS), LNCS 4307*, pages 519–528, December 2006.

[34] L. Wang and S. S. Kulkarni. Authentication for bulk data dissemination in sensor networks using symmetric keys. Technical Report MSU-CSE-07-14, Michigan State University, March 2007.