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

Limin Wang Sandeep S. Kulkarni

Software Engineering and Network Systems Laboratory

Department of Computer Science and Engineering

Michigan State University

East Lansing MI 48824 USA

Abstract

Authenticating bulk data dissemination in sensor networks is important as sensors need to verify that the data is truly from a trusted source. There are two ways to achieve authentication: asymmetric key based and symmetric key based. Although previous work has shown that asymmetric key authentication is feasible on sensor nodes if used sparingly, it is still quite expensive compared to symmetric key based approach. In this paper, we propose a symmetric key based protocol for authenticating data dissemination process. Our protocol uses the secret instantiation algorithm from [8,14] for distributing the keys. We apply the symmetric key signatures at the segment/group level and use hashed verification at the packet level. To improve the efficiency in the presence of packet loss/delay, we employ several techniques: the double connected hash chain, the caching scheme, and forward error correction (FEC). We show the effectiveness of our design through simulation. Moreover, since our protocol has much lower cost than the asymmetric key based approaches, it is especially valuable for the burst data dissemination, where the base station authenticates and transmits a moderate amount of data (1∼5KB) at a time.

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

[1] Email: {wanglim1,sandeep}@cse.msu.edu. Web: http://www.cse.msu.edu/~{wanglim1,sandeep}.

This work was partially sponsored by NSF CAREER CCR-0092724, DARPA Grant OSURS01-C-1901, ONR Grant N00014-01-1-0744, NSF equipment grant EIA-0130724, and a grant from Michigan State University.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2007		**00-00-2007 to 00-00-2007**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Authentication for Bulk Data Dissemination in Sensor Networks Using Symmetric Keys	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Michigan State University,Department of Computer Science and Engineering,Software Engineering and Network Systems Laboratory,East Lansing,MI,48824**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**25**	

1 Introduction

Reliably disseminating a large amount of data to every sensor in the network is a fundamental task for wireless sensor networks. We distinguish two types of bulk data dissemination. For one, the size of the data stream is *large* (>10KB), and the sender knows the entire data stream before authenticating/transmitting it. A typical example is network reprogramming, which re-tasks the network by disseminating a new program to all the sensors in the network. The base station authenticates the entire program and sends it all at once. The other case is what we call *burst* data dissemination, where burst/silent periods occur alternatively. A *moderate* amount of data (1∼5KB) is sent in each burst. The silent periods are typically long, during which the sender listens to the network or goes to sleep mode. An example of burst data dissemination is network monitoring, in which the base station monitors the health of the network, and broadcasts a sequence of commands once in a while. In this case, the data stream is generated and broadcast by the base station in real time. The entire data stream is not known to the base station in advance.

Data dissemination in sensor networks is performed via wireless radio, which is a broadcast medium, and is vulnerable to packet injection or corruption attacks. Moreover, the current data dissemination (reprogramming) protocols [10, 13, 16, 22, 27] are epidemic in nature. Once a false or viral command is accepted on one sensor, it could rapidly infect the entire network, and hence, lead to catastrophic damage. For these reasons, it is important that sensors be able to verify that the data/commands are from a trusted source.

In this paper, we are interested in providing authentication for this bulk data dissemination in sensor networks. Our goal is to provide a way that sensors can verify the authenticity and integrity of the received data.

Asymmetric key vs. symmetric key authentication. Authentication is a computation and energy expensive operation. Most authentication approaches rely on asymmetric digital signatures. In these approaches, a base station signs the packets using its private key. All the sensors have the public key of the base station, and can use it to verify the signatures. However, these asymmetric digital signatures are typically long (e.g., 50-1000 bytes per packet). Creating and verifying them have very high computation overhead. Communicating the long signatures to the sensors over radio also requires a lot of bandwidth and energy. 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 [9, 21, 28], they should still be avoided or used sparingly.

As an illustration of the cost of asymmetric key authentication, in a recent work [28], Wang and Li show that, if we use 1024-bit RSA on Mica motes, the time required for the public key operation is 0.79s. Since it takes 0.42ms to transmit a single byte of data over radio on Mica2 motes [11], the time for a public key operation is the same as the time for transmitting 1.88KB data. The asymmetric key authentication is especially inefficient for the burst data dissemination where the base

station sends a moderate amount of data at a time. For example, if the base station transmits 1.5KB data in each burst, the time required by the public key operation for authenticating a data burst even exceeds the time for the data transmission itself.

As an alternative, symmetric key based authentication needs much less energy/memory/computation resources, and hence, is expected to be more appropriate for resource constrained sensor nodes. For example, according to [11], 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. Hence, the cost of using symmetric key signatures is negligible. A simple approach is to use a single network-wide key shared by the base station and all the sensors [11]. The problem with this approach is that if one sensor is compromised and the key is captured (which has been shown to be relatively easy [4]), the entire network is no longer secure. Another symmetric key based approach is to share a pairwise key between the base station and each sensor. Although this approach is resistant to node compromise, it does not scale well to large networks. Moreover, with symmetric key based approaches, the number of keys that are stored at each sensor must be small.

Denial of service attack. Most sensor nodes have only a few kilobytes of RAM, yet the data from the base station are typically tens of kilobytes (consider the size of a program). A sensor must store the received data to the external flash (EEPROM) while it is accepting radio packets. If a large portion of the data that have been written to EEPROM fail in verification later, the data have to be erased. Since the energy cost of writing to external flash is high, the energy wasted in storing the false data is significant. Hence, the network is vulnerable to denial of service attacks, in which the adversary injects a lot of garbage packets that would eventually exhaust the sensors' energy.

According to their capacities, the adversaries can be categorized into two groups: mote-class adversaries and laptop-class adversaries. Mote-class adversaries have limited energy, and cannot launch extensive denial of service attacks. A laptop-class adversary can launch a denial of service attack by sending garbage data to the sensors. One way to mitigate laptop-class adversaries is to require that *only those packets that have been authenticated can be stored to flash* (as done in [5, 6]). In this way, the cost of failed verification is effectively reduced. However, this requirement can also increase the propagation time and energy usage significantly, because it potentially drops valid packets that cannot be verified.

Contribution of the paper. In this paper, we propose a symmetric key based protocol that authenticates the data dissemination process in sensor networks. In our work, the network consists of a base station and a collection of sensors. We note that the only communication that needs to be authenticated is the communication from the base station, rather than the communication between two arbitrary sensors. Utilizing this fact, we use a secret instantiation algorithm from [8, 14] to provide authentication. 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.

Signing each packet in the data stream to authenticate it is infeasible, as it incurs great computation and communication

overhead, even with a low cost signing algorithm. The practical alternative that reduces the overhead is to use a hash chain to link the packets, and sign the head of the hash chain [6, 7]. We apply a symmetric key signature to a group of packets, rather the entire data stream. For the reprogramming type of bulk data dissemination (which sends a large amount of data at once), we apply the symmetric key signature at the *segment* level. At a lower (i.e., packet) level, we use hashed verification, which has faster execution time and consumes less memory, compared to signatures. In the case of *burst* data dissemination such as network monitoring, the base station signs each data burst (in the case that a data burst is long, it divides the data burst into several parts, and signs each part), and uses the hash chain to link the packets within a burst. This approach essentially breaks one long hash chain into consecutive short hash chains. The advantage is that the base station only needs to know the group of packets it is going to transmit, rather than having to acquire/authenticate the entire data stream at once. As we pointed out, the symmetric key authentication has very low cost, hence, it does not incur much overhead even though we apply it multiple times. Hence, our protocol can be used for authenticating the online data stream (or burst data dissemination). Moreover, since the base station starts transmitting data as soon as it signs the first group of packets, it allows a quick start for both types of bulk data dissemination.

We consider both mote-class adversaries and laptop-class adversaries. Since one of our goals is to protect the sensor network in the event of a denial of service attack, we require the authentication for a packet be done before the packet is stored to EEPROM. For bulk data dissemination that only tolerates mote-class adversaries, we refer readers to [29].

We note that although the basic hash chain scheme has very little computation overhead, it does not tolerate packets arriving out of order, which is the typical case due to packet loss in wireless sensor networks. To overcome this problem, we design a *double connected hash chain* to achieve stronger resistance against packet loss. We will show in Section 4 that this scheme greatly improves performance of data dissemination in the presence of loss/delay. Moreover, on the receiver side, we introduce a group cache scheme to improve efficiency. We investigate the tradeoff between the memory consumption (the size of the cache) and performance. We also study the effect of applying forward error correction (FEC) to reduce the impact of packet loss/delay.

We apply our algorithm to MNP [16], a data dissemination algorithm for wireless sensor networks. We show the performance of our algorithm and the effectiveness of our design (the double connected hash chain, caching, FEC) through simulation on TOSSIM [18].

Organization of the paper. In Section 2, we describe the system model and security requirements of the secure data dissemination problem. In Section 3, we present our authentication protocol. We focus on the secret instantiation algorithm, the double connected hash chain, the caching scheme and applying FEC scheme. In Section 4, we evaluate our approach (integrated with MNP [16]) in terms of computation cost, memory overhead, delay, energy consumption, and communication cost. We also investigate the effectiveness of the techniques we have designed for our protocol. We survey related work in Section 5 and conclude in Section 6.

2 System Model and Security Requirements

The goal of this paper is to design a secure data dissemination protocol. Our protocol is targeted for TinyOS mote platforms, such as Mica2/XSM mote [1, 2] or TelosB mote [3]. It is supposed to be used with the existing data dissemination protocols, such as MNP [16] and Deluge [10], in which the code image is propagated from a base station to all the sensors in the network. In the remainder of this paper, we use MNP as an example. However, we note that our protocol can be applied to other data dissemination protocols, such as Deluge [10], Infuse [13], as well. In Section 2.1, we give an overview of MNP. In Section 2.2, we describe the threat model and security requirements.

2.1 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. MNP applies the advertise-request-data three-way handshake interface [12]. Data are sent in *segments*. Each segment contains K packets; the last segment may contain fewer 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 can 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. This model provides efficient and reliable transmission in a highly lossy and unstable wireless environment.

Every sensor in the network needs to receive the data stream from the base station. Once the sensors receive part of the data stream (one or more segments), they can advertise and forward the program to their neighbors. Message collision becomes a major problem when multiple sensors that are close to each other are trying to transmit at the same time. To reduce the message collision problem, MNP uses a sender selection algorithm to try to guarantee that there is only one sender in a neighborhood at a time (refer to [16] for details). If a sensor wins in the sender selection algorithm and becomes the sender, it broadcasts a "StartDownload" message several times, then starts transmitting the requested data packets in the segment. At the end of the transmission, the sender broadcasts a "TerminateDownload" message to inform the receivers that the transmission has finished. The sensors that are not transmitting or receiving data are put to sleep state to save energy.

Since some techniques we present in Section 3 are used to deal with packet loss, we describe the reliability scheme for MNP in more details. In MNP, each packet has a unique ID, from 1 to the size of the segment. Each receiver is responsible for detecting its own loss. A node maintains a bitmap (which we call *MissingVector*) of the current segment it is receiving in memory. Each bit in *MissingVector* corresponds to a packet. All the bits are initially set to 1. When a node receives a packet for the first time, it stores that packet in EEPROM and sets the corresponding bit in *MissingVector* to 0. The *MissingVector* is

attached in the request message and sent to the sender.

A node that is advertising maintains a *ForwardVector*, which is a bitmap of the advertised segment, and is an indicator of the packets the node needs to send if it becomes a sender. The *ForwardVector* of an advertising node is the union of the *MissingVectors* in the *request* messages that the node has received. A node only sends the packets indicated in the *ForwardVector*.

2.2 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.

Moreover, observe that authentication is sufficient for data dissemination in sensor networks. In other words, it suffices to ensure that the sensors can be assured that the data are from the trusted source. However, in this application, 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. Compromise resilience. Because current most-class devices are not tamper-resistant, it is relatively easy to compromise a sensor. It must not be possible that compromising a single sensor node will cause the other parts of the network insecure.
3. Low cost. The security scheme should be efficient in terms of computation, communication, and memory usage, and energy consumption. Moreover, it should not add long delay to the data dissemination process.
4. Denial of service attack resilience. 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

In this section, we describe our secure data dissemination protocol. In Section 3.1, we describe the secret instantiation algorithm [8, 14], which we use to distribute the secrets (i.e., keys) and create/verify the signatures. In Section 3.2, 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 3.3) and using forward error correction (FEC) (Section 3.4). We will study the effect of applying these schemes in Section 4.

3.1 Secret Instantiation

The base station has a collection of secrets. Initially, each sensor receives some subset of this collection. The base station knows the secret distribution, i.e., it knows the subset of secrets received by each sensor. 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 byte) 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, 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. As mentioned above, 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.

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: Each sensor has a unique ID that is of length $log_r n$, thus it is associated with a unique combination of $log_r n$ secrets. Only the base station contains all the secrets. Therefore, no other sensor, except the base station, has all the secrets that sensor j has. Hence, if j verifies $log_r n$ signatures, it is assured that the message originated at the base station. □

To illustrate the algorithm, we show an example in Figure 1. 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.

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 users may be able to obtain all the keys and, thereby,

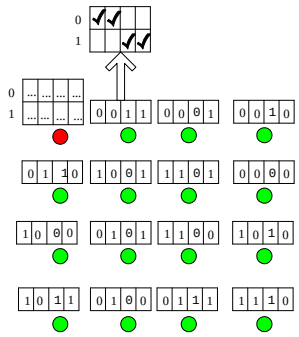


Figure 1. Secret instantiation: an example.

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.

In our experiments, for simplicity, we used the base $r = 2$ thereby choosing the least number of secrets at the base station. Hence, in a 10x10 network (100 sensors), the base station maintains 14 ($2log_2100$) secrets and each sensor maintains 7 (log_2100) secrets. In this case, collusion of 2 users with complementary IDs (e.g., a sensor with ID 1010 and a sensor with ID 0101) can allow them to pretend to be the base station. If higher collusion resistance is desired, the designer can choose a higher base. For example, if $r = 10$ is used for a 10x10 network, then the number of secrets maintained at the base station increases to 20 (as compared to 14 when $r = 2$). Since these secrets are used only a few times during data dissemination, it will not affect the performance (time/energy) significantly.

On the other hand, with increased value for r, not only the collusion resistance increases, but also the number of secrets maintained by each sensor ($log_r n$) decreases. Thus, providing higher level of collusion resistance does not adversely affect the sensors. For $r = \sqrt{n}$, the algorithm corresponds [14] to the grid algorithm in [15]. For $r = n$, the algorithm corresponds to the case where each sensor maintains a unique secret that is known only to that sensor and the base station. In this case, collusion between sensors does not allow them to pretend to be the base station.

3.2 Authenticate The Data Stream

In this section, we present our algorithm to authenticate the data stream using the secret instantiation algorithm and hashed verification. First, we show the basic hash chain approach from [7]. Then, we discuss the problem with the basic hash chain, and propose our double connected hash chain to reduce this problem.

Assume that the entire data stream (or a data burst, in the case of burst data dissemination) has N ($N \geq 1$) segments. Each segment contains K packets, possibly with the exception of the last segment. 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). As we apply one symmetric key signature to each segment, we can construct one hash chain per segment. In Figure 2, we

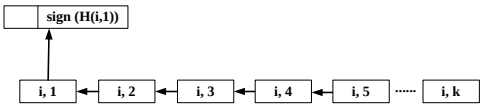


Figure 2. The basic hash chain (segment i).

show the basic hash chain approach for segment i. The hash of packet $P(i,j)$ is denoted as $H(i,j)$. A data packet $P(i,j)$ has two parts, the data part and a hash of the next packet (not shown in Figure 2 for clarity). In Figure 2, an arrow pointing from packet j to packet i indicates that packet i contains the hash of packet j. If $P(i,j)$ is the last packet of the data stream/burst, then the hash is 0. If $P(i,j)$ is the last packet of segment i ($1 \leq i < N$), the hash part can be either 0 or the hash of the first packet of segment $i+1$, depending on the availability of the first packet in segment $i+1$ at the moment that the hash chain of segment i is constructed (in the second case, the multiple hash chains for the individual segments are connected into a single chain). Hence, a data packet in a basic hash chain can be represented as in Figure 3 (assuming that the first packet of segment $i+1$ is available).

if $P(i,j)$ is the last packet of the data stream/burst

$P(i,j) = data(i,j)||0$

else if j is the last packet of segment i

$P(i,j) = data(i,j)||H(i+1,1), i = 1..N-1, j = K$

else

$P(i,j) = data(i,j)||H(i,j+1), i = 1..N, j = 1..K-1$

endif

Figure 3. Representation of a data packet $P(i,j)$ in the basic hash chain (assuming that the first packet of segment $i+1$ is available).

Note that the hash is computed over the entire packet, not just the data part. The base station signs the hash of the first packet in the segment, which is the head of the hash chain, using all the secrets. We denote the signatures of segment i as $sign(H(i,1))$ in Figure 2.

With the basic hash chain mechanism, a sensor can verify a data packet $P(i,j)$ if and only if it has received and verified all the packets in the hash chain proceeding $P(i,j)$. It implies that the data packets have to be *verified* in order. This is inefficient in the events of packet loss/delay. For example, if all the packets in segment i have been received except packet 2, none of the packets after packet 2 in the chain can be verified. They have to be thrown away if there is not enough memory to cache them.

As we can see, in the basic hash chain scheme, a single packet loss/delay can lead to erasure of many valid data packets. This leads to significant energy waste. Our simulation results (as well as the results from [5, 6]) show that if we require that

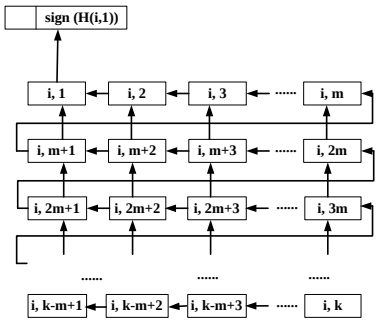


Figure 4. The double connected hash chain (segment i).

the data packets be received and stored in order, data dissemination process will be delayed significantly (e.g., the propagation time increases by 6 or 7 times if we use the default segment size 128 packets/segment in MNP). Correspondingly, the energy consumption also increases by a large amount.

To address this problem, we propose a double connected hash chain to enhance the inter-connection among the packets, as illustrated in Figure 4. As shown in Figure 4, 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 two hashes (rather than one): 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 this way, we construct multiple authentication paths for verifying a data packet. Consider the example we have discussed, i.e., a single packet (packet 2) is lost/delayed, while all other packets are received. If we use the double connected hash chain and suppose the size of hash group m is greater than 2, all the packets starting from packet m+1 can be authenticated because packet 1 contains the hash for packet m+1.

If $P(i, j)$ is the last packet of the data stream/burst, the two hashes are both 0. And similar to constructing the basic hash chain, if the first hash group of segment $i + 1$ is available at the moment that the base station computes signatures and hashes for segment i, the packets in last hash group of segment i contain the hashes of the corresponding packets in the first hash group of segment $i + 1$. Hence, we concatenate the hash chains for the individual segments into one chain. In the case that no data in the next segment is available, those hashes can be set to 0. Similarly, we represent a packet $P(i, j)$ in a double connected hash chain as in Figure 5 (assuming that the packets in the first hash group of segment $i + 1$ are available).

3.3 Caching

Data cache and hash cache. Caching is an effective technique to deal with packets that are delayed or temporarily lost. 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. For example, if we set the packet size to 70 bytes, excluding 6

if $P(i,j)$ is the last packet of the data stream/burst

$P(i,j) = data(i,j)||0||0$

else if j is the last packet of segment i

$P(i,j) = data(i,j)||H(i+1,1)||H(i+1,j+m-K), i = 1..N-1, j = K$

else if j is in the last hash group of segment i

$P(i,j) = data(i,j)||H(i,j+1)||H(i+1,j+m-K), i = 1..N-1, j = K-m+1..K$

else

$P(i,j) = data(i,j)||H(i,j+1)||H(i,j+m), i = 1..N, j = 1..K-m$

endif

Figure 5. Representation of a data packet $P(i,j)$ in the double connected hash chain (assuming that the packets in the first hash group of segment $i+1$ are available).

bytes for the header, storing one packet needs 64 bytes. In this case, caching the a 64-packet segment requires 4KB memory, which exceeds the memory capacity of some sensor platforms, such as Mica2 motes. To reduce the memory consumption, we choose to cache a *cache group* at a time, rather an entire segment.

We describe the strategy of using cache groups in the context of MNP. Recall that in MNP, if a sensor is selected as a sender for a segment, it transmits the packets that are requested by its neighbors. The packets that need to be transmitted are indicated in the sender's *ForwardVector* (c.f. Section 2). The sender divides the data packets that it is going to transmit (those indicated in its *ForwardVector*) into equal-sized *cache groups*. As the cache groups are computed based on the sender's *ForwardVector*, the receivers need to receive it as well. Hence, the sender puts its *ForwardVector* in the "StartDownload" message, and broadcasts it before the data transmission starts.

A receiver uses the data cache and the hash cache to store the data packets and hashes, respectively. A data cache can accommodate one cache group at a time. The cache group that is currently kept in the data cache is called *the active cache group*. When a receiver receives a data packet, it uses the sender's *ForwardVector* to determine the cache group this packet belongs to. Consider an example that the sender transmits packets 2, 3, 9, 10, 12, 15, 17, 24, 30, 32, 36, 40, 41, 42, 47, 50, 52, 59. The size of the segment is 64, and the size of the cache group is 8. The first 8 packets are in the first cache group, and the next 8 packets are in the second cache group, and the remaining 2 packets are in the third cache group. Moreover, each packet is assigned a specific slot. For example, packet 9 is the third packet in the first cache group, it will be written to the third slot in the data cache. Note that when we write a data packet to the data cache, we write the data part as well as the two hashes to the data cache. By doing this, when we authenticate a packet, we authenticate the hashes contained in the packet at the same time. Only those hashes that have been authenticated are written to the hash cache.

We show the data cache in Figure 6, assuming that the active cache group is 2. Each item in the data cache contains a packet ID, the data part, and two hashes. Moreover, it also contains a *valid* bit, which is either 0 or 1. This is used to identify

30	0	Data	H(i,31)	H(i,38)
32	1	Data	H(i,33)	H(i,40)
36	1	Data	H(i,37)	H(i,44)
40	1	Data	H(i,41)	H(i,48)
41	0	Data	H(i,42)	H(i,49)
42	1	Data	H(i,43)	H(i,50)
47	1	Data	H(i,48)	H(i,55)
50	0	Data	H(i,51)	H(i,58)

Figure 6. The data cache contains a cache group. The segment size is 64 packets, the size of the cache group is 8. Each item in the data cache contains the packet ID, the valid bit, the data part and the two hashes.

if the packet is in the data cache. If *valid* is 1, then the packet is in the data cache. Otherwise, it is not. There are two situations that a *valid* bit becomes 0: the receiver losses the packet due to radio interference; or the receiver does not need the packet since it has already received (and authenticated) it in a previous transmission.

Actions of a sensor when it receives a data packet. Similar to MNP, a receiver maintains a *MissingVector*, which is a bit map of the segment, indicating the packets that are missing in this segment. All the bits in a MissingVector are set to 1 when a sensor starts receiving a segment. If a packet is received *and* authenticated, the corresponding bit for this packet in the *MissingVector* is set to 0.

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 , all the items in the data cache are erased. If there are data packets in the data cache that are not yet authenticated, these packets are thrown away. 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 (i.e., external flash for motes), and the hashes are written to the hash cache (if they are not already in the hash cache). Also, the bit representing this packet in the *MissingVector* is cleared. The actions that a sensor takes when it receives a data packet is summarized in Figure 7.

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. We show this authentication process Authenticate(p) as a recursive function in Figure 8. It terminates when no more authentication occurs. Consider the example that packet 2 is received later than packets 3 and 4 in the active cache group. When packet 2 arrives, all the three packets 2, 3, 4 are authenticated and written to EEPROM. 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.

```
if a data packet p arrives & the sensor needs p
    compute p's cache group g(p)
    if g(p) != ActiveCacheGroup
        clear the data cache, and set ActiveCacheGroup to g(p)
    endif
    save p to the data cache, set the valid bit for this item to 1
    if the hash cache contains H(p)
        Authenticate(p) (c.f. Figure 8)
    endif
endif
```

Figure 7. Actions of a sensor when it receives a data packet.

Note that in our protocol, only the base station has all the secrets, and can compute the hashes and create the signatures for the hash of the first packet in a segment (i.e., $sign(H(i,1))$). All other sensors buffer the signatures and hashes, and forward them if they are selected as a sender. The sender puts the signature $sign(H(i,1))$ and $H(i,1)$ (as well as its *ForwardVector*, as we discussed in Section 3.3) in the "StartDownload" message. When a receiver receives the "StartDownload" message from a sender for a segment for the first time, it decrypts the signature using the collection of secrets it has, and verifies $H(i,1)$. If all the verifications are successful, it saves $H(i,1)$ to the hash cache.

In the double connected hash chain, if the packets in the last hash group of the current segment contain the hashes for the packets in the first hash group of the next segment, these hashes should also be temporarily cached. This extra hash cache is normally small. In the case that the size of a hash group (m) is 8 and a hash is 4 bytes long, this extra hash cache is only 32 bytes. These hashes can be used to authenticate the first m packets in the next segments.

```
Authenticate(p): p has a data part data(p) and two hashes H(p+1) and H(p+m)
    if p is in the data cache & p has not been authenticated (the bit for p in MissingVector is 1)
        write data(p) to EEPROM (or add data(p) to the writeEEPROM queue)
        clear the bit for p in MissingVector
        if the hash cache does not have H(p+1)
            add H(p+1) to the hash cache
            Authenticate(p+1)
        endif
        if the hash cache does not have H(p+m)
            add H(p+m) to the hash cache
            Authenticate(p+m)
        endif
    endif
```

Figure 8. Authenticate a data packet p.

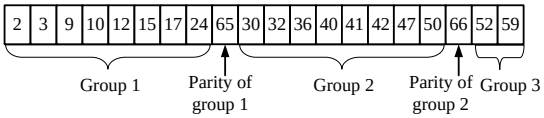


Figure 9. An example of the data and parity packets sent by a sender. The segment size is 64 packets, the size of FEC transmission group is 8.

3.4 Forward Error Correction

MNP, as well as all other existing data dissemination protocols [10, 13, 22, 27], 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. Using ARQ schemes can effectively recover packet loss as long as the network is connected. However, if the packet loss rate is high, the requests and retransmissions for the missing packets consume significant energy. In our protocol, we use forward error correction (FEC) to reduce packet loss.

FEC provides reliability by transmitting redundant packets in a proactive manner. There are different levels of FEC schemes: packet-level, byte-level, bit-level. We consider FEC at the packet level. The most commonly used FEC scheme is (n, k) FEC. The fundamental of (n, k) FEC is to add $n - k$ additional packets to a group of k source packets (called a *transmission group*) so that the receipt of any k packets at the receiver permits recovery of the original k ones. Consider the extremely limited computation and memory resources of a sensor node, 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. Therefore, simple XOR code is a $(k + 1, k)$ code. XOR code is very simple to implement. However, it can only 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. To distinguish a data packet and a parity packet, we set the packet ID of a parity packet to be the size of a segment plus its transmission group ID (which starts from 1). In Figure 9, we show an example of the data packets and parity packets sent by a sender, assuming the segment size is 64, and the size of FEC transmission group is 8.

A receiver uses the sender's *ForwardVector* to decide the transmission groups and decode the parity packet. 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 (by looking at its *MissingVector*). 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 (they are in the active cache group and their *valid* bits are 1), decoding for the parity packet can be conducted directly. In the case that some data packets are received in earlier transmissions and are not in the data cache, they must be read from EEPROM for decoding. As reading a packet from EEPROM is an optimized operation with low cost, employing simple XOR code in our protocol does not incur much computation overhead.

3.5 Hash Group/Cache Group/FEC Transmission Group

Now that we have defined three types of groups: hash group, cache group, and transmission group for FEC code. Hash group is used for constructing the double connected hash chain . As all the hashes and signatures are computed at the base station , the data part and hash part of a data packet are independent of the senders and remain unchanged during the data dissemination process.

The cache groups and FEC transmission groups are dependent on specific transmissions. In other words, to decide the cache group or FEC transmission group a data/parity packet belongs to, a receiver needs to know the sender's *ForwardVector*, i.e., the packets the sender transmits in this transmission. For example, if the size of the cache group is c (respectively, if the size of the FEC transmission group is t), then first c (respectively, t) packets that are transmitted by the sender are in the first cache group (respectively, FEC transmission group), the next c (respectively, t) packets that are transmitted are in the second cache group (respectively, FEC transmission group), and so on. The size of the cache group is decided by the size of the data cache. The size of the FEC transmission group should not be larger than the size of the cache group.

4. Evaluation

In this section, we evaluate the performance of our secure data dissemination protocol. We integrate our protocol with MNP [16] as described in Section 3. We refer to the integrated protocol as *MNP+Auth*. We simulate *MNP+Auth* using TOSSIM [18]. 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. 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 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. As we described in Section 3, before sending a segment, the base station computes all the hashes and signs the hash of the first packet of the segment using all the secrets. The signatures and the hash of the first packet, as well as the *ForwardVector* are contained in the "StartDownload" message. We assume that sensors use the last two bytes of the signatures, hence, the length of the 14 signatures is 28 bytes. The hash is 4 bytes, and the *ForwardVector* is 8 bytes. These fit into a single "StartDownload" message.

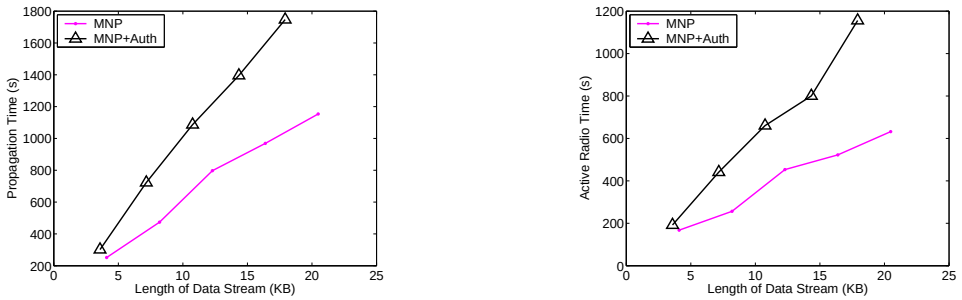
We analyze the computation cost and memory requirement of *MNP+Auth* in Sections 4.1 and 4.2. Then, in Section 4.3, we show the performance of *MNP+Auth* in terms of propagation time, active radio time, communication overhead. In MNP, the sensors are put to sleep when they are not expected to transmit or receive data. This effectively reduces energy consumption, because idle listening contributes to more than 90% of the energy consumption. The active radio time is a good indication of the amount of energy consumption during the data dissemination. In Section 4.4, we investigate how our design decisions (i.e., the use of the double connected hash chain, the caching and FEC techniques) affect the performance.

4.1 Computation Cost

The computation cost of the algorithm includes signing/verifying the hashes, computing the hashes, and computing FEC parity packets. As we mentioned, all the hashes and signatures are computed only once at the base station. The sensors simply use the signatures received from the base station. Moreover, since we use symmetric keys for signatures, the overhead is much lower than (about 0.03%$\sim$0.05%) 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. Also, because we use the simple XOR FEC code, encoding/decoding parity packets has very low computation cost.

4.2 Memory Requirement

Our authentication protocol has memory requirement in the following ways. First, $\log_r n$ secrets are maintained at each sensor. When r increases, the number of secrets maintained at the sensor decreases. In a 10x10 network, the number of secrets maintained at each sensor is at most 7. Second, the data cache and the hash cache (and possibly the extra hash cache, c.f. Section 3.3) 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 (this ID is used inside a segment) and the *valid* bit, the data cache requires 521 bytes. As we discussed in Section 3.3, the hash cache needs 256 bytes memory, assuming the size of a segment is 64 packets. The extra hash cache only requires 32 bytes. Third, the signatures from the base station for each segment need to be stored either in memory or in flash. Fourth, a variable to record the current active cache group. This only needs to be 1 byte long. Fifth, the signing/verification process consumes some amount of memory. This amount of memory is



(a) (b)

Figure 10. Propagation time and active radio time of MNP and *MNP+Auth*. (a) propagation time vs. length of data stream (b) active radio time vs. length of data stream.

much lower compared to the asymmetric key based approaches. And we discussed in Section 3.4, as the data cache provides the memory required by FEC encoding/decoding, there is no extra memory overhead by applying FEC scheme.

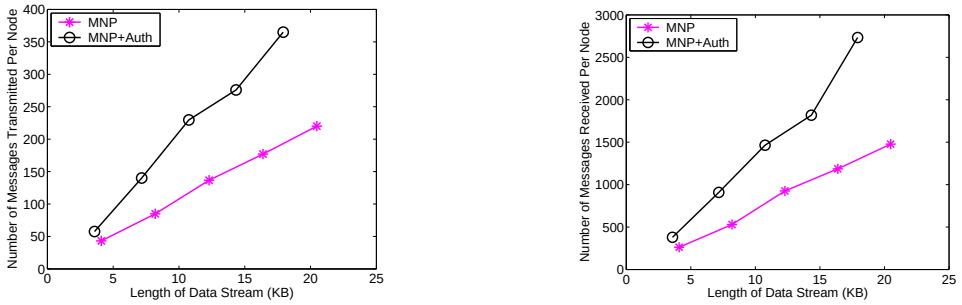
4.3 Performance of *MNP+Auth*

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 10, we show the propagation time and active radio 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. In both protocols, the active radio time of sensors is around 50-60% the propagation time. In Figure 10, we show that the active radio time of *MNP+Auth* is 30%-100% higher than the active radio time of MNP.

In Figure 11, we show the communication overhead of our authentication protocol. In Figure 11(a), we can see that for a given amount of data to be distributed, the message transmission *MNP+Auth* is 52%-92% higher than that of MNP. The message reception pattern is similar to the active radio time (Figure 10(b)).

4.4 Performance Analysis

The delay and communication overhead of our authentication protocol presented in Section 4.3 is mainly caused by two facts. For one, there is not enough memory to store all the received packets in a segment. If, some packets that have not yet been authenticated have to be removed from the data cache, in order to make room for the newly arrived packets, these items are thrown away. These packets are requested and retransmitted later. This results to delay and energy waste. For the other,



(a) (b)

Figure 11. Communication cost of authentication. (a) Number of message transmitted per node vs. length of data stream (b) Number of message received per node vs. length of data stream.

the hashes attached in each packet are the extra data to distribute. In our double connected hash chain, 8 out of 64 bytes are for hashes, i.e., 14.3% extra packets need to be transmitted for distributing the hashes.

We first compare the basic hash chain approach and the double connected hash chain approach in Section 4.4.1. Then, we study how the size of the data cache affects the performance in Section 4.4.2. As the limitation on memory space is one reason that causes authentication delay, we are interested in seeing how much performance improvement we can get by increasing the size of the data cache. Finally, in Section 4.4.3, we study the effect of employing FEC code in our protocol.

4.4.1 Comparing Basic Hash Chain with Double Connected Hash Chain

We apply the basic hash chain to MNP, and compare the performance with that of the double connected hash chain in Table 1. For better comparison of the two hash chains, we allow the basic hash chain approach to use caches in the same way as we designed for the double connected hash chain approach, i.e., use a data cache to temporarily buffer the received data packets, and a hash cache to buffer the hashes. We set the size of the data cache to 8 packets. And we also apply FEC in both cases.

The length of the data stream is 7.2KB. In the case of the double connected hash chain approach, the base station transmits 128 packets. Because in the basic hash chain, each packet carries only one hash (compared to two hashes in the double connected hash chain), the actual data that is carried in each packet is 4 bytes more than that using the double connected hash chain. Hence, we let the basic hash chain protocol to send 9 less packets (119 packets), so that the actual data sent by the two protocols are the same.

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 and the active radio time by 73%. The message transmission and reception using the double connected hash chain are also much

Table 1. Compare the basic hash chain with the double connected hash chain. Length of data stream: 7.2KB.

	Using the basic hash chain	Using the double connected hash chain
Propagation time	2654 (s)	723(s)
Active radio time /node	1652 (s)	441 (s)
Messages sent /node	602	140
Messages received /node	3815	908

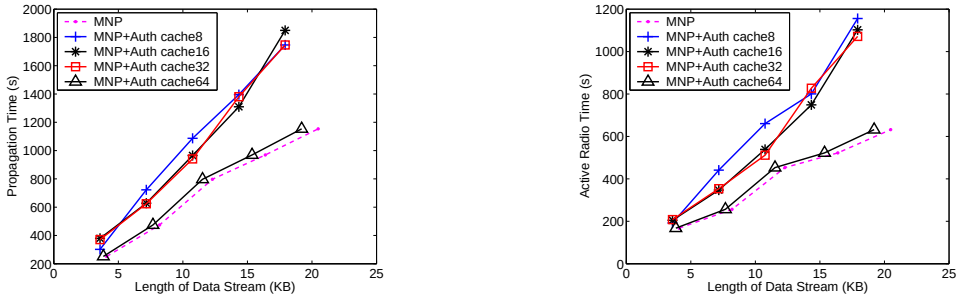
lower.

4.4.2 Varying the Size of the 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 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 12, we show the propagation time and the active radio 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 12, when the data cache size is 8 packets, 16 packets, and 32 packets, the lines that represent the propagation time in Figure 12(a) (and respectively, the active radio time in Figure 12(b)) intersect with each other. In other words, there is no significant performance improvement when we increase the size of the data cache.

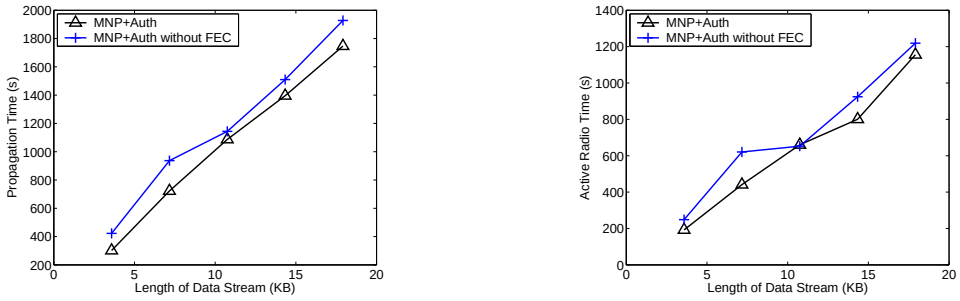
By increasing the cache size, we are able to cache more hash groups in memory. This helps to improve the performance only if some delayed packets in the earlier group(s) arrive later. However, this long delay of packets is uncommon (although short delays of packets within a cache group are common) for 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. On the other hand, if we reduce the size of the data cache to below 8 packets or completely remove the data cache, the propagation time and energy consumption increase significantly because a lot of valid packets that cannot be authenticated are discarded.

If we increase the data cache size to 64 packets, the propagation time and active radio 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 performance is only a little higher than that of MNP.



(a) (b)

Figure 12. Varying the data cache size from 8 packets to 64 packets. (a) propagation time vs. length of data stream (b) active radio time vs. length of data stream.



(a) (b)

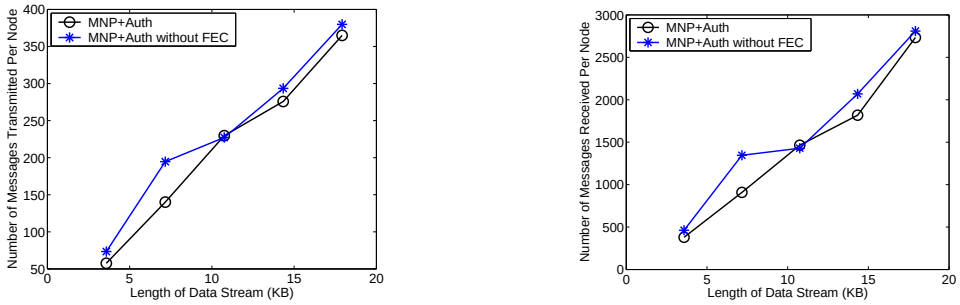
Figure 13. Effect of using FEC (propagation time and active radio time). The data cache size is 8 packets. (a) propagation time vs. length of data stream (b) active radio time vs. length of data stream.

4.4.3 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 13, 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 and the active radio time by up to 29%.

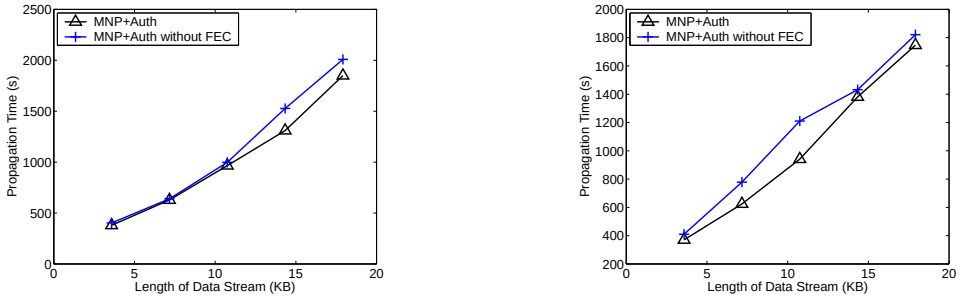
In Figure 14, we show the message transmission and reception in the two cases, *MNP+Auth* with FEC enabled (the default case) and with FEC disabled. We can see that using FEC also reduces message transmission and reception.

We change the size of the data cache to 16 packets and 32 packets, and repeat the same simulation. From Figure 15, we



(a) (b)

Figure 14. Effect of using FEC (communication overhead). The data cache size is 8 packets. (a) number of messages transmitted per node vs. length of data stream (b) number of messages received per node vs. length of data stream.



(a) (b)

Figure 15. Effect of using FEC under different data cache sizes. (a) when data cache is 16 packets, propagation time vs. length of data stream (b) when data cache is 32 packets, propagation time vs. length of data stream.

can see that using FEC can reduce the propagation time in both cases.

5. Related Work

In this section, we review related work in the areas of bulk data dissemination, authenticated broadcast, secure data dissemination, and FEC schemes.

Bulk Data Dissemination (reprogramming). The work on bulk data dissemination (reprogramming) for sensor networks includes MOAP [27], Deluge [10], MNP [16], Infuse [13], Sprinkler [22]. Although we only showed how to integrate our authentication protocol with MNP, it can be used with other reprogramming protocols as well. For example, if we want to authenticate Deluge reprogramming process, the only difference compared to *MNP+Auth* is that the signatures and the hash

of the first packet for each segment are attached with advertisement messages, rather than "StartDownload" messages in *MNP+Auth*. Our protocol can also be used to authenticate TDMA based reprogramming protocols, such as Infuse [13] and Sprinkler [22]. In this case, we can think of the whole program as a big segment. The integration process is straightforward.

Authenticated broadcast. A. Perrig *et. al.* proposed TESLA [24] and μTESLA [25] to provide broadcast authentication through a hash chain. μTESLA is designed to work on the resource-constrained sensor nodes. It applies symmetric keys, and achieves asymmetry for authentication by delaying the disclosure of the symmetric keys. BiBa [23] is another protocol that performs authenticated broadcast via pre-computed hash collisions and chains. However, all these protocols, TESLA, μTESLA, and BiBa require *loose time synchronization*, and hence, introduce additional constraints and overhead.

Secure data dissemination (reprogramming). Researchers have shown interest in secure data dissemination for sensor networks recently. The two protocols, Sluice [17], proposed by P. Lanigan *et. al.*, and SecureDeluge [6], proposed by P. Dutta *et. al.*, use the basic hash chain for authentication. Sluice verifies hashes at the segment level, while SecureDeluge verifies hashes at the packet level. Although segment-level hash chain has low computation, communication and memory cost, Sluice is vulnerable to some form of attacks, e. g., a single corrupted/lost packet will cause the entire segment to be discarded. The problem with the basic hash chain approach, as we discussed, is that it requires that packets be received/stored in order. All the packets that arrive out of order are thrown away. This requirement increases the delay and message cost significantly, especially when the network is lossy. There is another protocol proposed by J. Deng *et. al.* [5], which tries to address the problem by sending a hash tree over the data packets before sending the actual data packets. After sensors have received the entire hash tree, they can receive/verify data packets that arrive out of order.

All these three protocols mentioned above are based on the asymmetric key signatures. By contrast, our protocol is symmetric key based, which has much lower overhead (2000 to 3000 times faster in the example given in Section 1) than the asymmetric key based approaches. Our protocol is especially useful in the burst data dissemination, where the base station sends a moderate amount of data during a burst, and then waits for a long time before sending the next burst. Since the base station does not know the entire data stream in advance, it can only authenticate one burst of data at a time. It is desirable to use symmetric key signatures so that the cost of signing/verifying each data burst is low.

Forward error correction (FEC) codes. Simple XOR code and Reed-Solomon codes belong to block (n, k) FEC codes. A block code has the property that any k out of the n encoding packets can reconstruct the original k source packets. We have introduced simple XOR code in Section 3.4. Compared to XOR code, Reed-Solomon (RS) code [26] provides better flexibility at high processing cost, in terms of computation complexity and memory space.

Tornado codes [20] provide an alternative to RS codes. Tornado codes have lower computation complexity than RS codes, at the small cost of reception overhead, that is, a (n, k) Tornado code requires slightly more than k out of n encoding packets to recover k source packets.

Unlike the block (n, k) codes, Luby Transform (LT) codes [19] can generate as many unique encoding packets as required, using the k source packets as input. Each encoding packet is generated randomly and independently of all other encoding packets. LT codes have the property that the receiver is able to reassemble the original k source packets as long as it receives enough number (slightly more than k) of encoding packets. LT codes are designed for delivering a large amount of data over high bandwidth internet links. They have lower computation complexity on encoding and decoding than RS codes. However, they introduce higher recovery overhead because more redundant packets are transmitted. Normally, the number of repair packets are more than 10 times the number of source packets.

Due to the extreme resource constraints of sensor nodes, we use the simple XOR code, which requires the least amount of computation and memory. In our previous work [30], we proposed to apply simple XOR code or RS codes to the bulk data dissemination protocols to reduce packet loss. We did a case study on MNP, and showed that using simple XOR code to MNP can reduce the propagation time by up to 10% and reduce the active radio time by up to 18%. Using RS codes can achieve better performance, but has higher computation complexity.

6. 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 [8, 14] 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 applied the symmetric key signature at the segment level, and use hashed verification at the packet level. 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 illustrated this algorithm in the context of the MNP [16]. Our approach can also be easily applied to other data dissemination protocols such as [10, 13]. We analyzed/simulated the overhead of our protocol in terms of computation cost, memory requirement, propagation time, active radio time and communication cost. We also show the effectiveness of our design: the double connected hash chain, the caching scheme and using FEC code, through simulation.

As discussed in Section 3.1, 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 $rlog_rn$ (n is the number of sensors) secrets and each sensor maintains log_rn 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., the execution time of encrypting/decrypting a 8-byte block using RC5 is only 0.26ms, c.f. Section 1) 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 remains the same when we increase the level of collusion resistance.

As discussed in Section 5, our symmetric key based authentication algorithm is expected to be especially valuable for authenticating the burst data dissemination such as network monitoring, where the base station enters burst/silent periods alternatively, and sends a moderate amount of data during each burst. For such case, the base station does not know the entire data stream in advance. It must authenticate each data burst separately. Since each data burst is not long, it is desirable to use a low cost algorithm to authenticate them. Our algorithm uses symmetric key signatures to sign each data burst, and hence, provides authentication for such burst data dissemination with a low overhead.

References

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

[2] Crossbow Technology, Inc. *MSP410 Datasheet.* `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MSP410_Data%sheet.pdf`.

[3] Crossbow Technology, Inc. *TELOS-B Datasheet.* `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/TelosB_Data%sheet.pdf`.

[4] J. Deng, R. Han, and S. Mishra. Practical study of transitory master key establishment for wireless sensor networks. *the 1st IEEE/CreateNet Conference on Security and Privacy in Communication Networks (SecureComm)*, pages 289–299, September 2005.

[5] 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.

[6] 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.

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

[8] 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.

[9] 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 (CHES04)*, August 2004.

[10] 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.

[11] 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.

[12] J. Kulik, W. Heinzelman, and H. Balakrishnan. Negotiation-based protocols for disseminating information in wireless sensor networks. *Wireless Networks*, 8:169–185, 2002.

[13] 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.

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

[15] S. S. Kulkarni, M. G. Gouda, and A. Arora. Secret instantiation in ad hoc networks. *Special Issue of Elsevier Journal of Computer Communications on Dependable Wireless Sensor Networks*, 2005.

[16] 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.

[17] 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 06)*, July 2006.

[18] 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.

[19] M. Luby. LT codes. *In Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 271–282, 2002.

[20] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory, Special Issue: Codes on Graphs and Iterative Algorithms*, 47(2):569–584, February 2001.

[21] 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.

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

[23] 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.

[24] A. Perrig, R. Canetti, J. Tygar, and D. X. Song. Efficient authentication and signing of multicast streams over lossy channels. *IEEE Symposium on Security and Privacy*, pages 56–73, May 2000.

[25] 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 2001)*, July 2001.

[26] I. S. Reed and G. Solomon. Ploynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(10):300–304, 1960.

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

[28] H. Wang and Q. Li. Efficient 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.

[29] L. Wang and S. S. Kulkarni. Authentication in reprogramming of sensor networks for mote class adversaries. *The 15th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS 2007), under submission.* Available as Technical Report MSU-CSE-06-32, Computer Science and Engineering, Michigan State University, East Lansing, MI, November, 2006.

[30] L. Wang and S. S. Kulkarni. Proactive reliable bulk data dissemination in sensor networks. *The IASTED International Workshop on Distributed Algorithms and Applications for Wireless and Mobile Systems (DAAWMS)*, November 2005.