Authentication in Reprogramming of Sensor Networks for Mote Class Adversaries [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

Reprogramming is an essential service for wireless sensor networks. Authenticating reprogramming process is important as sensors need to verify that the code image is truly from a trusted source. There are two ways to achieve authentication: public key based and symmetric key based. Although previous work has shown that public 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 reprogramming process. Our protocol is based on the secret instantiation algorithm from [5, 11], which requires only $O(\log n)$ keys to be maintained at each sensor. We integrate this algorithm with the existing reprogramming protocol. Through simulation, we show that it is able to authenticate reprogramming process at very low communication cost, and has very short delay.

Keywords: Sensor networks, Reprogramming, Authentication, Symmetric keys

1 Introduction

Wireless reprogramming is an essential service for sensor networks due to the fact that sensor networks consist of hundreds or thousands of sensor nodes and they are often deployed in remote or hostile environments. It is demanding and sometimes impossible to collect all the sensor nodes from the field for reprogramming. Therefore, it is necessary to reprogram sensor networks in place.

[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.

Reprogramming is performed via wireless radio, which is a broadcast medium, and is vulnerable to packet injection or corruption attacks. Moreover, the current reprogramming protocols [7, 10, 13, 17, 21] are epidemic in nature. Once a false or viral code image is installed on one sensor, it could rapidly infect the entire network, and thus, lead to catastrophic damage. For these reasons, it is important that sensor nodes be able to verify that the code image is from a trusted source.

In this paper, we are interested in providing security for reprogramming. Specifically, we focus on authentication. Our goal is to provide a way that sensor nodes can verify program authenticity and integrity. Authentication can be achieved in two ways: public key based, or symmetric key based. In public key based authentication, 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 that the packets are from the base station. However, public key based authentication is computationally expensive. Although recent work has shown that elliptic curve cryptography (ECC) is feasible on Mica-2 motes [6, 16], it should still be avoided or used sparingly.

By contrast, symmetric key based authentication needs much less energy/memory/computation resources, and hence, is expected to be more appropriate for resource constrained sensor nodes. (As an illustration, on the Mica motes, encryption using public key is approximately 100-1000 times slower than the symmetric key encryption.) A simple approach is to use a single network-wide key shared by the base station and all the sensors [8]. 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 [2]), 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.

In this paper, we show how a symmetric key based algorithm can be used for authentication in reprogramming for mote-class adversaries. Examples of such adversaries are likely to occur in sensor network testbeds. Such testbeds are expected to be typically physically secure so that attacks from a laptop class adversary are prevented/mitigated. However, since the testbed relinquishes control of sensors to users for their experiments, one experiment can be affected by another concurrent experiment. In this case, a potential adversary is in mote-class, i.e., its computation and communication capability as well as battery power is similar to the sensors in the network. In our work, the network consists of a base station and a collection of sensors. The sensors need to verify that the code image is truly from the base station. 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 apply a secret instantiation algorithm from [5, 11] to provide authentication. Thus, in the protocol, only a very small number of keys are maintained at every sensor.

Finally, observe that authentication is sufficient for reprogramming in sensor networks. In other words, it suffices to ensure that the sensors can be assured that the code is from the trusted source. However, in this application, confidentiality is not required, i.e., the code being transmitted is public and can be acquired by the adversary. Hence, the code is sent in plain text along with appropriate authentication.

Contributions of the paper. We integrate the secret instantiation algorithm from [5, 11] with the existing reprogramming protocols. The algorithm requires only $O(\log n)$ keys to be maintained at each sensor, and a small signature is included in the messages from the base station that can be verified at the sensors. Through simulation on TOSSIM [15], we show that it is able to authenticate reprogramming process at low communication cost, and has short delay. We illustrate this by evaluating the effect of adding authentication to MNP [13], a multihop network reprogramming protocol.

Organization of the paper. In Section 2, we describe the system model and requirements of secure reprogramming problem. In Section 3, we present our authentication protocol, and illustrate the process of integrating it to the existing reprogramming protocols, MNP and Deluge. In Section 4, we evaluate our approach in terms of communication cost and delay. We survey related work in Section 5 and conclude in Section 6.

2. System Model

The goal of this paper is to add security to the existing reprogramming protocols, such as MNP [13] and Deluge [7], in which the code image is propagated from a base station to all the sensors in the network. These reprogramming protocols are all based on the advertise-request-data three-way handshake model [9]. The program to be distributed is divided into N segments. Each segment contains K packets; the last segment may contain fewer packets. The default packet payload is 23 bytes. Sensors must receive the segments in order. However, within a segment, the packets can be received out of order. The sensors broadcast advertisements of the available segments. When the neighbors receive the advertisements, if they haven't 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 is received by every receiver. This model provides efficient and reliable transmission in a highly lossy and unstable wireless environment.

We consider an adversary as one who tries to inject its own code into sensor nodes. It can eavesdrop on any communication in the network. It is able to compromise a sensor node, and acquire all information inside it. However, an adversary cannot compromise the base station. Initially, we do not consider collusion among sensor nodes. We discuss the effect of collusion in Section 3.1. As mentioned in the Introduction, we only consider a mote-class adversary, i.e., an adversary has limited computational resources, and cannot launch Denial of Service attack. We expect that the proposed algorithm would be useful in a testbed setting. To illustrate this, consider the case where a sensor network testbed [1] provides a user with a certain set of sensors for performing an experiment. In this case, typically, the testbed itself is physically secure from laptop-class adversaries. However, since the testbed may relinquish control of some sensors to users, such sensors may be able interfere with other experiments running on the testbed. Moreover, since an adversary is likely to be detected by auditing techniques that may exist in the testbed, the likelihood of security attacks is further reduced.

The goals of the proposed protocol are as follows:

1. Authenticity. Each sensor must be able to verify that a code image is from a trusted source and has not been changed during transit. We consider the base station as a trusted source, and is protected against compromise.

2. Compromise resilience. Because current mote-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. Moreover, it should not add long delay to the reprogramming process.

3. Protocol

Our authentication protocol for reprogramming is symmetric key based. In Section 3.1, we describe the secret instantiation algorithm [5, 11], which specifies how the secrets (i.e., keys) are distributed and used for authenticating the communication from the base station to sensors. In Section 3.2, we discuss the issues of integrating the secret instantiation algorithm with the existing reprogramming protocols.

3.1 Secret Instantiation Algorithm

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. If the verification of all the signatures a sensor has is 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.

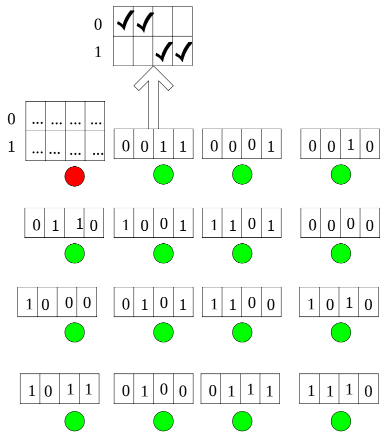


Figure 1. Secret instantiation: an example.

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 (the upper left node) 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, 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. The smallest value for r is 2; for this case, the base station maintains $2log_2 n$ secrets and each sensor maintains $log_2 n$ secrets. However, 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.

As the value of r increases, the number of secrets maintained by the base station ($rlog_r n$) increases. However, in this case, the number of secrets maintained by each sensor ($log_r n$) decreases. Moreover, when r increases, the collusion resistance also increases. For $r = \sqrt{n}$, the algorithm corresponds to the grid algorithm in [12]. 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 Integration with Reprogramming Protocols

In this section, we integrate our secret instantiation algorithm with reprogramming protocols. As a program normally consists of hundreds or thousands of packets, it would incur a lot of overhead if the base station signs every packet it sends. Hence, we use the chained hash approach proposed in [4].

We illustrate the chained hash mechanism in Figure 2. A program is divided into N segments, shown as N rows of packets. Each segment (row) has K packets. We represent the j^{th} data packet of the i^{th} segment as $P(i,j)$, $i = 1..N$, $j = 1..K$. The hash value of packet $P(i,j)$ is denoted as $H(i,j)$. A data packet $P(i,j)$ has two parts, the data part (denoted as $data(i,j)$) and a hash of the next packet. If $P(i,j)$ is the last packet in segment i ($1 \leq i < N$), then the next packet is the first packet of segment $i+1$. If $P(i,j)$ is the last packet in the program, then the hash value is 0. Hence, each data packet can be represented as

If $P(i,j)$ is the last packet of the program

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

else if j is the last packet in the segment

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

else

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

Note that the hash value is computed over the entire packet, not just the data part. As shown in Figure 2, the hash values of the packets construct a chain. The head of the hash chain, i.e., the hash value of the first packet, is signed by the base station. In our algorithm, the base station signs the first hash $H(1,1)$ using all the secrets. We denote the signature as $sign(H(1,1))$ in Figure 2.

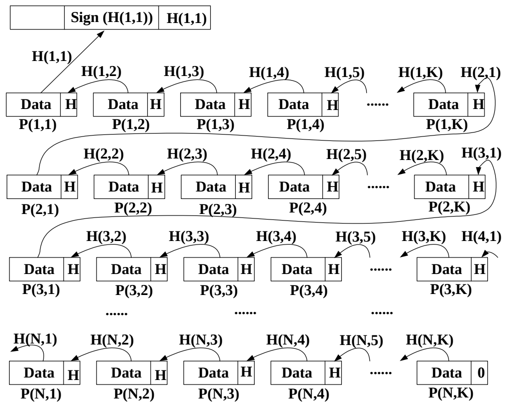


Figure 2. chained hash mechanism.

In multihop reprogramming, every sensor needs to receive the program. And once they receive part of the program (one or more segments), they can start advertising and forward the program to its neighbors. When sensors receive and forward the program, the signature and hash $H(1,1)$ are also buffered and forwarded. This signature and the hash $H(1,1)$ should be sent right before the transmission of the segment starts. In the remainder of this paper, we use MNP as an example. MNP uses a sender selection algorithm to try to guarantee that there is only one sender in a neighborhood at a time. 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. We add the signature and the hash $H(1,1)$ in the “StartDownload” message. If the signature plus hash is too long, and does not fit in one message, we can use multiple messages for “StartDownload”. In this case, the receiver needs to receive all of them in order to get the entire signature.

When a node receives the packet(s) that contains $sign(H(1,1))$ and $H(1,1)$, it decrypts the signature using the collection of secrets it has, and verifies $H(1,1)$. If all the signature verification are successful, it assumes that the packet is truly from the base station. When a node receives a data packet $P(i,j)$, it can verify its authenticity and integrity if and only if it has already received and verified $H(i,j)$ from a previously received packet. Therefore, it implies that the data packets have to be *verified* in order.

However, packets do not arrive in the same order as they are sent due to packet losses and/or delay. If we require that the data packets have to be *received/stored* in sequential order, we have to throw away a large portion of the data packets that have been received. Our simulation results (as well as the results from [3, 4]) show that if we require that the data packets can only be received and stored in order, reprogramming process will be delayed significantly (the completion time for reprogramming 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. On the other hand, if out of order delivery is allowed then an adversary can mount a denial of service attack by sending a large number of garbage packets, as storing the packets (to EEPROM, i.e., the external flash of motes) requires significant energy. However, since a mote class adversary has limited power and message transmission requires significant energy, the adversary cannot launch such attack for a large duration. Therefore, in our design, we allow the packets that arrive out of order (within one segment) to be received and stored immediately.

In our protocol, each sensor maintains a *received* vector (corresponds to *MissingVector* in MNP), which is a bitmap of the current segment, indicating which packets have been received. All the bits in the *received* vector are initialized to 0. Once a sensor receives a packet for the first time, it stores the packet (the data part) in EEPROM, and sets the corresponding bit in *received* to 1. The hash value part of the packet needs be buffered in memory so that it can be retrieved fast for verification. Moreover, each sensor contains a variable, *verifiedPackets*, which is the number of packets the sensor has received and verified.

When a sensor node receives a packet $P(i,j)$, if the *previous* packet has been verified, i.e., $j =$ *verifiedPackets*$+1$, then the node can verify packet $P(i,j)$ using the saved (and verified) $H(i,j)$ from the previous packet. If the verification is successful, the node will increase *verifiedPackets* by 1, and continue verifying the *next* received (but not verified) packet. (When we talk about *previous* or *next* packet, we refer to the previous or next node on the hash chain.) This verification process continues until we reach a missing packet. On the other hand, if the packet verification fails, the packet is thrown away (i.e., $received(j)$ is set to 0), and the verification process stops.

The operation a sensor performs when it receives a data packet $P(i,j)$ (the j^{th} packet in the i^{th} segment) is shown in Figure 3.

```
when a data packet P(i,j) arrives
    if P(i,j) is received the first time, i.e., received(j) == 0
        store P(i,j): store data part in EEPROM and the hash value
            for the next packet in memory, set received(j) to 1
        while ((received(j) == 1) and (j == verifiedPackets + 1))
            Compute H(i,j)
            if computed H(i,j) == saved H(i,j) from the previous packet
                verifiedPackets++, j++
            else // the packet cannot be verified, hence is thrown away
                set received(j) to 0, break while loop
            endif
        endwhile
    endif
```

Figure 3. Operation a node performs when it receives a data packet P(i,j)

4. Evaluation

In this section, we evaluate the performance of our secure reprogramming protocol. We integrate our protocol with MNP [13] as described in Section 3.2. We refer to the integrated protocol as SecureMNP. We simulate SecureMNP using TOSSIM. 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 128 data packets. The simulations are performed in a grid topology. 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.

The default payload size of each packet in MNP is 23 bytes. In SecureMNP, each packet carries a 4-byte hash, which is the hash value of the next packet. Hence, excluding the hash value, the *effective* data payload is 19 bytes. Therefore, in every data packet, 4 out of 23 bytes of the payload is consumed in authentication. Therefore, in order to transmit a program of certain size, more data packets need to be received at every sensor.

We consider a 20x20 network, i.e., the number of sensors in the network is 400. We set r to be 2. In this case, the base station contains 18 (i. e., $2\log_2 400$) secrets, and each sensor has 9 secrets. As we described in Section 3.2, the base station signs $H(1,1)$ using all the secrets, and attaches all the signatures and $H(1,1)$ in “StartDownload” messages. If sensors only verify the last bytes of the signatures, we only need 18 bytes for the signatures, and the signatures fit in a single packet. If two bytes of each signature are used, then the length of the 18 signatures is 36 bytes. In this case, two packets for “StartDownload” messages need to be sent.

Computation cost. For each program to be distributed, the signatures are only signed at the base station once. Sensors verify the signatures once, i.e., when they start receiving the first segment. When a sensor forwards the program to its neighbors, it simply uses the signatures received from the base station. Therefore, the computation cost is minimized, as encryption or decryption is only performed once on each sensor at the start of reprogramming. Finally, while hash computation is performed for every packet, it is very efficient (less than 10ms per packet). Hence, hash computation also does not significantly increase the computation cost.

Memory cost. Our authentication protocol has memory cost in the following ways. First, the algorithm uses a variable *verifiedPackets* to keep track of how many packets have been verified/authenticated. Second, $\log_r n$ secrets are maintained at each sensor. When r increases, the number of secrets maintained at the sensor decreases. In a 20x20 network, the number of secrets at each sensor is no more than 9. Third, since we allow packets to be received and stored out of order, we need to store all the hash values for the packets in the current segment. As we assume a 4-byte hash value is used and each segment contains 128 packets, the space that is used to store hash values is 512 bytes. The hash values can be stored in memory if reprogramming speed is important. Fourth, the signatures from the base station also need to be saved either in memory or in flash. Fifth, the encryption/decryption process consumes some amount of memory.

Delay. We assume that the last two bytes of each signature are used, then the collection of the signatures from the base station are contained in two "StartDownload" messages: "StartDownload1" and "StartDownload2". In order to get the entire set of signatures, each node needs to receive both messages. As we allow packets to be saved before verified, and hash values can be computed very fast, authentication process does not affect reprogramming time significantly. Given a certain number of data packets to be sent, the reprogramming time remains almost the same no matter whether the security protocol is used. The major overhead is the hash values that are carried in data payload. As we have pointed out earlier, 4 out of 23 bytes data payload in a data packet are used for authentication. The size of code image that is sent by a MNP segment is 2.94KB (128×23 bytes). By contrast, the size of the code image that is sent by a SecureMNP segment is 2.43KB (128×19 bytes). In Figure 4, we show that given a certain amount of code image to be sent, the reprogramming time required by SecureMNP is only a little higher than that required by MNP.

Communication cost. Similarly, the communication cost required by SecureMNP is a little higher than that is required by MNP due to fact that the hash values that are attached with every packet. In Figure 5, we show that for a given program size to be distributed, the message transmission and reception of SecureMNP is about 20% higher than that of MNP.

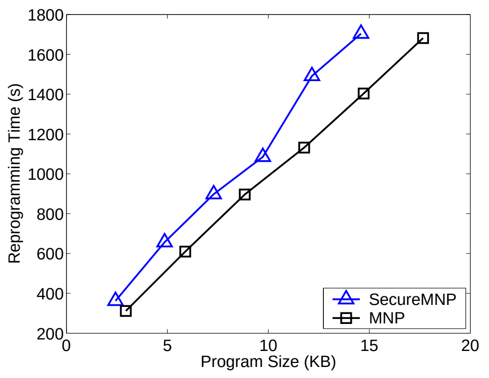
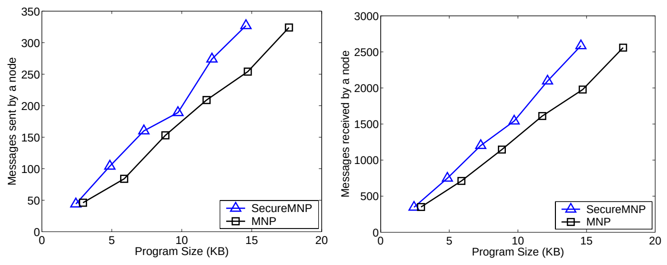


Figure 4. Delay of authentication under different program sizes.



(a) (b)

Figure 5. Communication cost of authentication under different program sizes.

5. Related Work

In this section, we review related work in the areas of network reprogramming, authenticated broadcast, and secure network reprogramming.

Network reprogramming. The work on network reprogramming include MOAP [21], Deluge [7], MNP [13], Infuse [10], Sprinkler [17]. Although these protocols differ in many aspects, such as suppression schemes, transmission scheduling, loss detection and recovery mechanisms, they are all designed to send the entire code image from one (or a few) base station to all the sensors in the network. 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 SecureMNP is that the signatures and the hash value of the first packet are attached with advertisement messages, rather than “StartDownload” messages in SecureMNP. Our protocol can also be used to authenticate TDMA based reprogramming protocols, such as Infuse [10] and Sprinkler [17]. 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 [19] and μTESLA [20] 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 [18] is another protocol that performs authenticated broadcast via precomputed hash collisions and chains. However, all these protocols, TESLA, μTESLA, and BiBa require *loose time synchronization*, and hence, introduce additional constraints and overhead.

Secure network reprogramming. Researchers have shown interest in secure network reprogramming recently. P. Lanigan *et. al.* proposed *Sluice* [14], which also uses a hash chain for authentication. However, the hashes are verified on the segment level rather than at the packet level. Although the computation, communication and memory cost are relatively less, Sluice is vulnerable to some form of attacks, e. g., a single bad packet will cause the entire segment to be discarded. P. Dutta *et. al.* [4] proposed a protocol, called SecureDeluge, which provides authentication through a packet level hash chain. In SecureDeluge, packets can only 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. J. Deng *et. al.* [3] 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. Receiving the hash tree itself requires a partial order. And sending the hash tree over the radio increases communication cost. In our protocol, we allow the packets that are out of order to be saved and wait for verification later. All these three protocols mentioned above are based on the public key algorithm. By contrast, our protocol is symmetric key based. We have shown that our algorithm authenticates reprogramming process without adding much delay and cost.

6. Conclusion

In this paper, we showed how authentication could be achieved for reprogramming protocols in sensor networks. We used symmetric key distribution algorithms from [5, 11] to ensure that the base station can communicate securely with each sensor in the network. Based on the security of the key distribution, the reprogramming protocol allows sensors to conclude that the code is truly transmitted by the base station. Thus, they will not accept unauthorized code. We illustrated this algorithm in the context of the MNP [13]. Our approach can also be easily applied to other reprogramming protocols such as [7, 10]. Our results show that the overhead added in terms of communication cost, increased delay and memory footprint is small.

Our focus in this paper was on mote-class adversary. Since such adversary has limited energy, it cannot use extensive denial of service attack. However, a laptop-class adversary can mount a denial of service attack by sending garbage data to the mote. In [4] authors provide an approach for mitigating laptop-class adversary by requiring that the sensor receive the data in order. By this requirement, the sensors will not save any packets to EEPROM (an energy consuming operation) unless the packet is authenticated. However, this requirement increases the reprogramming time and energy usage significantly, by as much as 6 to 7 times. (In [4], the use of public key also contributes to increased time/energy usage.) Moreover, even with this requirement, a laptop-class adversary can cause significant damage as message transmission and reception is also very energy consuming.

As discussed in Section 2, our algorithm is expected to be especially valuable for security in sensor network testbed. Such testbed is typically physically secure, thereby preventing/mitigating laptop-class attackers. However, the testbed typically relinquishes control of individual sensor nodes that are used in an experiment. Thus, an experiment could be interfered by other sensors in the testbed. Our algorithm provides protection from such interference/attacks with a low overhead.

Since the key distribution protocol used in this approach allows tradeoff between the number of secrets and level of collusion resistance, the designer can choose appropriate parameters to determine the desired level of collusion resistance. In our experiments, for simplicity, we used the base $r = 2$ thereby choosing the least number of secrets at the base station. However, if higher collusion resistance is desired, the designer can choose higher base; for example, for a 20x20 network (400 sensors), if $r = 10$ is used then the number of secrets maintained at the base station increases to 30 (as compared to 18 when $r = 2$). Moreover, since these secrets are used only a few times during reprogramming, it will not affect the reprogramming cost (time/energy) significantly. Additionally, with increased value for r, the number of secrets at the sensor is reduced. Thus, providing higher level of collusion resistance does not adversely affect the sensors.

References

[1] A. Arora, E. Ertin, R. Ramnath, W. Leal, and M. Nesterenko. Kansei: A high-fidelity sensing testbed. *IEEE Internet Computing, special issue on Large-Scale Sensor Networks*, March 2006.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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