

# Multi-Level $\mu$ TESLA: A Broadcast Authentication System for Distributed Sensor Networks\*

Donggang Liu Peng Ning  
Cyber Defense Laboratory  
Department of Computer Science  
North Carolina State University  
Raleigh, NC 27695-8207  
Emails: {dliu, pning}@ncsu.edu

## Abstract

Broadcast authentication is a fundamental security service in distributed sensor networks. This paper presents the development of a scalable broadcast authentication scheme named *multi-level  $\mu$ TESLA* based on  $\mu$ TESLA, a broadcast authentication protocol whose scalability is limited by its unicast based initial parameter distribution. Multi-level  $\mu$ TESLA satisfies several nice properties, including low overhead, tolerance of message loss, scalability to large networks, and resistance to replay attacks as well as denial of service attacks. This paper also presents the development of a multi-level  $\mu$ TESLA broadcast authentication system on TinyOS, an operating system for networked sensors, and experimental results obtained through simulation.

## 1 Introduction

A distributed sensor network usually consists of one or several computationally powerful nodes called *base stations* and a large number of inexpensive, low capacity nodes called *sensors* (or *sensor nodes*). The nodes in a distributed sensor network communicate through wireless communication, which is usually limited in bandwidth. Distributed sensor networks have extensive applications in military as well as civilian operations, in which it is necessary to deploy sensor nodes dynamically.

Broadcast authentication is an essential service in distributed sensor networks. Because of the large number of sensor nodes and the broadcast nature of wireless communication, it is usually desirable for the base stations to broadcast commands and data to the sensor nodes. The authenticity of such commands and data is critical for the normal operation of sensor networks. If convinced to accept forged or modified commands or data, sensor nodes may perform unnecessary or incorrect operations, and cannot fulfill the intended purposes of the network. Thus, in hostile environments (e.g., battle field, anti-terrorists operations), it is necessary to enable sensor nodes to authenticate broadcast messages received from the base station.

Providing broadcast authentication in distributed sensor networks turns out to be a non-trivial task. On the one hand, public key based digital signatures (e.g., RSA [29]), which are typically used for broadcast authentication in traditional networks, are too expensive to be used in sensor networks, due to the intensive computation involved in signature verification and the resource constraints on sensors. On the other hand,

---

\*A preliminary version of this paper has appeared in the Proceedings of the 10th ISOC Annual Network and Distributed Systems Security Symposium [20].

secret key based mechanisms (e.g., HMAC [18]) cannot be directly applied to broadcast authentication, since otherwise a compromised receiver can easily forge any message from the sender.

A protocol named  $\mu$ TESLA [27] has been proposed for broadcast authentication in distributed sensor networks, which is adapted from a stream authentication protocol called TESLA [25].  $\mu$ TESLA employs a chain of authentication keys linked to each other by a pseudo random function [14], which is by definition a one way function. Each key in the key chain is the image of the next key under the pseudo random function.  $\mu$ TESLA achieves broadcast authentication through delayed disclosure of authentication keys in the key chain. The efficiency of  $\mu$ TESLA is based on the fact that only pseudo random function and secret key based cryptographic operations are needed to authenticate a broadcast message. (More details of  $\mu$ TESLA can be found in Section 2.)

The original TESLA uses broadcast to distribute the initial parameters required for broadcast authentication. The authenticity of these parameters are guaranteed by a digital signature generated by the sender. However, due to the low bandwidth of a sensor network and the low computational resources at each sensor node,  $\mu$ TESLA cannot distribute these initial parameters using public key cryptography. Instead, the base station has to unicast the initial parameters to the sensor nodes individually. This feature severely limits the application of  $\mu$ TESLA in large sensor networks. For example, The implementation of  $\mu$ TESLA in [27] has 10 kbps at the physical layer and supports 30-byte packets. To bootstrap 2,000 nodes, the base station has to send or receive at least 4,000 packets to distribute the initial parameters, which takes at least  $\frac{4,000 \times 30 \times 8}{10,240} = 93.75$  seconds even if the channel utilization is perfect. Such a method certainly cannot scale up to very large sensor networks, which may have thousands of nodes.

In this paper, we present a series of techniques to extend the capabilities of  $\mu$ TESLA. The basic idea is to *predetermine* and *broadcast* the initial parameters required by  $\mu$ TESLA instead of unicast-based message transmission. In the simplest form, our extension distributes the  $\mu$ TESLA parameters during the initialization of the sensor nodes (e.g., along with the master key shared between each sensor and the base station). To provide more flexibility, especially to prolong the lifetime of  $\mu$ TESLA without requiring a very long key chain, we introduce a multi-level key chain scheme, in which the higher-level key chains are used to authenticate the commitments of lower-level ones. To further improve the survivability of the scheme against message loss and Denial of Service (DOS) attacks, we use redundant message transmissions and random selection strategies to deal with the messages that distribute key chain commitments. The resulting scheme, which is named *multi-level  $\mu$ TESLA*, removes the requirement of unicast-based initial communication between base station and sensor nodes while keeping the nice properties of  $\mu$ TESLA (e.g., tolerance of message loss, resistance to replay attacks).

We also report the implementation of a multi-level  $\mu$ TESLA broadcast authentication system on TinyOS [16]. In this paper, we describe the design and implementation of the multi-level  $\mu$ TESLA API as well as the experiments performed through simulation. Our experiments are intended to study the performance of multi-level  $\mu$ TESLA under severe attacks and poor channel quality. The experimental results demonstrate that our scheme can tolerate high channel loss rate and is resistant to known DOS attacks to a certain degree.

The rest of this paper is organized as follows. The next section gives a brief overview of  $\mu$ TESLA. Section 3 presents the development of the multi-level  $\mu$ TESLA scheme. Section 4 describes the design and implementation of the multi-level  $\mu$ TESLA API on TinyOS [16]. Section 5 presents our experiments performed through simulation. Section 6 discusses the related work, and section 7 concludes the paper and points out some future research directions. Appendix A presents the details of the two-level  $\mu$ TESLA scheme, from which the multi-level  $\mu$ TESLA is extended. Finally, appendix B gives the details of the multi-level  $\mu$ TESLA API.

## 2 An Overview of $\mu$ TESLA

Authentication of broadcast messages is an important security issue in wired or wireless networks. Generally, an asymmetric mechanism, such as public key cryptography, is required to authenticate broadcast messages. Otherwise, a malicious receiver can easily forge any packet from the sender. However, due to the high communication, computation and storage overhead of the asymmetric cryptographic mechanisms, it is impractical to implement them in resource constrained sensor networks.

$\mu$ TESLA introduced asymmetry by delaying the disclosure of symmetric keys [27]. A sender broadcasts a message with a Message Authentication Code (MAC) generated with a secret key  $K$ , which will be disclosed after a certain period of time. When a receiver receives this message, if it can ensure that the packet was sent before the key was disclosed, the receiver can buffer this packet and authenticate it when it receives the corresponding disclosed key. To continuously authenticate the broadcast packets,  $\mu$ TESLA divides the time period for broadcasting into multiple time intervals, assigning different keys to different time intervals. All packets broadcasted in a particular time interval are authenticated with the same key assigned to that time interval.

To authenticate the broadcast messages, a receiver first authenticates the disclosed keys.  $\mu$ TESLA uses a one-way key chain for this purpose. The sender selects a random value  $K_n$  as the last key in the key chain and repeatedly performs a pseudo random function  $F$  to compute all the other keys:  $K_i = F(K_{i+1}), 0 \leq i \leq n - 1$ , where the secret key  $K_i$  is assigned to the  $i^{th}$  time interval. With the pseudo random function  $F$ , given  $K_j$  in the key chain, anybody can compute all the previous keys  $K_i, 0 \leq i \leq j$ , but nobody can compute any of the later keys  $K_i, j+1 \leq i \leq n$ . Thus, with the knowledge of the initial key  $K_0$ , the receiver can authenticate any key in the key chain by merely performing pseudo random function operations. When a broadcast message is available in  $i^{th}$  time interval, the sender generates MAC for this message with a key derived from  $K_i$  and then broadcasts this message along with its MAC and discloses the key  $K_{i-d}$  assigned to the time interval  $I_{i-d}$ , where  $d$  is the disclosure lag of the authentication keys. The sender prefers a long delay in order to make sure that all or most of the receivers can receive its broadcast messages. But, for the receiver, a long delay could result in high storage overhead to buffer the messages.

Each key in the key chain will be disclosed after some delay. As a result, the attacker can forge a broadcast packet by using the disclosed key.  $\mu$ TESLA uses a security condition to prevent a receiver from accepting any broadcast packet authenticated with a disclosed key. When a receiver receives an incoming broadcast packet in time interval  $I_i$ , it checks the security condition  $\lfloor (T_c + \Delta - T_0) / T_{int} \rfloor < I_i + d$ , where  $T_c$  is the local time when the packet is received,  $T_0$  is the start time of the time interval 0,  $T_{int}$  is the duration of each time interval, and  $\Delta$  is the maximum clock difference between the sender and itself. If the security condition is satisfied, i.e., the sender has not disclosed the key  $K_i$  yet, the receiver accepts this packet. Otherwise, the receiver simply drops it. When the receiver receives the disclosed key  $K_i$ , it can authenticate it with a previously received key  $K_j$  by checking whether  $K_j = F^{i-j}(K_i)$ , and then authenticate the buffered packets that were sent during time interval  $I_i$ .

$\mu$ TESLA is an extension to TESLA [25]. The only difference between TESLA and  $\mu$ TESLA is in their key chain commitment distribution schemes. TESLA uses asymmetric cryptography to bootstrap new receivers, which is impractical for current sensor networks due to its high computation and storage overhead.  $\mu$ TESLA depends on symmetric cryptography with the master key shared between the sender and each receiver to bootstrap the new receivers individually. In this scheme, the receiver first sends a request to the sender, and then the sender replies a packet containing the current time  $T_c$  (for time synchronization), a key  $K_i$  of one way key chain used in a past interval  $i$ , the start time  $T_i$  of interval  $i$ , the duration  $T_{int}$  of each time interval and the disclosure lag  $d$ .

TESLA was later extended to include an immediate authentication mechanism [26]. The basic idea is to include an image under a pseudo random function of a late message content in an earlier message so that

once the earlier message is authenticated, the later message content can be authenticated immediately after it is received. This extension can also be applied to  $\mu$ TESLA protocol in the same way.

### 3 Multi-Level $\mu$ TESLA

The major barrier of using  $\mu$ TESLA in large sensor networks lies in its difficulty to distribute the key chain commitments to a large number of sensor nodes. In other words, the method for bootstrapping new receivers in  $\mu$ TESLA does not scale to a large group of new receivers, though it is okay to bootstrap one or a few. The essential reason for this difficulty is the mismatch between the *unicast*-based distribution of key chain commitments and the authentication of *broadcast* messages. That is, the technique is developed for broadcast authentication, but it relies on unicast-based technique to distribute the initial parameters.

In this section, we develop several techniques to extend the capability of  $\mu$ TESLA. The basic idea is to *predetermine* and *broadcast* the key chain commitments instead of unicast-based message transmissions. In the following, we present a series of schemes; each later scheme improves over the previous one by addressing some of its limitations except for scheme V, which improves over scheme IV only in special cases where the base station is very resourceful in terms of computational power. The final scheme, a multi-level  $\mu$ TESLA scheme, then has two variations based on schemes IV and V, respectively.

We assume each broadcast message is from the base station to the sensor nodes. Broadcast messages from a sensor node to the sensor network can be handled as suggested in [27]. That is, the sensor node unicasts the message to the base station, which then broadcasts the message to the other sensor nodes. The messages transmitted in a sensor network may reach the destination directly, or may have to be forwarded by some intermediate nodes; however, we do not distinguish between them in our schemes.

For the sake of presentation, we denote the key chain with commitment  $K_0$  as  $\langle K_0 \rangle$  throughout this paper.

#### 3.1 Scheme I: Predetermined Key Chain Commitment

A simple solution to bypass the unicast-based distribution of key chain commitments is to predetermine the commitments, the starting times, and other parameters of key chains to the sensor nodes during the initialization of the sensor nodes, possibly along with the master keys shared between the sensor nodes and the base station. (Unlike the master keys, whose confidentiality and integrity are both important, only the integrity of the key chain commitments needs to be ensured.) As a result, all the sensor nodes have the key chain commitments and other necessary parameters once they are initialized, and are ready to use  $\mu$ TESLA as long as the starting time is passed.

This simple scheme can greatly reduce the overhead involved in distribution of key chain commitments in  $\mu$ TESLA, since unicast-based message transmission is not required any more. However, this simple solution also introduces several problems.

First, a key chain in this scheme can only cover a fixed period of time. To cover a long period of time, we need either a long key chain, or long time intervals to divide the time period. However, both options may introduce problems. If a long key chain is used, the base station has to allocate a large amount of memory to store the key chain. For example, in our later experiments, the duration of each time interval is 100ms. To cover one day, the base station has to allocate  $24 \times 60 \times 60 \times 10 \times 8 = 6,912,000$  bytes memory to store the keys. This may not be desirable in some applications. In addition, the receivers has to perform intensive computation of pseudo random functions if there is a long delay (which covers a large number of time intervals) between broadcast messages in order to authenticate a later disclosed key. Continuing from the previous example, if the time between two consecutive messages received in a sensor is one hour, the sensor has to perform  $60 \times 60 \times 10 = 36,000$  pseudo random operations to verify the disclosed key,

which may be prohibitive in resource constrained sensors. If a long interval is used, there will be a long delay before the authentication of a message after it is received, and it requires a larger buffer at each sensor node. Though the extensions to TESLA [26] can remove the delay in authenticating the data payload and the buffer requirement at the sensor nodes, the messages will have to be buffered longer at the base station.

Second, it is difficult to predict the starting time of a key chain when the sensor nodes are initialized. If the starting time is set too early, the sensor nodes will have to compute a large number of pseudo random functions in order to authenticate the first broadcast message. As we see in the previous example, one hour delay will introduce a huge number of pseudo number operations. In addition, the key chain must be fairly long so that it does not run out before the sensor network's lifetime ends. If the starting time is set too late, messages broadcasted before it cannot be authenticated via  $\mu$ TESLA.

These problems make this simple scheme not a practical one. In the following, we propose several additional techniques so that we not only avoid the problems of unicast-based distribution of key chain commitment, but also those of this simple scheme.

### 3.2 Scheme II: Naive Two-Level $\mu$ TESLA

The essential problem of scheme I lies in the fact that it is impossible to use both a short key chain and short time intervals to cover a long period of time. This conflict can be mitigated by using multiple levels of key chains. In the following several subsections, we first investigate the special case of two level key chains to enhance its security and robustness, and then extend the results to multi-level key chains in Section 3.6.

The two-level key chains consist of a high-level key chain and multiple low-level key chains. The low-level key chains are intended for authenticating broadcast messages, while the high-level key chain is used to distribute and authenticate commitments of the low-level key chains. The high-level key chain uses a long enough interval to divide the time line so that it can cover the lifetime of a sensor network without having too many keys. The low-level key chains have short enough intervals so that the delay between the receipt of broadcast messages and the verification of the messages is tolerable.

The lifetime of a sensor network is divided into  $n_0$  (long) intervals of duration  $\Delta_0$ , denoted as  $I_1, I_2, \dots$ , and  $I_{n_0}$ . The high-level key chain has  $n_0 + 1$  elements  $K_0, K_1, \dots, K_{n_0}$ , which are generated by randomly picking  $K_{n_0}$  and computing  $K_i = F_0(K_{i+1})$  for  $i = 0, 1, \dots, n_0 - 1$ , where  $F_0$  is a pseudo random function. The key  $K_i$  is associated with each time interval  $I_i$ . We denote the starting time of  $I_i$  as  $T_i$ . Thus, the starting time of the high-level key chain is  $T_1$ .

Since the duration of the high-level time intervals is usually very long compared with the network delay and clock discrepancies, we choose to disclose a high-level key  $K_i$  used for  $I_i$  in the following time interval  $I_{i+1}$ . Thus, we use the following security condition to check whether the base station has disclosed the key  $K_i$  when a sensor node receives a message authenticated with  $K_i$  at time  $t$ :  $t + \delta_{Max} < T_{i+1}$ , where  $\delta_{Max}$  is the maximum clock discrepancy between the base station and the sensor node.

Each time interval  $I_i$  is further divided into  $n_1$  (short) intervals of duration  $\Delta_1$ , denoted as  $I_{i,1}, I_{i,2}, \dots, I_{i,n_1}$ . If needed, the base station generates a low-level key chain for each time interval  $I_i$  by randomly picking  $K_{i,n_1}$  and computing  $K_{i,j} = F_1(K_{i,j+1})$  for  $j = 0, 1, \dots, n_1 - 1$ , where  $F_1$  is a pseudo random function. The key  $K_{i,j}$  is intended for authenticating messages broadcasted during the time interval  $I_{i,j}$ . The starting time of the key chain  $\langle K_{i,0} \rangle$  is predetermined as  $T_i$ . The disclosure lag for the low-level key chains can be determined in the same way as  $\mu$ TESLA and TESLA [25, 27]. For simplicity, we assume all the low-level key chains use the same disclosure lag  $d$ . Further assume that messages broadcasted during  $I_{i,j}$  are indexed as  $(i, j)$ . Thus, the security condition for a message authenticated with  $K_{i,j}$  and received at time  $t$  is:  $i' < (i - 1) * n_1 + j + d$ , where  $i' = \lfloor \frac{t - T_1 + \delta_{Max}}{\Delta_1} \rfloor + 1$ , and  $\delta_{Max}$  is the maximum clock discrepancy between the base station and the sensor node.

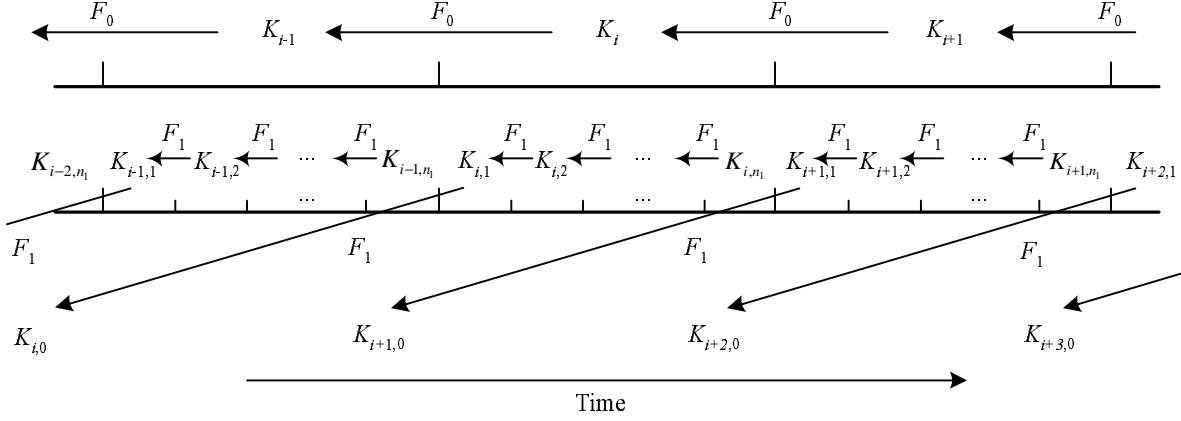


Figure 1: The two levels of key chains in Scheme II. Each key  $K_i$  is used for the high-level time interval  $I_i$ , and each key  $K_{i,j}$  is used for the low-level time interval  $I_{i,j}$ .  $F_0$  and  $F_1$  are different pseudo random functions. Each commitment  $K_{i,0}$  is distributed during the time interval  $I_{i-2}$ .

When sensor nodes are initialized, their clocks are synchronized with the base station. In addition, the starting time  $T_1$ , the commitment  $K_0$  of the high-level key chain, the duration  $\Delta_0$  of each high-level time interval, the duration  $\Delta_1$  of each low-level time interval, the disclosure lag  $d$  for the low-level key chains, and the maximum clock discrepancy  $\delta_{Max}$  between the base station and the sensor nodes throughout the lifetime of the sensor network are distributed to the sensors.

In order for the sensors to use a low-level key chain  $\langle K_{i,0} \rangle$  during the time interval  $I_i$ , they must authenticate the commitment  $K_{i,0}$  before  $T_i$ . To achieve this goal, the base station broadcasts a *commitment distribution message*, denoted as  $CDM_i$ , during each time interval  $I_i$ . (In the rest of this paper, we use commitment distribution message and its abbreviation  $CDM$  interchangeably.) This message consists of the commitment  $K_{i+2,0}$  of the low-level key chain  $\langle K_{i+2,0} \rangle$  and the key  $K_{i-1}$  in the high-level key chain. Specifically, the base station constructs the  $CDM_i$  message as follows:

$$CDM_i = i|K_{i+2,0}|MAC_{K'_i}(i|K_{i+2,0})|K_{i-1}, \text{ where “|” denotes message concatenation, and } K'_i \text{ is derived from } K_i \text{ with a pseudo random function other than } F_0 \text{ and } F_1.$$

Thus, to use a low-level key chain  $\langle K_{i,0} \rangle$  during  $I_i$ , the base station needs to generate the key chain during  $I_{i-2}$  and distribute  $K_{i,0}$  in  $CDM_{i-2}$ .

Since the high-level authentication key  $K_i$  is disclosed in  $CDM_{i+1}$  during the time interval  $I_{i+1}$ , each sensor needs to store  $CDM_i$  until it receives  $CDM_{i+1}$ . Each sensor also stores a key  $K_j$ , which is initially  $K_0$ . After receiving  $K_{i-1}$  in  $CDM_i$ , the sensor authenticates it by verifying that  $F_1^{i-1-j}(K_{i-1}) = K_j$ . Then the sensor replaces the current  $K_j$  with  $K_{i-1}$ .

Suppose a sensor has received  $CDM_{i-2}$ . Upon receiving  $CDM_{i-1}$  during  $I_{i-1}$ , the sensor can authenticate  $CDM_{i-2}$  with  $K_{i-2}$  disclosed in  $CDM_{i-1}$ , and thus verify  $K_{i,0}$ . As a result, the sensor can authenticate broadcast messages sent by the base station using the  $\mu$ TESLA key chain  $\langle K_{i,0} \rangle$  during the high-level time interval  $I_i$ .

This scheme uses  $\mu$ TESLA in two different levels. The high-level key chain relies on the initialization phase of the sensor nodes to distribute the key chain commitment, and it only has a single key chain throughout the lifetime of the sensor network. The low-level key chains depend on the high-level key chain to distribute and authenticate the commitments. Figure 1 illustrates the two-level key chains, and Figure 2 displays the key disclosure schedule for the keys in these key chains.

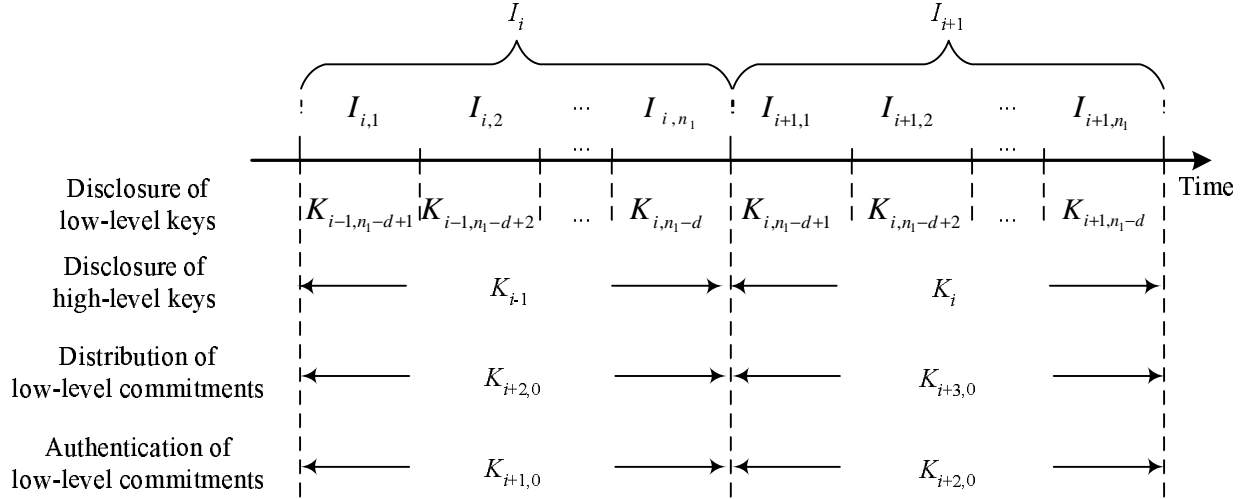


Figure 2: Key disclosure schedule in Scheme II

The two-level key chains scheme mitigates the problem encountered in scheme I. On the one hand, by having long time intervals, the high-level key chain can cover a long period of time without having a very long key chain. On the other hand, the low-level key chain has short time intervals so that authentication of broadcast messages does not have to be delayed too much.

Similar to  $\mu$ TESLA and TESLA, a sensor can detect forged messages by verifying the MAC with the corresponding authentication key once the sensor receives it. In addition, replay attacks can be easily defeated if a sequence number is included in each message.

In the preliminary version of this paper [20], we used a variation of this naive two-level key chains scheme based on the immediate authentication extension to TESLA [26]. The intention was to enable a sensor to authenticate the key included in a *CDM* message immediately after it receives the message. Specifically, we included an image of the key chain commitment contained in the next *CDM* message under a pseudo random function in the current *CDM* message. Once this *CDM* message is authenticated (after receiving the next *CDM* message), the key chain commitment in the next *CDM* message can be authenticated immediately. However, a further investigation reveals that this alternative does not save much. Unlike the data to be immediately authenticated in [26], a key chain commitment usually has the same length as its image under a pseudo random function. Thus, the above variation is equivalent to having each key chain commitment included in two consecutive *CDM* messages.

### 3.3 Scheme III: Fault Tolerant Two-Level $\mu$ TESLA

Scheme II does not tolerate message losses as well as  $\mu$ TESLA and TESLA. There are two types of message losses: the losses of normal messages, and the losses of *CDM* messages. Both may cause problems for scheme II. First, the low-level keys are not entirely chained together. Thus, losses of a key disclosure messages for later keys in a low-level key chain cannot be recovered even if the sensor can receive keys in some later low-level key chains. For example, consider the last key  $K_{i, n_1}$  that is used to authenticate the packet in the key chain of time interval  $I_i$ . If the *CDM* message that carries the disclosure of  $K_{i, n_1}$  is lost, the sensor then has no way to authenticate this packet. As a result, a sensor may not be able to authenticate a stored message even if it receives some key disclosure messages later. In contrast, with  $\mu$ TESLA a receiver can authenticate a stored message as long as it receives a later key. Second, if  $CDM_{i-2}$  does not reach

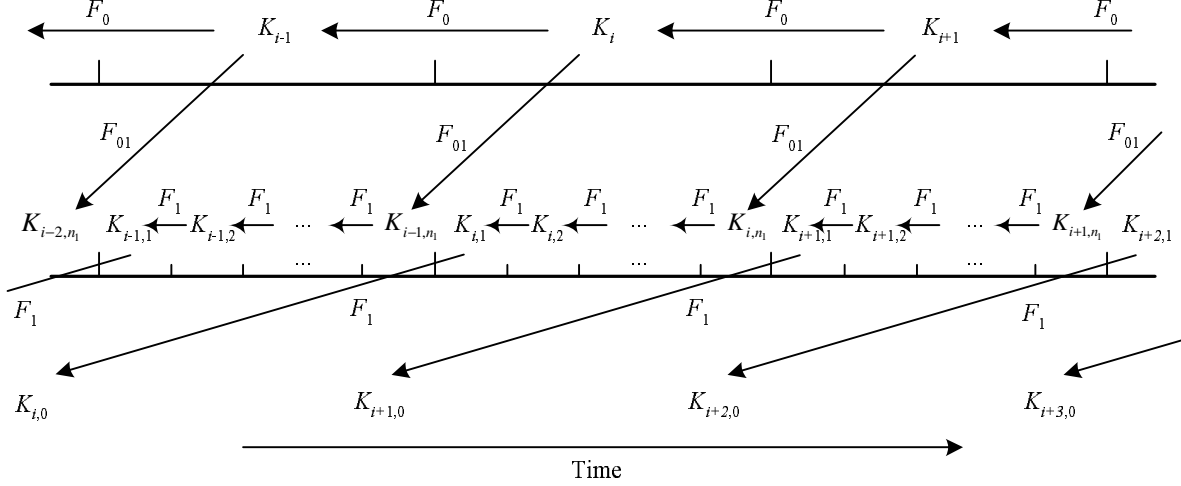


Figure 3: The two levels of key chains in Scheme III. It differs from Figure 1 in that each  $K_{i,n_1}$  is derived from  $K_{i+1}$  using an additional pseudo random function  $F_{01}$ .

a sensor, the sensor will not be able to use the key chain  $\langle K_{i,0} \rangle$  for authentication during the entire time interval  $I_i$ , which is usually pretty long (to make the high-level key chain short).

To address the first problem, we propose to further connect the low-level key chains to the high-level one. Specifically, instead of choosing each  $K_{i,n_1}$  randomly, we derive each  $K_{i,n_1}$  from a high-level key  $K_{i+1}$  (which is to be used in the next high-level time interval) through another pseudo random function  $F_{01}$ . That is,  $K_{i,n_1} = F_{01}(K_{i+1})$ . As a result, a sensor can recover any authentication key  $K_{i,j}$  as long as it receives a *CDM* message that discloses  $K_{i'}$  with  $i' \geq i + 1$ , even if it does not receive any later low-level key  $K_{i,j'}$  with  $j' \geq j$ . Thus, the first problem can be resolved. Figure 3 illustrates this idea.

The second problem does not have an ultimate solution; if the base station cannot reach a sensor at all during a time interval  $I_i$ , *CDM*<sub>*i*</sub> will not be delivered to the sensor. However, the impact of temporary communication failures can be reduced by standard fault tolerant approaches.

One possible solution to mitigate the second problem is to include each key chain commitment in multiple *CDM* messages. For example, we may include each key chain commitment  $K_{i,0}$  in  $l$  consecutive *CDM* messages,  $CDM_{i-2}, \dots, CDM_{i-(l+1)}$ . As a result, *CDM*<sub>*i*</sub> includes the key chain commitments  $K_{i+2,0}, \dots, K_{i+1+l,0}$ . A sensor can recover and authenticate  $K_{i,0}$  if it receives either any two of the above  $l$  *CDM* messages, or one of the  $l$  *CDM* messages and *CDM*<sub>*i-1*</sub>. However, this also increases the size of *CDM* messages as well as the *CDM* buffer on sensor nodes. Moreover, the larger a packet is, the more possible that it is lost in wireless communication. Considering the fact that packets in distributed sensor networks usually have limited size (e.g., the payload of each packet in TinyOS [16] is at most 29 bytes), we decide not to go with this solution.

Instead, we propose to have the base station periodically broadcast the *CDM* message during each time interval. Assuming that the frequency of this broadcast is  $F$ , each *CDM* message is therefore broadcasted  $F \times \Delta_0$  times. To simplify the analysis, we assume the probability that a sensor cannot receive a broadcast of a *CDM* message is  $p_f$ . Thus, the probability that a sensor cannot receive any copy of the *CDM* message is reduced to  $p_f^{F \times \Delta_0}$ .

Note that even if a sensor cannot receive any *CDM* message during a time interval  $I_i$ , it still has the opportunity to authenticate broadcast messages in time intervals later than  $I_{i+1}$ . Not having the *CDM* message in time interval  $I_i$  only prevents a sensor from authenticating broadcast messages during  $I_{i+1}$ . As long as the sensor gets a *CDM* message, it can derive all the low-level keys in the previous time intervals.

By periodically broadcasting *CDM* messages, scheme III introduces more overhead than scheme II. Let us consider the overhead on the base station, the sensors, and the communication channel, respectively. Compared with Scheme II, this scheme does not change the computation of *CDM* messages in the base station, but increases the overhead to transmit *CDM* messages by  $F \times \Delta_0$  times. Base stations in a sensor network are usually much more powerful than the sensor nodes. Thus, the increased overhead on base stations may not be a big problem as long as  $F \times \Delta_0$  is reasonable.

The sensors are affected much less than the base station in a benign environment, since each sensor only needs to process one *CDM* message for each time interval. Thus, the sensors have roughly the same overhead as in scheme II. However, we will show that a sensor has to take a different strategy in a hostile environment in which there are DOS attacks. We will delay the discussion of sensors' overhead until we introduce our counter measures.

This approach increases the overhead in the communication channel by  $F \times \Delta_0$  times, since the *CDM* message for each time interval is repeated  $F \times \Delta_0$  times. Assume the probability that a sensor cannot receive a *CDM* message is  $p_f = 1/2$  and  $F \times \Delta_0 = 10$ . Under our simplified assumption, the probability that the sensor cannot receive any of the 10 *CDM* messages is  $p_f^{F \times \Delta_0} < 0.1\%$ . Further assume that  $\Delta_0$  is 1 minutes, which is quite short as the interval length for the high-level key chain. Thus, there is one *CDM* message per 6 seconds. Assume the bandwidth is 10 kbps and each *CDM* packet is 36 bytes = 288 bits, which includes the 29 byte *CDM* message and the 7 byte packet header as in our experiments (Section 5). Then the relative communication overhead is  $\frac{288}{10240 \times 6} = 0.47\%$ . This is certainly optimistic, since we assume perfect channel utilization. However, it still shows that scheme III introduces very reasonable communication overhead in typical sensor networks.

One limitation of Scheme III is that if a sensor misses all copies of  $CDM_i$  during the time interval  $I_i$ , it cannot authenticate any data packets received during  $I_{i+2}$  before it receives an authentic  $K_j$ ,  $j > i + 2$ . (Note that the sensor does not have to receive an authentic *CDM* message. As long as the sensor can authenticate a high-level key  $K_j$  with  $j > i + 2$ , it can derive the low-level keys through the pseudo random functions  $F_0, F_{01}$ , and  $F_1$ .) Since the earliest high-level key  $K_j$  that satisfies  $j > i + 2$  is  $K_{i+3}$ , and  $K_{i+3}$  is disclosed during  $I_{i+4}$ , the sensor has to buffer the data packets received during  $I_{i+2}$  for at least the duration of one high-level time interval.

### 3.4 Scheme IV: DOS-Tolerant Two-Level $\mu$ TESLA

In scheme III, the usability of a low-level key chain depends on the authentication of the key chain commitment contained in the corresponding *CDM* message. A sensor cannot use the low-level key chain  $\langle K_{i,0} \rangle$  for authentication before it can authenticate  $K_{i,0}$  distributed in  $CDM_{i-2}$ . This makes the *CDM* messages attractive targets for attackers. An attacker may disrupt the distribution of *CDM* messages, and thus prevent the sensors from authenticating broadcast messages during the corresponding high-level time intervals. Although the high-level key chain and the low-level ones are chained together, and such sensors may store the broadcast messages and authenticate them once they receive a later commitment distribution message, the delay between the receipt and the authentication of the messages may introduce a problem: Indeed, an attacker may send a large amount of forged messages to exhaust the sensors' buffer before they can authenticate the buffered messages, and force them to drop some authentic messages.

The simplest way for an attacker to disrupt the *CDM* messages is to jam the communication channel. We may have to resort to techniques such as frequency hopping if the attacker completely jam the communication channel. This is out of the scope of this paper. The attacker may also jam the communication channel only when the *CDM* messages are being transmitted. If the attacker can predict the schedule of such messages, it would be much easier for the attacker to disrupt such message transmissions. Thus, the base station needs to send the *CDM* messages randomly or in a pseudo random manner that cannot be

predicted by an attacker that is unaware of the random seed. For simplicity, we assume that the base station sends the  $CDM$  messages randomly.

An attacker may forge commitment distribution messages to confuse the sensors. If a sensor does not have a copy of the actual  $CDM_i$ , it will not be able to get the correct  $K_{i+2,0}$ , and cannot use the low-level key chain  $\langle K_{i+2,0} \rangle$  during the time interval  $I_{i+2}$ .

Consider a commitment distribution message:  $CDM_i = i|K_{i+2,0}|MAC_{K'_i}(i|K_{i+2,0})|K_{i-1}$ . Once seeing such a message, the attacker learns  $i$  and  $K_{i-1}$ . Then the attacker can replace the actual  $K_{i+2,0}$  or  $MAC_{K'_i}(i|K_{i+2,0})$  with arbitrary values  $K'_{i+2,0}$  or  $MAC'$ , and forge another message:  $CDM'_i = i|K'_{i+2,0}|MAC'|K_{i-1}$ . Assume a sensor has an authentic copy of  $CDM_{i-1}$ . The sensor can verify  $K_{i-1}$  with  $K_{i-2}$ , since  $K_{i-2}$  is included in  $CDM_{i-1}$ . However, the sensor has no way to verify the authenticity of  $K'_{i+2,0}$  or  $MAC'$  without the corresponding key, which will be disclosed later. In other words, the sensor cannot distinguish between the authentic  $CDM_i$  messages and those forged by the attacker. If the sensor does not save an authentic copy of  $CDM_i$  during  $I_i$ , it will not be able to get an authenticated  $K_{i+2,0}$  even if it receives the authentication key  $K_i$  in  $CDM_{i+1}$  during  $I_{i+1}$ . As a result, the sensor cannot use the key chain  $\langle K_{i+2,0} \rangle$  during  $I_{i+2}$ .

One may suggest to distribute each  $K_{i,0}$  in some earlier time intervals than  $I_{i-2}$ . However, this does not solve the problem. If a sensor does not have an authentic copy of the  $CDM$  message, it can never get the correct  $K_{i,0}$ . To take advantage of this, an attacker can simply forge  $CDM$  messages as discussed earlier.

We propose a random selection method to improve the reliable broadcast of commitment distribution messages. For the  $CDM_i$  messages received during each time interval  $I_i$ , each sensor first tries to discard as many forged messages as possible. There is a simple test for a sensor to identify some forged  $CDM_i$  messages during  $I_i$ . The sensor can verify if  $F_0^{i-1-j}(K_{i-1}) = K_j$ , where  $K_{i-1}$  is the high-level key disclosed in  $CDM_i$  and  $K_j$  is a previously disclosed high-level key. (Note that such a  $K_j$  always exists, since the commitment  $K_0$  of the high-level key chain is distributed during the initialization of the sensor nodes.) Messages that fail this test are certainly forged and should be discarded.

The simple test can filter out some forged messages; however, they do not rule out the forged messages discussed earlier. To further improve the possibility that the sensor has an authentic  $CDM_i$  message, the base station uses a random selection method to store the  $CDM_i$  messages that pass the above test. Our goal is to make the DOS attacks so difficult that the attacker would rather use constant signal jamming instead to attack the sensor network. In other words, we want to prevent the DOS attacks that can be achieved by sending a few packets. Some of the strategies are also applicable to the low-level key chains as well as the (extended) TESLA and  $\mu$ TESLA protocols.

Without loss of generality, we assume that each copy of  $CDM_i$  has been weakly authenticated in the time interval  $I_i$  by using the aforementioned test.

### 3.4.1 Single Buffer Random Selection

Let us first look at a simple strategy: *single buffer random selection*. Assume that each sensor node only has one buffer for the  $CDM$  message broadcasted in each time interval. In a time interval  $I_i$ , each sensor node randomly selects one message from all copies of  $CDM_i$  it receives. The key issue here is to make sure all copies of  $CDM_i$  have equal probability to be selected. Otherwise, an attacker who knows the protocol may take advantage of the unequal probabilities and make a forged  $CDM$  message be selected.

To achieve this goal, for the  $k$ th copy of  $CDM_i$  a sensor node receives during the time interval  $I_i$ , the sensor node saves it in the buffer with probability  $1/k$ . Thus, a sensor node will save the first copy of  $CDM_i$  in the buffer, substitute the second copy for the buffer with probability  $1/2$ , substitute the third copy for the buffer with probability  $1/3$ , and so on. It is easy to verify that if a sensor node receives  $n$  copies of  $CDM_i$ ,

all copies have the same probability  $1/n$  to be kept in the buffer.

The probability that a sensor node has an authentic copy of  $CDM_i$  can be estimated as  $P(CDM_i) = 1 - p$ , where  $p = \frac{\# \text{forged copies}}{\# \text{total copies}}$ . To maximize his attack, an attacker has to send as many forged copies as possible.

### 3.4.2 Multiple Buffer Random Selection

The single buffer random selection can be easily improved by having additional buffers for the  $CDM$  messages. Assume there are  $m$  buffers. During each time interval  $I_i$ , a sensor node can save the first  $m$  copies of  $CDM_i$ . For the  $k$ th copy with  $k > m$ , the sensor node keeps it with probability  $\frac{m}{k}$ . If a copy is to be kept, the sensor node randomly selects one of the  $m$  buffers and replaces the corresponding copy. It is easy to verify that if a sensor node receives  $n$  copies of  $CDM_i$ , all copies have the same probability  $\frac{m}{n}$  to be kept in one of the buffers.

During the time interval  $I_{i+1}$ , a sensor node can verify if it has an authentic copy of  $CDM_i$  once it receives and weakly authenticates a copy of  $CDM_{i+1}$ . Specifically, the sensor node uses the key  $K_i$  disclosed in  $CDM_{i+1}$  to verify the MAC of the buffered copies of  $CDM_i$ . Once it authenticates a copy, the sensor node can discard all the other buffered copies.

If a sensor node cannot find an authentic copy of  $CDM_i$  after the above verification, it can conclude that all buffered copies of  $CDM_i$  are forged and discard all of them. The sensor node then needs to repeat the random selection process for the copies of  $CDM_{i+1}$ . Thus, a sensor node needs at most  $m + 1$  buffers for  $CDM$  messages with this strategy:  $m$  buffers for copies of  $CDM_i$ , and one buffer for the first weakly authenticated copy of  $CDM_{i+1}$ .

With  $m$  buffer random selection strategy, the probability that a sensor node has an authentic copy of  $CDM_i$  can be estimated as  $P(CDM_i) = 1 - p^m$ , where  $p = \frac{\# \text{forged copies}}{\# \text{total copies}}$ .

### 3.4.3 Effectiveness of Random Selection

In the rest of this subsection, we perform a further analysis using Markov Chain theory to understand the effectiveness of the random selection strategy. Specifically, we would like to compute the probability that a sensor has an authentic low-level key chain commitment before the key chain is used.

We assume that the base station sends out multiple  $CDM$  messages in each high-level time interval so that the probability of all these  $CDM$  messages being lost due to lossy channel is negligible. Since our concern is about the availability of an authentic commitment for the low-level key chain before it is used, we consider the state of a sensor only at the end of each high-level time interval.

At the end of each high-level time interval, we use  $Q_1$  to represent that a sensor buffers at least one authentic  $CDM$  message in the previous high-level time interval, and  $Q_2$  to represent that a sensor buffers at least one authentic  $CDM$  message in the current high-level time interval. We use  $\neg Q_1$  (or  $\neg Q_2$ ) to represent that  $Q_1$  (or  $Q_2$ ) is not true. Thus, with  $Q_1$ ,  $Q_2$ , and their negations, we totally have four combinations, each of which makes one possible state of the sensor. Specifically, state 1 represents  $Q_1 \wedge Q_2$ , which indicates the sensor has an authentic copy of  $CDM$  message in both the previous and the current high-level time interval. Similarly, state 2 represents  $Q_1 \wedge \neg Q_2$ , state 3 represents  $\neg Q_1 \wedge \neg Q_2$ , and state 4 represents  $\neg Q_1 \wedge Q_2$ . A sensor may transit from one state to another when the current time moves from the end of one high-level time interval to the end of the next high-level time interval.

Figure 4 shows the state transition diagram, which is equivalent to the following transition matrix:

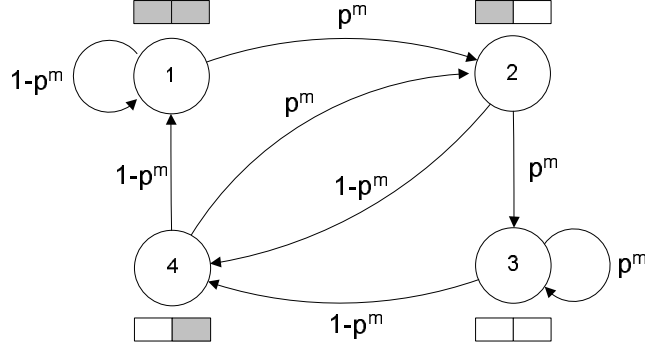


Figure 4: State transition diagram for Scheme IV

$$\mathbf{P} = \begin{pmatrix} 1-p^m & p^m & 0 & 0 \\ 0 & 0 & p^m & 1-p^m \\ 0 & 0 & p^m & 1-p^m \\ 1-p^m & p^m & 0 & 0 \end{pmatrix},$$

where  $p = \frac{\# \text{forged copies of each CDM message}}{\# \text{total copies of each CDM message}}$  and  $m$  is the number of buffers for *CDM* messages in each sensor.

Among the four states, both states 1 and 2 imply that the sensor gets an authentic key chain commitment for the low-level key chain to be used in the next high-level time interval. The reason is as follows: In both states 1 and 2, the sensor already has an authentic *CDM* message in the previous high-level time interval. Thus, it only needs a disclosed key to authenticate this message. If an attacker wants the DOS attack to be successful, he/she has to ensure the forged *CDM* messages can be weakly authenticated. As a result, the sensor can obtain a key to authenticate the *CDM* message distributed in the previous high-level time interval, and then obtain an authenticated commitment of the low-level key chain to be used in the next high-level time interval, even if it does not have an authentic copy of the *CDM* message. Therefore, the overall probability of having an authentic key chain commitment for the next key chain is the sum of the probabilities in state 1 and state 2.

To determine the probability of a sensor being in each state, we need to find the steady state of the above process. Thus, we need to solve the equation  $\Pi = \Pi \times \mathbf{P}$ , where  $\Pi = (\pi_1, \pi_2, \pi_3, \pi_4)$  and  $\pi_i$  represents the probability of the sensor being in state  $i$ . That is,

$$(\pi_1, \pi_2, \pi_3, \pi_4) = (\pi_1, \pi_2, \pi_3, \pi_4) \times \begin{pmatrix} 1-p^m & p^m & 0 & 0 \\ 0 & 0 & p^m & 1-p^m \\ 0 & 0 & p^m & 1-p^m \\ 1-p^m & p^m & 0 & 0 \end{pmatrix}.$$

By solving the above equation and considering that  $\pi_1 + \pi_2 + \pi_3 + \pi_4 = 1$ , we get

$$\begin{cases} \pi_1 = (1-p^m)^2 \\ \pi_2 = p^m(1-p^m) \\ \pi_3 = p^{2m} \\ \pi_4 = p^m(1-p^m). \end{cases}$$

Therefore, the probability that a sensor has an authentic key chain commitment for the next low-level key chain is  $P = \pi_1 + \pi_2 = 1 - p^m$ . This result shows that the more buffers we have, the more effective this

random selection strategy is. Moreover, according to the exponential form of the above formula, having a few more buffers can significantly increase the availability of an authenticated key chain commitment before the key chain is used.

### 3.4.4 Frequency of *CDM* Messages

One critical parameter in our proposed technique is the frequency of *CDM* messages. We describe one way to determine this parameter. Consider a desirable probability  $P$  that a sensor has an authenticated copy of a key chain commitment before the key chain is used. Let  $R_d$ ,  $R_c$  and  $R_a$  denote the fractions of bandwidth used by data, authentic *CDM* messages, and forged *CDM* messages, respectively. Assume each message has the same probability  $p_l$  of being lost in the communication channel. To simplify the analysis, we assume an attacker uses all available bandwidth to launch a DOS attack. Then we have  $R_d + R_c + R_a = 1$ . (Note that increasing the transmission of any type of messages will reduce the bandwidth for the other two types of messages. Thus, it is usually difficult in practice to *choose*  $R_d$ ,  $R_c$ , and  $R_a$  as desired. Here we consider the relationship among the actual rates as they happen in communication.) To ensure the probability that a sensor has an authentic low-level key chain commitment (before the use of the key chain) is at least  $P$ , we have

$$1 - \left( \frac{R_a \times (1 - p_l)}{R_c \times (1 - p_l) + R_a \times (1 - p_l)} \right)^m \geq P.$$

This implies

$$R_a \leq \frac{\sqrt[m]{1 - P}}{1 - \sqrt[m]{1 - P}} \times R_c.$$

Together with  $R_d + R_c + R_a = 1$ , we have

$$R_c \geq (1 - R_d)(1 - \sqrt[m]{1 - P}). \quad (1)$$

Equation 1 presents a way to determine the frequency of *CDM* messages to mitigate severe DOS attacks that use all available bandwidth to prevent the distribution and authentication of low-level key chain commitments. In other words, if we can determine the number  $m$  of *CDM* buffers based on resources on sensors, the fraction  $R_d$  of bandwidth for data packets based on the expected application behaviors, the probability  $P$  of a sensor authenticating a low-level key chain commitment before the key chain is used based on the expected security performance under severe DOS attacks, we can compute  $R_c$  and then determine the frequency of *CDM* messages. Moreover, we may examine different choices of these parameters and make a trade-off most suitable for the sensor networks.

Figure 5 shows the fraction of bandwidth required for *CDM* messages for different combinations of  $R_d$  and  $m$  given  $P = 0.9$ . We can see that the bandwidth required for *CDM* messages in order to ensure  $P = 0.9$  is substantially more than that required to deal with message losses. For example, as shown in Figure 5(a), when there are few data packets and each sensor has only 10 buffers for *CDM* messages, about 20% of the bandwidth must be used for *CDM* messages in order to ensure 90% authentication rate for low-level key chain comments when there are severe DOS attacks. This is understandable since under such circumstances the sensor network is facing aggressive attackers that try everything possible to disrupt the normal operations of the network.

It is also shown in Figure 5(b) that the increase in the number of *CDM* buffers can significantly reduce the requirement for *CDM* messages. As shown in Figure 5(b), when each sensor has 40 *CDM* buffers, less than 5% of the bandwidth is required for *CDM* messages. In addition, the shape of the curves in Figure 5(b) also shows that the smaller  $m$  is, the more effective an increase in  $m$  is.

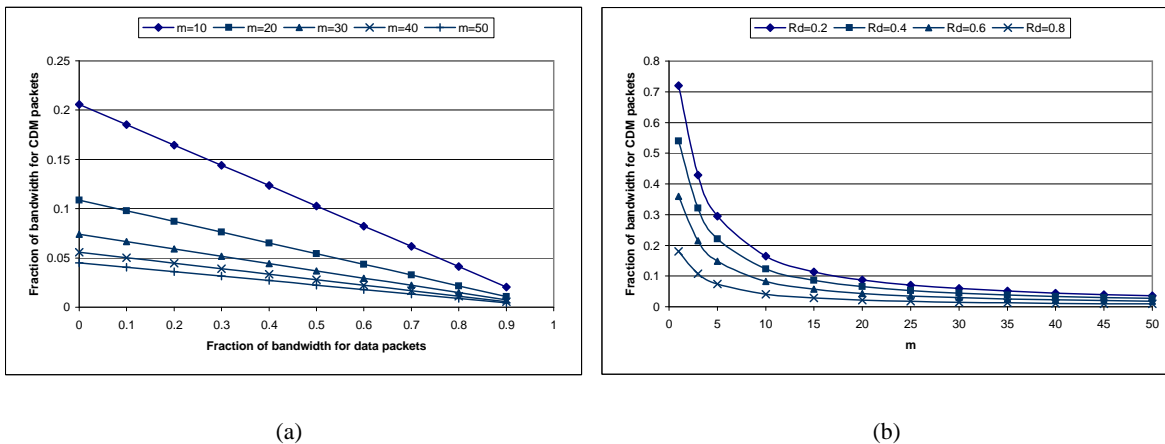


Figure 5: Bandwidth required for  $CDM$  messages to ensure 90% of low-level key chain commitments are authenticated before the key chains are used.

Figure 5(a) further shows that the increase in data rate results in the decrease in the fraction of bandwidth required for  $CDM$  messages. This is because when the data consume more bandwidth, there is less bandwidth for the DOS attacks, and in effect the requirement for  $CDM$  messages is also reduced.

It is worth noting that the fractions for data and  $CDM$  messages are the *actual* fractions of  $CDM$  messages that the sensors receive, not the fractions *planned* by the base station. A message scheduled for transmission by the base station is not guaranteed to be transmitted if the DOS attack consumes too much bandwidth. Nevertheless, the above analysis provides a target frequency of  $CDM$  messages, and the base station can adaptively change its transmission strategy to meet this target.

### 3.5 Scheme V: DOS-Resistant Two-Level $\mu$ TESLA

Scheme IV can be further improved if the base station has enough computational and storage resources. Indeed, when at least one copy of each  $CDM$  message can reach the sensors, we can completely defeat the aforementioned DOS attack without the random selection mechanism.

The solution can be considered a variation of the immediate authentication extension to TESLA [26]. The idea is to include in  $CDM_i$  the image  $H(CDM_{i+1})$  for each  $i$ , where  $H$  is a pseudo random function. As a result, if a sensor can authenticate  $CDM_i$ , it can get authentic  $H(CDM_{i+1})$  and then authenticate  $CDM_{i+1}$  when it is received. Specifically, the base station constructs  $CDM_i$  for the high-level time interval  $I_i$  as follows:

$$CDM_i = i|K_{i+1,0}|H(CDM_{i+1})|MAC_{K'_i}(i|K_{i+1,0}|H(CDM_{i+1}))|K_{i-1},$$

where “|” denotes message concatenation,  $H$  is a pseudo random function other than  $F_0$  and  $F_1$ , and  $K'_i$  is derived from  $K_i$  with a pseudo random function other than  $H$ ,  $F_0$  and  $F_1$ .

Suppose a sensor has received  $CDM_i$ . Upon receiving  $CDM_{i+1}$ , the sensor can authenticate  $CDM_i$  with  $K_i$  disclosed in  $CDM_{i+1}$ . Then the sensor can immediately authenticate  $CDM_{i+1}$  by verifying that applying  $H$  to  $CDM_{i+1}$  results in the same  $H(CDM_{i+1})$  included in  $CDM_i$ . As a result, the sensor can authenticate a commitment distribution message immediately after receiving it.

Alternatively, if  $H(CDM_1)$  is pre-distributed before deployment, the sensor can immediately authenticate  $CDM_1$  when receiving it, and then use  $H(CDM_2)$  included in  $CDM_1$  to authenticate  $CDM_2$ , and so on.

One may observe that in this case, a sensor does not use the disclosed high-level keys in  $CDM$  messages directly. However, including such keys in  $CDM$  messages is still useful. Indeed, when a sensor fails to receive or keep an authentic  $CDM$  message, it can use the random selection mechanism and the approach described in the previous paragraph to recover from the failure.

The cost, however, is that the base station has to compute the  $CDM$  messages in the reverse order. That is, in order to include  $H(CDM_{i+1})$  in  $CDM_i$ , the base station has to have  $CDM_{i+1}$ , which implies that it also needs  $CDM_{i+2}$ , and so on. Therefore, the base station needs to compute both the high-level and the low-level key chains completely to get the commitments of these key chains, and construct all the  $CDM$  messages in the reverse order before the distribution of the first one of them. (Note that in scheme IV, the base station only needs to compute the high-level key chain but not all the low-level ones during initialization. The base station may delay the computation of a low-level key chain until it needs to distribute the commitment of that key chain.)

This imposes additional computation during the initialization phase. Assume that all the key chains have 1,000 keys. The base station needs to perform about 1,001,000 pseudo random function operations to generate all the key chain commitments, and 1,000 pseudo random function operations and 1,000 MAC operations to generate all the  $CDM$  messages. Due to the efficiency of pseudo random functions, such computation is still practical if the base station is relatively resourceful. For example, using MD5 as the pseudo random function, a modern PDA can finish the above computation in several seconds. Moreover, the base station does not have to save the low-level key chains. Indeed, to reduce the storage overhead, the base station may compute a low-level key chain (again) when the key chain is needed. Thus, the base station only needs to store the high-level key chain and the MACs of all the  $CDM$  messages. Further assume both the authentication key and the image of a pseudo random function are 8 bytes. To continue the earlier example, the base station needs  $(8 + 8) \times 1,000 = 16,000$  bytes to store the high-level key chain and the MACs.

The immediate authentication of  $CDM_i$  depends on the successful receipt of  $CDM_{i-1}$ . However, if a sensor cannot receive an authentic  $CDM_i$  due to communication failure or an attacker's active disruption, the sensor has to fall back to the techniques introduced in Scheme IV (i.e., the random selection strategies). This implies that the base station still needs to distribute  $CDM$  messages multiple times in a random manner. The combination of these techniques is straightforward; we do not discuss it further in this paper.

Now let us assess how difficult it is for a sensor to recover if it fails to receive an authentic  $CDM$  message. We assume an attacker will launch a DOS attack to deter this recovery. To recover from the failure, the sensor has to buffer an authentic  $CDM$  message by the end of a later high-level time interval and then authenticate this message. For example, suppose a sensor buffers an authentic  $CDM_{i+j}$ . If it receives a disclosed key in interval  $I_{i+j+1}$ , it can authenticate  $CDM_{i+j}$  immediately and gets  $H(CDM_{i+j+1})$ . The sensor then recovers from the failure. Thus, if a sensor fails to receive an authentic  $CDM_i$ , the probability that it recovers from this failure within the next  $l$  high-level time intervals is  $1 - p^{m \times l}$ , where  $p = \frac{\# \text{forged copies of each } CDM \text{ message}}{\# \text{total copies of each } CDM \text{ message}}$  and  $m$  is the number of buffers for  $CDM$  messages.

It is sensible to dynamically manage  $CDM$  buffers in sensors in this scheme. There are three cases: (1) During normal operations, each sensor only needs one buffer to save an authenticated  $CDM$  message during each high-level time interval; (2) When a sensor tries to recover from communication failures, it needs a relatively small number of  $CDM$  buffers to tolerate communication failures, as discussed in Section 3.3; (3) When a sensor tries to recover from a loss of authentic  $CDM$  messages under severe DOS attacks, the sensor needs as many buffers as possible to increase its chance of recovery. Once a sensor recovers an authentic  $CDM$  message, it can fall back to only one  $CDM$  buffer, since it can authenticate the next  $CDM$  message once the message is received. This requires that each sensor be able to detect the presence of DOS attacks. Fortunately, this can be done easily with high precision: If most buffered  $CDM$  messages are forged, there must be a DOS attack.

The base station needs to broadcast each *CDM* message multiple times to mitigate communication failures and to help sensors recover from failures under potential DOS attacks. The frequency of *CDM* messages required in this scheme can be determined in a similar way to scheme IV. However, a sensor in this scheme only needs a large number of *CDM* buffers temporarily during recovery. Moreover, a sensor only needs to recover one authentic *CDM* message in order to go back to normal operations, and the sensor may recover over several high-level time intervals. Indeed, if we allow a sensor to recover from such a failure over  $l$  high-level time intervals, by using the same process to derive Equation 1, we can get the following equation:

$$R_c \geq (1 - R_d)(1 - \sqrt[m \cdot l]{1 - P}), \quad (2)$$

where  $R_c$  is the fraction of bandwidth required for *CDM* messages,  $R_d$  is the fraction of bandwidth used by data packets,  $m$  is the number of buffers for *CDM* messages, and  $P$  is the desired probability to recover from the failure over the next  $l$  high-level time intervals. It is easy to see that  $R_c$  decreases when  $m$  and  $l$  increase. Thus, the bandwidth required for *CDM* messages can be much less than in scheme IV.

Since the probability that a sensor fails to receive an authentic *CDM* message is unknown, it is not possible to derive the probability that the sensor has an authentic low-level key chain commitment before the key chain is used. Nevertheless, this probability can be easily computed in the same way as in Section 3.4 if the aforementioned information is available.

From the above analysis, we can see that this scheme introduces additional computation requirement before deployment, though it can defeat the DOS attacks when at least one copy of each *CDM* message reaches the sensors. Fortunately, such computation is affordable if the base station is relatively resourceful. It is also possible to perform such computation on powerful machines and then download the result to the base station before deployment. In addition, the communication overhead and the sensor storage overhead in this scheme is potentially much less than that in scheme IV, as discussed earlier. Thus, when the required computational resources are available (on either the base station or some other machines), scheme V is more desirable. Otherwise, scheme IV could be used to mitigate the DOS attacks.

### 3.6 Scheme VI: Multi-Level $\mu$ TESLA

Both scheme IV and scheme V can be extended to  $M$ -level key chain schemes. The  $M$ -level key chains are arranged from level 0 to level  $M - 1$  from top down. The keys in the  $(M - 1)$ -level key chains are used for authenticating data packets. Each higher-level key chain is used to distribute the commitments of the immediately lower-level key chains. Only the last key of the top-level (level 0) key chain needs to be selected randomly; all the other keys in the top-level key chain are generated from this key, and all the key chains in level  $i$ ,  $1 \leq i \leq M - 1$ , are generated from the keys in level  $i - 1$ , in the same way that the low-level key chains are generated from the high-level keys in the two-level key chain schemes. For security concerns, we need a family of pseudo random functions. The pseudo random function for each level and between adjacent levels should be different from each other. Such a family of pseudo random functions has been proposed in [25].

The benefit of having multi-level key chains is that it is more flexible in providing short key chains with short delays in authenticating data packets, compared with the two-level key chain schemes. As a result, a multi-level  $\mu$ TESLA scheme can scale up to cover a long period of time. In practice, a three-level scheme is usually sufficient to cover the lifetime of a sensor network. For example, if the duration of a lowest-level time interval is 100ms, and each key chain has 1,000 keys, then a three-level scheme can cover a period of  $10^8$  seconds, which is over three years. In the following, we still present our techniques as generic multi-level key chains schemes for the sake of generality.

In addition to multi-level  $\mu$ TESLA schemes directly extended from schemes IV and V, we can combine them into a hybrid scheme to achieve a trade-off between precomputation and operational overheads. Thus,

we have three variations of multi-level  $\mu$ TESLA schemes. The first variation, which is named *DOS-tolerant multi-level  $\mu$ TESLA*, is extended from scheme IV and is suitable for sensor networks where the base station is not very resourceful. The second variation, which is named *DOS-resistant multi-level  $\mu$ TESLA*, is extended from scheme V. This variation is suitable for sensor networks with relatively short lifetime and relatively powerful base stations. The third variation, which is named *hybrid multi-level  $\mu$ TESLA*, is a trade-off between the above two variations. It sacrifices certain immediate authentication capability to exchange for less precomputation requirement.

In the following, we describe and analyze these variations, respectively.

### 3.6.1 Variation I: DOS-Tolerant Multi-Level $\mu$ TESLA

This variation of multi-level  $\mu$ TESLA scheme is a direct extension to scheme IV. Each *CDM* message has the same format as in scheme IV, and each sensor uses the multiple buffer random selection mechanism to save *CDM* messages. The only difference is that this variation may have more than two key chain levels.

Compared with scheme IV, this variation is not more vulnerable to DOS attacks. The success of the DOS attacks depends on the percentage of forged *CDM* messages and the buffer capacity in sensor nodes. As long as the base station maintains a certain authentic *CDM* message rate, this variation will not have higher percentage of forged *CDM* messages than scheme IV. The base station can further piggy-back the *CDM* messages for different levels of key chains so as to reduce the communication cost.

Having more levels of key chains does increase the overhead at both the base station and the sensor nodes. This variation requires the base station to maintain one active key chain at each level. Because of the available resource in typical bases stations, this overhead is usually tolerable. Similarly, sensor nodes have to maintain more buffers for the key chain commitments as well as *CDM* messages in different key chain levels. This is usually not desirable because of the resource constraints in sensors. In addition, the more levels we have, the more bandwidth is required to transmit the *CDM* messages. Thus, we should use as few levels as possible to cover the lifetime of a sensor network.

Now let us consider the frequency of *CDM* messages in DOS-tolerant multi-level  $\mu$ TESLA. To increase the chance to succeed, the attacker may target at a particular key chain level instead of attacking all levels simultaneously. Further assume that the base station sends out the *CDM* messages of each key chain level in the same frequency, and the buffer in each sensor can accommodate  $m$  (authentic and/or forged) copies of a *CDM* message. Thus, for DOS-tolerant  $M$ -level  $\mu$ TESLA, Equation 1 can be generalized to

$$R_c \geq \frac{(M-1)(1-R_d)(1-\sqrt[m]{1-P})}{(M-1)(1-\sqrt[m]{1-P}) + \sqrt[m]{1-P}}, \quad (3)$$

where  $R_c$  is the fraction of bandwidth required for *CDM* messages in all key chain levels, and  $R_d$  is the fraction of bandwidth used for data packets,  $m$  is the number of *CDM* buffers in each key chain level, and  $P$  is the desired probability that a sensor has an authenticated key chain commitment before the key chain is used.

We may still use the approach in Section 3.4.4 to determine the frequency of *CDM* messages in order to maintain broadcast authentication service when the network is under severe DOS attacks. Figure 6 shows the required fraction of bandwidth for *CDM* messages to guarantee that each sensor has the probability  $P = 0.9$  to have an authenticated low-level key chain commitment before the key chain is used. It is easy to see that the addition of more key chain levels does introduce additional communication overhead. Similar to Figure 5, Figure 6 shows smaller fraction of bandwidth required for *CDM* messages when the data rate is higher. As discussed earlier, the increase in data rate consumes more bandwidth for data and leaves less bandwidth for forged *CDM* messages. As a result, the requirement for *CDM* messages is also reduced.

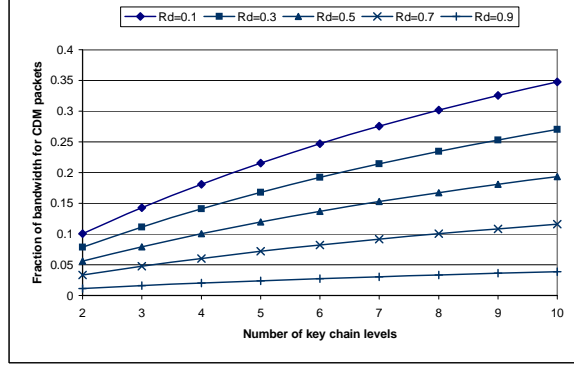


Figure 6: Bandwidth for *CDM* messages v.s. number of key chain levels. Assume the number of *CDM* buffers in each key chain level is  $m = 40$ .

In the following, we give an analysis of the overheads introduced by the DOS-tolerant multi-Level  $\mu$ TESLA scheme. For simplicity, we assume there are totally  $M$  levels in our scheme and  $L$  keys in each key chain. Thus, if the duration of each lowest-level time interval (level  $M - 1$ ) is  $\Delta$ , the duration of each level  $i$  time interval is  $\Delta_i = \Delta \times L^{M-i-1}$ , and the maximum lifetime of the scheme is  $\Delta \times L^M$ .

The storage overhead in sensors is mainly due to the buffer of *CDM* messages. Each sensor has to buffer weakly authenticated *CDM* messages for the top  $M - 1$  levels. Assuming a sensor uses  $m$  *CDM* buffers, this totally requires about  $m \cdot (M - 1)$  buffers. (Note that for each *CDM* message, only the disclosed key chain commitment and the MAC need to be stored.) In addition, each sensor needs to store 1 most recently authenticated key for level 0 key chain and 3 most recently authenticated keys for each of the other levels (one for the previous key chain because it is possible that the sensor receives a packet which discloses a key in the previous key chain, another for the current key chain, and a third for the next key chain). Thus, each sensor needs to store  $3M - 2$  more keys.

A base station only needs to keep the current key chain for each level, which occupies at most  $M \times L$  storage space in total. This is because a lower-level key chain can be generated directly from a key in its adjacent upper-level key chain, and the length of key chain in our technique can be short enough to allow computation of a key chain in real time. In contrast, in the original  $\mu$ TESLA scheme [27], the base station has to precompute and store  $L^M$  keys to cover the same period of time as in our scheme.

Consider the communication overhead due to the *CDM* messages. In order to mitigate severe DOS attacks, the base station has to use a fair amount of bandwidth to broadcast *CDM* messages, as indicated by Equation 3. For example, Figure 6 shows that when the fraction of bandwidth for data packets is 0.1, the number of key chain levels is 3, and each sensor has 40 buffers for each *CDM* message, the base station needs about 15% of the bandwidth for *CDM* messages.

The computational overhead in sensors is mainly due to the authentication of disclosed keys and MACs. A sensor's computation for data packets is dependent on the number of data packets the sensor receives. However, a sensor's computation for *CDM* packets is bounded by the number  $m$  of *CDM* buffers, since the sensor has at most  $m$  copies of each *CDM* message, and it can stop once it authenticates a copy.

As discussed earlier, in the original  $\mu$ TESLA protocol, if there is a long delay between the receipts of two data packets, a sensor has to perform a large number of pseudo random functions in order to authenticate the key disclosed in the packet. In the worst case, it has to perform about  $L^M$  pseudo random functions if it only receives the first and the last packets. In contrast, with the DOS-tolerant multi-level  $\mu$ TESLA scheme, such a sensor needs to perform at most  $M \times L$  pseudo random functions. In general, if a sensor does not

receive packets for  $n_l$  lowest-level time intervals, the number of pseudo random functions that it needs to perform in order to authenticate a key received later never exceeds  $L \times \log_L(n_l)$ .

It appears that the overheads in this scheme, especially the communication overhead and the storage overhead in sensors, are not negligible. In the following, we introduce the second variation of multi-level  $\mu$ TESLA scheme that is more efficient in terms of communication overhead and storage overhead in sensors.

### 3.6.2 Variation II: DOS-Resistant Multi-Level $\mu$ TESLA

The DOS-resistant multi-level  $\mu$ TESLA scheme is extended directly from scheme V. There are multiple key chain levels, with lower-level key chains generated from keys in the immediately higher-level key chains. There are multiple key chains in all levels except for level 0. Among these levels, only level  $M - 1$  is used to authenticate data packets; all the other levels are used to distribute the key chain commitments in the immediately lower-level. Each  $CDM$  message consists of the image of the next  $CDM$  message under a pseudo random function. In level  $i$ ,  $0 < i < M - 1$ , the last  $CDM$  message in an earlier key chain contains the image of the first  $CDM$  message in the immediately next key chain. As a result, the end of a key chain does not interrupt the immediate authentication of later  $CDM$  messages in the same level.

Similar to its two-level counter part, this scheme requires precomputation to generate all the key chains in each level and all the  $CDM$  messages. This computation cost could be prohibitive if the lifetime of a sensor network is very long. However, it may be tolerable for relatively short-lived sensor networks. For example, consider a three-level scheme with 100 keys in each key chain and 100ms lowest-level time intervals. Such a scheme can cover  $10^5$  seconds, which is about 27 hours. The precomputation required to initialize the scheme consists of 1,010,100 pseudo random functions to generate all the key chains, and 10,100 pseudo random functions to generate all the  $CDM$  messages. Such computation can be finished in several seconds on a modern PC or PDA. Thus, the precomputation can be either performed on base stations directly, or performed on a regular PC and then downloaded to the base station.

The base station does not have to store all these values due to the low cost involved in computing pseudo random functions. To continue the above example, the base station may simply store the keys for the active key chain of each level and the images of  $CDM$  messages under pseudo random functions. Assume that both a key and an image of a pseudo random function takes 8 bytes. Then the base station only needs to save about  $8 \times 300 + 8 \times 10,100 \approx 82$  KBytes.

In general, for a DOS-resistant  $M$ -level  $\mu$ TESLA scheme, where each key chain consists of  $L$  keys, a base station needs to precompute  $L + L^2 + \dots + L^M = \frac{L^{M+1}-L}{L-1}$  keys and  $L + L^2 + \dots + L^{M-1} = \frac{L^M-L}{L-1}$   $CDM$  messages, respectively. In addition, the base station needs to store  $M \times L$  keys and  $\frac{L^M-1}{L-1}$   $CDM$  images, respectively. Additional trade-off is possible to reduce the storage requirement (by not saving but computing some  $CDM$  images when they are needed) if the base station does not have space for all these keys and  $CDM$  images.

This scheme inherits the advantage of its two-level counter part. That is, a sensor can get an authenticated key chain commitment as long as it receives one copy of the corresponding  $CDM$  message. As we discussed in Section 3.5, this property substantially reduces the communication overhead introduced by  $CDM$  messages, since the base station only needs to send enough copies of a  $CDM$  message to make sure the sensors have a high probability to receive  $CDM$  messages during normal operations, and have a high probability to recover from failures over a period of time when the sensors are under DOS attacks. Specifically, if we would like a sensor to recover from a failure of receiving a  $CDM$  message within  $l$  time intervals (in the same level), by using the same process to get Equation 3, we have the following equation:

$$R_c \geq \frac{(M-1)(1-R_d)(1-\sqrt[m]{1-P})}{(M-1)(1-\sqrt[m]{1-P}) + \sqrt[m]{1-P}}, \quad (4)$$

where  $R_c$  is the fraction of bandwidth required for *CDM* messages in all key chain levels, and  $R_d$  is the fraction of bandwidth used for data packets,  $m$  is the number of *CDM* buffers in each key chain level, and  $P$  is the desired probability that a sensor recovers from the failure over the next  $l$  time intervals. It is easy to verify that when  $m$  and  $l$  increase, the right hand side of Equation 4 decreases, and so does the requirement for  $R_c$ . Moreover, a sensor may use dynamic buffer management as discussed in Section 3.5 to arrange buffers for *CDM* messages. Though a *CDM* message in this scheme is slightly larger than that in variation I (by one pseudo random function image per *CDM* message), the frequency of *CDM* messages can be reduced substantially. Thus, the overall storage requirement in sensors can be much less than that in variation I.

The computational overhead in a sensor is not as clear as in variation I. In variation I, the number of authentication a sensor needs to perform is bounded by the number of *CDM* buffers. In contrast, in this scheme, a sensor may only need to authenticate one copy of *CDM* message if the first received message is authentic, but may also have to authenticate every received copy of a *CDM* message if no copy is authentic in the worst case.

The limitation of this variation is its scalability. It is easy to see that the precomputation cost is linear to the number of lowest-level time intervals. Consider a long-lived sensor network that requires a 3-level key chains scheme, where each key chain consists of 1,000 keys and the duration of each lowest-level time interval is 10ms. The lifetime of this scheme is  $10^7$  seconds, which is about 116 days. Using 3-level key chains implies that the base station needs to precompute about 1,001,001,000 pseudo random functions to compute the key chains and another 1,001,000 pseudo random functions to compute the images of *CDM* messages. In addition, the base station needs to store about 3000 keys and 1,001,000 images of pseudo random functions, which take about 8 MBytes memory. Though this is still feasible for typical PCs and workstations, it may be too expensive for base stations that are not very resourceful.

### 3.6.3 Variation III: Hybrid Multi-Level $\mu$ TESLA

Variation III is essentially a trade-off between the first two variations. To make the techniques in variation II practical for low-end base stations, we reduce the precomputation and storage overheads by sacrificing certain immediate authentication capability. Specifically, we limit the precomputed *CDM* messages to the active key chain being used in each level. For a given key chain in a particular level, the base station computes the images of the *CDM* messages (under the pseudo random function  $H$ ) only when the first key is needed for authentication, and this computation does not go beyond this key chain in this level. As a result, the *CDM* message authenticated with the last key in a key chain will not include the image of the next *CDM* message in the same level, because this information is not available yet. The base station may simply set this field as NULL. For the first key chain in each level  $i$ , where  $0 \leq i \leq M - 1$ , the image of the first *CDM* message can be distributed during the initialization phase.

The behavior of a sensor is still very simple. If the sensor has an authentic image of the next *CDM* message in a certain level, it can authenticate the next *CDM* message immediately after receiving it. Otherwise, the sensor simply uses the random selection strategy to buffer the weakly authenticated copies. To increase the chance that the sensors receive an authentic image of the first *CDM* message for a key chain, the base station may also broadcast it in data packets.

Such a method reduces the computation and storage requirement significantly compared with variation II. For an  $M$ -level  $\mu$ TESLA with  $L$  keys in each key chain, the base station only needs to precompute around  $M \cdot L$  pseudo random functions and store  $(M - 1) \cdot L$  images of *CDM* messages. In the earlier example with 3-level key chains and 1,000 keys per key chain, the base station only needs to compute about 3,000 (instead of 1,001,001,000 in variation II) pseudo random operations during initialization and store 2,000 (instead of 1,001,000 in variation II) *CDM* images.

An obvious weak point of this multi-level  $\mu$ TESLA scheme is the handover of two consecutive key chains in the same level. Consider two consecutive key chains in level  $i$ , where  $i < M - 1$ . These key chains are used to distribute  $CDM$  messages for the immediately lower-level key chains. For all the keys except for the last one in each key chain, the corresponding  $CDM$  messages include an image of the next  $CDM$  message, which enables a sensor to authenticate the next  $CDM$  message immediately after receiving it. However, the last  $CDM$  message corresponding to the earlier key chain does not have an image of the first  $CDM$  message corresponding to the later key chain, as discussed earlier. Thus, the first  $CDM$  message of the later key chain cannot be authenticated immediately after it is received, though the commitment of this key chain can be authenticated with the immediately upper-level  $CDM$  message. As a result, a sensor has to wait for the next  $CDM$  message to disclose the corresponding  $\mu$ TESLA key in order to authenticate the first  $CDM$  message.

An attacker may take advantage of this opportunity to launch DOS attacks. However, this scheme will not perform worse than variation I, since each sensor can always fall back the random selection mechanism to mitigate the impact of such an attack. In addition to the dynamic buffer management discussed in Section 3.5, the base station can also use an adaptive method to determine the frequency of  $CDM$  messages to improve the resistance against DOS attacks without substantially increasing the communication overhead. That is, the base station may use a low frequency to send out  $CDM$  messages corresponding to later intervals in a key chain, and use a high frequency for the early ones. The analysis performed for variation I to decide the desirable frequency of  $CDM$  messages is also applicable to variation III.

Though having less overhead than variation II, variation III introduces more overheads into base stations than variation I. Besides computing a key chain before using it, a base station using this variation has to compute all the corresponding  $CDM$  messages, since each earlier  $CDM$  message includes the image of the immediately following  $CDM$  message. The storage overhead in the base station in this scheme is also higher than that in variation I due to the storage of these  $CDM$  messages.

Variation III introduces lower overheads in sensors than variation I, but has higher overheads than variation II. In normal situations when a sensor has an authenticated image of the following  $CDM$  message, it only needs to save one copy of that  $CDM$  message. A sensor's computation and storage overheads are the same as in variation II. During the handover of two key chains (in the same level), a sensor needs to increase the number of  $CDM$  buffers to mitigate potential DOS attacks. This is similar to variation I. However, unlike in variation I, a sensor using variation III can recover to the above normal situation once it authenticates one  $CDM$  message. This is essentially the same as recovering from failures (to receive an authentic  $CDM$  message) in variation II. As discussed earlier, the storage overhead in sensors is much smaller than that in variation I when the sensors are allowed to recover over several time intervals. But such overheads in a sensor using variation III are higher than in variation II, since such recovery processes are "scheduled" in addition to those due to failures.

A sensor using variation III may use an adaptive approach to save  $CDM$  messages during handover of key chains. Specifically, a sensor may just save a few (or even a single copy of) of the first  $CDM$  message corresponding to a new key chain. When the next  $CDM$  message arrives, the sensor can then decide whether there is an on-going DOS attack by attempting to authenticate the earlier  $CDM$  message. If the earlier  $CDM$  message is authenticated, the sensor can continue to authenticate later  $CDM$  messages with the corresponding image; otherwise, the sensor can determine that there is a DOS attack and adaptively increase the number of  $CDM$  buffers.

Consider the communication overhead in variation III introduced by  $CDM$  messages. We can use Equation 4 to determine the frequency of  $CDM$  messages given the fraction of bandwidth used by data packets, the number  $M$  of key chain levels, the number  $m$  of  $CDM$  buffers in each sensor, and the probability  $P$  that a sensor recovers from a failure (or get the first authenticated  $CDM$  message for a key chain) over  $l$  time intervals. The base station may increase the frequency of the first several  $CDM$  messages in a key chain

based on Equation 3 to increase their probability to be authenticated by sensors. Thus, the communication overhead in variation III is between those of variation I and variation II.

Among these variations, variation II has a distinctive advantage over the other two variations. Indeed, variation II can substantially reduce the impact of DOS attacks. In order to get an authentic key chain commitment in a *CDM* message, a sensor only needs to receive an authentic copy of this message in most of cases, since the sensor can immediately authenticate it. Though a sensor has to rely on the random selection mechanism to recover from failures, the cost is much less than those required by variations I and III. The disadvantage of variation II is its precomputation and storage overhead. Thus, if the base station has enough resources, variation II should be used. Variation III sacrifices some immediate authentication capability to reduce the precomputation and storage requirements in variation II. Thus, if the base station has certain, but not enough resources, variation III should be used. If the base station cannot afford the precomputation and storage overheads required by variation III at all, variation I can be used to mitigate the potential DOS attacks.

## 4 Multi-Level $\mu$ TESLA on TinyOS

We have implemented the DOS-tolerant multi-level  $\mu$ TESLA scheme on TinyOS [16], which is an operating system for networked sensors. The software package can be downloaded on our website<sup>1</sup>. We cannot directly implement the DOS-resistant multi-level  $\mu$ TESLA scheme on TinyOS due to the maximum payload size (29 bytes) supported by TinyOS. It might be possible to implement the DOS-resistant multi-level  $\mu$ TESLA scheme by transmitting a *CDM* message over multiple packets, or modify TinyOS to support larger packets; however, we do not pursue such methods in this paper, but consider them possible future work.

Following [27], we implemented pseudo random functions with a MAC, which was implemented using the CBC-MAC [32] with RC5 [28] as the block cipher. Our implementation uses RC5 with 32 bit words, 12 rounds, and 8 byte keys.

In our system, the number of levels, denoted *MAX\_LEVEL*, is predetermined at the compiling time. The type of a packet, which is either a data packet or a *CDM* packet, is indicated by the first byte in the packet. (*MAX\_LEVEL* - 1) indicates that the packet is a data packet, while other smaller, non-negative integer indicates that the packet is a *CDM* packet in the corresponding level.

Each *CDM* packet consists of the following fields: level (1 byte), index (of time interval) (4 bytes), key chain commitment (8 bytes), MAC (8 bytes), and disclosed key (8 bytes). Thus, the total length of a *CDM* message is 29 bytes. Note that this is already the maximum payload size supported by TinyOS. The DOS-resistant variation of the multi-level  $\mu$ TESLA scheme requires an additional field of image of next *CDM* message (8 bytes), and thus cannot be directly accommodated by TinyOS. Details about *CDM* as well as data packets can be found in Appendix B.

The architecture of the broadcast authentication system is illustrated in Figure 7. The system consists of 5 components: *RC5*, *CBCMAC*, *SecPrimitive*, *Sender*, and *Receiver*. The components in the dashed box, *RC5* and *CBCMAC*, are directly adopted from the TinySec package<sup>2</sup>. The *SecPrimitive* component, which is developed based on the *CBCMAC* component, provides security primitives, including pseudo random function, pseudo random number generator, generation of key chains, and generation and verification of MAC.

These components were written in nesC [12], a C-like programming language used to develop TinyOS and its applications. All the three components we developed are quite simple. In the current version, the

<sup>1</sup><http://discovery.csc.ncsu.edu/software/ML-microTESLA/>.

<sup>2</sup><http://www.cs.berkeley.edu/~nks/tinysec/>.

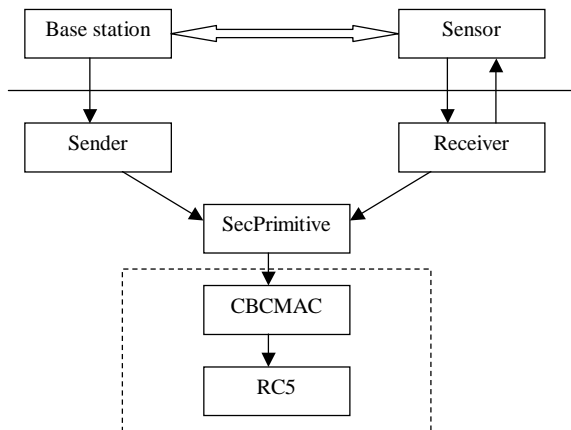


Figure 7: The multi-level  $\mu$ TESLA broadcast authentication system on TinyOS.

Sender component has about 160 lines of source code, the Receiver component has about 280 lines of source code, and the SecPrimitive component has about 60 lines of source code.

The Sender and the Receiver components are intended to provide broadcast authentication services for the base station application and the sensor application, respectively, using the interfaces provided by the SecPrimitive component. The application in the base station may call the corresponding interfaces in the Sender component to initialize the system, or send authenticated data packets. The application in a sensor node may call the interfaces in the Receiver component to initialize the system. Whenever a data packet is authenticated, the Receiver component triggers an event to pass the authentic data packet to the sensor application.

We use the multiple buffer random selection strategy at sensors to deter DOS attacks against the *CDM* packets. To simplify the buffer management at the resource constrained sensors, we choose to predetermine the number of data and *CDM* buffers. The sensor application may decide these numbers during the initialization phase. The numbers of data and *CDM* buffers are certainly important to the overall performance. In Section 5, we perform a series of experiments to show how these parameters may be selected to deter intensive DOS attacks.

The generation and processing of *CDM* packets are handled by the broadcast authentication system, and hidden from the applications on base station or sensors after system initialization. The base station has to determine the frequency and the sending time of *CDM* packets. The frequency of *CDM* packet can be determined in many ways. (See Section 3.6 for one way to determine this parameter.) Thus, we let the application determine the frequency of *CDM* packets and passes it to the Sender component. Whenever the frequency of *CDM* packets is determined, it is trivial to decide when to send *CDM* packets. The generation and broadcast of *CDM* packets are triggered by a timer event in the Sender component, which is set during the initialization of the system.

#### 4.1 Multi-Level $\mu$ TESLA API

We have developed a broadcast authentication API on TinyOS to support sensor network applications that require broadcast authentication services. The API for base station applications is contained in the Sender component, while the API for sensor applications is provided in the Receiver component. In the following, we first present the API, and then discuss how to build applications based on this API. The details of data structures and functions related to this API can be found in Appendix B.

The configuration and security of the broadcast authentication system depends on several parameters

such as number of levels and maximum clock discrepancy between the base station and the sensors. It may be possible to set default values of these parameters for certain applications when the configuration (e.g., lifetime, maximum clock discrepancy) of the sensor network and acceptable risks are well understood. However, these are not possible without knowing the actual hardware and the applications. Therefore, we do not consider this option in this paper, but leave the options to the sensor applications.

The Sender component provides the following functions:

- **Sender.init:** It initializes the sender based on the given configuration, which includes the master key used to generate all the key chains, the length of the key chains and the durations of time intervals in each level, the starting time of multi-level  $\mu$ TESLA protocol, the key disclosure lag of the data packets and the frequency of *CDM* packets.
- **Sender.sendData:** It generates a MAC for a given data packet. The caller needs to fill in the data payload field in the data packet. This function then fills in all the other fields, including the level (indicating a data packet), the index of current time interval for the lowest-level key chain, MAC, and the disclosed key for an earlier time interval. Finally, this function invokes the send function in communication module to broadcast the data packet.
- **Sender.generateConf:** It generates the configuration parameters for the receivers that will receive the authentic packets from this particular sender. The configuration parameters include the length of the key chains as well as the duration of time intervals in each level, the starting time of multi-level  $\mu$ TESLA protocol, the key disclosure lag of the data packet, the commitments of the first key chain in all levels, and the commitments of the second key chain in all levels other than level 0. We assume that there are other ways to derive the maximum clock discrepancy and maintain loose time synchronization between the sender and the receivers. These configuration parameters should be given to the receivers through a secure channel that can ensure the integrity of these parameters.

The Receiver component provides the following functions:

- **Receiver.init:** It initializes the receiver based on a given configuration, which includes the length of the key chains as well as the duration of time intervals in each level, the starting time of multi-level  $\mu$ TESLA protocol, the key disclosure lag of the data packet, the commitments of the first key chain in all levels, the commitments of the second key chain in all levels other than level 0, and the maximum clock discrepancy.
- **Receiver.authenticDataReady:** This is an event in TinyOS; it is signaled whenever a data packet is authenticated and ready to be passed to the receiver application. The sensor application must implement an event handler to process the authenticated data packet.

## 4.2 Building Applications Based on Multi-Level $\mu$ TESLA API

We use some simple examples to show how to build applications with multi-level  $\mu$ TESLA for base stations and sensors, respectively. Figure 8 shows a base station as well as a sensor application. For brevity, the arguments of function calls are omitted. As shown in the left column, a typical base station application needs to have the following steps:

1. **Initialization:** The base station application calls `Sender.init` to initialize the Sender components.
2. **Sending a data packet:** The base station first checks data are ready or not. If the data are ready, the application calls `Sender.sendData` to fill in the authentication information and send the data out.

Base station application	Sensor application
<pre> void init() {     call Sender.init; } void process() {     if (data ready in data){         call Sender.sendData;     } } </pre>	<pre> void init() {     call Receiver.init; } event Receiver.authenticatedDataReady() {     process the authenticated data; } </pre>

Figure 8: Example applications in base stations and sensor nodes.

The right column of Figure 8 shows a typical sensor application, which has the following steps.

1. Initialization: The sensor application calls `Receiver.init` to initialize the Receiver components.
2. Processing the authenticated data Packet: Whenever a data packet is authenticated, the receiver component triggers an event to pass the authenticated data packet to the sensor application. The sensor application needs to implement an event handler function to process the received data packet.

We have built simple applications on base stations and sensors in our experiments. We report our experiences and the experimental results in the following section.

## 5 Experimental Results

We have performed a series of experiments using our implementation to evaluate the performance of the DOS-tolerant multi-level  $\mu$ TESLA when there are packet losses and DOS attacks against *CDM* messages. The communication, storage, and computation overheads are discussed in earlier sections. The focus of the evaluation in this section is on the overall effectiveness of the proposed techniques (e.g., multi-buffer random selection) in tolerating packet losses and DOS attacks, and the impact of different choices of certain parameters (e.g., buffer size, percentage of forged *CDM* packets). The experiments were performed using Nido, the TinyOS simulator. To simulate the lossy communication channel, we have each sensor drop each received packet with a given probability.

To further study the performance of the scheme in presence of attacks, we also implemented an attacker component, which listens to the *CDM* messages broadcasted by the base station and inserts forged *CDM* messages into the broadcast channel to disrupt the broadcast authentication. We assume that the attacker is intelligent in that it uses every piece of authentic information that a sensor node can determine in the forged messages. That is, it only modifies  $K_{i+2,0}$  and the MAC value in a *CDM* message, since any other modification can be detected by a sensor node immediately. There are other attacks against the scheme. Since they are either defeatable by the scheme (e.g., modification of data packets), or not specific to our extension (e.g., DOS attacks against the data packets), we did not consider them in our experiments.

To concentrate on the design decisions we made in our schemes, we fix the following parameters in all the experiments. We only performed the experiments with DOS-tolerant two-level  $\mu$ TESLA, since the only purpose of having multiple levels is to scale up to a long period of time. We assume the duration of each low-level time interval is 100 ms, and each low-level key chain consists of 600 keys. Thus, the duration of each time interval for the high-level key chain is 60 seconds. We put 200 keys in the high-level key chain,

which covers up to 200 minutes in time. We also set the data packet rate at base station to 100 data packets per minute. Our analysis and experiments indicate that the number of high-level keys does not have an obvious impact on the performance measures. Nevertheless, the lifetime of the two-level key chains can be extended by having more keys in the high-level key chain or another higher level of key chain. Since our purpose is to study the performance of the scheme w.r.t. to packet losses and DOS attacks, we did not do so in our evaluation.

The performance of our techniques depends on the probability of having an authentic key chain commitment, which is mainly affected by the number of *CDM* buffers in sensors and the percentage of forged *CDM* packets in the communication channel as we discussed before. Thus, in our experiments, we simply fix the *CDM* packet rate but use different attack rates to evaluate the performance of our system.

The performance of our system is evaluated with the following metrics: average percentage of authenticated data packets (i.e.,  $\frac{\# \text{authenticated data packets}}{\# \text{received data packets}}$  averaged over the sensor nodes) and average data packet authentication delay (i.e., the average time between the receipt and the authentication of a data packet). In these experiments, we focused on the impact of the following parameters on these performance metrics: sensor node's buffer size for data and *CDM* messages, percentage of forged *CDM* packets and the packet loss rate.

Because of the extremely limited memory available on sensor nodes, the buffer allocation for data packets and *CDM* messages becomes a major concern when we deploy a real sensor network. We evaluate the performance of different memory allocation schemes with a memory constraint. The format of data packet in our proposed technique is the same as in the original  $\mu$ TESLA except for a level number, which only occupies one byte. In our implementation, both *CDM* and data packets consist of 29 bytes. The data packet includes a level number (1 bytes), an index (4 bytes), data (8 bytes), MAC (8 bytes) and a disclosed key (8 bytes). A *CDM* packet includes a level number (1 byte), an index (4 bytes), a key chain commitment  $K_{i+2,0}$  (8 bytes), a MAC (8 bytes), and a disclosed key (8 bytes).

It is true that our schemes (and  $\mu$ TESLA) have relatively high overhead in data packets with the above settings. This is in some sense because of the small packet size. However, broadcast authentication is usually used to broadcast commands or control data from the base station to sensors. We expect typical commands or control data can fit in the 8 bytes payload. The base station also has the option to split long commands or data into multiple packets. Moreover, it is possible to modify the maximum packet size in TinyOS to decrease the overhead. In our experiment, we only consider the default maximum packet size supported by TinyOS, because the effect of *CDM* packets is our main concern.

When a sensor node receives a data packet, it does not need to buffer the level number and the disclosed key for future authentication; only the other 20 bytes need to be stored. For *CDM* packets, all copies of the same *CDM* message have the same values for the fields other than the key chain commitment and the MAC value (i.e.,  $K_{i+2,0}$  and MAC in  $CDM_i$ ), since all forged messages without these values can be filtered out by the weak authentication mechanism. As a result, for all copies of  $CDM_i$ , the only fields that need saving are  $K_{i+2,0}$  (8 bytes) and MAC (8 bytes), assuming that the level number and the index are used to locate the buffer and the disclosed key  $K_{i-1}$  is stored elsewhere to authenticate later disclosed keys. Further assume the totally available memory for data and *CDM* messages is  $C$  bytes, and the sensor node decides to store up to  $x$  data packets. Then the sensor can save up to  $y = \lfloor \frac{C-20 \times x}{16} \rfloor$  copies of *CDM* messages.

Figure 9 shows the performance of different memory allocation schemes under severe DOS attacks against *CDM* messages (95% forged *CDM* packets). In these experiments, we have total memory of 512 bytes or 1K bytes. As shown in Figure 9, three data buffers (60 bytes) are enough to authenticate over 90% of the received data packets when the total memory is 1K bytes. This is because the data packet arrived in later time interval carries the key that can be used to authenticate the data packets arrived in earlier time intervals. If there are no DOS attacks on data packets (such attacks are not considered in our experiments),

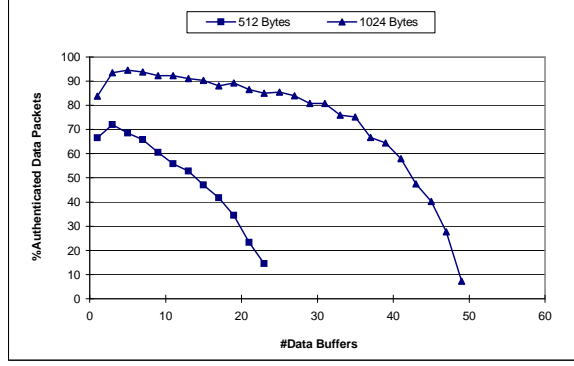


Figure 9: The performance with different buffer allocation schemes for total memory 512 and 1024 bytes to buffer data and *CDM* messages. Assume 95% of *CDM* packets are forged and 50% of packets are lost when transmitted over the channel.

the sensor can authenticate those data packets that arrived no less than  $d$  time intervals earlier and remove them from the buffer. Thus, the buffer size for data packets depends on the data rate, the key disclosure lag  $d$  and the duration of the lowest key chain time interval. In practice, it only need to be large enough to hold all data packets within one lowest-level time interval.

The figure also shows that after a certain point, having more data buffers does not increase the performance. Instead, it decreases the performance, since less memory is left for buffering the *CDM* messages.

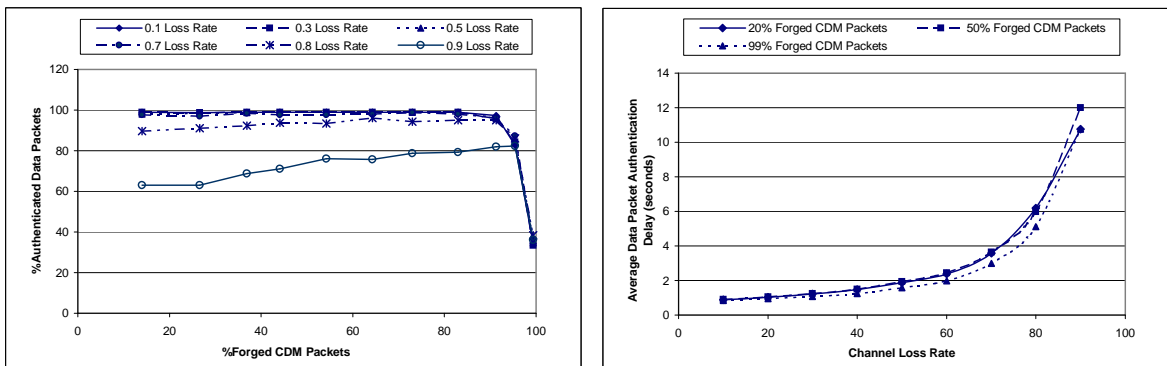
To measure the performance under intensive DOS attacks, we assume that each sensor node can store up to 3 data packets and 39 *CDM* packets, which totally occupy 684 bytes memory space. The experimental results are shown in Figures 10(a) and 10(b). Figure 10(a) shows that our system can tolerate DOS attacks to a certain degree; however, when there are extremely severe DOS attacks (over 95% of forged *CDM* packets), the performance decreases dramatically. This result is reasonable; a sensor node is certainly not able to get an authentic *CDM* message if all of the *CDM* messages it receives are forged. Nevertheless, an attacker has to make sure he/she sends much more forged *CDM* packets than the authentic ones to increase his/her chance of success.

Figure 10(a) also shows that if the base station rebroadcasts sufficient number of *CDM* messages so that on average, at least one copy of such authentic *CDM* message can reach a sensor node in the corresponding high-level time interval (e.g., when loss rate  $\leq 70\%$ ), the channel loss rate does not affect our scheme much. When the loss rate is large (e.g., 90% as in Figure 10(a)), we can observe the drop of data packet authentication rate when the percentage of forged *CDM* packets is low.

An interesting result is that when the channel loss rate is 90%, the data packet authentication rate initially increase when the percentage of forged *CDM* packets increases. This is because the sensor nodes can get the disclosed key from forged *CDM* packets when they cannot get it from the authentic ones.

The channel loss rate does affect the average authentication delay, which can be seen in Figure 10(b). The reason is that a sensor node needs to wait a longer time to get the disclosed key. Though the number of dropped packets increases dramatically under sever DOS attack (over 95%) as seen in Figure 10(a), Figure 10(b) shows that the percentage of forged *CDM* message does not have a significant impact on the average data packet authentication delay for those packets that have been authenticated.

In summary, the experimental results demonstrate that our system can maintain reasonable performance even with high channel loss rate under severe DOS attacks.



(a) Percentage of authenticated data packets

(b) Average data packet authentication delay

Figure 10: Experimental results under different channel loss rate and percentage of forged *CDM* packets. Assuming 3 data packet buffers, 39 *CDM* buffers and fixed data rate (100 *data packets/minute*).

## 6 Related Work

Security in sensor networks has attracted intensive research efforts recently (e.g., [6, 27, 31]). Due to the limited resources at sensor nodes, solutions based on asymmetric cryptography [13, 30, 33] are usually impractical for sensor networks. In the following, we restrict our discussion to related techniques based on symmetric cryptography.

One way hash functions play an important role in our schemes. The use of one way hash functions for authentication can be traced back to Lamport [19], which was later implemented as the S/Key one-time password system [15]. Cheung proposed OLSV that uses delayed disclosures of keys by the sender to authenticate link-state routing updates between routers [8]. Anderson et al. used the same technique in their Guy Fawkes protocol to authenticate messages between two parties [1]. Briscoe proposed the FLAMeS protocol [4], and Bergadano et al. presented an authentication protocol for multicast [3]. Both are similar to the OLSV protocol [8]. Canetti et al. proposed to use  $k$  different keys to authenticate the multicast messages with  $k$  different *MAC*'s for sender authentication [5]. But, their scheme has high communication overhead because of the  $k$  *MAC*'s for each message. Perrig introduced a verification efficient signature scheme named BiBa based on one-way hash functions without trapdoors [23]; however, BiBa has high overhead in signature generation and public key distribution. These techniques either do not address, or cannot be applied to broadcast authentication in sensor networks.

Our techniques in this paper are closely related to TESLA, a well-known broadcast authentication protocol, which has been described in Section 2. TESLA was originally proposed in [25] to efficiently authenticate multicast streams over lossy channels. Though TESLA requires loose time synchronization between a sender and multiple receivers, it is extremely efficient it mainly uses symmetric cryptography for authentication. TESLA was later extended to provide additional capabilities such as immediate authentication (to remove the delay between the receipt and the authentication of a data packet) and concurrent TESLA instances (to accommodate heterogeneous networks with different bandwidths) [26]. TESLA requires a digital signature operation to bootstrap itself, and thus is impractical in resource constrained sensor networks. As an adaption of TESLA,  $\mu$ TESLA uses symmetric cryptography to distribute initial parameters to the sensor nodes individually [27]. As discussed earlier, the drawback of this solution is the high communication overhead required for initializing sensors when the number of sensor nodes is large. The work in this paper is to

address this problem.

Perrig et al. proposed to use an earlier key chain to distribute the commitments of the next key chain [24]. Multiple early TESLA packets are used to tolerate packet losses. However, since reliable distribution of later commitment cannot be fully guaranteed, if all the packets used to distribute commitments are lost (e.g., due to temporary network partition), a receiver will not be able to recover the commitment of the later key chain. As a result, the sender and the receivers will have to repeat the costly bootstrap process. In contrast, because of the connection between two consecutive levels of key chains, our techniques allow a receiver to recover the key chains even if all the commitment distribution messages during one high-level time interval are lost, .

Besides broadcast authentication, key management is also a fundamental security service in sensor networks. (In some sense, our techniques can also be considered key management techniques for broadcast authentication.) Based on the assumption of tamper-resistant hardware, Basagni et al. presented a key management scheme to periodically update the symmetric keys shared by all sensor nodes [2]. With this key shared among all sensor nodes, authenticated broadcast can be easily implemented. However, this scheme cannot prevent a (compromised) sensor from sending forged messages if an attacker can reuse the tamper-resistant hardware.

Due to the resource constraints in sensors, several new key management techniques have been proposed recently. A probabilistic key predistribution technique was first proposed in [11]. The basic idea is to let each sensor randomly pick a set of keys from a key pool so that two sensors can have a certain probability of sharing a common key. Chan et al. improved this idea to a  $q$ -composite key predistribution scheme, which requires at least  $q$  shared common keys in order to set up a pairwise key [7]. Moreover, Chan et al. also investigated a random pairwise keys scheme, which predistributes a unique random pairwise key between a random pair of sensors [7]. Liu and Ning developed a framework to predistribute pairwise keys using bivariate polynomials, and two efficient instantiations, a random subset assignment scheme and a grid-based key predistribution scheme, to establish pairwise keys in sensor networks [21]. Instead of using a bivariate polynomial, Du et al. proposed another approach based on Blom's key predistribution scheme [10]. Liu and Ning, Du et al. later independently developed techniques to use sensors' expected locations to improve the performance of pairwise key predistribution [9, 22]. Zhu et al. proposed a protocol suite named LEAP (Localized Encryption and Authentication Protocol) to help establish individual keys between sensors and a base station, pairwise keys between sensors, cluster keys within a local area, and a group key shared by all nodes [35]. These techniques address the fundamental problem of secure communication between (or among) sensors. However, they cannot provide broadcast authentication capabilities. Thus, we consider them complementary to our techniques in this paper.

Wood and Stankovic identified a number of DOS attacks in sensor networks [34]. Karlof and Wagner analyzed the vulnerabilities and the countermeasures for a number of routing protocols for sensor networks [17]. The broadcast authentication techniques proposed in this paper may help address some attacks identified in these papers.

## 7 Conclusion and Future Work

In this paper, we developed a multi-level key chain scheme to efficiently distribute the key chain commitments for the broadcast authentication scheme named  $\mu$ TESLA. By using pre-determination and broadcast, our approach removed  $\mu$ TESLA's requirement of a unicast-based distribution of initial key chain commitments, which introduces high communication overhead in large distributed sensor networks. We also proposed several techniques, including periodic broadcast of commitment distribution messages and random selection strategies, to improve the survivability of our scheme and defeat some DOS attacks. The resulting

protocol, named multi-level  $\mu$ TESLA, satisfies several nice properties, including low overhead, tolerance of message loss, scalability to large networks, and resistance to replay attacks.

We also developed a broadcast authentication system on TinyOS, using the techniques proposed in this paper. Our system provides a simple and easy-to-use API, which can be used to build base station and sensor applications on TinyOS. This implementation shows that our technique is practical for resource constrained sensors. Our experiments further demonstrated that the broadcast authentication system can tolerate message loss and DOS attacks to a certain degree.

There are several possible future directions that worth while studying. First, the authentication and the failure recovery delay are still not fully solved in our techniques. For example, when a sensor node does not get a key chain commitment during a time interval, it must wait for a relatively long period of time to recover from this failure. We will seek solutions to this problem in our future research. Second, the assumption of loose time synchronization in sensor network may not be true in some applications. There are many mechanisms to disrupt the time synchronization method. Thus, it may be desirable to have alternative ways of authenticating broadcast messages without assumption of time synchronization. Third, in this paper, we assume a single base station in the whole network; and this base station is assumed to be secure. However, in some scenarios, there may exist multiple base stations and one or some of those base stations may be compromised by the attacker. We would like to study broadcast authentication which may involve multiple base stations and can tolerate compromised base stations.

## Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments.

## References

- [1] R. Anderson, F. Bergadano, B. Crispo, J. Lee, C. Manifavas, and R. Needham. A new family of authentication protocols. In *Operating Systems Review*, October 1998.
- [2] S. Basagni, K. Herrin, D. Bruschi, and E. Rosti. Secure pebblenets. In *Proceedings of ACM International Symposium on Mobile ad hoc networking and computing*, pages 156–163, 2001.
- [3] F. Bergadano, D. Cavagnino, and B. Crispo. Individual single source authentication on the mbone. In *IEEE International Conference on Multimedia & Expo (ICME)*, August 2000.
- [4] B. Briscoe. FLAMeS: Fast, loss-tolerant authentication of multicast stream. Technical report, BT Research, 2000.
- [5] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and some efficient constructions. In *Proceedings of IEEE INFOCOMM*, pages 708–716, 1999.
- [6] D.W. Carman, P.S. Kruus, and B.J.Matt. Constrains and approaches for distributed sensor network security. Technical report, NAI Labs, 2000.
- [7] H. Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. In *IEEE Symposium on Research in Security and Privacy*, 2003.
- [8] S. Cheung. An efficient message authentication scheme for link state routing. In *13th Annual Computer Security Applications conference*, San Diego, Calif, December 1997.

- [9] W. Du, J. Deng, Y. S. Han, S. Chen, and P. Varshney. A key management scheme for wireless sensor networks using deployment knowledge. Submitted for publications.
- [10] W. Du, J. Deng, Y. S. Han, and P. Varshney. A pairwise key pre-distribution scheme for wireless sensor networks. In *10th ACM Conference on Computer and Communications Security*, October 2003.
- [11] L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 41–47, November 2002.
- [12] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proceedings of Programming Language Design and Implementation (PLDI 2003)*, June 2003.
- [13] R. Gennaro and P. Rohatgi. How to sign digital streams. Technical report, IBM T.J.Watson Research Center, 1997.
- [14] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.
- [15] N. M. Haller. The S/KEY one-time password system. In *Proceedings of the ISOC Symposium on Network and Distributed System Security*, pages 151–157, 1994.
- [16] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D.E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.
- [17] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. In *First IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003.
- [18] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. IETF RFC 2104, February 1997.
- [19] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.
- [20] D. Liu and P. Ning. Efficient distribution of key chain commitments for broadcast authentication in distributed sensor networks. In *Proceedings of the 10th Annual Network and Distributed System Security Symposium*, pages 263–276, February 2003.
- [21] D. Liu and P. Ning. Establishing pairwise keys in distributed sensor networks. In *10th ACM Conference on Computer and Communications Security*, October 2003.
- [22] D. Liu and P. Ning. Location-based pairwise key establishments for static sensor networks. In *2003 ACM Workshop on Security in Ad Hoc and Sensor Networks*, October 2003.
- [23] A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 28–37, November 2001.
- [24] A. Perrig, R. Canetti, Briscoe, J. Tygar, and D. Song. TESLA: Multicast source authentication transform. IRTF draft, draft-irtf-smug-tesla-00.txt, November 2000.
- [25] A. Perrig, R. Canetti, D. Song, and D. Tygar. Efficient authentication and signing of multicast streams over lossy channels. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, May 2000.

- [26] A. Perrig, R. Canetti, D. Song, and D. Tygar. Efficient and secure source authentication for multicast. In *Proceedings of Network and Distributed System Security Symposium*, February 2001.
- [27] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J.D. Tygar. SPINS: Security protocols for sensor networks. In *Proceedings of Seventh Annual International Conference on Mobile Computing and Networks*, July 2001.
- [28] R. Rivest. The RC5 encryption algorithm. In *Proceedings of the 1st International Workshop on Fast Software Encryption*, volume 809, pages 86–96, 1994.
- [29] R.L. Rivest, A. Shamir, and L.A. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [30] P. Rohatgi. A compact and fast hybrid signature scheme for multicat packet authentication. In *6th ACM Conference on Computer and Communications Security*, November 1999.
- [31] F. Stajano and R. Anderson. The resurrecting duckling: security issues for ad hoc networks. In *Proc. of Security Protocols: 7th International Workshop*, pages 172–194, 1999.
- [32] U.S. National Institute of Standards and Technology. DES modes of operation. Federal Information Processing Standards Publication 81 (FIPS PUB 4-3), December 1980.
- [33] C.K. Wong and S. S. Lam. Digital signatures for flows and multicasts. In *Proc. IEEE ICNP'98*, 1998.
- [34] A. D. Wood and J. A. Stankovic. Denial of service in sensor networks. *IEEE Computer*, 35(10):54–62, October 2002.
- [35] S. Zhu, S. Setia, and S. Jajodia. Leap: Efficient security mechanisms for large-scale distributed sensor networks. In *10th ACM Conference on Computer and Communications Security*, October 2003.

## A A Detailed Description of Scheme IV

### Initialization

During the initialization phase, all the sensor nodes synchronize their clocks with the base station. (Alternatively, the base station and all the sensor nodes may synchronize their clocks with a time service.) In addition, the base station generates the following parameters: (1) the initial random key  $K_{n_0}$  for the high-level key chain; (2) a sequence of keys  $K_i = F_0(K_{i+1})$  in the high-level key chain, where  $i = 0, 1, \dots, n_0 - 1$ , and  $F_0$  is a pseudo random function; (3) the duration  $\Delta_0$  of each time interval for the high-level key chain; (4) the starting time  $T_1$  for the high-level key chain; (5) duration  $\Delta_1$  of the low-level time intervals; (6) the disclosure lag  $d$  for the low-level key chains; (7) the maximum clock discrepancy  $\delta_{Max}$  during the lifetime of the sensor network; (8) the frequency of *CDM* packets.

A constraint for these parameters is that  $\Delta_1 \times d + \delta_{Max} <$  the duration of the time interval for the high-level key chain. Otherwise, the disclosure of a high-level key may disclose a low-level key that should not be disclosed.

The base station distributes the following parameters to the sensor nodes: (1)  $K_0$ , (2)  $\Delta_0$ , (3)  $T_1$ , (4)  $\Delta_1$ , (5)  $d$ , and (6)  $\delta_{Max}$ . Here we predetermine all the parameters for the low-level key chains except for the commitments. Alternatively, we may allow the base station to dynamically choose these parameters and distribute them to the sensors in the commitment distribution messages. In this case, the authentication procedure below should be changed slightly. In addition, if the base station wants to enable the sensors to

authentication broadcast messages during the high-level time intervals  $I_1$  and  $I_2$ , the base station needs to distribute  $K_{1,0}$  and  $K_{2,0}$  to the sensors.

Note that the initialization phase does not introduce significantly more overhead than the original  $\mu$ TESLA. In the original  $\mu$ TESLA, it is at least necessary to distribute the master keys to the sensor nodes so that the base station shares some common keying material with each sensor node. The aforementioned parameters can be distributed to the sensor nodes along with the master keys.

### Broadcast of Commitment Distribution Messages

When the base station needs to broadcast authenticated messages to the sensors, it generates parameters for each low-level key chain in a similar way to TESLA and  $\mu$ TESLA [25–27]. Assume the base station decides to divide each time interval  $I_i$  into  $n_1$  smaller intervals, denoted  $I_{i,1}, I_{i,2}, \dots, I_{i,n_1}$ . The base station generates the low-level key chain by computing  $K_{i,n_1} = F_{01}(K_{i+1})$ , and  $K_{i,j} = F_1(K_{i,j+1})$ , where  $j = 0, 1, \dots, n_1 - 1$  and  $F_1$  is a pseudo random function. Thus, the base station has the low-level key chain  $\langle K_{i,0} \rangle$ . The base station distributes the relevant information about the low-level key chain  $\langle K_{i,0} \rangle$  in  $CDM_{i-2}$  during the time interval  $I_{i-2}$ .

Each commitment distribution message  $CDM_i$  contains the index of the high-level time interval, the commitment of the low-level key chain  $\langle K_{i+2,0} \rangle$ , the MAC generated over the above fields with the key  $K'_i$ , which is derived from the high-level key  $K_i$ , and the disclosed high-level authentication key  $K_{i-1}$ .

$$Base\ Station \rightarrow Sensors : CDM_i = i | K_{i+2,0} | MAC_{K'_i}(i | K_{i+2,0}) | K_{i-1}.$$

The base station randomly chooses  $F \times \Delta_0$  points during each time interval  $I_i$ , and broadcasts  $CDM_i$  at these time points.

### Authentication of Commitment Distribution Messages

Assume that a sensor node  $S$  has  $m + 1$  buffers for commitment distribution messages. When  $S$  receives a copy of  $CDM_i$  at time  $t_i$  during the time interval  $I_i$ , it processes this message according to the following procedure.

1.  $S$  checks the security condition for  $CDM_i$ , i.e.,  $t_i + \delta_{Max} < T_{i+1}$ .  $S$  discards the packet and stops if the security condition is not satisfied.
2.  $S$  authenticates  $K_{i-1}$  against a previously disclosed key  $K_j$  by verifying that  $K_{i-1} = F^{i-1-j}(K_j)$ . (Note that  $K_j$  always exists since  $K_0$  was distributed to each sensor node during initialization.) If this verification fails,  $S$  discards the message and stops. Otherwise,  $S$  replaces  $K_j$  with  $K_{i-1}$ .
3. For each copy  $c$  of  $CDM_{i-1}$ ,  $S$  authenticates  $c$  by verifying its MAC with  $K_{i-1}$  disclosed in  $CDM_i$ . If this verification fails,  $S$  discards  $c$  and continues the verification for the next copy of  $CDM_{i-1}$ . Otherwise,  $S$  discards all the other copies of  $CDM_{i-1}$  and makes  $c$  the authenticated copy of  $CDM_{i-1}$ . The key chain commitment  $K_{i+1,0}$  contained in this copy of  $CDM_{i-1}$  is then selected as the commitment of the low-level key chain  $\langle K_{i+1,0} \rangle$  for the next high-level time interval  $I_{i+1}$ .
4.  $S$  uses the random selection strategy discussed in 3.4 to decide whether to save the current copy of  $CDM_i$  or not. (Note that if the current step is being executed, all the copies of  $CDM_{i-1}$  should have been discarded.) Further assume the current copy of  $CDM_i$  is the  $j$ th copy. If  $j < m$ ,  $S$  still has free buffers available, and  $S$  saves it in one of the empty buffers. Otherwise,  $S$  keeps this copy with the probability  $m/j$ , and places it in a randomly selected buffer (among the  $m$  occupied buffers).

### Broadcast and Authentication of Normal Messages

Broadcast and authentication of normal messages are performed in the same way as in the extended TESLA [26], except for the distribution of the key chain commitments, which is handled in the distribution and authentication of commitment distribution messages.

## B The Multi-Level $\mu$ TESLA API on TinyOS

### B.1 Data Structure

- struct `Data_Packet_t`. It defines the format of a multi-level  $\mu$ TESLA data packet.

name	type	description
level	char	The key chain level. 1 byte. For data packet, it is always $(MAX\_LEVEL-1)$ , where $MAX\_LEVEL$ is the total number of levels, which is predefined in the application.
index	long	The index of the time interval for the data packet. 4 bytes.
data	array of char	Actual payload. The array size is define by $DATA\_SIZE$ , which is predefined in the application. Because of the limitation of active message size in TinyOS, it is set to 8 currently. If longer message size is supported in the future, this value can be easily changed.
mac	array of char	Message Authentication Code computed for this packet. The array size is 8.
dis	array of char	Disclosed key. The array size is 8.

- struct `CDM_Packet_t`. It defines the format of a multi-level  $\mu$ TESLA *CDM* packet.

name	type	description
level	char	The key chain level this <i>CDM</i> packet belongs to. 1 byte.
index	long	The index of time interval. 4 bytes
kc_2_0	array of char	The commitment for chain $(index+2)$ of its next lower-level key chain. The array size is 8
mac	array of char	Message Authentication Code computed for this packet. The array size is 8
dis	array of char	Disclosed key for the previous time interval. The array size is 8

- struct `Sender_Config_t`. It defines the configuration information for a sender.

name	type	description
MT_key	array of char	The master multi-level $\mu$ TESLA key, which is used to generate all the other multi-level $\mu$ TESLA keys. 8 bytes
kc_len	array of long	The length of key chain for each level. The array size is MAX_LEVEL, which is the total number of levels.
kc_int	array of long	The duration of the time interval for each level. The array size is MAX_LEVEL, which is the total number of levels. Actually, we only need the duration of time interval of the lowest level key chain, and all the others can be computed from this value and the length of the key chain in each level.
start_time	long long	Start time of multi-level $\mu$ TESLA. 8 bytes
delay	long	Key disclosure delay for data packet. 4 bytes
freq	long	Frequency of <i>CDM</i> packets. 4 bytes

- struct KCC\_t. It defines buffer to store a single key chain commitment.

name	type	description
index	long	The index of this key chain commitment. 4 bytes
key	array of char	Key chain commitment. The array size is 8

- struct LC\_t. It defines the buffer to store the key chain commitments for a particular level.

name	type	description
chain	long	The chain saved in this buffer. 4 bytes
commit	array of KCC_t	The buffer that stores key commitments. The array size is 3. This is used to store the key commitment for the previous, the current, and the next key chain.

- struct Receiver\_Config\_t. It defines the configuration information for a receiver.

name	type	description
kc_len	array of long	The length of key chain for each level. It is the same as the kc_len field in Sender_Config_t struct.
kc_int	array of long	The duration of time interval for each level. It is the same as the kc_int field in Sender_Config_t struct.
start_time	long long	Start time of multi-level $\mu$ TESLA. 8 bytes
delta	long	the maximum clock discrepancy between sender and receiver. 4 bytes.
delay	long	Key disclosure delay for data packet. 4 bytes
lc	array of LC_t	The buffer to store all the key chain commitments. The array size is MAX_LEVEL, which is the total number of levels

## B.2 API Description

### Sender component

- Sender.init(config)
  - caller: The application
  - return value: None
  - Parameters

name	type	description
config	Sender_Config_t*	configuration of the sender.

- Sender.sendData(time,packet)
  - caller: The application
  - return value:
    1. SUCCESS: Successfully send out an authenticated data packet.
    2. FAIL: Fail to send out the data packet. This happens when all the keys for data packets are used up or communication module is busy.

-Parameters

name	type	description
time	long long	The time when this data packet is generated. 8 bytes
packet	Data_Packet_t*	The buffer that stores the data packet

- Sender.generateConf(config)
  - caller: The application
  - return value: None
  - Parameters

name	type	description
config	Receiver_Config_t*	The buffer to store the generated configuration for the receivers

## Receiver component

- Receiver.init(config)
  - caller: The application
  - return value: None
  - Parameters

name	type	description
config	Receiver_Config_t*	configuration of the Receiver.

- Receiver.authenticDataReady(packet, delay)
  - caller: The Receiver component
  - return value: Depends on how the application implement it
  - parameters

name	type	description
packet	Data_Packet_t*	The buffer that stores the authenticated data packet
delay	long	The delay between the reception and the authentication of this data packet. 4 bytes