Mitigating DoS Attacks against Signature-Based Broadcast Authentication in Wireless Sensor Networks

Peng Ning An Liu
North Carolina State University

Wenliang Du
Syracuse University

Abstract

Broadcast authentication is a critical security service in wireless sensor networks. There are a number of benefits to provide broadcast authentication with digital signatures, such as immediate authentication capability and the ease of managing cryptographic keys, compared with the alternative of TESLA-based approaches. Though previously considered infeasible, recent results have demonstrated that it is possible to perform public key cryptography on resource constrained sensor nodes efficiently. However, using digital signatures for broadcast authentication still faces a great challenge of denial of service (DoS) attacks: *An attacker can inject bogus broadcast packets to force sensor nodes to perform expensive signature verifications, and thus exhaust their limited battery power.* This paper presents an efficient mechanism called *message specific puzzle* to mitigate such DoS attacks. In addition to a digital signature, this approach adds a weak authenticator in each broadcast packet, which can be efficiently verified by a regular sensor node, but takes a computationally powerful attacker a substantial amount of time to forge. Upon receiving a broadcast packet, each sensor node first verifies the weak authenticator, and performs the expensive signature verification operation only when the weak authenticator is valid. A weak authenticator cannot be pre-computed without a non-reusable key disclosed only in a valid broadcast packet. As a result, an attacker cannot start the expensive computation to forge a weak authenticator without seeing a valid broadcast packet. Even if an attacker has sufficient computational resources to forge one or more weak authenticators, it is difficult to reuse these forged weak authenticators. Thus, this weak authentication mechanism substantially increases the difficulty of launching successful DoS attacks against signature verifications. This paper also reports an implementation (called *TinySigGuard*) of the proposed techniques on TinyOS, as well as the experimental evaluation in a network of MICAz motes.

1 Introduction

Recent technological advances have made it possible to deploy large scale sensor networks consisting of a large number of low-cost, low power, and multi-functional sensor nodes that communicate in short distances through wireless links [2]. These sensor nodes are typically battery-powered, and are expected to run in an unattended fashion for a long period of time. Such sensor networks have a wide range of applications in civilian and military operations such as monitoring of critical infrastructure and battlefield surveillance. Many attempts have been made to develop protocols that can fulfill the requirements of these applications (e.g., [2, 9, 16, 29, 31, 37]).

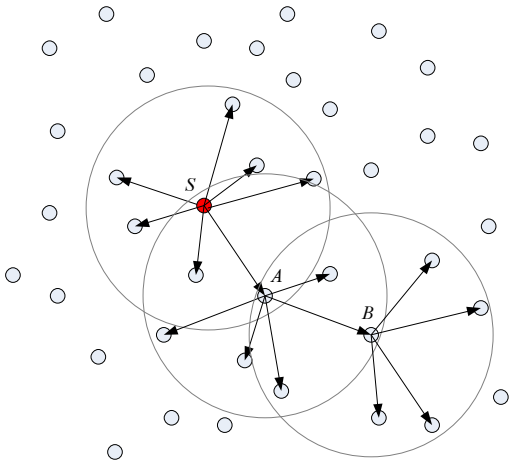

Figure 1: Broadcast in wireless sensor networks. A broadcast packet is usually re-broadcast multiple times before all reachable nodes receive it.

Broadcast is an important communication primitive in wireless sensor networks. It is highly desirable to broadcast commands (e.g., queries used to collect sensor data) and data (e.g., global clock value distributed for time synchronization) to the sensor nodes due to the large number of sensor nodes and the broadcast nature of wireless communication. Due to the limited signal range, it is usually necessary to have some receivers of a broadcast packet re-broadcast it in order to propagate the packet throughout the network (e.g., through flooding, or probabilistic broadcasting [24, 30, 41]). As illustrated in Figure 1, node *S* first broadcasts a packet (locally within the signal range), and *some* nodes that receive this packet for the first time (e.g., node *A*) re-broadcast it (locally) to propagate this packet to more nodes (e.g., node *B*). This process continues until all the reachable nodes receive the broadcast packet.

For clarify, we refer to the node that originally generates the broadcast packet as the *sender*, and a node that receives a broadcast packet as a *receiver*. As discussed earlier, a receiver may re-broadcast the received packet.

1.1 Broadcast Authentication in Sensor Networks

In hostile environments, broadcast authentication (i.e., authentication of broadcast packets) is a critical security service to ensure the trustworthiness of sensor network applications. Due to the resource constraints on sensor nodes, especially the limited battery power, broadcast authentication in wireless sensor networks is by no means a trivial problem.

Several approaches [25–27, 37] have been developed in the past several years for scalable broadcast authentication in wireless sensor networks. All of these approaches are based on TESLA [35, 36], which provides broadcast authentication based on symmetric cryptography by delayed disclosure of authentication keys. The main motivation of these approaches is to provide broadcast authentication without using public key cryptography, which was expected to be infeasible in resource constrained sensor networks.

However, the TESLA-based approaches have a fundamental limitation: A broadcast packet cannot be authenticated immediately after it is received, due to the delayed disclosure of authentication keys. Perrig et al. provided an immediate authentication mechanism by including the hash image of the *packet content* in a previous broadcast *packet* [36]. However, such immediate authentication does not cover, for example, the hash image for later packet content. As a result, an attacker can forge a large number of packets by modifying the uncovered part without being immediately detected, force sensor nodes to propagate these packets across the network, and eventually exhaust their limited battery power. Appendix A gives additional details on TESLA-based approaches and their limitation regarding denial of service (DoS) attacks.

Broadcast authentication has been traditionally provided with digital signatures in wired networks. Though digital signatures are substantially more expensive than TESLA-based approaches, they have a number of benefits, such as the immediate authentication capability and the ease of managing cryptographic keys. Recently, Gura et al. demonstrated that it is feasible to perform public key cryptographic operations on low end sensor nodes [13]. For example, consider the 160-bit elliptic curve `secp160r1` [6] recommended by the Standards for Efficient Cryptography Group (SECG), which provides cryptography strength similar to 1024-bit RSA or DSA. It takes 0.81 seconds to perform a point multiplication on this elliptic curve on Atmega128, the processor used in many sensor nodes such as MICA2 and MICAz motes [13]. This implies that it would take about 1.62 seconds to verify an ECDSA signature on the same elliptic curve, since the dominant operations in signature verification are two point multiplications. Meanwhile, the recent advances in sensor wireless communication allow relatively large packets to be transmitted. In particular, IEEE 802.15.4, the standard for low power sensor networks, allows a maximum packet size of 127 bytes, including a variable payload of up to 102 bytes [20]. Such a packet provides enough space to include a digital signature for broadcast authentication, such as a 40-byte ECDSA signature on the above 160-bit elliptic curve. Thus, it is highly desirable to consider digital signatures for broadcast authentication in wireless sensor networks.

1.2 DoS Attacks against Signature-Based Broadcast Authentication

Although it is possible to perform digital signature operations on sensor nodes, the cost of such operations is still substantially higher than that of symmetric cryptographic operations, and will result in the depletion of battery power if frequently performed. This leads to a fatal threat to signature-based broadcast authentication: *An attacker may simply forge a large number of broadcast messages with digital signatures, force sensor nodes to verify these signatures, and eventually deplete their battery power.* Benign sensor nodes may certainly decide not to forward broadcast messages before their signatures are verified. However, a single malicious node can still overload and disable many benign nodes in its local region with forged messages. Moreover, an attacker may generate much higher impact by increasing the signal strength or deploying multiple malicious nodes.

1.3 Proposed Approach

In this paper, we develop an approach to mitigate the DoS attacks against signature-based broadcast authentication. The basic idea is to use an efficiently verifiable weak authenticator along with a digital signature, so that a sensor node performs the expensive signature verification only when the weak authenticator can be verified. We develop a weak authentication mechanism called *message specific puzzle* to achieve this goal. This mechanism has a number

of nice properties:

- A weak authenticator can be efficiently verified by a regular sensor node; however, it takes a computationally powerful attacker a substantial amount of time to forge a valid weak authenticator.
- A weak authenticator cannot be pre-computed without a non-reusable key disclosed only in a valid broadcast packet. Thus, an attacker cannot start the expensive computation to forge a weak authenticator without seeing a valid broadcast packet.
- Even if an attacker has sufficient computational resources to forge one or more weak authenticators, it is difficult to reuse these forged weak authenticators. Thus, this weak authentication mechanism substantially increases the difficulty of launching successful DoS attacks against signature verifications.

These desirable properties come with a cost. First, the proposed message specific puzzles require a computationally powerful sender with sufficient power supply. Second, the generation of weak authenticators introduces a delay at the sender. However, considering the benefits brought by message specific puzzles, we believe the proposed techniques are useful and practical in wireless sensor networks.

To facilitate the evaluation of message specific puzzles, we have implemented on TinyOS [16] a software package called *TinyECC* for Elliptic Curve Cryptography (ECC), which currently provides support for ECDSA signature generation and verification, and a software package called *TinySigGuard*, which uses the ECDSA signature support in TinyECC and provides the proposed weak authentication for signature-based broadcast authentication. We have performed initial experimental evaluation of TinySigGuard in a network of MICAz motes. The experimental results indicate that the proposed techniques are promising for secure sensor network applications.

1.4 Organization

The remainder of this paper is organized as follows. The next section describes the assumptions of the proposed techniques. Section 3 presents the proposed techniques to mitigate DoS attacks against signature-based broadcast authentication in wireless sensor networks. Section 4 describes the implementation and evaluation of the proposed techniques on TinyOS, an operating system for networked sensors [16]. Section 5 discusses related work. Section 6 concludes this paper and points out some future research directions. The appendices give more information related to the proposed techniques.

2 Assumptions

Assumptions of Sensor Networks. We assume the (original) senders of authenticated broadcast messages are computationally powerful nodes (e.g., laptops), which also have sufficient power supply (e.g., charged in a vehicle). There are certainly scenarios where the senders of broadcast messages are regular, resource-constrained sensor nodes. Our techniques proposed in this paper do not apply to such cases. Nevertheless, we will investigate techniques for these scenarios in our future research.

We assume regular sensor nodes can perform a limited number of public key cryptographic operations, and can finish each operation in a reasonable amount of time. As discussed in the Introduction, this assumption has been validated in [13]. For example, based on the results in [13], a MICA2 mote can finish a 1024-bit RSA signature verification in about 0.43 seconds, and a 160-bit ECDSA signature verification in about 1.62 seconds. However, such public key cryptographic operations still consume substantially more resources (e.g., battery power) than symmetric cryptographic operations, and can be exploited by attackers to launch DoS attacks.

We also assume that a packet transmitted in a sensor network is large enough to accommodate a public key signature. As discussed earlier, using IEEE 802.15.4, ZigBee-compliant sensor nodes (e.g., MICAz [7]) can support packet payload up to 102 bytes [20], despite the fact that the default payload size in TinyOS is only 29 bytes [16]. Such a packet can certainly include, for example, a 160-bit ECDSA signature, which requires 40 bytes. To confirm this assumption, we performed experiments with MICAz motes to measure the packet delivery rate at different distances when the packet payload size is 102 bytes. In our indoor experiments, the packet delivery rate for MICAz is over 90% when the distance is 100 feet, compared with close to 0% packet delivery rate for MICA2. Appendix B shows more details.

We do not assume any specific broadcast protocol. The broadcast protocol can be simply flooding, or probabilistic broadcast (e.g., [24, 30, 41]). However, we do assume that in order to propagate a broadcast packet to the

entire network, it is necessary for *some* receivers to re-broadcast the packet. This is certainly true for all existing broadcast protocols for wireless sensor networks due to the limited signal range.

Assumptions of Attackers. We assume the attacker can eavesdrop, inject, and modify packets transmitted in the network. We assume the attacker has access to computationally resourceful nodes such as laptops and workstations. We assume the attacker may use multiple colluding nodes in different parts of the network. Thus, the attacker will be able to create wormholes between different parts of a network. We assume the attacker may compromise some nodes and learn the cryptographic secrets on them. However, the attacker cannot compromise the broadcast sender, and cannot forge valid signatures.

Our goal in this paper is to develop lightweight techniques to mitigate the DoS attacks against signature verification launched by such attackers. In other words, we would like to enable regular sensor nodes to quickly identify most forged broadcast packets (if not all) without performing the costly signature verifications.

3 Mitigating DoS Attacks against Signature Verification

Our general approach is to use efficient cryptographic primitives to provide a weak authenticator along with a digital signature in each broadcast packet, so that a sensor node does not have to verify the digital signature if the weak authenticator cannot be verified. As discussed in the Introduction, the proposed weak authentication mechanism has some nice properties: The verification of a weak authenticator takes substantially less resources than the verification of a digital signature; however, forging a weak authenticator is time-consuming, though not infeasible. Moreover, it is computationally infeasible to forge a weak authenticator before the broadcast sender discloses some secret information. As a result, weak authenticators cannot be pre-computed. Even if an attacker has sufficient computational resources to forge one or more weak authenticators, it is difficult to reuse these forged weak authenticators. Thus, this weak authentication mechanism substantially increases the difficulty of launching successful DoS attacks against signature verifications.

We would like to emphasize that weak authenticators are not intended as a replacement of digital signatures. Instead, they are used as an additional layer of protection to filter out forged broadcast packets to reduce the resource consumption of unnecessary signature verifications.

In the following, we first present a strawman approach to illustrate the basic idea and the potential threats. We then gradually enhance this approach to obtain the final solution. For simplicity, we assume there is one broadcast sender and many receivers in the later presentation. But our techniques can certainly be used when there are multiple broadcast senders.

3.1 Weak Authentication through One-Way Key Chains: A Strawman Approach

This strawman approach uses one-way key chains to provide weak authentication. One-way key chains have been used in several scenarios to provide efficient authentication. Examples include S/Key [14], TESLA [35], and its variations [25, 26, 36].

To generate a one-way key chain, the sender first selects a random value K_n as the last key in the key chain, and then repeatedly performs a (cryptographic) hash function, which is a one-way function, to compute all the other keys. That is, $K_i = F(K_{i+1})$, where F is the hash function and $0 \leq i \leq n-1$. With the hash function F, given K_j in the key chain, it is easy to compute all the previous keys K_i ($0 \leq i \leq j$), but it is computationally infeasible to compute any of the later keys K_i ($j+1 \leq i \leq n$). Thus, with the knowledge of the initial key K_0, a receiver can authenticate any key in the key chain by merely performing hash function operations. The initial key K_0 is often called the *commitment* of the key chain. Figure 2 illustrates an example of one-way key chain.

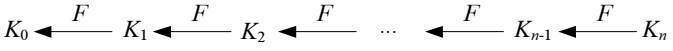

Figure 2: An example one-way key chain. K_n is randomly generated. $K_i = F(K_{i+1})$, where F is a pseudo random function and $0 \leq i \leq n-1$. K_0 is used as the commitment of the key chain.

The sender can use these keys as weak authenticators. Before transmitting the broadcast packets, the sender distributes the commitment K_0 of the key chain to all the receivers. This can be done through pre-distribution or signature-based broadcast packets. In this paper, we assume the commitment has been reliably distributed to all receivers. When the sender is ready to broadcast the i-th packet with message M_i, where $1 \leq i \leq n$, it first generates the digital signature $Sig(i|M_i)$. It then broadcasts the i-th packet, which includes the index i, the

message M_i, the signature $Sig(i|M_i)$, and the i-th weak authenticator K_i. Notation-wise, we represent the i-th broadcast packet as $i|M_i|Sig(i|M_i)|K_i$, where "|" represents concatenation.

Each receiver keeps the most recently authenticated weak authenticator K_j and the corresponding index j. Initially, $j = 0$ and $K_j = K_0$. Upon receiving a packet with index i, each receiver first verifies if the i-th broadcast packet from the sender has been previously received and authenticated (i.e., the signature has been verified successfully). If yes, this packet is potentially replayed or forged, and the receiver simply drops it. Otherwise, the receiver verifies the weak authenticator K_i by checking whether $K_j = F^{i-j}(K_i)$. It then verifies the digital signature $Sig(i|M_i)$ if the weak authentication succeeds. If both the weak authenticator and the signature are valid, the receiver replaces j with i, and K_j with K_i.

The use of one-way key chains provides some nice properties: Each weak authenticator K_i can be easily verified by regular sensor nodes. Moreover, before the broadcast of the i-th packet, an attacker does not have access to K_i, and thus cannot forge the weak authenticator (due to the one-way property of hash function F).

Weakness of the Strawman Approach. This strawman approach also has an obvious weakness, as the reader may have observed. A malicious node may exploit an observed weak authenticator and the communication delay (or network partition) to forge broadcast packets, though it cannot forge a weak authenticator directly. Specifically, once a malicious node receives a broadcast packet, it may replace the actual message and/or the signature and re-broadcast the modified packet. Moreover, it may use a fast channel (e.g., a wormhole [19]) to transmit the weak authenticator to another malicious node in a region that has not received the broadcast packet. The latter malicious node can then forge broadcast packets using this weak authenticator.

If the valid broadcast packet reaches the nodes being attacked within a reasonable amount of time, the number of signature verifications can still be bounded, since a receiver drops the packets whose index numbers belong to some previously verified packets. However, if the nodes being attacked are isolated from the sender due to network partition, the attacker can force these nodes to perform an infinite number of signature verifications, and eventually exhaust their battery power.

3.2 Message Specific Puzzles Based on One-Way Key Chains

In this subsection, we develop an initial version of message specific puzzles based on the strawman approach, assuming there is no network partition. We then address the network partition problem in the next subsection.

Our idea is to use cryptographic puzzles to reduce the possibility that an attacker may exploit an observed weak authenticator to forge broadcast packets. Intuitively, a sender (or an attacker) has to solve a cryptographic puzzle [21] in order to generate a valid weak authenticator. The puzzle solution is then used as the weak authenticator. Though it is a bit time-consuming to solve a cryptographic puzzle, it is very efficient to verify a puzzle solution. Thus, while a receiver can efficiently verify a weak authenticator, it takes a substantial amount of time for an attacker to forge a weak authenticator.

Traditional cryptographic puzzles (e.g., client puzzles [4, 21, 43], congestion puzzles [42]) require interactions between a client and a server. However, broadcast in sensor networks, which involves one sender and a large number of receivers, does not permit such interactions. Moreover, we have to prevent an attacker from pre-computing puzzle solutions. Thus, we have to develop additional techniques to make this idea feasible.

Our solution is *message specific puzzles based on one-way key chains*. Intuitively, we consider each broadcast message, along with the message index and the digital signature, as a (message specific) puzzle. To prevent an attacker from pre-computing puzzle solutions to forged messages, we further add into such a message specific puzzle a previously undisclosed key in the one-way key chain. As a result, an attacker cannot pre-compute a puzzle solution until such a key is released by the sender. Upon receiving such a packet, any node can easily verify the puzzle solution. However, we develop the puzzle system in such a way that it will take any node a substantial amount of time to solve a puzzle. As a result, even if the key K_i is released in a broadcast packet, an attacker cannot immediately solve the puzzle for a forged packet, and thus cannot immediately launch DoS attacks.

Basic Construction. Now let us describe the details of message specific puzzles. As in the strawman approach, we assume the sender has generated a one-way key chain consisting of K_0, K_1, ..., K_n, and distributed K_0 to all potential receivers. The i-th key K_i ($1 \leq i \leq n$) in the one-way key chain is used for the weak authentication of the i-th broadcast packet. We also assume there is a hash function F_p known to the sender and all the receivers.

Given the i-th message M_i, the sender first generates the signature $Sig(i|M_i)$ using its private key. The index i, the message M_i, the signature $Sig(i|M_i)$, and K_i then constitute the puzzle, which we call the *i-th message-specific puzzle*. For the sake of presentation, we call K_i the *(i-th) puzzle key*, and denote the solution for this puzzle as P_i. As illustrated in Figure 3, a valid solution P_i to the i-th message-specific puzzle, where $1 \leq i \leq n$, must satisfy the following two conditions:

1. The puzzle key K_i is the i-th key in the one-way key chain, and
2. After applying the hash function F_p to the i-th message specific puzzle and its solution, we get an image where the first l bits are all "0". That is,

$$F_p(i|M_i|Sig(i|M_i)|K_i|P_i) = \underbrace{00...0}_{l \text{ bits}} \text{xx...x},$$

where "xx...x" represents any bit pattern. The parameter l is called the *strength* of the puzzle.

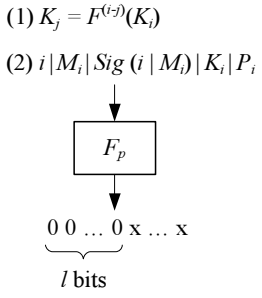

Figure 3: Message specific puzzles

Because of the one-way property of the hash function F_p, one has to search through the space of possible solutions to solve the puzzle. In other words, given i, M_i, $Sig(i|M_i)$, and K_i, for each candidate solution P_i', the sender (or an attacker) has to verify if the first l bits of $F_p(i|M_i|Sig(i|M_i)|K_i|P_i')$ are all "0". On average, the sender needs to try 2^{l-1} possible solutions before finding the right one.

To take advantage of message specific puzzles, we use the puzzle key K_i (i.e., the i-th key in the one-way key chain) and the puzzle solution P_i together as the *weak authenticator for the i-th broadcast packet*. Given the i-th broadcast message M_i, the sender first generates the digital signature $Sig(i|M_i)$, retrieves the puzzle key K_i, and computes the puzzle solution P_i. The sender then broadcasts the packet with the payload $i|M_i|Sig(i|M_i)|K_i|P_i$. Upon receiving a broadcast packet, each receiver first verifies the puzzle key using F_p and K_0 (or a previously verified puzzle key). Only when this verification is successful does the node verify the puzzle solution, and only when the puzzle solution is valid does it verify the signature. Thus, an attacker cannot force the nodes to verify digital signatures in forged packets without first solving some message specific puzzles.

Since the sender needs to solve a message specific puzzle before sending a broadcast packet, the computation involved in finding the puzzle solution should finish in a reasonable amount of time, though it should not be trivial to solve such a puzzle. Thus, an attacker may commit significant computational resources (e.g., multiple powerful computers) to compute puzzle solutions (and thus the weak authenticator) for forged packets once it obtains the puzzle key in a valid broadcast packet. If the attacker is able to use a fast channel (e.g., a wormhole [18]) to send the forged packet to nodes that have not received the valid broadcast packet, it may force these nodes to perform an unnecessary signature verification. (Note that such forged packets do not have much impact on the nodes that have received the valid packet, because they can easily identify the duplicated use of the puzzle key in the forged packets.) Moreover, the attacker may repeatedly send the same forged packet to force the sensor nodes to verify a large number of (the same) signatures, if there is no further protection.

Minimizing the Reuse of Forged Puzzle Solutions. Though we cannot prevent the nodes from verifying the signatures in the forged packets if the attacker has access to sufficient computational resources and fast communication channels, we can still take measures to minimize the impact of such attacks. In particular, we would like to minimize the reuse of forged puzzle solutions by the attacker.

We consider a puzzle solution in a received broadcast packet as a *forged* one if the puzzle solution can be verified but the signature in the same packet cannot. To minimize the impact from attacker reusing forged puzzle solutions, we propose to keep a buffer at each node for the forged puzzle solutions, and use the *multi-buffer random selection* strategy in [25, 26] to manage the buffer. Specifically, assume each node has m entries in the buffer for forged puzzle solutions. For each incoming broadcast packet, each node first checks if the puzzle solution in the packet already exists in the buffer, and drops the packet if yes. Otherwise, for the k-th forged puzzle solution, if $k \leq m$, the node simply saves the puzzle solution in an empty buffer entry. If $k > m$, the node does not have enough buffers to save all forged puzzle solutions. In this case, the node saves it with probability $\frac{m}{k}$. If the puzzle solution is to be saved, the node randomly picks a buffer entry and replaces the old entry with the new puzzle solution.

It is easy to see that when the attacker has more than m forged puzzle solutions, the more frequently the attacker uses one particular forged puzzle solution, the more possible this puzzle solution is in the buffer when it reaches a

sensor node (and is then discarded). Thus, a good strategy for the attacker is to use these forged puzzle solutions at the same frequency. In this case, it is also easy to verify that given k' ($m < k' \leq k$) distinct forged puzzle solutions, each of them solution has the same probably $\frac{m}{k'}$ to be kept in the buffer.

We will provide more details about how well the proposed approach can mitigate the reuse of forged puzzle solutions during the security analysis. In the following, we first describe the procedure with which each node processes incoming broadcast packets.

Processing Broadcast Packets at Receivers. Based on the above discussion, we describe the following procedure for each receiver to process an incoming broadcast packet with payload $i|M_i|Sig(i|M_i)|K_i|P_i$, assuming that the most recently authenticated puzzle key is K_j.

1. If $j > i$ (i.e., the i-th packet is an earlier packet), or $j = i$ but an authenticated copy of the i-th broadcast packet has been received, simply drop this packet and stop.
2. Authenticate the puzzle key K_i with K_j by checking if $K_j = F^{i-j}(K_i)$. (Note that it is possible to have $j = i$ when the node has received forged puzzle solutions.) If K_i is not authenticated, simply drop the packet and stop. Otherwise, if $j < i$, replace K_j with K_i, and empty the buffer for forged puzzle solutions.
3. If P_i is in the buffer for forged puzzle solutions, drop the packet and stop.
4. Verify the puzzle solution by computing $F_p(i|M_i|Sig(i|M_i)|K_i|P_i)$. If the first l bits of the image are not all "0", simply drop the packet and stop.
5. Verify the digital signature $Sig(i|M_i)$ using the sender's public key. If the verification passes, go to Step 7.
6. Add P_i into the buffer for forged puzzle solutions when space is available. Otherwise, suppose this P_i is the k-th puzzle solution received for the i-th broadcast packet. Keep P_i with probability m/k, where m is the number of buffer entries. If P_i is to be kept, randomly pick any previously saved puzzle solution P_i', and replace P_i' with P_i. Drop the packet and stop.
7. Re-broadcast the received packet if necessary[1].

Security Analysis. The one-way property of the hash function F_p brings a nice feature to message specific puzzles: An attacker has to search in a solution space in order to find a weak authenticator for a forged packet. As discussed earlier, given the puzzle strength l, an attacker needs to try 2^{l-1} hash function operations on average in order to find a puzzle solution. Moreover, the use of one-way key chains prevents an attacker from pre-computing puzzle solutions. In other words, it is computationally infeasible for an attacker to compute a puzzle key that has not been disclosed by the sender. Thus, the attacker cannot solve the message specific puzzle to forge the i-th broadcast packet until it has received a valid puzzle key K_i in the (real) i-th broadcast packet.

We temporarily assume that there is no network partition so that all broadcast packets can reach all the nodes in a finite amount of time. (We will discuss the case where there are network partitions in Section 3.3.) Considering the difficulty of solving message specific puzzles, given an appropriate puzzle strength, an attacker may not have enough time to forge a weak authenticator before the broadcast packet reaches all sensor nodes if the attacker does not have substantial computational resources.

An attacker can certainly commit a lot of computational resources to forging weak authenticators. For each forged packet, the attacker has to solve a message specific puzzle, which involves on average 2^{l-1} hash function operations. Since each receiver has m entries in the buffer for forged puzzle solutions, the attacker cannot reuse any forged packet before he/she solves more than m puzzles. Consider the puzzle strength $l = 22$, which requires less than 3 bytes per buffer entry. Based on the benchmark result for Crypto++ 5.2.1 [8], it takes about 7.5 seconds on average for a 2.1 GHz Pentium 4 processor running Windows XP to solve one puzzle. Thus, a 150 byte buffer for forged puzzle solutions will force an attacker with the aforementioned system to compute for about 376.6 seconds on average before the attacker has a chance to reuse a forged puzzle solution.

Suppose the attacker has finished computing k' ($k' > m$) puzzle solutions, and is sending the k-th ($k > k'$) forged packet. In the best case, the attacker can succeed in reusing a previous puzzle solution with probability $1 - \frac{m}{k-1}$. This happens when the attacker sends a newly forged puzzle solution (i.e., the k'-th one) as the $(k-1)$-th packet and attempts to reuse it in the k-th packet. This probability will drop quickly as the attacker attempts to

[1] In some broadcast protocols (e.g., probabilistic broadcast [24, 41]), not all the nodes need to re-broadcast the received packet.

reuse the same forged puzzle solution. It is very likely that the real broadcast packet has reached most (if not all) sensor nodes before the attacker can reuse any forged puzzle solution many times.

Compared with the simple signature-based broadcast authentication where an attacker can claim an arbitrary message as a signed broadcast packet and force many sensor nodes to verify signatures, message specific puzzles have substantially increased the cost of DoS attacks. Moreover, as discussed earlier, even if the attacker has enough resources to launch such attacks, the forged weak authenticators are valid only for a limited period of time. A forged broadcast packet has to arrive at a sensor node before the real packet to generate a significant impact.

Since each broadcast packet includes a message index for the sender, each message specific puzzle is unique. Moreover, the puzzle keys also change from packet to packet. Thus, puzzle solutions will also change with a high probability (approximately $1 - 2^{-l}$), and cannot be reused for later messages.

Performance Analysis. Message specific puzzles introduce light computational overhead on regular sensor nodes. For each broadcast packet, a receiver needs to perform an extra hash function operation to first verify the weak authenticator. When there are DoS attacks against signature verifications, the proposed approach can reduce the computational cost significantly by reducing the number of expensive signature verifications. However, the broadcast sender has to solve a message specific puzzle with strength l in order to generate a valid weak authenticator, which involves 2^{l-1} hash function operations on average per broadcast packet. Moreover, the sender needs to pre-compute a one-way key chain before the deployment of the network. This includes, for example, 10,240 hash function operations for a chain of 10,240 puzzle keys. As discussed earlier, we assume the broadcast sender is a powerful computer with external power supply, and can perform such operations.

Message specific puzzles require some space in each broadcast packet. Besides the message index, the message content, and the digital signature, each packet has to include a puzzle key and a puzzle solution. In general, a 64-bit puzzle key is sufficient to prevent attacks against the one-way key chain, and the solution to a message specific puzzle with strength l requires at least l bits space in the packet. They together require $8 + \lceil \frac{l}{8} \rceil$ bytes space in the packet (e.g., 11 bytes when $l = 24$). Considering the importance of broadcast authentication and the maximum payload size of 102 bytes in ZigBee-compliant sensor nodes (e.g., MICAz), such an overhead is acceptable.

The storage overhead on regular sensor nodes is reasonable. Each node has to maintain the index of the most recently verified broadcast packet, the corresponding puzzle key, and the buffer for m forged puzzle solutions. When 16-bit indexes, 64-bit puzzle keys, and l-bit puzzles are used, these require $10 + m \cdot \lceil \frac{l}{8} \rceil$ bytes space for each sender. For example, these require 160 bytes when $l = 24$ and $m = 50$. However, the storage requirement on the broadcast sender is much heavier. The sender has to keep at least the unused part of the one-way key chain, unless it computes the puzzle key every time it is needed. This requires, for example, 80,960 bytes for a chain of 10,240 64-bit keys. Given the assumption that the sender is a powerful node (e.g., a laptop), this is not a problem at all.

Choice of Parameters. We need to decide several parameters before we can use the message specific puzzles: the hash functions, the puzzle strength l, and the buffer size m for forged puzzle solutions.

Similar to the existing cryptographic puzzles (e.g., client puzzles [4, 21, 43], congestion puzzles [42]), we only use the one-way property of hash functions in message specific puzzles. Thus, as indicated in [21], we may use a fast hash function such as MD4 [38], or a fast block cipher such as RC6 [39] as the hash function. To save the code size, our implementation reuses SHA-1, which is necessary to generate ECDSA signatures, as the hash function for both the one-way (puzzle) key chain and message specific puzzles.

Puzzle strength l is an important parameter for message specific puzzles. The decision of this parameter should follow two principles: First, the sender should be able to solve the puzzle within a reasonable amount of time. An overly large value for l will result in a long delay before transmitting broadcast packets on the sender's side. Second, the parameter should not be too small. In other words, the attacker should not be able to solve a large number of puzzles before the valid broadcast packet is propagated throughout the network. Based on these two principles, the network designer should determine the value l through balancing the maximum delay the sender can tolerate before sending the broadcast packet and the risk of DoS attacks against signature verifications.

The larger buffer a node has for forged puzzle solutions, the better it can minimize the reuse of forged puzzle solutions. In practice, parameter m should be determined based on the available storage on sensor nodes and the threat model. For example, when $l = 22$ and there are more than 150 bytes available on each node, we may set $m = 50$. This requires at most 150 bytes for the buffer, and can force an attacker with one 2.1 GHz Pentium 4

processor to spend about 376.6 seconds on average in order to reuse a previously forged puzzle solution.

A Remaining Threat. A threat still remains when there are network partitions, even if we use message specific puzzles as weak authenticators. Consider the following scenario: A computationally resourceful attacker observes the i-th broadcast packet transmitted by the sender, and learns the key K_i included in this packet. As a result, the attacker can forge the i-th broadcast packet with an invalid signature but valid weak authenticator. This is in general not a big threat to a connected sensor network, because a node will discard the forged packet after it receives the valid broadcast packet. However, when some nodes are isolated from the sender (i.e., they cannot receive the packet from the sender), the attacker can repeatedly forge packets and send to these nodes, and thus force them to verify the (invalid) signatures. The attacker will eventually exhaust their battery power.

3.3 Time Limited Message Specific Puzzles

In this subsection, we further enhance message specific puzzles to mitigate the aforementioned attack against nodes isolated from the sender. For brevity, we refer to a node isolated from the sender as an *isolated node*.

The essential reason for the above attack is that a puzzle key remains valid for a node as long as this node has not authenticated a broadcast packet that uses this or a later key in the key chain. An attacker can use this puzzle key repeatedly to exhaust the battery power of isolated nodes. Our solution is thus to invalidate this condition.

Our solution is inspired by TESLA [35]. As shown in Figure 4, we divide the time period for broadcasting into multiple time intervals, labeled as I_1, I_2, ..., I_n. Each puzzle key K_i in the one-way key chain is associated with the time interval I_i, where $1 \leq i \leq n$. The sender uses K_i for weak authentication only during the time interval I_i. For convenience, we denote the starting point and the end point of interval I_i ($1 \leq i \leq n$) as T_{i-1} and T_i, respectively.

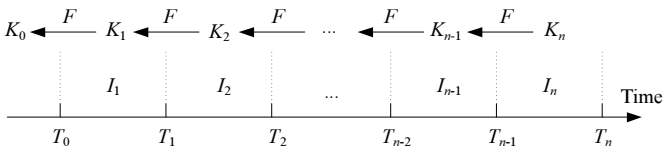

Figure 4: One-way key chain in time-limited message specific puzzles. Each K_i is only valid between $T_{i-1} - \delta_c$ and $T_i + \delta_c + \delta_p$, where δ_c is the maximum clock difference between the sender and any receiver, and δ_p is the maximum propagation delay.

We assume the clocks of the sender and all receivers are loosely synchronized. More precisely, we assume the clock difference between any two nodes is bounded by δ_c. Moreover, we assume the propagation delay of any broadcast packet in a network without partition is bounded by δ_p. This delay may also include the time required by signature verification at intermediate forwarding nodes. At each receiver, each key K_i is only valid between $T_{i-1} - \delta_c$ and $T_i + \delta_c + \delta_p$ (in the local clock). When a node receives a broadcast packet at local time t with a weak authenticator, which is composed of the puzzle key K_i and the puzzle solution P_i, it first verifies the condition $T_{i-1} - \delta_c < t < T_i + \delta_c + \delta_p$, and continues to perform the steps specified in Section 3.2 only when this condition is satisfied. As a result, even if a node is isolated from the sender, an attacker can only use a cracked weak authenticator for a limited period of time.

The reader may have noticed that when the sender has not broadcast for a relatively long period of time, all the receivers have to perform a potentially large number of hash function operations to verify the key in a new broadcast packet. A simple solution to this problem is to have the sender periodically (e.g., for every 100 time intervals) broadcast the most recently expired key to the network. (Note that such keys are self-authenticated because of the one-way key chain.) After receiving and authenticating such a key, each receiver replaces the most recently authenticated puzzle key with the new one. As a result, the receiving nodes can spread the verification of the puzzle keys in the one-way key chain over time.

Time limited message specific puzzles retain the security and performance properties of message specific puzzles discussed earlier. Moreover, it can prevent attackers from launching unlimited DoS attacks against isolated nodes, as discussed earlier. This extension does bring a restriction along with the benefit: The sender cannot send more than one broadcast packet per time interval, since each time interval has only one puzzle key. This can be addressed by having short time intervals, or having multiple puzzle keys per interval. The sender may need a potentially large number of puzzle keys, many of which are not used. Such a problem can be potentially addressed by using sandwich chains [17] or multi-level key chains [25, 26]. These approaches provide more complex but efficient ways to organize key chains, and allow receivers to skip the computation of intermediate keys when authenticating later keys. Since these are not the focus of this paper, we do not discuss them in detail.

3.4 Adaptive Verification

As discussed earlier, broadcast in a wireless sensor network typically requires that some nodes receiving an authenticated broadcast packet re-broadcast it (locally) to propagate the packet across the network. In the proposed (time limited) message specific puzzles, such a node verifies the puzzle solution and the digital signature before forwarding the broadcast packet. Though the verification of solutions to message specific puzzles is trivial, signature verification takes much longer time. This will certainly introduce undesirable delays in large sensor networks.

An alternative approach is to have each node re-broadcast the packet right after verifying the puzzle solution but before verifying the signature. However, message specific puzzles are *weak* authenticators intended for mitigating DoS attacks against signature verifications. As discussed earlier, they can be forged if the attacker devotes significant computational resources. If a node uses this alternative approach, it may forward forged packets before realizing that they are forged, thus wasting the limited battery power on unnecessary packet transmissions.

It seems that both approaches are not satisfactory. To address this dilemma, we propose an adaptive approach to determining the order of signature verification and forwarding of broadcast packets. Intuitively, this approach tries to detect attempts of DoS attacks against signature verifications. In normal situations where there are no such attacks, each node re-broadcasts a broadcast packet once the weak authenticator is verified, and then verifies the signature. However, when there are DoS attacks against signature verifications, each node first verifies the digital signature, and then re-broadcasts the packet if the signature is valid.

Figure 5 illustrates this approach. Each node works in two modes: *optimistic mode* and *pessimistic mode*. In the optimistic mode, a node re-broadcasts the packet locally once it verifies the weak authenticator. In contrast, in the pessimistic mode, a node verifies both the weak authenticator and the signature, and re-broadcasts the packet only when both verifications pass. The switch between these two modes is determined by a detection metric N_f, *the number of failed signature verifications in the past* w *time units*, where w is a system parameter determined by the security policy. Note that a node verifies a signature only when the weak authenticator is valid. Thus, N_f represents the number of forged broadcast packets with valid weak authenticators but invalid signatures. A node initially works in the optimistic mode. It switches to pessimistic mode if N_f becomes greater than 0, and may switch back to the optimistic mode when N_f drops to 0.

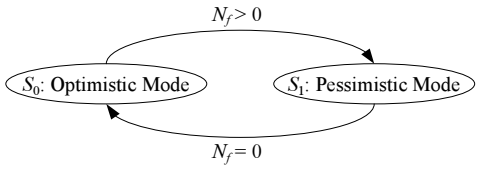

Figure 5: Adaptive verification (N_f: # of failed signature verifications in the past w time units)

Adaptive verification can be used with either message specific puzzles or time limited message specific puzzles, and retains the same security properties. When there are DoS attacks, this approach is exactly the same as proposed earlier. However, in normal situations where there are no such attacks, adaptive verification can substantially reduce the broadcast delay.

3.5 Discussion

The proposed message specific puzzle technique can be more secure against external attackers with a slight change. If all the normal sensor nodes share a key, we can include this key in the input to the hash function F_p. As a result, only valid nodes can solve the message specific puzzles. However, this does not add any defense against compromised nodes, which have access to the global key.

Though we presented message specific puzzles as a defense mechanism for signature-based broadcast authentication, it can potentially be used with μTESLA-based broadcast authentication [25, 26, 37] as well. The concern in this case is not the expensive computation, but the buffer of broadcast packets before they can be verified with an authentication key disclosed later. Nevertheless, the use of message specific puzzles for μTESLA and its variations is not the focus of this paper, and may entail additional research. We will consider it in future work.

Despite the useful properties, message specific puzzles also have some limitations. First, the broadcast sender has to solve a puzzle before broadcasting a message. This requires that the sender must be a computationally powerful node with sufficient power supply, and also implies that there will be a delay before the transmission of the packet. However, in certain applications, these problems are tolerable in exchange of the mitigation of the DoS attacks. Moreover, message specific puzzles add moderate communication overhead and storage overhead on regular sensor nodes. As discussed in the security analysis in Section 3.2, these overheads are generally acceptable in the current generation of sensor networks.

4 Implementation and Experimental Evaluation

We have implemented the proposed techniques on TinyOS, an operating system for networked sensors [16], and performed initial experimental evaluation. To facilitate the evaluation, we have implemented a software package called *TinyECC* for Elliptic Curve Cryptography (ECC) on TinyOS. The current version of TinyECC includes big integer operations (e.g., modular addition, modular multiplication), elliptic curve operations (i.e., point addition, point doubling, point multiplication) over finite field F_p, ECDSA over F_p, and SHA-1 (which is required for ECDSA). More details of TinyECC can be found in Appendix C.

The proposed techniques are implemented in a software package called *TinySigGuard*, which uses TinyECC for digital signature support on TinyOS. To reuse the code in TinyECC, we used SHA-1 as the hash functions for both message specific puzzles and the one-way key chain. To reduce the size of the puzzle keys included in broadcast packets, when generating the one-way key chain, we randomly generate a 64-bit key as the last puzzle key (K_n), and truncate the output of SHA-1 function to 64 bits. Thus, all puzzle keys in the one-way key chain have 64 bits. Note that truncating each SHA-1 output to 64 bits does not necessarily provide the expected security as in a 64-bit one-way function. This is simply an implementation decision, and can be replaced if such a truncation significantly weakens the one-way property.

TinySigGuard consists of two parts: *TSGSender* and *TSGReceiver*. TSGSender is a Java program running on a PC. It communicates with the sensor network through a regular sensor node attached to the PC, which runs *TOSBase*, an application (in the TinyOS distribution) that simply forwards packets between the sensor network and the PC. To broadcast an authenticated packet, TSG_sender first generates the packet by signing the broadcast data and solving the message specific puzzle, and then sends it to the sensor node running TOSBase, which then broadcasts the packet. TSGReceiver is responsible for verifying the message specific puzzles and digital signatures in broadcast packets and re-broadcasting them. In this implementation, we take the simplest flooding approach as the broadcast protocol. That is, each receiver re-broadcasts a packet once receiving and authenticating it. It is certainly desirable to experiment with other more efficient broadcast protocols; we will do so in our future research.

To allow the transmission of broadcast packets authenticated with ECDSA signatures, we modified the maximum payload size in TinyOS from 29 bytes to 102 bytes, which is the maximum payload size in IEEE 802.15.4 standard specification [20].

Table 1: Code Size (Bytes) on MICAz

	ROM	RAM
TSGReceiver	1,978	398

Table 1 shows the code size of TSGReceiver on MICAz, which was obtained using the `check_size.pl` script in the TinyOS CVS repository. This does not include the code in TinyECC and other TinyOS components. In our experiments, the sender is a DELL Latitude D510 laptop with a 1.6 GHz Pentium M 730 processor and 512 MB DDR SDRAM. Each sensor node is a MICAz mote, which has an 8-bit Atmega128 processor and an IEEE 802.15.4 compliant RF transceiver. (More details about MICAz can be found in [1].) Table 2 shows some timing results on solving message specific puzzles (on the PC) and verifying puzzle solutions (on a MICAz mote). Each number is the average of the results from 100 random test cases. Note that the timing results for the sender is obtained with a Java program; it can be more efficient if a good implementation is used on the native machine (rather than the Java virtual machine).

Table 2: Time (milliseconds) Required for Solving and Verifying Message Specific Puzzles

Puzzle Strength (l)	Solving a Puzzle (on PC)	Verifying a Solution Puzzle (on MICAz)
20	8,835	14.6
22	28,203	14.6
24	132,250	14.6
26	581,258	14.6

We have performed experiments in a network of 30 MICAz motes. We used ECDSA on the 160-bit elliptic curve `secp160k1` specified by SECG [6]. As discussed earlier, we used a simple flooding protocol for broadcasting. That is, each node re-broadcasts an authenticated packet when it receives this packet for the first time. To avoid packet collision, each node randomly delays between 0 and 50 milliseconds before re-broadcasting. Since the computation and storage overheads are already clear from the analysis and the earlier experiments, these experiments are focused on the communication delay introduced by message specific puzzles.

Figure 6 shows the broadcast delays at different hops from the sender for four cases: (1) no broadcast authentication, (2) our approach in optimistic mode using TinyECC, (3) our approach in pessimistic mode using TinyECC, and (4) our approach in pessimistic mode using the optimized ECC implementation in [13]. The first three cases

were obtained in our experiments, while the last case is estimated based on the timing results in [13] (i.e., each ECDSA signature verification takes about 1.62 seconds).

It is easy to see that the proposed approach introduces light delays when used in optimistic mode. However, in pessimistic mode (i.e., when there are DoS attacks), the proposed approach does add significant delays (e.g., about 10 seconds to reach 7 hops with the optimized ECC implementation in [13]). Though these results do not justify the immediate use of these techniques, they are close to acceptable performances. We expect these techniques will be practical when sensor nodes with better processing power are available.

We have provided security analysis for the proposed techniques, and performed initial experimental evaluation in normal situations. It is also desirable to experiment with these techniques when there are attacks. We will perform such experiments in our future research.

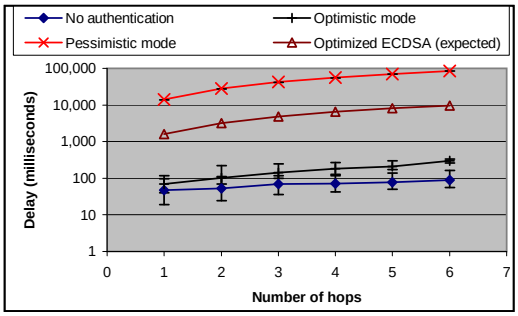

Figure 6: Delay with and without Signature-Based Authentication

5 Related Work

Broadcast authentication has been traditionally achieved with digital signatures, where the sender signs the messages and all the receivers can authenticate the messages by verifying the signatures. In the past few years, many researchers have been working on how to reduce the number of signature operations in, for example, streaming applications over lossy channels (e.g., graph-based broadcast authentication [10, 28, 40], forward error correction based approaches [32, 33]) and to address the resulting problems (e.g., DoS attacks [12, 22]). These techniques certainly suffer from the DoS attacks against signature verifications in wireless sensor networks, though message specific puzzles can potentially be useful to mitigate such attacks.

Researchers have been working on broadcast authentication purely based on symmetric cryptography, such as TESLA [35] and its variations [25–27, 37], which have been discussed in the Introduction, and BiBa [34]. In particular, μTESLA [37] and the later variations have been considered a good candidate for broadcast authentication wireless sensor networks. However, μTESLA and the other variations require a loose clock synchronization, which cannot always be guaranteed. Moreover, they introduce authentication delays, and could be exploited by an attacker to launch DoS attacks. (See Appendix A.1 for more details.) The techniques developed in this paper provide a way to use digital signatures efficiently in wireless sensor networks, and thus provide a solution complementary to μTESLA-based approaches.

Message specific puzzles are essentially an integration of client puzzles and one-way hash chains. Client puzzles were proposed in [21] and later used in several applications (e.g., [4, 42, 43]). However, all the previous cryptographic puzzle techniques require interactions between a client and a server. Our innovation in this paper is to integrate cryptographic puzzles, one-way key chains, and broadcast messages together to achieve weak authentication without requiring interaction between the sender and many receivers.

6 Conclusion

In this paper, we developed message specific puzzles, a weak authentication mechanism, to mitigate DoS attacks against signature-based broadcast authentication in wireless sensor networks. This approach has a number of nice properties: First, a weak authenticator can be efficiently verified by a regular sensor node, but takes a computationally powerful attacker a substantial amount of time to forge. Second, a weak authenticator cannot be pre-computed without a non-reusable key disclosed only in a valid broadcast packet. Thus, an attacker cannot start the expensive computation to forge a weak authenticator without seeing a valid broadcast packet. Third, even if an attacker has sufficient computational resources to forge one or more weak authenticators, it is difficult to reuse these forged weak authenticators. Thus, this weak authentication mechanism substantially increases the difficulty of launching successful DoS attacks against signature verifications. We have implemented the proposed techniques in a software package named TinySigGuard on TinyOS, and performed initial experimental evaluation on MICAz motes. In our future research, we will continue the experimental evaluation in large-scale sensor networks, and investigate the effective integration with efficient broadcast protocols for wireless sensor networks.

References

[1] Micaz: Wireless measurement system. http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38(4):393–422, 2002.

[3] American Bankers Association. *ANSI X9.62-1998: Public Key Cryptography for the Financial Services Industry: the Elliptic Curve Digital Signature Algorithm (ECDSA)*, 1999.

[4] T. Aura, P. Nikander, and J. Leiwo. DOS-resistant authentication with client puzzles. In *Proceedings of the 8th International Workshop on Security Protocols, LNCS 2133*, pages 170–177, 2001.

[5] I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography*. Cambridge University Press, 1999. London Mathematical Society Lecture Note Series 265.

[6] Certicom Research. Standards for efficient cryptography – SEC 2: Recommended elliptic curve domain parameters. http://www.secg.org/collateral/sec2_final.pdf, September 2000.

[7] Crossbow Technology Inc. Wireless sensor networks. http://www.xbow.com/Products/Wireless_Sensor_Networks.htm. Accessed in May 2005.

[8] W. Dai. Crypto++ 5.2.1 benchmarks. http://www.eskimo.com/~weidai/benchmarks.html, July 2004.

[9] 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.

[10] R. Gennaro and P. Rohatgi. How to sign digital streams. In *Advances in Cryptology – CRYPTO '97*, pages 180–197, 1997.

[11] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[12] C.A. Gunter, S. Khanna, K. Tan, and S. Venkatesh. DoS protection for reliably authenticated broadcast. In *Proceedings of the 11th Network and Distributed Systems Security Symposium (NDSS '04)*, pages 17–36, 2004.

[13] N. Gura, A. Patel, and A. Wander. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In *Proceedings of the 2004 Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004)*, August 2004.

[14] 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.

[15] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[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] Y. Hu, M. Jakobsson, and A. Perrig. Efficient constructions for one-way hash chains. In *Proceedings of the 3rd International Conference on Applied Cryptography and Network Security*, pages 423–441, June 2005.

[18] Y. Hu, A. Perrig, and D. B. Johnson. Wormhole detection in wireless ad hoc networks. Technical Report TR01-384, Department of Computer Science, Rice University, Dec 2001.

[19] Y.C. Hu, A. Perrig, and D.B. Johnson. Packet leashes: A defense against wormhole attacks in wireless ad hoc networks. In *Proceedings of INFOCOM 2003*, April 2003.

[20] IEEE Computer Society. IEEE 802.15.4: Ieee standard for information technology – telecommunications and information exchange between systems local and metropolitan area networks – specific requirements part 15.4: Wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANs). `http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf`, October 2003.

[21] A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proceedings of the 6th Network and Distributed Systems Security Symposium (NDSS '99)*, February 1999.

[22] C. Karlof, N. Sastry, Y. Li, A. Perrig, and J. Tygar. Distillation codes and applications to dos resistant multicast authentication. In *Proceedings of the 11th Network and Distributed Systems Security Symposium (NDSS '04)*, pages 37–56, 2004.

[23] RSA Laboratories. RSAREF: A cryptographic toolkit (version 2.0), March 1994.

[24] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the 1st Symposium on Network System Design and Implementation (NSDI '04)*, March 2004.

[25] 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 (NDSS'03)*, pages 263–276, February 2003.

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

[27] D. Liu, P. Ning, S. Zhu, and S. Jajodia. Practical broadcast authentication in sensor networks. In *Proceedings of the 2nd Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous 2005)*, July 2005.

[28] S. Miner and J. Staddon. Graph-based authentication of digital streams. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 232–246, 2001.

[29] J. Newsome and D. Song. GEM: graph embedding for routing and data-centric storage in sensor networks without geographic information. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys '03)*, pages 76–88, Nov 2003.

[30] S. Ni, Y. Tseng, Y. Chen, and J. Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99)*, pages 151–162, 1999.

[31] D. Niculescu and B. Nath. Ad hoc positioning system (APS). In *Proceedings of IEEE GLOBECOM '01*, 2001.

[32] A. Pannetrat and R. Molva. Efficient multicast packet authentication. In *Proceedings of the 10th Network and Distributed Systems Security Symposium (NDSS '03)*, pages 251–262, 2003.

[33] J.M. Park, E.K.P. Chong, and H.J. Siegel. Efficient multicast stream authentication using erasure codes. *ACM Transactions on Information and System Security*, 6(2):258–285, 2003.

[34] 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.

[35] 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.

[36] 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.

[37] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and D. Tygar. SPINS: Security protocols for sensor networks. In *Proceedings of Seventh Annual International Conference on Mobile Computing and Networks*, July 2001.

[38] R. Rivest. The MD4 message-digest algorithm. RFC 1320, April 1992. `http://www.ietf.org/rfc/rfc1320.txt`.

[39] R. Rivest, M. Robshaw, R. Sidney, and Y.L. Yin. The RC6 block cipher. Presented at the NIST Fist AES Candidate Conference, 1998.

[40] D. Song, D. Zuckerman, and J.D. Tygar. Expander graphs for digital stream authentication and robust overlay networks. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, 2002.

[41] I. Stojmenovic, M. Seddigh, and J. Zunic. Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):14–25, 2002.

[42] X. Wang and M.K. Reiter. Mitigating bandwidth-exhaustion attacks using congestion puzzles. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS '04)*, pages 257–267, October 2004.

[43] B. Waters, A. Juels, J.A. Halderman, and E.W. Felten. New client puzzle outsourcing techniques for dos resistance. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS '04)*, pages 246–256, October 2004.

A A Brief Overview of μTESLA and Its Limitations

An asymmetric mechanism such as public key cryptography is generally required for broadcast authentication [35]. Otherwise, a malicious receiver can easily forge any packet from the sender, as discussed earlier. μTESLA introduces asymmetry by delaying the disclosure of symmetric keys [37]. A sender broadcasts a message with a Message Authentication Code (MAC) generated with a secret key K, which is disclosed after a certain period of time. When a receiver gets this message, if it can ensure that the packet was sent before the key was disclosed, the receiver buffers this packet and authenticates the packet when it later receives the disclosed key. To continuously authenticate broadcast packets, μTESLA divides the time period for broadcast into multiple intervals, assigning different keys to different time intervals. All packets broadcast 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. μTESLA 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 (cryptographic) hash 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 (except for K_0) is assigned to the i-th time interval. Because of the one-way property of the hash function, 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 ones $K_i, j+1 \leq i \leq n$. Thus, with the knowledge of the initial key K_0, which is called the *commitment* of the key chain, a receiver can authenticate any key in the key chain by merely performing hash function operations. When a broadcast message is available in the i-th time interval, the sender generates a MAC for this message with a key derived from K_i, broadcasts this message along with its MAC, and discloses the key K_{i-d} for time interval I_{i-d} in the broadcast message (where d is the disclosure lag of the authentication keys).

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. μTESLA uses a security condition to prevent such situations. When a receiver receives

an incoming broadcast packet in time interval I_i, it checks the security condition $\lfloor (T_c + \Delta - T_1)/T_{int} \rfloor < i + d - 1$, where T_c is the local time when the packet is received, T_1 is the start time of the time interval 1, T_{int} is the duration of each time interval, and Δ 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.

μTESLA is an extension to TESLA [35]. The only difference between TESLA and μTESLA 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 overheads. μTESLA depends on symmetric cryptography (with the master key shared between the sender and each receiver) to bootstrap the new receivers individually. TESLA was later extended to include an immediate authentication mechanism [36]. The basic idea is to include an image under a hash 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 being received. This extension can also be applied to μTESLA.

A.1 Limitations of μTESLA in Wireless Sensor Networks

A major limitation of μTESLA [37] and its variations [25, 26] is the authentication delay. In other words, a receiver cannot authenticate a broadcast packet immediately after receiving it. Consider the fact that a broadcast packet typically has to be forwarded (via local re-broadcast) multiple times before it can reach all the nodes. This means that a sensor node has to forward a broadcast packet before properly authenticating it. The key disclosed in a broadcast packet can provide some weak authentication. However, once an attacker receives a normal broadcast packet, he/she can reuse this key to forge many packets that can pass this weak authentication. As a result, an attacker can easily inject a large number of forged broadcast packets, force regular nodes to forward these bogus packets, and eventually exhaust their battery power.

Perrig et al. provided an immediate authentication mechanism by including the hash image of the *packet content* in a previous broadcast *packet* [36]. This extension ensures the immediate authentication of the content of a broadcast packet. However, such immediate authentication does not cover the hash image for the later packet content, nor the MAC. As a result, an attacker can forge a large number of packets by modifying the uncovered part without being immediately detected, force sensor nodes to propagate these packets across the network, and eventually exhaust their limited battery power.

B Packet Delivery Rates for MICA2 and MICAz in Indoor Environments

We performed some in-door experiments to confirm the packet loss rate for MICAz with large packet sizes. We used both MICA2 and MICAz in our experiments for comparison purposes. The Radio Frequency (RF) module of MICA2 runs at frequency 916.7MHz, while that of MICAz runs at frequency 2.425GHz. We set the transmission power as -10dbm on both MICA2 and MICAz. Figure 7 shows the comparison of packet delivery rates for MICA2 and MICAz when the packet payload size is 102 bytes (i.e., the maximum payload size in IEEE 802.15.4). It is easy to see that the packet delivery rate for MICAz remains above 95% in all test cases, when this rate quickly drops to 0 as the distance between the sender and the receiver increases from 50 feet to 90 feet. This results confirms our assumption that it is practical to have a large enough packet that can accommodate a digital signature on IEEE 802.15.4 compliant sensor nodes.

C Implementation and Performance of TinyECC

We have been implementing a software package for Elliptic Curve Cryptography (ECC) on TinyOS, which is called *TinyECC*. Currently, we have finished the implementation of elliptic curve operations over finite field F_p and ECDSA [3] on such curves. In this appendix, we briefly describe this implementation and the performance results.

The big integer operations in TinyECC are based on RSAREF 2.0 [23]. We first ported the natural number operations in RSAREF 2.0 to TinyOS, and then added a few known optimizations to speed up ECC operations.

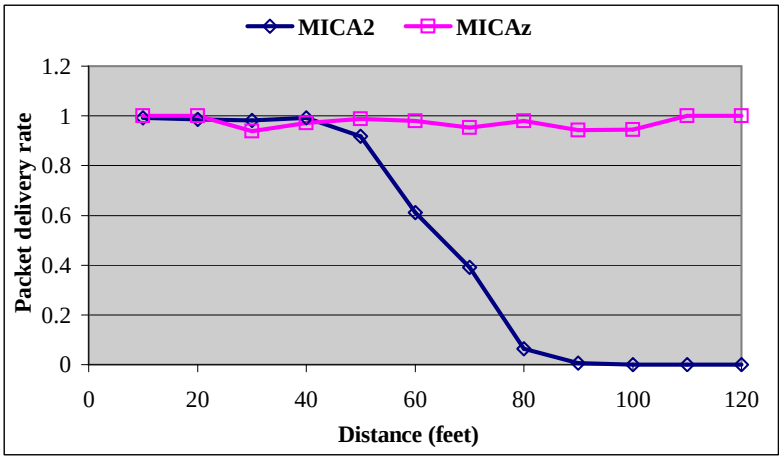

Figure 7: Packet delivery rates for MICA2 and MICAz in our indoor experiments (Payload size: 102 bytes)

These include the modular square, modular addition, and modular subtraction operations described in [15], as well as the optimization for modular multiplication described in [13].

We adopted Jacobian projective coordinates to speed up the ECC operations [15]. In particular, we used the algorithms for elliptic curve point addition and point doubling described in [5]. To optimize for special elliptic curves, we used the point addition and doubling algorithms in [15] for elliptic curves $y^2 = x^3 + ax + b$ with $a = 0$ or -3. Finally, we used the sliding window method (i.e., process w bits a time, where w is the window size) described in [15] for point multiplication.

Table 3 shows some performance results of ECDSA in TinyECC for the SECG-defined elliptic curves [6]. In Table 3, the name of each curve starts with "secp", indicating that the curve is over F_p by SECG. The number after "secp" indicates the number of bits in the prime number defining the finite field. The letter "k" or "r" then indicates if the elliptic curve is a Koblitz curve or a verifiably random curve. The name of each curve is then finished with a sequence number. The exact parameters of each curve can be found in [6].

Note that the code sizes shown in Table 3 are those for the testing program, which signs and verifies random messages, and then transmits the timing results back to a PC. The performance results may change as we improve our implementation.

Table 3: Code size and execution time of ECDSA in TinyECC on MICAz motes. (Note that the code size changes with different parameters due to the changes in the ECC parameters and some internal state. Moreover, the code for SHA-1, which is 31,866 bytes, is the dominant part of TinyECC.)

Curve	Window Size (w)	Code Size (ROM)	Code Size (RAM)	Signature Generation	Signature Verification
secp128r1	$w = 2$ bits	44,438 bytes	442 bytes	9,966 ms	19,201 ms
	$w = 4$ bits	44,446 bytes	1,298 bytes	8,716 ms	17,080 ms
secp128r2	$w = 2$ bits	44,468 bytes	442 bytes	11,405 ms	22,499 ms
	$w = 4$ bits	44,476 bytes	1,298 bytes	10,150 ms	20,270 ms
secp160k1	$w = 2$ bits	44,530 bytes	522 bytes	8,189 ms	16,272 ms
	$w = 4$ bits	44,538 bytes	1,570 bytes	7,012 ms	13,956 ms
secp160r1	$w = 2$ bits	44,546 bytes	522 bytes	8,464 ms	16,395 ms
	$w = 4$ bits	44,554 bytes	1,570 bytes	7,317 ms	14,630 ms
secp160r2	$w = 2$ bits	44,550 bytes	522 bytes	8,843 ms	17,393 ms
	$w = 4$ bits	44,558 bytes	1,570 bytes	7,727 ms	15,257 ms

TinyECC is implemented primarily for research purposes, due to the lack of publicly available implementation of public key cryptosystems on the current generation of sensor networks. It does not include all the known

optimizations for big integer and EC operations. Moreover, because it is entirely implemented in nesC, it cannot take advantage of some techniques (e.g., hybrid multiplication aimed at optimizing the registers and memory accesses) described in [13].