Secure Data Discovery and Dissemination based on Hash Tree for Wireless Sensor Networks

Daojing He*, Sammy Chan, Shaohua Tang*, and Mohsen Guizani

***Abstract*—Wireless sensor networks (WSNs) are widely applicable in monitoring and control of environment parameters. It is sometimes necessary to disseminate data through wireless links after they are deployed in order to adjust configuration parameters of sensors or distribute management commands and queries to sensors. Several approaches have been proposed recently for data discovery and dissemination in WSNs. However, they all focus on how to ensure reliability and usually overlook security vulnerabilities.**

This paper identifies the security vulnerabilities in data discovery and dissemination when used in WSNs. Such vulnerabilities allow an adversary to update a network with undesirable values, erase critical variables, or launch denial-of-service (DoS) attacks. To address these vulnerabilities, this paper presents the design, implementation, and evaluation of a secure, lightweight, and DoS-resistant data discovery and dissemination protocol named *SeDrip* for WSNs. Our protocol takes into consideration the limited resources of sensor nodes, packet loss and out-of-sequence packet delivery. Also, it can provide instantaneous authentication without packet buffering delay, and tolerate node compromise. Besides the theoretical analysis that demonstrates the security and performance of SeDrip, this paper also reports the experimental evaluation of SeDrip in a network of resource-limited sensor nodes, which shows its efficiency in practice.

***Index Terms*—Data discovery and dissemination, security, wireless sensor networks, efficiency.**

I. Introduction

WIRELESS sensor networks (WSNs) have been attracting great interest in a wide range of applications related to monitoring and control of environmental or physical conditions, such as industry monitoring and military operations. After a WSN is deployed in the field, it may be necessary to update the installed programs or stored parameters in sensor nodes. This can be achieved by dissemination services which ensure new programs or parameter values to be propagated throughout the WSN so that all nodes have a consistent copy. Normally, a new program is of the order of kilobytes while a parameter is just few bytes long. Due to such a vast disparity between their sizes, the design considerations of their dissemination protocols are different. As a result, two types of dissemination protocols are developed in the literature. Code dissemination (also referred to as data dissemination or reprogramming) protocols [1] are developed to efficiently distribute long messages into a network, enabling complete system reprogramming. On the other hand, *data discovery and dissemination protocols* are used to distribute short messages, such as several two-byte configuration parameters, within a WSN. Common uses of this kind of protocols include injecting small programs, commands, queries, and configuration parameters.

Recently, several data discovery and dissemination protocols [2]–[5] have been proposed. Among them, Drip [2], DIP [3] and DHV [4] are most well known and included in TinyOS distributions [6]. However, to the best of our knowledge, all existing data discovery and dissemination protocols only address reliable data transmission, but provide no security mechanism. Certainly, this is a critical issue that needs to be addressed. Otherwise, adversaries could, for example, distribute viral or false data to cripple a WSN deployed in the battlefield.

In an effort to make these protocols secure, this paper has the following main contributions:

1) We first investigate the security issues in data discovery and dissemination procedure of WSNs and point out that the lack of authentication of the disseminated data introduces a vulnerability to the update of arbitrary data in WSNs.
2) We then develop a secure, lightweight, and Denial-of-Service (DoS)-resistant data discovery and dissemination protocol named *SeDrip* for WSNs, which is a secure extension of Drip. To achieve DoS-attack resilience and allow immediate verification of any received packets, SeDrip is based on a signed Merkle hash tree. This way the base station of a WSN needs to sign only the root of this tree. Also, SeDrip can tolerate the compromise of some sensor nodes. To further improve the security and efficiency, some additional mechanisms such as the message specific puzzle approach are incorporated into the design of SeDrip.
3) We also implement the proposed protocol in networks of MicaZ and TelosB motes, respectively. Experimental results demonstrate its high efficiency in practice. To the best of our knowledge, this is also the first implemented secure data discovery and dissemination protocol for WSNs.

The rest of the paper is organized as follows. Section II analyzes security vulnerabilities in data discovery and dissemination. The design considerations of securing this procedure

Manuscript received January 13, 2013; revised May 21, 2013; accepted July 2, 2013. The associate editor coordinating the review of this paper and approving it for publication was N. Kato.

This work was supported by the Guangxi Key Laboratory of Trusted Software, National Natural Science Foundation of China under Grant No. U1135004 and 61170080 and Guangdong Province Universities and Colleges Pearl River Scholar Funded Scheme (2011).

D. He is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, P.R.China, and also with the Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, P.R. China (e-mail: hedaojinghit@gmail.com).

S. Chan is with the Department of Electronic Engineering, City University of Hong Kong, Hong Kong SAR, P.R. China (e-mail: eeschan@cityu.edu.hk).

S. Tang is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, P.R. China (e-mail: csshtang@scut.edu.cn).

M. Guizani is with Qatar University, Qatar (e-mail: mguizani@ieee.org).

*D. He and S. Tang are co-corresponding authors.

Digital Object Identifier 10.1109/TWC.2013.090413.130072

are described in Section III. SeDrip is presented in Section IV in detail. Some approaches to defeat DoS attacks and improve the efficiency are suggested in Section V. Section VI provides theoretical analysis of the security of SeDrip. Section VII describes the implementation and experimental evaluation of SeDrip. Section VIII concludes this paper.

II. Security Vulnerabilities in Data Discovery and Dissemination

A. Review on Data Discovery and Dissemination

Both Drip and DIP are based on the same algorithm - Trickle [7]. According to the algorithm, a node periodically broadcasts a summary of the data that it has, unless it has recently received the same summary from other nodes. If a node receives an old summary, it returns an update to the source of the old summary. Once the data in all nodes are consistent, the broadcast interval is exponentially increased to save energy. On the other hand, if a node detects that other nodes have new data, it starts reporting more quickly. Thus, once new data are injected to a node by the base station, they will be disseminated by Trickle quickly. Among the existing data discovery and dissemination protocols, Drip is a simple and basic implementation that establishes an independent trickle for each data item.

In practice, each data item contains a unique key to identify which variable it will update and a value to indicate its freshness. For example, in Drip, DIP and DHV, each data item is represented as a 3-tuple *(key, version, data)*, where *key* uniquely identifies the variable to be updated, *version* indicates if the data item is old or new (the larger the *version*, the newer the data), and *data* denotes the disseminated data (e.g., parameter, command or query).

B. Security Vulnerabilities

All existing data discovery and dissemination protocols developed for WSNs assume benign environments. However, in hostile environments, data discovery and dissemination would face both external and insider attacks.

External attacks include eavesdropping for sensitive information, injecting forged messages, replaying previously intercepted messages, and impersonating valid sensor nodes. Also, the adversary may launch wormhole attacks to fake non-existing links, or DoS attacks by forging, for example, a lot of signature/data packets to consume the limited resources (e.g., battery power, memory) on selected sensor nodes. Insider attacks can be launched by compromising some nodes to attack the rest of the network. Insider attacks can be launched when one or more nodes are compromised. These compromised nodes are totally controlled by adversaries to attack other nodes in the network. They are regarded as insiders because they are still members of the network until they are identified and excluded from the network. For example, they may be instructed by the adversary to launch Sybil attacks, inject false data, exploit specific weaknesses of various protocols, or not to cooperate with other nodes.

With forged messages, an adversary can reboot the entire network, or erase an important variable from all sensor nodes. For example, to reboot a network with a wrong data $data^*$ for the variable identified by *key*, the adversary simply needs to inject a fake data item $(key, version, data^*)$ to the network using a data discovery and dissemination protocol, where *version* is larger than all version numbers of the variable stored on the sensor nodes. Any receiving node will update the variable with the received $data^*$ due to the larger version number. Similarly, to erase an important variable identified by key on all sensor nodes, the adversary can simply send a fake data item $(key, version, 0)$ using a data discovery and dissemination protocol, where $version$ is a large enough number.

III. Assumptions and Design Considerations of Secure Data Discovery and Dissemination

A. Assumptions

Our protocol is based on the following assumptions.

- As the source of disseminated data, the base station cannot be compromised, and is trustworthy. In data discovery and dissemination, the base station is the origin of all legitimate data updates. If it were not trustworthy, nothing else in the network could be trusted.
- The base station has unlimited computational power compared with sensor nodes.
- Although the sensor nodes are resource constrained in terms of memory space, computation power, bandwidth and energy, it can perform a limited number of public key cryptographic operations during the lifetime of its battery.

B. Design Considerations

According to the analysis in Section II, it is vital to provide security measures to data discovery and dissemination. The root of the vulnerability is the lack of authentication of incoming data items. However, providing security measures for WSNs encounters many challenges. The primary one is the limited computing and storage capabilities of sensor nodes. For example, a seemingly straight forward solution is that the base station signs each packet individually using digital signature technique and each node verifies the signature before processing the packet. However, this is inapplicable because it suffers from expensive computational and communication cost. To address this problem, TESLA and its various extensions have been proposed [8], [9], which are based on the delayed disclosure of authentication keys, i.e., the key used to authenticate a message is disclosed in the next message. However, these mechanisms require time synchronization for the whole network and are vulnerable to a DoS attack due to the authentication delay. Through a simple flooding attack, each sensor node has to buffer all forged data items claimed to be from the base station until it receives the disclosed key.

Therefore, our security goal is straightforward: All data items broadcast by the base station should be authenticated so that any bogus data item inserted by an adversary can be efficiently rejected/filtered. Other desirable properties include energy efficiency (with respect to both communication and computation), low storage overhead, tolerance to out-of-sequence packet delivery, resistance against DoS attacks and no time synchronization between the base station and sensor nodes.

TABLE I
NOTATIONS

Notation	Description
$SIG_k(M)$	the signature on message M with the key k
, or $\|$	concatenation operator of the two bit streams
$H(.)$	public one-way cryptographic hash function (e.g., SHA-1)
$H(M)$	the hash value of message M

IV. SeDrip: The Basic Protocol

Before giving the detailed description of SeDrip, we first provide an overview of SeDrip.

A. Overview of SeDrip

Compared with the traditional approaches, elliptic curve cryptography (ECC) is a better approach to public-key cryptography in terms of key size, computational efficiency, and communication efficiency. However, while ECC is feasible on resource-limited sensor motes, heavily involving ECC-based authentication is still not practical.

SeDrip combines ECC public key algorithm and Merkle hash tree [10] to avoid frequent public key operations and achieve strong robustness against various malicious attacks. Also, SeDrip inherits robustness to packet loss from underlying Trickle algorithm, because Trickle uses periodic retransmissions to ensure eventual delivery of the message to every node in the network.

SeDrip consists of three phases: system initialization, packet pre-processing, and packet verification. The system initialization phase is carried out before network deployment. In this phase, the base station creates its public and private keys, and loads the public parameters on each sensor node. Then, before disseminating data, the base station executes the packet pre-processing phase in which packets and their corresponding Merkle hash tree are constructed from data items. Finally, in the packet verification phase, a node verifies each received packet. If the result is positive, it updates the data according to the received packet. In the following, each phase is described in detail. The notations used in the description are listed in Table I.

B. System Initialization Phase

In this stage, the base station sets up an ECC by deriving a private key SK and public parameters $\{PK, Q, p, q, H(.)\}$ by performing the following operations. It selects an elliptic curve E over $GF(p)$, where p is a big prime number. Here Q denotes the base point of E while q is also a big prime number and represents the order of Q. It then picks the private key $SK \in GF(q)$ and generates the public key $PK = SK \cdot Q$. Afterwards, the public parameters are pre-loaded in each node of the network. As an illustrative example, for 160-bit ECC, both PK and Q are 320 bits long, and both p and q are 160 bits long.

In SeDrip, we extend the 3-tuple $(key, version, data)$ of Drip into a 4-tuple $(round, key, version, data)$ to represent a data item, where $round$ refers to which round of data dissemination this data item belongs to (the higher the round, the newer the data dissemination), and the other three elements bear the same meaning as existing protocols. Same as the Drip implementation, key and $version$ are 2 bytes and 4 bytes long, respectively. For the *round* field, it can be just as short as 4 bits because we can allow a wrap around in the number space to take place. This is possible based on two characteristics of the dissemination process. First, the configuration of a WSN is not expected to change frequently and hence the dissemination rate would be low. Second, only a small amount of data is disseminated in each round, so the time required to complete one round of dissemination should be very short. As a result, each sensor node would not experience any ambiguity in determining which round number is the latest even if there is a wrap around in a round number.

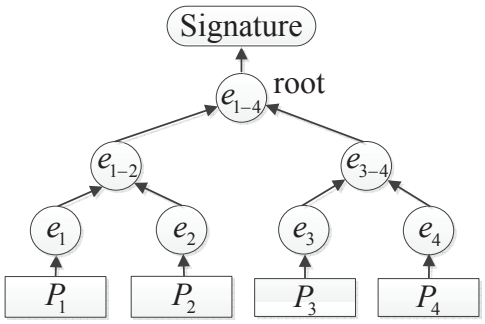


Fig. 1. Example of the Merkle hash tree.

C. Packet Pre-processing Phase

After system initialization, if the base station wants to disseminate n data items: $d_i = \{round, key_i, version_i, data_i\}$, $i = 1, 2, \ldots, n$, it uses the Merkle hash tree [10] method to construct the packets of the respective data as follows.

Merkle hash tree is a tree of hashes, where the leaves in the tree are hashes of the authentic packets P_i, $i = 1, 2, \ldots, n$. Here the hash function is calculated over packet header and data item $d_i (= \{round, key_i, version_i, data_i\})$. Nodes further up in the tree are the hashes of their respective children. Fig. 1 shows the construction of the Merkle hash tree when $n = 4$. More exactly, the base station computes $e_i = H(P_i)(i = 1, 2, 3, 4)$, and builds a binary tree by computing internal nodes from adjacent children nodes. Each internal node of the tree is the hash value of the two children nodes. For example, in Fig. 1, $e_{1-2} = H(e_1 \| e_2)$ and $e_{1-4} = H(e_{1-2} \| e_{3-4})$. Subsequently, the base station constructs n packets based on this Merkle hash tree. For packet P_i, it consists of the packet header, the data item d_i and the values in its authentication path (i.e., the siblings of the nodes in the path from P_i to the root) in the Merkle hash tree. For example, packet P_1 contains the header, d_1, e_2, and e_{3-4} in Fig. 1.

In addition, the base station creates a signature packet P_0 which includes the root of the Merkle hash tree (denoted by $root$) and a signature over the root. The base station assigns a pre-defined key to identify this signature packet. That is, packet P_0 consists of the packet header, the data item $d_0 (= \{round, key, version, root\})$ and the signature $SIG_{SK}(H(d_0))$. For example, in Fig. 1, packet P_0 consists of the packet header and the data

$< round, key, version, e_{1-4}, SIG_{SK}(H(d_0)) >$. The base station first broadcasts the signature packet, which serves as the advertisement of a new round. This root allows each node to immediately verify each received packet P_i $(i = 1, \ldots, n)$ upon its arrival, using the values in the authentication path included in the same packet. For example, in Fig. 1, if e_{1-4} has been authenticated in the signature packet, upon receiving a packet consisting d_1, e_2 and e_{3-4}, a node can immediately verify whether $H(H(H(d_1)\|e_2)\|e_{3-4}) = e_{1-4}$. Therefore, SeDrip enables each node to authenticate and verify the integrity of data packets quickly, even when the packets may arrive out of sequence.

Here we assume that the signature packet in each round of dissemination has been received by each sensor node before the node receives any data packet of the same round. This assumption can be achieved due to the following two reasons. Firstly, no signature packet loss is ensured by Trickle because it reliably disseminates packets to each node. Secondly, in order to satisfy this assumption, the base station needs to keep a long enough time interval between the signature packet and the first data packet sent. Let T_d be the maximum packet propagation delay in the network. So, it would suffice for the base station to set the time interval to be larger than T_d.

D. Packet Verification Phase

Upon receiving a packet (from any one-hop neighboring node or the base station), each sensor node, say S_i, first checks the key field of the packet:

1) If this is a signature packet P_0, node S_i runs the following operations:
   a) If this is a new round (i.e., the $round$ included in this packet is newer than that of its stored $< round, root >$), node S_i uses the public key PK of the base station to run an ECDSA verify operation to authenticate the received signature packet. If this verification passes, node S_i accepts the root of the Merkle hash tree and then updates its stored $< round, root >$ by the corresponding values in packet P_0; otherwise, node S_i simply drops the signature packet P_0.
   b) If node S_i has recently heard an identical signature packet (i.e., the $round$ included in this packet is same as that of its stored $< round, root >$), it increases the broadcast interval of this packet through the Trickle algorithm, thereby limiting energy costs when a network is consistent.
   c) If this is an old round (i.e., the $round$ included in this packet is older than that of its stored $< round, root >$. That is, the signature packet distributed by its one-hop neighboring node is old), node S_i broadcasts its stored signature packet.
2) Otherwise (i.e., it is a data packet which contains some data item), node S_i picks up the tuple $< key, version >$ from this packet and checks $version$.
   a) If this is a new version, node S_i picks the root of the Merkle hash tree from its storage according to the $round$ included in this packet, and then uses the root to verify this packet by performing hash function operation. If yes, node S_i updates the data according to the key of this packet; otherwise, node S_i simply discards this packet.
   b) If node S_i has recently heard an identical data (i.e., the $version$ included in this packet is same as that stored in this node), it increases the broadcast interval of this packet through the Trickle algorithm.
   c) If this is an old version, node S_i broadcasts its stored data packet.

$$K_0 \xleftarrow{H} K_1 \xleftarrow{H} \cdots \xleftarrow{H} K_{b-1} \xleftarrow{H} K_b$$

Fig. 2. One-way key chain for puzzle keys.

V. Further Improvements in Security and Efficiency

In this section, we point out further improvements to enhance SeDrip. However, such improvements require more changes to the basic protocol of SeDrip. For brevity, we just present the parts that need to be changed.

A. Message Specific Puzzle Approach for Security Improvement

Recall that SeDrip uses a digital signature to bootstrap the authentication of each round of disseminated data. By flooding the WSN with a large number of illegal signature packets, an adversary can impose a DoS attack to the nodes because their resources are exhausted to process the illegal signature packets rather than the legitimate ones. To resist this attack, the message specific puzzle approach [1] can be directly applied in SeDrip as follows:

1) System Initialization Phase: As illustrated in Fig. 2, the base station randomly chooses a number K_b, and then generates a one-way key chain consisting of $K_0, K_1, \ldots, K_b$, where $K_j = H(K_{j+1})$ $(j = b-1, b-2, \ldots, 0)$. Then the base station pre-distributes the *key chain commitment* K_0 into each sensor node before deployment. The keys $K_0, K_1, \ldots, K_b$ are referred to as *puzzle keys*, and K_j is used for the j-th round of the disseminated data items, where $j > 0$.

2) Packet Pre-processing Phase: To mitigate the above-mentioned DoS attack, a weak authenticator such as a message specific puzzle can be attached to the signature packet as a defense. With the use of puzzle key K_j in round j, the payload of the signature packet $d_0\|SIG_x(H(d_0))$ and K_j constitute a message specific puzzle. As shown in Fig. 3, a valid solution L_j is a value such that, after applying the hash function $H(.)$ to $d_0\|SIG_x(H(d_0))\|K_j\|L_j$, the first l bits of the resulting image are all "0". The parameter l determines the strength of the puzzle, and the base station fixes the value of l and then loads it into all sensor nodes before deployment. Before transmitting the signature packet, the base station needs to find the puzzle solution L_j first. Subsequently, the base station broadcasts the final signature packet with payload $d_0\|SIG_x(H(d_0))\|K_j\|L_j$.

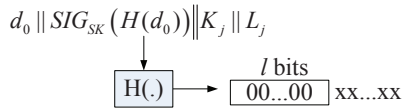


Fig. 3. Message specific puzzle.

3) Packet Verification Phase:

- 1.a) If this is a new round, node S_i first verifies that the puzzle key included in the signature packet P_0 is valid using $H(.)$ and K_0 (or a previously verified puzzle key for efficiency improvement) and that the puzzle key has not been used along with a valid signature packet before. Only if this verification is successful, the node goes to the next step (i.e., ECDSA verification operation); otherwise, node S_i simply discards the packet.

It can be seen from above that, without first solving a message specific puzzle, the adversary cannot produce any forged packet which triggers a node to carry out the signature verification function. The salient feature of this approach is that, due to the one-way property of a hash function, a puzzle can only be solved by exhaustive search, but a puzzle solution can be efficiently verified by a hash function operation and comparison. Therefore, heavy computational burden is imposed on the adversary to deter it from launching the DoS attack. Note that although it requires the same effort from both the base station and an adversary to solve a puzzle, the base station has the advantage that it has no tight time limit to solve the puzzle. To generate a valid signature packet, the base station picks the appropriate puzzle key and solves the puzzle without a deadline as tight as the adversary. On the other hand, the adversary has to solve the puzzle after seeing a puzzle key but before it is invalidated by the arrival of the signature packet at the target nodes. Therefore, with an appropriate puzzle strength, the message specific puzzle method substantially increases the difficulty of launching the DoS attack against signature packets.

Another merit of the usage of message specific puzzle method is as follows: Assume the message specific puzzle method is not used. At step 1.a) of the packet verification phase, when a node, say S_i, receives the signature packet of a new round, it broadcasts the signature packet using the Trickle algorithm after performing signature verification operation. Clearly, this will result in a large propagation delay. More specifically, the propagation delay for all nodes to receive the signature packet is more than $Time_{\text{ver}} \times Num_{\text{hop}}$, where $Time_{ver}$ and Num_{hop} denote the signature verification time on a node and the maximum hop number of the network. To address this problem, a naive solution is to ask a node to directly broadcast the signature packet before the signature verification. Unfortunately, this method opens a door to another DoS attack in which an adversary sends a lot of bogus signature packets to force all nodes to waste energy in broadcasting fake packets. A feasible solution is that once the verification of the puzzle solution in the signature packet passes, node S_i broadcasts the signature packet using the Trickle algorithm. The efficacy of this solution will be demonstrated by the experiment of the propagation delay described in Section VII.B.

B. Adding Salt Number as Input of a Hash Function for Efficiency Improvement

SeDrip uses a Merkle hash tree to allow each packet to be immediately authenticated upon receipt. However, this method adds additional overhead due to the transmission of part of a Merkle hash tree for each data item. For a round of n data items, the tree has height $f = log_2(n)$, and the base station needs to include f hash values in each packet. Despite the logarithmic tree height, this is still too large for most WSNs. For example, when $n = 32$ and SHA-1 (with 20-byte output) is used, the overhead due to the authentication path is $5 \times 20 = 100$ bytes. This is far too much for most WSNs, because for sensors with IEEE 802.15.4 compliant radios, the maximum payload size is 102 bytes. Moreover, energy is an extremely scarce resource on sensor nodes while radio communication consumes the most amount of energy, thus an appropriate approach should be provided to tackle this problem.

In order to reduce the transmission overhead, often it is suggested that only the hash function with truncated output (say, 4, 6, or 8 bytes) is used to reduce the hash overhead [11]. However, the suggested size of 4 bytes is too small to defeat DoS attacks, because an adversary may use a pre-built dictionary of 2^{32} packets covering all possible 4-byte hash values. The required storage for the dictionary is 192 GB for 48-byte packets, which is readily available on PCs. A well-known mechanism to resist such off-line dictionary attacks without increasing the hash size is as follows.

1) Packet Pre-processing Phase: The base station chooses a random 8-byte number $salt$ (called salt number) for each round of data dissemination, and add $salt$ into the data d_0 of the signature packet, that is, $d_0 = \{round, key, version, root, salt\}$. Similarly, instead of $\{round, key, version, root\}$, $\{round, key, version, root, salt\}$ will be the input of the ECC signature generating operation. Also, for the construction of the leaves of Merkle hash tree, the base station uses $salt$ as part of the input in the hash operation, i.e., $e_i = H(P_i \| salt)$.

2) Packet Verification Phase:

- 1.a) After node S_i ensures that a signature packet P_0 is authentic, the node updates its stored $< round, salt, root >$ by the corresponding values in packet P_0.
- 2.a) Node S_i picks $< round, salt, root >$ from its storage according to the $round$ included in this packet, and then uses the root and salt number to verify this packet by performing hash function operation.

VI. Security and Efficiency Analysis

A. Security Analysis

Integrity of Data Items: In SeDrip, the base station is trustworthy. It signs the root of the Merkle hash tree constructed from all data items with the private key, which is obviously only known to itself. Since all the sensor nodes know the public key of the base station and it is assumed that the adversary cannot compromise the base station, it is certain that the sensor nodes can authenticate the signature packet and the root of the Merkle hash tree carried by the packet. This

means that all the nodes can authenticate the data packets upon receiving them, based on the security of Merkle hash tree.

Resistance to DoS Attacks: As discussed above, there are three types of DoS attacks against basic SeDrip: (1) DoS attacks exploiting authentication delays, (2) DoS attacks exploiting the expensive signature verifications, and (3) DoS attacks exploiting the Trickle algorithm.

SeDrip is resistant to all three types of DoS attacks. Due to the use of Merkle hash tree, upon receiving a data packet, each node can immediately authenticate it through performing a few efficient hash function operations, and successfully defeat DoS attacks exploiting authentication delays. Further, because of the use of message specific puzzles, each node can efficiently detect fake signature packets. Thus, SeDrip provides resistance to DoS attacks that send fake signature packets. Finally, SeDrip can ensure the authentication of the disseminated data items on each node. As a result, an external attacker cannot convince regular nodes to misuse the Trickle mechanism.

SeDrip can successfully defeat all three types of DoS attacks even if there are compromised nodes. Indeed, without the private key and the unreleased puzzle keys on the base station, even an inside attacker cannot forge any signature/data packets.

B. Efficiency Analysis

Only if the verification of the puzzle key included in the signature packet is successful, each node executes the signature verification operation. Also, only one signature packet is sent in each round of data dissemination no matter how many data packets are disseminated. Thus, the computation complexity of the proposed system on each node only involves one ECC public key verification operation on a small size value ($H(d_0)$) in each round, independent of how many data packets are disseminated in a round or how big the sizes of data packets are. Here, the computational complexity of a hash function is omitted since this operation is very fast on sensor nodes. Thus, our protocol incurs very little overhead of computing. Also, a node only needs to save a 2-tuple $< round, root >$ in RAM to verify the integrity of each data packet. So, the protocol is memory-efficient as well.

VII. Implementation and Performance Evaluation

We evaluate SeDrip by implementing all its components in an experimental test-bed. Also, we choose Drip for performance comparison.

A. Implementation and Experimental Setup

Our implementation has the base station and sensor node side programs. The base station side programs are C programs using OpenSSL [12] and running on laptop PCs (with 2 GB RAM and the Ubuntu 11.04 operating system) with different processor speeds. Also, the sensor node side programs are written in nesC and run on resource-limited motes (MicaZ and TelosB). The MicaZ mote has an 8-bit 8-MHz Atmel microcontroller, 128-kB ROM and 4-kB RAM. Also, the TelosB mote has an 8-MHz CPU, 10-kB RAM, 48-kB ROM, 1MB of flash memory, and an 802.15.4/ZigBee radio. Our motes run TinyOS [6] 2.x. Additionally, the key size of ECC is set to 160 bits and SHA-1 is used. Throughout this paper, unless otherwise stated, all experiments on PCs (resp., sensor nodes) were repeated one million times (resp., one thousand times) for each measurement in order to obtain accurate average results.

To implement SeDrip, we add the following functionalities in the C tools on the base station side: construction of Merkle hash tree of all data items, generation of the signature packet (from the root of the Merkle hash tree) and all data packets (attached with the authentication path of the Merkle tree). To obtain $version$ of each data item, we modify *DisseminatorC* and *DisseminatorP* modules of Drip nesC library to provide an interface named *DisseminatorVersion*. Also, we have implemented two versions of SeDrip, i.e., SeDrip1 (forwarding the signature packet when the signature verification passes), and SeDrip2 (forwarding the signature packet when the message specific puzzle verification passes). In the implementation of Drip, when a packet with a new version number is received, a node first stores the packet and then resets the Trickle timer. For the two versions of SeDrip, we modify the above procedure as follows. In SeDrip1, when a signature/data packet with a new version number is received, a node stores the packet and then authenticates it. If the result is positive, the node resets the Trickle timer; otherwise, the node simply discards the packet. In SeDrip2, when a signature packet with a new version number is received, a node first checks the puzzle solution in the packet. If the result is positive, the node broadcasts the signature packet to its one-hop neighboring nodes, and then stores the packet and resets the Trickle timer; otherwise, the node simply discards it.

Following the design of SeDrip described above, we employ the *ECDSA verify* and *SHA-1 hash* operations of TinyECC 2.0 library [13] to add the verification function of signature packet and data packets into the Drip nesC library. In our implementation, the base station (i.e., a laptop PC) first sends the signature and data packets through a serial port to a particular sensor node which is referred to as *retransmitter*. Then, the retransmitter disseminates these packets by Drip or SeDrip on behalf of the base station.

Similar to [14], we have built a circuit as shown in Fig. 4 to measure the power consumption in resource-limited sensor nodes when they execute various cryptographic operations considered in this paper. The Tektronix TDS 3034C digital oscilloscope is used to accurately measure the voltage V_r across the 20.36 Ω resistor. Given that the voltage of the batteries is V_b, the voltage V_m across the mote is given by $V_b - V_r$. In our experiments, V_b=3.0 V. The current I through the circuit can simply be obtained by applying Ohm's law to the resistor. The power P consumed by the sensor node is then given by $V_m \times I$. Also, we have measured the execution times of the operations. Finally, the energy consumption of each operation is given by the product of power and execution time.

B. Evaluation Results

We use the following four metrics to evaluate SeDrip, namely, execution time, memory overhead, propagation delay,

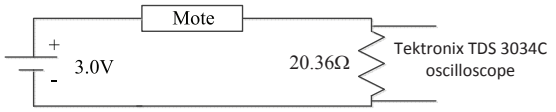


Fig. 4. Experimental setup for investigating the energy consumption.

TABLE II
RUNNING TIME FOR EACH PHASE OF THE BASIC PROTOCOL OF SEDRIP (EXCEPT THE SENSOR NODE VERIFICATION PHASE).

	System Initialization	Signing an 8-byte data	Signing a 20-byte data
Time (1.6 GHz) (μs)	1805.0	707.3	704.6
Time (2 GHz) (μs)	1446.4	565.0	565.2
Time (2.4 GHz) (μs)	1205.6	473.2	475.6
Time (3.1 GHz) (μs)	931.1	368.4	372.3

and energy overhead. The execution time measures the time duration for each operation of SeDrip. The memory overhead measures the exact amount of data space required in the real implementation. Also, the propagation delay is the time required to finish disseminating a round of data items to all the nodes in the network.

Table II gives the execution times of some major operations in the basic protocol of SeDrip. For example, the execution times for the system initialization phase and signing a random 20-byte message (i.e., the output of SHA-1 function) are 1.446 ms and 0.565 ms on a 2-GHz Laptop PC, respectively. Fig. 5 shows the execution times of SHA-1 hash function (extracted from TinyECC 2.0 [13]) on MicaZ and TelosB motes. The inputs to the hash function are randomly generated numbers with length varying from 20 bytes to 140 bytes in increments of 2 bytes. We perform the same experiment ten thousand times and take an average over them. For example, the execution times on a MicaZ mote for inputs of 54 bytes, 56 bytes, 118 bytes, and 120 bytes are 9.6788 ms, 18.4678 ms, 18.9767 ms, and 27.7641 ms, respectively. Also, the execution times on a TelosB mote for inputs of 54 bytes, 56 bytes, and 120 bytes are 5.7263 ms, 10.3161 ms, and 15.3941 ms, respectively. From Fig. 5, it can be seen that the execution time remains very stable when the byte length of the input falls in the interval $[0, 55]$, $[56, 119]$, or $[120, 140]$. Thus, it is suggested that in the application of SeDrip, the length of the input of a hash function should be chosen according to the above intervals to achieve low computation complexity on sensor nodes.

To compare with the efficiency of public key cryptography, we have implemented the 160-bit ECC algorithm of TinyECC 2.0 library [13] on MicaZ and TelosB motes, it is measured that the signature verification (with a random 20-byte number as the output) times are 2.436 seconds and 3.968 seconds, which are 131 and 692 times longer than SHA-1 hash operation with a 56-byte random number as input on MicaZ and TelosB motes, respectively. It is demonstrated that packet authentication based on the use of Merkle hash tree is much more efficient than that based on verifying the digital signature of each packet. Thus, SeDrip is suitable for resource-limited sensor nodes.

Next, the energy consumption of ECC verification and SHA-1 hash function operations are investigated (when the

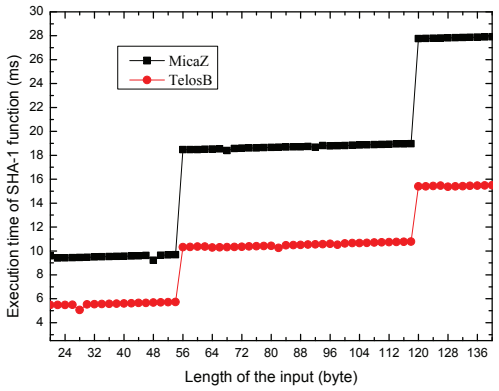


Fig. 5. The execution times of SHA-1 hash function on MicaZ and TelosB motes.

TABLE III
THE EXECUTION TIMES FOR SOLVING MESSAGE SPECIFIC PUZZLE IN SEDRIP.

Value of l	20	22	24	26
Time (CPU = 1.6 GHz) (ms)	489	1882	7919	32825
Time (CPU = 2.4 GHz) (ms)	318	1283	4945	20759
Time (CPU = 3.1 GHz) (ms)	246	1000	3986	16028

radio of the mote is off). When a MicaZ mote is used in the circuit, $V_r = 138$ mV, $I = 6.7779$ mA, $V_m = 2.8620$ V, $P = 19.3983$ mW. When a TelosB mote is used, $V_r = 38$ mV, $I = 1.8664$ mA, $V_m = 2.9620$ V, $P = 5.5283$ mW. During the whole procedure of the small data dissemination, each sensor node's radio is always on. That is, the RF transceiver of each sensor node is either in receive/listen mode or transmit mode. From the TelosB datasheet, the current is 23 mA when the node is in receive/listen mode while the current is 17.4 mA when the node is in transmit mode (this measurement is obtained when the voltage across the mote is 3 V and the mote does not carry out any computations' operations). Thus, the powers consumed by each TelosB mote are 69 mW and 52.2 mW when a TelosB mote is in receive/listen mode and in transmit mode, respectively. Hence, the impact on energy consumption from cryptographic computation operation is very low. By multiplying the power with the execution time obtained from Fig. 5, we can determine the total energy consumption of SHA-1 operation on the motes. For example, the energy consumption of SHA-1 operation with a random 56-byte number as input on MicaZ and TelosB motes are 0.3582 mJ and 0.0317 mJ, respectively. Also, the energy consumption of ECC signature verification operation on MicaZ and TelosB motes are 2835.2555 mJ and 1316.1777 mJ, respectively.

Table III gives the time required to solve the message-specific puzzles in OpenSSL implementation on laptop PCs. In each experiment, the parameters d_0, $SIG_x(H(d_0))$ and K_j are randomly generated for five thousand times, where the byte lengths of these three parameters are 27, 40, and 20, respectively. Also, the length of L_j is set to 4 bytes. For example, the time required to solve a puzzle on a 2.4-GHz laptop PC is 4.945 seconds when the parameter l is set to 24. Since, as described in Section V.A, the puzzle imposes

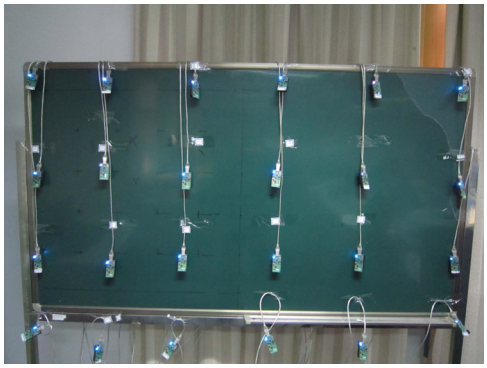


Fig. 6. The testbed (24 TelosB motes)

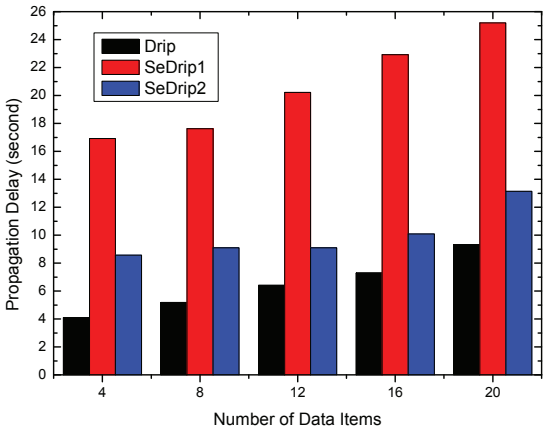


Fig. 7. Propagation delay comparison of three protocols.

a very tight time constraint for adversaries to obtain the puzzle solution, this time consumption is enough to defeat DoS attacks.

To further investigate the impact of security protection mechanisms on the propagation delay of dissemination in multi-hop networks, we perform experiments in an indoor testbed (as shown in Fig. 6) comprising of 24 TelosB motes in a 4×6 grid. The inter-node spacing is about 35 cm and the transmission power of each node is configured to the lowest power level in order to simulate the multi-hop behavior. The retransmitter is performed by a TelosB node which is located at the vertex of the grid.

In this experiment, the base station disseminates data items to the TelosB motes in the grid. The packet delivery rate at the base station is 5 packets/s. Fig. 7 shows the propagation delays of Drip, SeDrip1, and SeDrip2 measured from the experiment. The propagation delay of SeDrip is the time from construction of Merkle hash tree, until the corresponding variables on all sensor nodes are updated. In our implementation, the lengths of $round$ and $data$ in each data item are 4 bits and 2 bytes, respectively. In the construction of a Merkle hash tree, only the hash function with 8-byte truncated output is used. The byte length of ECC-160 signature is 40. In order to obtain accurate average results, for each experiment, we have executed each dissemination operation for 20 times and taken an average over them. As the number of data items increases, the propagation delays of all protocols increase almost linearly. For example, when the number of data items is 12 (resp. 20), the propagation delays of Drip, SeDrip1, and SeDrip2 are 6.413 seconds (resp. 9.334 seconds), 20.218 seconds (resp. 25.195 seconds), and 9.096 seconds (resp. 13.133 seconds), respectively. Comparing the propagation delays of Drip and SeDrip2, it is concluded that the signature verification by sensor nodes in SeDrip2 has low impact on the propagation delay of dissemination. It is clear that the increase in propagation delay is mainly due to the ECC signature verification operations on the nodes. This is because, upon receiving a new signature packet, each sensor node broadcasts the signature packet to its one-hop neighboring nodes after the message specific puzzle verification passes. Subsequently, it starts to check the validity of the signature packet by performing signature verification operation. Only the signature verification time of the one-hop neighboring nodes of the base station has impact on the propagation delay. Furthermore, the increased propagation delay due to signature verification is constant, which is independent of the number of the disseminated data items of each round. Statistically, SeDrip2 is just about 4 seconds more than that of Drip.

Table IV shows the ROM and RAM usage of SeDrip (with 4 data items) on MicaZ and TelosB motes. The code sizes of Drip (with 4 data items) and the verification function of TinyECC (a set of implementations in various elliptic curve domains according to the Standards for Efficient Cryptography Group, e.g., secp128r1, secp160r1 and secp192r1) are also included for reference purposes. For example, the implementation of SeDrip (with secp160r1) on a TelosB mote uses 2,637 bytes of RAM and 28,050 bytes of ROM, respectively. The resulting size of our implementation corresponds to 25.75% and 57.07% of the RAM and ROM capacities of TelosB, respectively.

VIII. Conclusion

In this paper, we have identified the security vulnerabilities in data discovery and dissemination of WSNs. We then developed a lightweight protocol named SeDrip to allow efficient authentication of the disseminated data items by taking advantage of efficient Merkle tree algorithm. SeDrip is designed to work within the computation, memory and energy limits of inexpensive sensor motes. In addition to analyzing the security of SeDrip, this paper has also reported the evaluation results of SeDrip in an experimental network of resource-limited sensor nodes, which show that SeDrip is efficient and feasible in practice.

References

[1] D. He, C. Chen, S. Chan, and J. Bu, "DiCode: DoS-resistant and distributed code dissemination in wireless sensor networks," *IEEE Trans. Wireless Commun.*, vol. 11, no. 5, pp. 1946–1956, May 2012.

[2] G. Tolle and D. Culler, "Design of an application-cooperative management system for wireless sensor networks," in *Proc. EWSN*, pp. 121–132, 2005.

[3] K. Lin and P. Levis, "Data discovery and dissemination with DIP," in *Proc. 2008 ACM/IEEE IPSN*, pp. 433–444.

[4] T. Dang, N. Bulusu, W. Feng, and S. Park, "DHV: a code consistency maintenance protocol for multi-hop wireless sensor networks," in *Proc. 2009 EWSN*, pp. 327–342.

[5] R. Panta, M. Vintila, and S. Bagchi, "Fixed cost maintenance for information dissemination in wireless sensor networks," in *Proc. 2010 IEEE SRDS*, pp. 54–63.

TABLE IV
CODE SIZES (BYTES) ON MICAZ AND TELOSB MOTES.

		Drip	SeDrip			TinyECC in SeDrip		
			secp128r1	secp160r1	secp192r1	secp128r1	secp160r1	secp192r1
MicaZ	ROM	14,756	31,930	32,834	32,232	16,986	17,950	17,370
	RAM	423	2,212	2,512	2,812	1,370	1,670	1,970
TelosB	ROM	14,808	28,002	28,050	28,096	13,908	13,992	14,062
	RAM	453	2,337	2,637	2,937	1,467	1,767	2,067

[6] "TinyOS: an open-source OS for the networked sensor regime." Available: http://www.tinyos.net/

[7] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: a self-regulating algorithm for code maintenance and propagation in wireless sensor networks," in *Proc. 2004 NSDI*, pp. 15–28.

[8] A. Perrig, R. Canetti, D. Song, and J. Tygar, "Effcient and secure source authentication for multicast," in *Proc. 2001 NDSS*, pp. 35–46.

[9] Y. Chen, I. Lin, C. Lei, and Y. Liao, "Broadcast authentication in sensor networks using compressed bloom filters," in *Proc. 2008 IEEE DCOSS*, pp. 99–111.

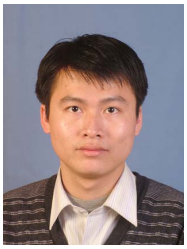
[10] R. Merkle, "Protocols for public key cryptosystems," in *Proc. 1980 IEEE S&P*, pp. 122–134.

[11] S. Hyun, P. Ning, A. Liu, and W. Du, "Seluge: secure and DoS-resistant code dissemination in wireless sensor networks," in *Proc. 2008 ACM/IEEE IPSN*, pp. 445–456.

[12] "OpenSSL." Available: http://www.openssl.org

[13] A. Liu and P. Ning, "TinyECC: a configurable library for elliptic curve cryptography in wireless sensor networks," in *Proc. 2008 ACM/IEEE IPSN*, pp. 245–256.

[14] J. Lee, K. Kapitanova, and S. Son, "The price of security in wireless sensor networks," *Comput. Netw.*, vol. 54, no. 17, pp. 2967–2978, Dec. 2010.



Daojing He is currently an Associate Professor, South China University of Technology, China. He received the B.Eng.(2007) and M. Eng. (2009) degrees from Harbin Institute of Technology (China) and the Ph.D. degree (2012) from Zhejiang University (China), all in Computer Science. His research interests include network and systems security. He is an associate editor or on the editorial board of some international journals such as Springer *Journal of Wireless Networks*, Wiley's *Wireless Communications and Mobile Computing* Journal, Wiley's *Security and Communication Networks* Journal, and KSII *Transactions on Internet and Information Systems*. He has been serving as a TPC member for leading conferences including IEEE WCNC, GLOBECOM, and ICC.

Sammy Chan received his B.E. and M.Eng.Sc. degrees in electrical engineering from the University of Melbourne, Australia, in 1988 and 1990, respectively, and a Ph.D. degree in communication engineering from the Royal Melbourne Institute of Technology, Australia, in 1995. From 1989 to 1994, he was with Telecom Australia Research Laboratories, first as a research engineer, and between 1992 and 1994 as a senior research engineer and project leader. Since December 1994, he has been with the Department of Electronic Engineering, City University of Hong Kong, where he is currently an associate professor.

Shaohua Tang received the B.Sc. and M.Sc. degrees in applied mathematics from South China University of Technology, China, in 1991 and 1994, respectively, and the Ph.D. degree in communication and information system from South China University of Technology, in 1998. He was a visiting scholar with North Carolina State University, USA, 2001-2002, and a visiting professor with University of Cincinnati, USA, 2009-2010. He has been a full professor with the School of Computer Science and Engineering, South China University of Technology since 2004. His current research interests are in information security and networking. He is a member of the IEEE and the IEEE Computer Society.

Mohsen Guizani (S'85-M'89-SM'99-F'09) is currently a Professor and the Associate Vice President for Graduate Studies at Qatar University, Doha, Qatar. He was the Chair of the Computer Science Department at Western Michigan University from 2002 to 2006 and Chair of the Computer Science Department at University of West Florida from 1999 to 2002. He also served in academic positions at the University of Missouri-Kansas City, University of Colorado-Boulder, Syracuse University and Kuwait University. He received his B.S. (with distinction) and M.S. degrees in Electrical Engineering; M.S. and Ph.D. degrees in Computer Engineering in 1984, 1986, 1987, and 1990, respectively, from Syracuse University, Syracuse, New York. His research interests include Computer Networks, Wireless Communications and Mobile Computing, and Optical Networking. He currently serves on the editorial boards of six technical Journals and the Founder and EIC of *Wireless Communications and Mobile Computing* Journal published by John Wiley (http://www.interscience.wiley.com/jpages/1530-8669/). He is the author of eight books and more than 300 publications in refereed journals and conferences. He guest edited a number of special issues in IEEE Journals and Magazines. He also served as member, Chair, and General Chair of a number of conferences. Dr. Guizani served as the Chair of IEEE Communications Society Wireless Technical Committee (WTC) and Chair of TAOS Technical Committee. He was an IEEE Computer Society Distinguished Lecturer from 2003 to 2005. Dr. Guizani is an IEEE Fellow and a Senior member of ACM.