DiCode: DoS-Resistant and Distributed Code Dissemination in Wireless Sensor Networks

Daojing He, *Student Member, IEEE*, Chun Chen, *Member, IEEE*,
Sammy Chan, *Member, IEEE*, and Jiajun Bu, *Member, IEEE*

***Abstract*—Code dissemination in a wireless sensor network (WSN) is the process of propagating a new program image or relevant commands to sensor nodes. As a WSN is usually deployed in hostile environments, secure code dissemination is and will continue to be a major concern. Most code dissemination protocols are based on the centralized approach in which only the base station has the authority to initiate code dissemination. However, it is desirable and sometimes necessary to disseminate code images in a distributed manner which allows multiple authorized network users to simultaneously and directly update code images on different nodes without involving the base station. Motivated by this consideration, we develop a secure and distributed code dissemination protocol named *DiCode*. A salient feature of DiCode is its ability to resist denial-of-service attacks which have severe consequences on network availability. Further, the security properties of our protocol are demonstrated by theoretical analysis. To verify the efficiency of the proposed approach in practice, we also implement the proposed mechanism in a network of resource-constrained sensor nodes.**

***Index Terms*—Sensor networks, code dissemination, security, denial-of-service, user privilege.**

I. Introduction

CODE dissemination is the process of propagating a new program image[1] or relevant commands to sensor nodes through wireless links after a wireless sensor network (WSN) is deployed. Due to the need of removing bugs and adding new functionalities, code dissemination is an important operation function of WSNs. As a WSN is usually deployed in hostile environments such as the battlefield, an adversary may exploit the code dissemination mechanism to launch various attacks. For example, the adversary may inject bogus code images to take over the control of the whole WSN. Thus, secure code dissemination is and will continue to be a major concern.

Several code dissemination protocols have been proposed to propagate new code images in WSNs (e.g., [1]–[4]). Among these protocols, Deluge [2] is included in the TinyOS distributions [5]. However, since the design of Deluge did not take security into consideration, there have been several extensions to Deluge to provide security protection for code dissemination [6]–[12]. Among them, Seluge [12] enjoys both strong security and high efficiency.

However, all these code dissemination protocols ([2]–[4], [6]–[12]) are based on the centralized approach which assumes the existence of a base station and only the base station has the authority to reprogram sensor nodes. As shown in Fig. 1(a), when the base station wants to disseminate a new code image, it broadcasts the signed code image and each sensor node only accepts code images signed by it. Unfortunately, there are WSNs having no base station at all. Examples of such networks include a military WSN in a battlefield to monitor enemy activity (e.g., troop movements), a WSN deployed along an international border to monitor weapons smuggling or human trafficking, and a WSN situated in a remote area of a national park monitoring illegal activities (e.g., firearm discharge, illicit crop cultivation). Having a base station in these WSNs introduces a single point of failure and a very attractive attack target. Obviously, the centralized approach is not applicable to such WSNs. Also, the centralized approach is inefficient, weakly scalable (i.e., inefficient for supporting a large number of sensor nodes and users), and vulnerable to some potential attacks along the long communication path.

Alternatively, a distributed approach can be employed for code dissemination in WSNs. It allows multiple authorized network users to simultaneously and directly update code images on different nodes without involving the base station. Another advantage of distributed code dissemination is that different authorized users may be assigned different privileges of reprogramming sensor nodes. This is especially important in large scale WSNs owned by an owner and used by different users from both public and private sectors [13], [14]. Distributed service protocol in WSNs (e.g., decentralized sensing [15]) is a research field that is getting increasingly more attention. Very recently, an identity-based signature scheme to achieve secure and distributed code dissemination is proposed [16]. In this paper, we further extend this scheme in three important aspects. Firstly, we consider denial-of-service (DoS) attacks on code dissemination, which have severe consequences on network availability, as well as propose and implement two approaches to defeat DoS attacks. Secondly, the proposed code dissemination protocol is based on a secure and efficient proxy signature by warrant (PSW) technique,

Manuscript received October 15, 2011; revised December 31, 2011; accepted February 5, 2012. The associate editor coordinating the review of this paper and approving it for publication was Z. Han.

This work was supported by Scholarship Award for Excellent Doctoral Student granted by Ministry of Education, National Science Foundation of China (Grant No. 61070155), the Program for New Century Excellent Talents in University (NCET-09-0685), and a grant from the Research Grants Council of the Hong Kong SAR, China [Project No. City U 111208].

D. He, C. Chen, and J. Bu are with the National Engineering Research Center for Intelligent Train, College of Computer Science, Zhejiang University, 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).

Digital Object Identifier 10.1109/TWC.2012.030812.111857

[1]Note that "program image" and "code image" will be used interchangeably throughout this paper.

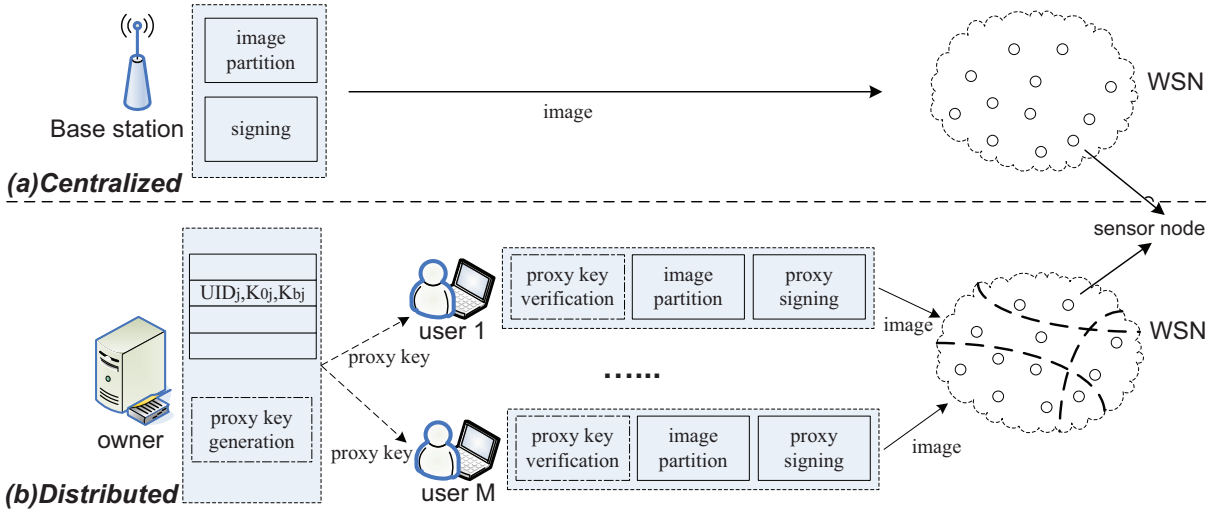

Fig. 1. A system overview of (a) centralized and (b) distributed reprogramming approaches.

which makes it stronger than the scheme of [16]. Thirdly, we consider how to avoid reprogramming conflict and support dynamic participation.

Similar to the centralized code dissemination protocols, a secure distributed code dissemination protocol should satisfy the following requirements: (1) **Integrity of Code Images**: The source of a program image must be verified by a sensor node prior to installation, ensuring that only a trusted source can install a program. In addition, it must be possible to ensure that a program has not been altered during its transmission. (2) **Freshness**: An earlier version of a program image cannot be installed over the program with the same or greater version number, ensuring a node always installs the newest version of a program image. (3) **DoS Attacks Resistance**: A practical code dissemination mechanism should maintain service availability even in the presence of DoS attacks [17]. (4) **Node Compromise Tolerance**: A compromised node must be prevented from causing an uncompromised node to violate the above security requirements.

Other than meeting the above requirements, a distributed code dissemination protocol should also have the following properties. (5) **Distributed**: Multiple authorized network users are able to update code images on different nodes simultaneously without involving the base station. At the same time, the protocol should prevent unauthorized users from updating sensor nodes. (6) **Supporting Different User Privileges**: To ensure smooth functioning for a WSN, the level of each user privilege should be limited by the network owner. For example, a user is only allowed to reprogram the nodes set with specified identities or/and within a particular localized area during his/her subscription period. (7) **Partial Reprogram Capability**: To prevent sensor nodes from being totally controlled by network users, the special modules (e.g., authentication module for each new program image) on each node cannot be overwritten by anyone except the network owner. (8) **Avoiding Reprogramming Conflicts**: Multiple users may be allowed to reprogram a node, which may result in reprogramming conflicts. Some mechanisms should be provided to avoid such conflicts. (9) **User Traceability**: In most application scenarios, traceability is highly desirable, particularly for code dissemination, where it is used for collecting the users' activities for some purposes. (10) **Scalability**: Firstly, the protocol needs to be efficient even in a large-scale WSN with thousands of sensor nodes, and secondly, the protocol should be able to support a large number of users. (11) **Dynamic Participation**: New nodes can be supplemented whenever they are needed, and the scheme should allow dynamic addition of new users.

To satisfy the above requirements, we propose in this paper a practical secure and distributed code dissemination protocol named *DiCode*, which is built on the PSW technique. Since the PSW was not originally designed for distributed code dissemination, a direct application of the method cannot satisfy requirements (3), (7), (8) and (11), which are very challenging for ensuring feasibility, security and efficiency for distributed code dissemination. To address these issues, some additional mechanisms are incorporated into the design of DiCode to enable it to achieve all requirements of distributed code dissemination listed above, while keeping the merits of both Deluge and Seluge.

We also implement the proposed protocol in a network of MicaZ motes. Experimental results show its efficiency in practice. This is also the first implemented DoS-resistant and distributed code dissemination protocol for WSNs.

The rest of the paper is organized as follows. The network, trust and threat models are summarized in Section II. DiCode is presented in Section III in detail. Some approaches to defeat DoS attacks are suggested in Section IV. Section V provides theoretical analysis of the security of DiCode. Then in Section VI, some important issues about DiCode are discussed. Section VII describes the implementation and experimental evaluation of DiCode in a network of MicaZ motes. Section VIII concludes this paper.

II. Network, Trust and Threat Models

A. Network Model

As shown in Fig. 1(b), a WSN consists of a large number of resource-constrained sensor nodes, many sensor network

users and a single network owner. The network users use mobile devices such as PDAs or laptop PCs to reprogram the nodes. The network owner can be off-line, who has bootstrapped the keying materials for the mobile devices to enforce reprogramming privilege policy. It is assumed that the owner cannot be compromised and has unlimited computational power compared with sensor nodes. The nodes can perform a limited number of asymmetric cryptographic operations including signature verification in RSA, but they cannot afford to perform many such operations due to their large energy consumption. Note that all existing secure code dissemination techniques [6]- [12] are based on public key cryptography such as RSA. Also, many novel security techniques (e.g., [18]) based on public key cryptography have been proposed for sensor networks. We assume Deluge as the underlying code dissemination protocol. We also assume sensor nodes are able to establish pair-wise keys between neighbor nodes, for example, using the scheme of [19]. To enable each node to check whether the subscription period of each authorize user has expired, we assume there is a loose time synchronization among the nodes with the help of some existing secure time synchronization scheme [20].

B. Trust Model

The network owner delegates his/her code dissemination privilege to the network users who are willing to register. We assume the special modules (e.g., authentication module for each new program image proposed in this paper, the user access log module) reside in the bootloader section of the program flash on each sensor node which cannot be overwritten by anyone except the network owner. To achieve this goal, some existing approaches can be employed such as hardware-based approaches (e.g., security chips) and software-based approaches (e.g., program code analysis [21]).

C. Threat Model

We assume that an adversary can launch both outsider and insider attacks. In outsider attacks, the adversary does not control any valid nodes in the WSN. The adversary may eavesdrop, copy or replay the transmitted messages in the WSN. He/she may also inject false messages or forge non-existing links in the network by launching a wormhole attack (e.g., [22]). With insider attacks, the adversary can compromise some users (or sensor nodes) and then inject forged code dissemination packets, or exploit specific weakness of the secure protocol architecture.

Additionally, the adversary may launch DoS attacks. The adversary may jam the communication channel; however, we assume that the adversary cannot constantly jam the communication channel without being detected and removed. As described in Section II.B, an authorized user cannot totally control a node. However, the user may load malicious program on some nodes. DiCode can provide user traceability, which will be described in Section V. That is, a node can inform the owner by delivering the identity of such a malicious user.

III. DiCode

Before giving the detailed description of DiCode, we first discuss what kind of cryptographic techniques are suitable for distributed code dissemination.

A. Cryptographic Techniques for Distributed Code Dissemination

Fig. 1(b) shows that a distributed code dissemination protocol consists of three kinds of participants, the network owner, authorized network users and all sensor nodes. After the users register to the owner, they can enter to the WSN and then have pre-defined privileges to simultaneously and directly reprogram the sensor nodes without involving the owner.

A naive solution is to pre-quip each node with multiple public key/reprogramming privilege pairs, each of which corresponds to one authorized user. This scheme allows a user to sign a program image with his/her private key such that each node can verify if the program image originates from an authorized user. However, resource constraints on sensor nodes often make it undesirable to implement such an expensive algorithm. For example, in RSA-1024 public key cryptosystem, the length of each public key is more than 1026 bits. Assuming that the length of reprogramming privilege (including the valid periods of delegation and the nodes set with specified identities or/and within a specific region) is 48 bytes, the length of each public key/reprogramming privilege pair is more than 176 bytes. This means that not too many public key/reprogramming privilege pairs can be stored in a node. In this case, not too many users can be supported. Moreover, it is clear that the network owner has no ability to pre-define the reprogramming privileges of new users who will join after the WSN deployment. Once a new user registers to the network owner, the owner needs to sign a new public key/reprogramming privilege pair and then broadcasts it to all nodes. A more suitable approach is for each authorized user to send a new program image to the nodes through a standard group signature technique [23]. A group signature scheme allows one member of the group to sign a message such that any verifier can verify that the message originated from a group member. Thus, only the group public key is pre-loaded onto each sensor node. Unfortunately, a group signature algorithm does not allow the network owner to specify different reprogramming privileges for different users.

B. Overview of DiCode

In this paper, PSW is introduced into the design of DiCode. This technique involves two kinds of participants, an original signer and proxy signers. The original signer gives the proxy signer a warrant, which specifies the identity of the proxy signer, the identity of the original signer, the range of messages to sign, the expiration time of the delegation of signing power, etc. The proxy signer generates proxy signatures only with the proxy signature key given by the original signer. Verifiers validate proxy signatures only with the public key of the original signer and pay attention to the legality of the warrant. The detailed information about applying the PSW technique into DiCode is as follows. The network owner plays the role of original signer while the network users play the role of

proxy signers. Through registration, the users obtain one or more proxy signature keys from the network owner before they enter a WSN. The key can subsequently be used to make signature on a new code image sent to the sensor nodes. Thus, authorized users generate valid code dissemination packets only with the proxy signature keys given by the network owner. The validity of each code dissemination packet can be verified by any sensor node with the public key of the network owner. In this way, the network owner can prevent unauthorized program updates on sensor nodes and only the public key of the network owner is pre-loaded on each node. However, we notice that a direct application of the PSW algorithms is still unable to meet requirements (3), (7), (8) and (11) of a distributed code dissemination protocol. To address these issues, some additional mechanisms are incorporated into the design of DiCode.

Dozens of PSW schemes have been proposed since 1996. However, some proposed PSW schemes are questionable in the security assurance. Moreover, most of the reported PSW schemes (e.g., [24]) take very long time to verify a signature, which is critical for resource-limited WSNs. Therefore, after a thorough evaluation, we choose the PSW scheme that was introduced by Shao [25], which is secure and considered to be best suited to the WSN application. However, as described above, any other secure and effective PSW technique can be applied easily in our protocol.

Referring to Fig. 1(b), DiCode consists of three phases, system initialization, user pre-processing, and sensor node verification. For our basic protocol, in system initialization phase, the network owner as the original signer creates its public and private keys and then delivers one or more proxy signature keys to the authorized users. Only the network owner's public key is loaded on each node before deployment. In user pre-processing phase, if a user enters to the WSN and has a new program image, he/she will need to construct the code dissemination packets and then send them to the nodes. In sensor node verification phase, if the packets verification passes then the nodes accept the program image. Based on the basic protocol, our improved protocol goes one step further by providing the functionalities of limiting the number of reprogramming times of each authorized user and more efficient DoS attack resistance, which are demanded by some WSN application scenarios. The detailed description of each phase is as follows.

C. The Basic Protocol

1) ***System Initialization****:* In this phase, the network owner executes the following steps:

a) Randomly pick two large safe primes p and q, and compute a public modulus $n = pq$. Then the network owner chooses a public one-way hash function $h()$.

b) Choose a pair of integers e and d satisfying the properties $e{\cdot}d{\equiv}1\,(\bmod \phi(n))$ and d is a large positive number, where $\phi(n)$ is the Euler-Totient function and e should be larger than the output of the one-way hash function $h()$. Thus, the network owner creates its RSA public/private key pair as $\{n, e\}$ and d.

c) Choose the identity UID_j for the jth network user in advance, where $1{\leq}j{\leq}M$. Here we assume that the total number of users is M, which should be set according to the specific application scenario. Here the bit length of the identity of each network user is set to 8. Only the public key of the network owner is pre-loaded on each node before deployment.

d) We assume a user U_j registers to the network owner. After verifying his/her registration information, the network owner assigns an identity, say UID_j, for him. Then the network owner computes a proxy signature key υ_j for user U_j, which is given by

$$\upsilon_j{\equiv}[h(m_w)]^{-d}\,(\bmod n) \tag{1}$$

Here n is the public modulus defined earlier. The warrant m_w records UID_j, the identity of the network owner and the user privilege such as the sensor nodes set with specified identities or/and within a specific region that user U_j is allowed to reprogram, and valid periods of delegation (i.e., the beginning time and the end time). Note that equation (1) involves modular exponentiation with a negative exponent, which can be performed by finding the modular multiplicative inverse u of $h(m_w)$ modulo n using the extended Euclidean algorithm. That is, $\upsilon_j{\equiv}[h(m_w)]^{-d}\,(\bmod n){\equiv}u^d\,(\bmod n)$, where $h(m_w){\cdot}u{\equiv}1(\bmod n)$. It should be noted that the multiplicative inverse u of $h(m_w)$ modulo n exists if and only if $h(m_w)$ and n are coprime (i.e.,$gcd(h(m_w), n) = 1$). To ensure the existence of u, a feasible approach is when the network owner computes m_w for a user, redundant bits may be appended into m_w such that $h(m_w)$ and n are coprime.

Fig. 2 shows an example of the format of the warrant m_w in DiCode. Here we assume that the length of the warrant m_w is 32 bytes. In future applications, the network users may be allowed to update the program images by involving the specified types of sensor data (e.g., humidity, light, temperature). In that case, the reserved field of m_w may contain the sensor types set which user U_j is allowed to reprogram. For example, the first bit represents the light. If the bit is "1", it indicates that the light sensor can be activated by the network user. Additionally, we observe that in some of the existing reprogramming protocols (e.g., [4]), the code image on each sensor node is divided into multiple modules, each of which corresponds to different functionality. Thus, in DiCode, the network owner can define the reprogramming privilege of each user according to different functionality. Of course, any other simple ways to specify the reprogramming privilege can be applied easily here.

Taking the lengths of m_w and $n\,(=\ p{\times}q)$ as 32 bytes and 1024 bits, respectively, we carry out the experiment of coprime checking on desktop PCs with 2.4 GHz processor. The experimental results show that 1 byte of redundancy data is enough, and the search of appropriate redundancy data is very fast (i.e., the execution time is 3 μs). In our experiment, the same execution is run for 10-thousand times, and m_w, p, and q are randomly chosen for each time.

e) For each user U_j, the network owner returns the message $\{\upsilon_j, m_w\}$ using a secure transmission protocol, such as the wired Transport Layer Security (TLS) protocol. Afterward, U_j can verify the proxy signature key υ_j by checking if equation (2) is satisfied:

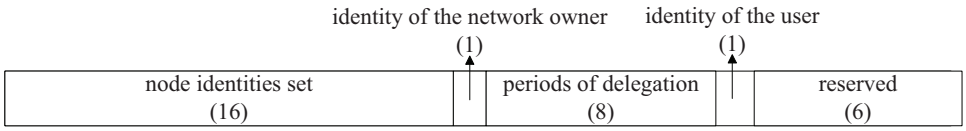

Fig. 2. An example of the format of the warrant m_w in DiCode. The byte size of each field is indicated below the label.

$$v_j{}^e h(m_w) \equiv 1 \pmod n \tag{2}$$

Because $v_j \equiv u^d \pmod n$, $u \equiv v_j{}^e \pmod n$, $u \equiv h(m_w)^{-1} \pmod n$, $v_j{}^e \pmod n \equiv h(m_w)^{-1} \pmod n$.

*2) **User Pre-processing**:* Assume that user U_j enters to the WSN and has a new program image. U_j takes the following actions:

a) U_j partitions the program image to P fixed-size pages, denoted as page 1 through page P. U_j splits page i $(1 \le i \le P)$ into N fixed-size packets, denoted as $Pkt_{i,1}$ through $Pkt_{i,N}$. The hash value of each packet in page P is appended to the corresponding packet in page $P-1$. For example, the hash value of packet $Pkt_{P,1}$, $h(Pkt_{P,1})$, is included in packet $Pkt_{P-1,1}$. Here $Pkt_{P,1}$ presents the first packet of page P. Similarly, the hash value of each packet in page $P-1$ is included in the corresponding packet in page $P-2$. This process continues until U_j finishes hashing all the packets in page 2 and including their hash values in the corresponding packets in page 1. Then a Merkle hash tree [26] is used to facilitate the authentication of the hash values of the packets in page 1. We refer to the packets related to this Merkle hash tree collectively as page 0. The root of the Merkle hash tree, the meta data about the code image (e.g., version number, targeted node identities set, program image size), and a signature over all of them are included in a signature message. The detailed information can be referred to [12]. Here we assume that the message m represents the root of the Merkle hash tree and the meta data about the code image. Essentially, the message m contains security bootstrap header, which an authorized user sends to bootstrap the authentication procedure. An example format of the message m is depicted in Fig. 3. Note that in order to support a variety of applications, the formats and lengths of the message m and the warrant m_w in DiCode should be set according to the specified application scenario. Here the *targeted node identities set* field indicates the identities of the sensor nodes which the user wishes to reprogram. We assume that the length of the identity of each sensor node is 2 bytes. Obviously, the targeted node identities set field is set according to the warrant m_w of the user.

Then U_j takes the following actions to construct the signature message:

b) U_j randomly chooses an integer $t \in [1, n]$ and computes $r = t^e \pmod n$.

c) With the message m, U_j computes $k = h(m \| r)$.

d) U_j computes $y = t v_j{}^k \pmod n$. Thus, U_j generates the signature message $\{m, m_w, y, k\}$, which serves as the notification of the new code image.

Note that there are two different cases for user U_j to reprogram the nodes. One case is that U_j wants to reprogram one or more particular sensor nodes, say $\{S_1, ..., S_v\}$, with identities $\{ID_1, ..., ID_v\}$. Here $v \ge 1$. We assume that sensor nodes do not know their geographical locations. Obviously, this assumption makes DiCode more applicable in the real world. In this case, the identities are added into the targeted node identities set field of m. As will be proved in Section V, the integrity of the message m (including the identities of the targeted nodes) can be ensured by proxy-unprotected signature. Therefore, no adversary can modify the identities and then pass the verification of any sensor node. Of course, reprogramming all nodes via broadcast also belongs to this case. Referring to Fig. 3, by setting the targeted node identities set field to "0", it indicates that U_j wants to reprogram all nodes. The other case is that U_j wants to reprogram the nodes in a specific region. We assume that the nodes know their geographical locations which can be acquired via deployment knowledge or many existing secure localization schemes (e.g., [22]). In this case, U_j needs to add the information about the specific region into the targeted node identities set field of m.

*3) **Sensor Node Verification**:* Upon receiving a signature message $\{m, m_w, y, k\}$, each sensor node verifies it as follows:

a) The node firstly pays attention to the legality of the warrant m_w and the message m. For example, the node needs to check whether the identity of itself is included in the node identities set of the warrant m_w. Also, according to the valid periods of delegation field of warrant m_w, the node can check whether reprogramming service to a user is expired. Only if they are valid, the verification procedure goes to the next step.

b) The sensor node computes $r^* = y^e h(m_w)^k \pmod n$.

c) The sensor node checks whether $h(m \| r^*) = k$. Because

$$\begin{aligned} v_j^e &= h(m_w)^{-1} \pmod n, \\ r^* &= t^e v_j^{ke} h(m_w)^k = t^e = r \pmod n \end{aligned}$$

Thus, $h(m \| r^*) = h(m \| r) = k$.

If the above verification passes, the node believes that the message m and the warrant m_w are from an authorized user. Hence the node accepts the root of the Merkle hash tree constructed for page 0. Thus, the node can authenticate the hash packets in page 0 once it receives such packets, based on the security of Merkle hash tree. The hash packets include the hash values of the data packets in page 1. Therefore, after verifying the hash packets, a node can easily verify the data packets in page 1 based on the one-way property of hash functions. Likewise, the data packets in page i have been verified, a node can easily authenticate the data packets in page $i+1$, where $i = 1, 2, \ldots, P-1$. Only if all verification procedures described above pass, the sensor node accepts the code image.

version num (1)	targeted node identities set (6)	code image size (2)	root of the Merkle hash tree (20)

Fig. 3. An example of the format of the message m of DiCode. The byte size of each field is indicated below the label.

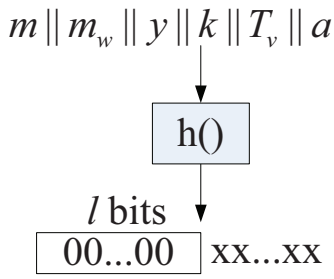

Fig. 4. Message specific puzzle.

IV. Some approaches to Complement the Basic Protocol for Resisting DoS attacks

Similar to most secure centralized schemes of [6]–[12], DiCode uses a digital signature technique for the authentication of the program image. This signature is vulnerable to DoS attacks. That is, the adversary may flood a large number of illegal signature messages to the nodes to exhaust their resources and render them less capable of serving legitimate users. To prevent the attacks, message specific puzzle and the improved message specific puzzle are described here, which can complement the basic protocol of DiCode. We assume the entire network is partitioned into a set of regions. Each sensor node belongs to exactly one region and a region may contain multiple nodes. The size of a region is application specific and sufficiently small to support the distributed reprogramming resolution. For reducing communication costs we choose to elect region (cluster) heads. That is, each region has a head sensor node which is responsible for generating a puzzle and then distributing it to all other nodes of the region.

A. Message Specific Puzzle Approach

To prevent the attacks, we adopt message specific puzzle (also called client puzzles) of [27] into DiCode. The idea is summarized as follows. In the system initialization phase, the network owner sets a threshold for user reprogramming rate. When there is no evidence of such an attack, each sensor node processes the signature messages normally, that is, indiscriminately. On the other hand, once a node finds the rate of incoming signature message is more than the threshold, it believes that it is under a DoS attack and only performs verification on signature messages selectively. In particular, the node attaches a unique puzzle into the *beacon messages* which is periodically broadcasted to declare service existence, and requires the solution of the puzzle to be attached in each signature message. The node commits resources to process a signature message only when the solution is correct. In general, solving a puzzle requires a brute-force search in the solution space, while solution verification is very fast. Additionally, puzzles are deployed in conjunction with conventional time-outs on node resources. Thus, in order to create an interruption in service, an adversary must have abundant resources to be able to promptly compute a large enough number of puzzle solutions in line with his sending rate of illegal signature messages. In contrast, although puzzles slightly increase legitimate users' computational load, they are still able to obtain reprogramming services regardless the existence of the attack. To incorporate this method into our protocol, we add a MESSAGE SPECIFIC PUZZLE flag in the *beacon messages*. If a node, say S_v, is not under attack, it sets the MESSAGE SPECIFIC PUZZLE flag to "No". It indicates to the users that no puzzles are being distributed. And our basic protocol is executed normally. If S_v is under attack, it sets the MESSAGE SPECIFIC PUZZLE flag to "Yes", and adds a puzzle (i.e., a timestamp T_v, a random number a and an integer l) into the *beacon messages*. In order to update the program on S_v, a user must solve the puzzle within a specified time interval. A valid solution L_i is such a value that after applying the hash function $h()$ to $(m\|m_w\|y\|k\|T_v\|a\|L_i)$, the first l bits of the resulting image are all "0", as illustrated in Fig. 4. The parameter l determines the strength of the puzzle. Before transmitting the signature message $\{m, m_w, y, k\}$, a user first tries to solve the puzzle by finding the puzzle solution L_i. Subsequently, the user sends the final signature message $\{m, m_w, y, k, T_v, a, L_i\}$ to S_v. Obviously, the puzzle solution in every signature message can be efficiently verified by S_v via a hash function operation and comparison. Only if this verification is successful, S_v performs expensive verification on the signature message $\{m, m_w, y, k\}$.

B. An Improved Message Specific Puzzle Approach

To provide a more effective message specific puzzle approach, we can make use of the one-way hash chain [28]. However, this requires more changes to the basic protocol of DiCode. For brevity, we just present the parts that need to be changed.

1) ***System Initialization****:* c) In addition to the activity described in step c) of the system initialization phase of the basic protocol, the network owner generates M one-way key chains. A one way key chain is based on a public cryptographic hash function $h()$, which is easy to compute but computationally hard to invert. A key chain with length b is generated by applying $h()$ on the initial selected element K_{bj} repeatedly b times. The last output of applying $h()$ b times, is called the committed value of the key chain. In DiCode, the length (i.e., b) of every key chain must be exactly the number of times the respective user U_j is allowed to reprogram the sensor nodes. For user U_j, the ith element in the hash chain is denoted as K_{ij}. Then we have $K_{ij} = h(K_{(i+1)j}), 0 \leq i \leq b$. Here a key chain is not only used to limit the number of reprogramming times, but also generate a message specific puzzle to mitigate DoS attacks against signature messages. The detailed description will be given in the following. The mapping table between the jth user's identity UID_j and the committed value of the corresponding key chain K_{0j}

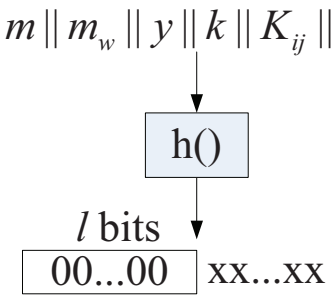

Fig. 5. The Improved Message specific puzzle.

$(UID_j \leftrightarrow K_{0j})$ are pre-loaded on every sensor node, where $1 \leq j \leq M$. As the output of a hash function (e.g., SHA-1), the byte length of K_{0j} is 20.

e) In addition to the message transmitted by step e) of the system initialization phase of our basic protocol, the network owner delivers K_{bj} to user U_j using the wired TLS protocol.

*2) **User Pre-processing**:* d) In addition to the activity described in step d) of the user pre-processing phase of the basic protocol, user U_j does $b - i$ hash operations on the value K_{bj}, and obtains K_{ij}, and then uses it to generate a puzzle. As described in Section III.C.2).d, here we consider the signature message of version i code image, denoted as $\{m, m_w, y, k\}$. The signature message $\{m, m_w, y, k\}$ and the puzzle key K_{ij} constitute a message specific puzzle. A valid solution L_i is such a value that after applying the hash function $h()$ to $(m \| m_w \| y \| k \| K_{ij} \| L_i)$, the first l bits of the resulting image are all "0", as illustrated in Fig. 5. The parameter l determines the strength of the puzzle. Before transmitting the signature message, user U_j first tries to solve the puzzle by finding the puzzle solution L_i. Subsequently, user U_j sends the finial signature message $\{m, m_w, y, k, K_{ij}, L_i\}$ to the nodes.

*3) **Sensor Verification**:* a) As step a) of the sensor verification phase of the basic protocol, if the warrant m_w and the message m are valid, the sensor node obtains the identity UID_j of user U_j from the warrant m_w. According to UID_j, the node can extract the corresponding puzzle key $K_{(i-1)j}$ from its memory. Thus the node can verify whether the received puzzle key K_{ij} is valid by comparing whether $h(K_{ij})$ equals $K_{(i-1)j}$. At the same time, the node needs to confirm that K_{ij} has not been used along with a valid signature message before. Only if these two verifications are successful, the node replaces $K_{(i-1)j}$ with K_{ij}. If the puzzle solution is valid, the node goes to the next step. Therefore, without first solving some message specific puzzles with a fresh puzzle key, the adversary cannot force a node to verify signatures in forged messages. Also, in the above procedure, since the node replaces $K_{(i-1)j}$ with K_{ij} for each successful verification and does just one hash operation for comparison, an adversary cannot claim a puzzle key close to the end of key chain and fool the node to perform a large number of unnecessary hash operations, causing a DoS attack.

Compared to the message puzzle approach presented in Section IV.A, this improved approach can more effectively mitigate DoS attacks against signature message, especially in some application scenarios where the adversary and a legitimate user may be incomparable in computation power. This is because that although it takes the same effort for both an authorized user and adversary to solve a puzzle, the authorized user has a clear advantage over the adversary due to the prior knowledge of the puzzle keys. The authorized user has enough time to solve a puzzle off-line before disseminating a new code image. In contrast, the adversary has a very tight time limit of solving the puzzle; it cannot begin to solve the puzzle until it has intercepted the puzzle key when the user transmits the signature massage, but has to finish solving the puzzle before the puzzle key becomes invalid when the signature message reaches the targeted sensor nodes. Although this improved approach requires the mapping table to be stored on each node and the scale of the mapping table increases linearly with the number of authorized users, the evaluation results in Section VII.B will show that this improved approach is able to support a large number of users.

Note that the proposed approach is also robust against distributed DoS attacks with multiple collaborative attackers in two aspects. Firstly, because each sensor node can detect DoS attacks, the attacked node(s) can marshal the aid of other legitimate nodes and impose filters to discard the attacking traffic closer to its point of origin. Secondly, all sensor nodes will notify the network owner of each distributed DoS attack. Thus, the network owner can analyze all attacks and then take some more effective measures.

V. Security Analysis of DiCode

In the following, we will analyze the security of DiCode to verify that the security requirements mentioned in Section I are satisfied.

Integrity of Code Images: In DiCode, an authorized network user uses a digital signature technique (more exactly, proxy-unprotected signature by warrant) to authenticate the root of the Merkle hash tree in page 0, with the proxy signature key only known to himself/herself and the network owner. All the sensor nodes know the public key of the network owner, and thus can verify the signature message. Under the assumption that the adversary cannot compromise the network owner, it is guaranteed that all sensor nodes can authenticate any received signature message as well as the root of the Merkle hash tree contained there. As described in Section III.C.3, if an adversary injects a forged modified program image, each receiving node can detect it easily because of the (immediate) authentication of code dissemination packets.

Ensurance of Freshness: Similar to [16], there are two cases for the users to administrate the program update of a WSN. In the first case, each user has the privilege to reprogram the nodes in different zones (or different sets of sensor nodes according to their identities) and there exists no node which is allowed to be reprogrammed by two users. In step a) of the sensor node verification phase, a node first checks whether the version number from the received signature message is valid. Only if it is valid, the verification procedure goes to the next step. Therefore, the use of the version number of the updated program image can ensure the freshness of DiCode. The other case is that a node may be assigned to multiple users by the network owner. A feasible approach is that a timestamp is used instead of the version number of the updated code

image. In step a) of the sensor node verification phase, a node first checks whether the timestamp included in the message m is fresh. This can ensure that a node always installs the most recent version of a program. More information about this issue will be given in Section VI.A.

Resistance to DoS Attacks: As described in [12], there are three types of DoS attacks against Deluge-based code dissemination: One is DoS attacks exploiting authentication delays, another is DoS attacks exploiting the expensive signature verifications, and the other is DoS attacks exploiting the Deluge propagation and suppression mechanisms. Since DiCode is a secure extension to Deluge, these attacks are also applicable in DiCode.

Because of the page-by-page dissemination strategy, DiCode can immediately authenticate any packet it receives in the current page, and successfully prevent from DoS attacks exploiting authentication delays. Further, due to the use of message specific puzzles, each node can perform an efficient hash function operation and comparison to detect fake signature message. Therefore, DiCode provides the resistance to DoS attacks exploiting the expensive signature verifications. Finally, similar to Seluge, DiCode uses cluster keys to authenticate each advertisement or SNACK packet. As a result, an adversary cannot convince regular nodes to misuse the propagation or suppression mechanisms. For a more detailed analysis, readers are referred to [12].

Resistance to Node Compromised Attack: As described in Section IV.B, even for the improved protocol, only the public key of the network owner and the mapping table ($UID_j \leftrightarrow K_{0j}$) are pre-loaded on every sensor node, where $1 \leq j \leq M$. That is, even if an adversary compromises some sensor nodes, the adversary just obtains the public key of the network owner and the mapping table ($UID_j \leftrightarrow K_{0j}$). Thus, the adversary cannot impersonate any authorized network user by compromising sensor nodes.

Distributed: As described in Sections III.C, in order to pass the signature verification of sensor nodes, each user has to obtain a proxy signature key from the network owner. In addition, it is clear that the authorized users are able to carry out code dissemination in a distributed manner.

Supporting Different User Privileges: The network owner can restrict each network user's activities by defining the warrant m_w, which records the user privilege. Because every proxy signature key is generated based on the corresponding warrant m_w, nobody except the network owner can modify the user privilege included in the warrant m_w and then pass the signature verification from the sensor nodes. Moreover, by introducing the key chain, the number of times each user is allowed to reprogram the sensor nodes is limited.

User Traceability: In DiCode, a node obtains the identity of a user from the warrant m_w included in the signature message, and then informs the network owner the reprogramming activity of the user through submitting the signature message $\{m, m_w, y, k\}$. Visits by the network owner are sporadic. Consequently, sensor nodes must accumulate the records in situ and wait for an explicit upload signal. Alternatively, there are also hybrid WSNs where the set of nodes, in addition to sensors, includes some small number of collector nodes, each serving as a temporary repository of the reprogramming records for their constituent sensors. The network owner later obtains the records directly from the collector nodes. Since every proxy signature key is generated based on the corresponding warrant m_w, nobody except the network owner can modify the identity of the network user included in the warrant m_w and then pass the signature verification from the network owner. Also, without knowing the proxy signature key of a user, the adversary (including some compromised sensor nodes) cannot modify the user identity information in signature messages. Therefore, DiCode can provide user traceability.

Note that each node only submits the whole signature messages to the network owner, without explicitly stating whether a network user is adversary. Thus, bad-mouthing attack cannot be launched by compromised sensor nodes. Though the network owner needs to go through the same procedure as the sensor node verification, it will not be overloaded for two reasons. Firstly, experimental results described in Section VII have shown that the signature verification cost is low even for resource-limited sensor nodes such as MicaZ motes. Since the network owner usually has much more processing power, it is thus able to verify a large number of signature messages forwarded by sensor nodes. Secondly, in real-world applications, it is desirable for a WSN to have a long lifetime but reprogramming is an operation with high energy consumption. Therefore, it is expected that reprogramming frequency is low.

VI. Discussion

So far, we have elaborated the operations of DiCode. By the protocol, we can achieve secure and distributed code dissemination. However, some important issues need further discussion.

A. Avoiding Reprogramming Conflicts

When multiple users are allowed to reprogram a node, some approaches should be employed as follows. In the user preprocessing phase, each network user, say U_j, needs to add the chosen T_P to every updated program image, where T_P denotes the time duration that a node will be occupied for reprogramming. Therefore, during this period, only user U_j is permitted to reprogram the nodes. Here T_P must be less than a fixed number, which is set by the network owner and then preloaded on each node. Generally, a user needs to reprogram a set of nodes which may only be available at different time. Our protocol allows a user to execute the reprogramming of nodes individually, without waiting for all nodes to be available.

B. Dynamic Participation

1) New User Joining Phase: This phase is invoked when a user, say U_{j+1}, hopes to obtain code dissemination privilege after the network is deployed. Here we consider the improved protocol, as described in Section IV.B, when user U_{j+1} registers to the network owner, the network owner chooses an identity, say UID_{j+1}, and the respective key chain. Then the network owner computes a proxy signature key υ_{j+1} for user U_{j+1}. As before, the warrant m_w records the identity UID_{j+1}

TABLE I
THE CODE SIZE OF VERIFICATION IMPLEMENTATION OF SIGNATURE MESSAGES IN DICODE

The length of n (bits)	1024	768	512
Code size on ROM (bytes)	40,840	40,572	40,316
Code size on RAM (bytes)	1,572	1,334	1,096

TABLE II
THE WHOLE CODE SIZE ON MICAZ

	ROM	RAM
Deluge (bytes)	50,562	673
Seluge (bytes)	92,642	2,556
DiCode (n=1024 bits) (bytes)	75,222	2,931
DiCode (n=768 bits) (bytes)	74,960	2,675
DiCode (n=512 bits) (bytes)	74,706	2,419

of user U_{j+1} and the user privilege. Subsequently, the network owner delivers the message $\{v_{j+1}, m_w, K_{n(j+1)}\}$ to user U_{j+1} using the wired TLS protocol. Note that the mapping information ($UID_{j+1} \leftrightarrow K_{0(j+1)}$) has been pre-loaded in each sensor node. Now this new user U_{j+1} can reprogram the sensor nodes. For the basic protocol, in this phase, the network owner just needs to transmit the message $\{v_{j+1}, m_w\}$ to user U_{j+1} using the wired TLS protocol.

2) Adding New Node Phase: This phase is invoked when a new node, say S_{w+1}, is to be added into the network by the network owner. For the basic protocol, the public key of the network owner is preloaded on S_{w+1}. For the improved protocol, the public key of the network owner and the mapping table $(UID_j \leftrightarrow K_{0j})$ needs to be preloaded on S_{w+1}, where $1 \leq j \leq M$. After that, S_{w+1} can be deployed in the network.

Additionally, when a network user leaves the system or has his/her privilege updated, whether the old warrant should be invalidated or not should be set according the specific application. If it does not need to be invalidated, it is clear that no extra processing is needed. Otherwise, the network owner needs to notify the related sensor nodes (i.e., these nodes' identities have been included in the *targeted node identities set* field of the old warrant).

VII. IMPLEMENTATION AND PERFORMANCE EVALUATION

We evaluate DiCode by implementing all components on an experimental test-bed. Since it has been demonstrated that Seluge [12] exceeds the security and efficiencies of other centralized code dissemination techniques, here we choose Seluge for performance comparison.

A. Implementation and Experimental Setup

Our implementation has the network owner, network user and sensor node side programs. The owner side programs are C programs using OpenSSL [29] running on a 3.2 GHz desktop PC. The network user side programs are C programs using OpenSSL running on a 2.4 GHz laptop PC. In addition, the sensor node side programs are written in nesC and runs on MicaZ motes. Each MicaZ platform uses an ATmega128 CPU that operates at 7.7 MHz. Our MicaZ motes run TinyOS [5] version 1.x. We use an indoor test-bed consisting of MicaZ motes to evaluate the efficiency of DiCode.

We add the following functionalities in the Java tools on the network user side: Computation of the hash values of the data packets from the last to the first page, construction of page 0 (i.e., Merkle hash tree) and hashing packets from the hash values of the page 1, and construction of the proxy signature message from the root of the Merkle hash tree and the meta data of the program image. We include the message specific puzzle method developed in both the Java tools and the sensor programs. When a network user generates a message specific puzzle, the lengths of T_v, a and L_i are assumed to be 2, 2, and 4 bytes, respectively.

TABLE III
THE EXECUTION TIME FOR EACH PHASE OF DICODE

The length of n (bits)	1024	768	512
Proxy signature key generation (ms)	9.7	4.4	1.8
Proxy signature key verification (ms)	1.8	1.2	0.5
Proxy signing (ms)	3.3	2.2	1.0
Verifying a puzzle solution(ms)	32.2	27.6	23.0
Signature verification (s)	19.3	7.4	3.6
System initialization (ms)	5.5	1.7	0.6

We modify the *PacketVerifier* module of the Seluge nesC library to perform verification of signature messages (including both puzzle and proxy signature verification), hash packets, and data packets. The public key of the network owner is generated by OpenSSL [29], and then pre-distributed to all sensor nodes. The pairwise keys used to distribute cluster keys and the mapping table $(UID_j \leftrightarrow K_{0j})$ are also pre-distributed to all nodes. The $h()$ used throughout DiCode is SHA-1.

B. Evaluation Results

We use the memory overhead, execution time and propagation delay to evaluate DiCode. The memory overhead measures the exact amount of data space required in the real implementation. Similarly, the execution time measures the time duration for each operation of DiCode. The propagation delay is the time required to finish disseminating a code image to all the nodes in the network. Table I shows the code size of verification implementation of signature message in DiCode. Among this code, the parameters (e.g., public key $\{n, e\}$ and proxy signature $\{y, k\}$) take a lot of memory space.

Table II shows the ROM and RAM usages of DiCode on MicaZ motes. The code size of Deluge and Seluge are also included for reference purposes. It is clear that DiCode takes less ROM than Seluge. Note that in the improved protocol, the mapping table $(UID_j \leftrightarrow K_{0j})$ is pre-loaded in each sensor node. We assume that the byte lengths of UID_j and K_{0j} are 1 and 20, respectively. Therefore, assuming that the improved protocol supports 1,000 network users, the code size of the mapping table is about 20 KB.

Table III gives the execution time of each operation in DiCode when the length of the parameter modulus n varies. Here system initialization indicates the generation of the network owner's public key and private key. As shown in Table III, the time for proxy signing on 2.4 GHz laptop PC is 3.3 ms when the length of n is 1024 bits. Considering the clock frequency of a typical PDA is more than 1 GHz, our protocol is efficient for most of mobile devices (e.g., laptop PCs or PDAs). Note that for the cryptographic operations on desktop PC and laptop PC, we perform the same operation one thousand times and take an average over them. It is easy to see

TABLE IV
THE EXECUTION TIME FOR SOLVING BASIC AND IMPROVED MESSAGE-SPECIFIC PUZZLES IN DICODE

The length of n (bits)	1024		768		512	
	basic	improved	basic	improved	basic	improved
The execution time (l=16)(s)	0.081	0.084	0.078	0.079	0.062	0.063
The execution time (l=18)(s)	0.327	0.328	0.318	0.322	0.241	0.248
The execution time (l=20)(s)	1.297	1.326	1.257	1.295	0.979	1.009
The execution time (l=22)(s)	5.099	5.156	5.005	5.119	3.938	3.971
The execution time (l=23)(s)	10.309	10.399	10.299	10.386	7.806	7.819

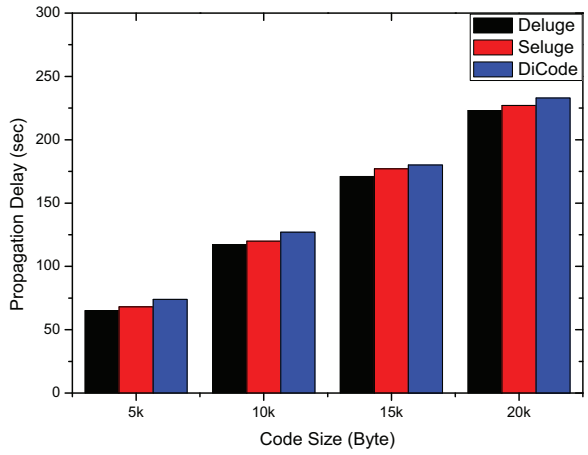

Fig. 6. Propagation delay comparison of three mechanisms.

that verifying a puzzle solution at a receiver (i.e., MicaZ mote) is extremely efficient. For example, the time for verifying a message-specific puzzle on a MicaZ mote is 27.6 ms when the length of n is 768 bits. Our experiments show that the time for signature verification on a MicaZ mote is 7.4 seconds when the length of n is 768 bits. Note that the proportion of this signature verification time in the total code dissemination time is very small. Considering Seluge as an example, the total code dissemination time for a 40-KByte program image is longer than 380 seconds in a WSN consisting of 65 MicaZ nodes. Further, the signature verification time of DiCode takes less than 2% of the total dissemination time. Considering the benefits that DiCode provides, this time consumption is acceptable.

Table IV gives the time required to solve the basic and improved message-specific puzzles for different length of n in OpenSSL implementation. For example, the time required to solve a basic puzzle and an improved puzzle are 5.005 seconds and 5.119 seconds, respectively, when the parameter l is set to 22 and the bit length of n is 768. As described in Section IV, these time consumptions are enough to defeat DoS attacks. Moreover, since the improved puzzle imposes a very tight time constraint for adversaries to obtain the puzzle solution, it is much more effective to mitigate DoS attacks than the basic puzzle even though they require similar computation time to obtain puzzle solutions.

To further investigate the impact of signature verification on the propagation delay of code dissemination in multi-hop networks, a multi-hop experiment was conducted. In this experiment, 6 MicaZ motes were deployed in a line with the same intervals, where a code image is propagated from one side to the other side. Here the length of n is set to 768 bits. Fig. 6 shows the propagation delays of Deluge, Seluge and DiCode measured from the experiment. As the code image size increases, the propagation delays of all schemes increase almost linearly. From Fig. 6, it is concluded that the signature verification by sensor nodes in DiCode only has low impact on the propagation delay of code dissemination. For example, when the code size of a program is 20-KByte, propagation delay of DiCode is only 4.5% more than that of Deluge. This is because upon receiving the signature packet, each sensor node can start to check the validity of the signature packet. At the same time, it transmits the packet to next-hop node. Considering the security benefits that DiCode provides, this cost is acceptable.

Researchers have justified [30] that when a sensor node's radio is always on during the code dissemination process, the energy consumption of the node depends chiefly on the completion time (i.e., propagation delay). The above experiment shows that the energy overhead of DiCode is similar to Deluge or Seluge.

VIII. CONCLUSION

In this paper, a distributed and DoS-resistant code dissemination protocol named DiCode has been proposed. Besides analyzing the security of DiCode, this paper has also reported the evaluation results of DiCode in a network of resource-limited sensor nodes, which shows the efficiency of our protocol in practice. We believe the performance can be improved by using more powerful sensor node (e.g., Imote2 has 416 MHz processor). To the best of our knowledge, until now our protocol is the only one that allows authorized users to reprogram sensor nodes in a DoS-resistant and distributed manner.

REFERENCES

[1] Crossbow Technology Inc., note in-network programming user reference, 2003.
[2] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proc. SenSys'04*.
[3] V. Naik, A. Arora, P. Sinha, and H. Zhang, "Sprinkler: a reliable and energy efficient data dissemination service for extreme scale wireless networks of embedded devices," *IEEE Trans. Mobile Comput.*, vol. 6, no. 7, pp. 762-776, 2007.
[4] R. K. Panta, I. Khalil, and S. Bagchi, "Stream: low overhead wireless reprogramming for sensor networks," in *Proc. 2007 IEEE INFOCOM*.
[5] TinyOS: an open-source OS for the networked sensor regime, http://www.tinyos.net/.
[6] J. Deng, R. Han, and S. Mishra, "Secure code distribution in dynamically programmable wireless sensor networks," in *Proc. IPSN'06*.
[7] P. K. Dutta, J. W. Hui, D. C. Chu, and D. E. Culler, "Securing the deluge network programming system," in *Proc. IPSN'06*.
[8] P. E. Lanigan, R. Gandhi, and P. Narasimhan, "Sluice: secure dissemination of code updates in sensor networks," in *Proc. ICDCS'06*.

[9] Y. Law *et al.*, "Secure rateless deluge: pollution-resistant reprogramming and data dissemination for wireless sensor networks," *EURASIP J. Wireless Commun. and Networking*, vol. 2011, pp. 1–21, 2010.
[10] C. Parra *et al.*, "A protocol for secure and energy-aware reprogramming in WSN," in *Proc. IWCMC'09*.
[11] N. Bui *et al.*, "An integrated system for secure code distribution in wireless sensor networks," in *Proc. Percom'10*.
[12] S. Hyun, P. Ning, A. Liu, and W. Du, "Seluge: secure and DoS-resistant code dissemination in wireless sensor networks," in *Proc. IPSN'08*.
[13] Geoss, http://www.epa.gov/geoss/.
[14] NOPP, http://www.nopp.org/.
[15] A. Lim, "Distributed services for information dissemination in self-organizing sensor networks," *J. Franklin Institute*, vol. 38, no. 6, pp. 707–727, 2001.
[16] D. He, C. Chen, S. Chan, and J. Bu, "SDRP: a secure and efficient reprogramming protocol for wireless sensor networks," *IEE Trans. Ind. Electron.*, accepted, 2011, DOI: 10.1109/TIE.2011.2178214.
[17] D. He, J. Bu, S. Chan, C. Chen, and M. Yin, "Privacy-preserving universal authentication protocol for wireless communications," *IEEE Trans. Wireless Commun.*, vol. 10, no. 2, pp. 431–436, 2011.
[18] X. Du, M. Guizani, Y. Xiao, and H.-H. Chen, "A routing-driven elliptic curve cryptography based key management scheme for heterogeneous sensor networks," *IEEE Trans. Wireless Commun.*, vol. 8, no. 3, pp. 1223–1229, 2009.
[19] D. Liu, P. Ning, and R. Li, "Establishing pairwise keys in distributed sensor networks," *ACM Trans. Inf. and Syst. Security*, vol. 8, no. 1, pp. 41–77, 2005.
[20] M. Rahman and K. EI-Khatib, "Secure time synchronization for wireless sensor networks based on bilinear pairing functions," *IEEE Trans. Parallel Distrib. Systems*, accepted, 2010.
[21] P. Li and J. Regehr, "T-Check: bug finding for sensor networks," in *Proc. IPSN'10*.
[22] D. He, L. Cui, H. Huang, and M. Ma, "Design and verification of enhanced secure localization scheme in wireless sensor networks," *IEEE Trans. Parallel Distrib. Systems*, vol. 20, no. 7, pp. 1050–1058, 2009.
[23] J. Camenisch and M. Stadler, "Efficient group signature schemes for large groups," *Advances in Cryptology -CRYPTO '97*, vol. 1296 of LNCS, pp. 410–424. Springer Verlag, 1997.
[24] F. Zhang and K. Kim, "Efficient ID-based blind signature and proxy signature from bilinear pairings," in *Proc. ACISP'03*.
[25] Z. Shao, "Proxy signature schemes based on factoring," *Inf. Process. Lett.*, vol. 85, no. 3, pp. 137–143, 2003.
[26] R. Merkle, "Protocols for public key cryptosystems," in *Proc. 1980 IEEE S&P*.
[27] A. Juels and J. Brainard, "Client puzzles: a cryptographic countermeasure against connection depletion attacks," in *Proc. NDSS'99*.
[28] Y.-C. Hu, M. Jakobsson, and A. Perrig, "Efficient constructions for one-way hash chains," in *Proc. ANCS'05*.
[29] OpenSSL, http://www.openssl.org.
[30] R. Fonseca, P. Dutta, P. Levis, and I. Stoica, "Quanto: tracking energy in networked embedded systems," in *Proc. USENIX OSDI'08*, pp. 323–328.

Daojing He received his B.Eng. and M.Eng. degrees in Computer Science from Harbin Institute of Technology in 2007 and 2009, respectively. He is currently a Ph.D. student in Zhejiang University, P.R. China. His research interests include network and systems security with focuses on wireless security. He serves as TPC for IEEE Globecom 2011, IEEE PIMRC 2011, IEEE ICC 2012, IEEE PIMRC 2012, IEEE WCNC 2012, etc.

Chun Chen received the bachelor degree in mathematics from Xiamen University, China, in 1981, and the masters and PhD degrees in computer science from Zhejiang University, China, in 1984 and 1990, respectively. He is a professor in the College of Computer Science, and the director of the Institute of Computer Software at Zhejiang University. His research activity is in image processing, computer vision, and embedded system.

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.

Jiajun Bu received the BS and PhD degrees in computer science from Zhejiang University, China, in 1995 and 2000, respectively. He is currently a professor in the College of Computer Science and the deputy dean of the Department of Digital Media and Network Technology at Zhejiang University. His research interests include embedded system, mobile multimedia, and data mining.