Effective Key Management in Dynamic Wireless Sensor Networks

Seung-Hyun Seo, *Member, IEEE*, Jongho Won, *Student Member, IEEE*, Salmin Sultana, *Member, IEEE*, and Elisa Bertino, *Fellow, IEEE*

***Abstract*— Recently, wireless sensor networks (WSNs) have been deployed for a wide variety of applications, including military sensing and tracking, patient status monitoring, traffic flow monitoring, where sensory devices often move between different locations. Securing data and communications requires suitable encryption key protocols. In this paper, we propose a certificateless-effective key management (CL-EKM) protocol for secure communication in dynamic WSNs characterized by node mobility. The CL-EKM supports efficient key updates when a node leaves or joins a cluster and ensures forward and backward key secrecy. The protocol also supports efficient key revocation for compromised nodes and minimizes the impact of a node compromise on the security of other communication links. A security analysis of our scheme shows that our protocol is effective in defending against various attacks. We implement CL-EKM in Contiki OS and simulate it using Cooja simulator to assess its time, energy, communication, and memory performance.**

***Index Terms*— Wireless sensor networks, certificateless public key cryptography, key management scheme.**

I. Introduction

DYNAMIC wireless sensor networks (WSNs), which enable mobility of sensor nodes, facilitate wider network coverage and more accurate service than static WSNs. Therefore, dynamic WSNs are being rapidly adopted in monitoring applications, such as target tracking in battlefield surveillance, healthcare systems, traffic flow and vehicle status monitoring, dairy cattle health monitoring [9]. However, sensor devices are vulnerable to malicious attacks such as impersonation, interception, capture or physical destruction, due to their unattended operative environments and lapses of connectivity in wireless communication [20]. Thus, security is one of the most important issues in many critical dynamic WSN applications. Dynamic WSNs thus need to address key security requirements, such as node authentication, data confidentiality and integrity, whenever and wherever the nodes move.

To address security, encryption key management protocols for dynamic WSNs have been proposed in the past based on symmetric key encryption [1]–[3]. Such type of encryption is well-suited for sensor nodes because of their limited energy and processing capability. However, it suffers from high communication overhead and requires large memory space to store shared pairwise keys. It is also not scalable and not resilient against compromises, and unable to support node mobility. Therefore symmetric key encryption is not suitable for dynamic WSNs. More recently, asymmetric key based approaches have been proposed for dynamic WSNs [4]–[7], [10], [15], [18], [25], [27]. These approaches take advantage of public key cryptography (PKC) such as elliptic curve cryptography (ECC) or identity-based public key cryptography (ID-PKC) in order to simplify key establishment and data authentication between nodes. PKC is relatively more expensive than symmetric key encryption with respect to computational costs. However, recent improvements in the implementation of ECC [11] have demonstrated the feasibility of applying PKC to WSNs. For instance, the implementation of 160-bit ECC on an Atmel AT-mega 128, which has an 8-bit 8 MHz CPU, shows that an ECC point multiplication takes less than one second [11]. Moreover, PKC is more resilient to node compromise attacks and is more scalable and flexible. However, we found the security weaknesses of existing ECC-based schemes [5], [10], [25] that these approaches are vulnerable to message forgery, key compromise and known-key attacks. Also, we analyzed the critical security flaws of [15] that the static private key is exposed to the other when both nodes establish the session key. Moreover, these ECC-based schemes with certificates when directly applied to dynamic WSNs, suffer from the certificate management overhead of all the sensor nodes and so are not a practical application for large scale WSNs. The pairing operation-based ID-PKC [4], [18] schemes are inefficient due to the computational overhead for pairing operations. To the best of our knowledge, efficient and secure key management schemes for dynamic WSNs have not yet been proposed.

In this paper, we present a certificateless effective key management (CL-EKM) scheme for dynamic WSNs. In certificateless public key cryptography (CL-PKC) [12], the user's full private key is a combination of a partial private key generated by a key generation center (KGC) and the user's own secret value. The special organization of the full private/public key pair removes the need for certificates and also resolves the key escrow problem by removing the responsibility for the user's full private key. We also take the benefit of ECC keys defined on an additive group with a 160-bit length as secure as the RSA keys with 1024-bit length.

Manuscript received August 6, 2014; revised October 17, 2014; accepted November 18, 2014. Date of publication December 4, 2014; date of current version January 13, 2015. This work was supported in part by the Brain Korea 21 Plus Project. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Kui Q. Ren.

S.-H. Seo is with the Center for Information Security Technologies, Korea University, Seoul 136-701, Korea (e-mail: seosh77@gmail.com).

J. Won, S. Sultana, and E. Bertino are with the Department of Computer Science, Purdue University, West Lafayette, IN 47907 USA (e-mail: won12@purdue.edu; ssultana@purdue.edu; bertino@purdue.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2014.2375555

In order to dynamically provide both node authentication and establish a *pairwise key* between nodes, we build CL-EKM by utilizing a pairing-free certificateless hybrid signcryption scheme (CL-HSC) proposed by us in an earlier work [13], [14]. Due to the properties of CL-HSC, the pairwise key of CL-EKM can be efficiently shared between two nodes without requiring taxing pairing operations and the exchange of certificates. To support node mobility, our CL-EKM also supports lightweight processes for cluster key updates executed when a node moves, and key revocation is executed when a node is detected as malicious or leaves the cluster permanently. CL-EKM is scalable in case of additions of new nodes after network deployment. CL-EKM is secure against node compromise, cloning and impersonation, and ensures forward and backward secrecy. The security analysis of our scheme shows its effectiveness. Below we summarize the contributions of this paper:

- We show the security weaknesses of existing ECC based key management schemes for dynamic WSNs [10], [15], [25].
- We propose the first certificateless effective key management scheme (CL-EKM) for dynamic WSNs. CL-EKM supports four types of keys, each of which is used for a different purpose, including secure pair-wise node communication and group-oriented key communication within clusters. Efficient key management procedures are defined as supporting node movements across different clusters and key revocation process for compromised nodes.
- CL-EKM is implemented using Contiki OS and use a TI exp5438 emulator to measure the computation and communication overhead of CL-EKM. Also we develop a simulator to measure the energy consumption of CL-EKM. Then, we conduct the simulation of node movement by adopting the Random Walk Mobility Model and the Manhattan Mobility Model within the grid. The experimental results show that our CL-EKM scheme is lightweight and hence suitable for dynamic WSNs.

The remainder of this paper is organized as follows: In Section 2, we briefly discuss related work and show the security weaknesses of the existing schemes. In Section 3, we provide our network model and adversary model. In Section 4, we provide an overview of our CL-EKM. In Section 5, we introduce the details of CL-EKM. In Section 6, we analyze the security of CL-EKM. In Section 7, we evaluate the performance of CL-EKM, conduct the simulation of node movement in Section 8, and conclude in Section 9.

II. Related Work

Symmetric key schemes are not viable for mobile sensor nodes and thus past approaches have focused only on static WSNs. A few approaches have been proposed based on PKC to support dynamic WSNs. Thus, in this section, we review previous PKC-based key management schemes for dynamic WSNs and analyze their security weaknesses or disadvantages. Chuang et al. [7] and Agrawal et al. [8] proposed a two-layered key management scheme and a dynamic key update protocol in dynamic WSNs based on the Diffie-Hellman (DH), respectively. However, both schemes [7], [8] are not suited for sensors with limited resources and are unable to perform expensive computations with large key sizes (e.g. at least 1024 bit). Since ECC is computationally more efficient and has a short key length (e.g. 160 bit), several approaches with certificate [5], [10], [15], [25] have been proposed based on ECC. However, since each node must exchange the certificate to establish the pairwise key and verify each other's certificate before use, the communication and computation overhead increase dramatically. Also, the BS suffers from the overhead of certificate management. Moreover, existing schemes [5], [10], [15], [25] are not secure. Alagheband et al. [5] proposed a key management scheme by using ECC-based signcryption, but this scheme is insecure against message forgery attacks [16]. Huang et al. [15] proposed a ECC-based key establishment scheme for self-organizing WSNs. However, we found the security weaknesses of their scheme. In step 2 of their scheme, a sensor node U sends $z = q_U \cdot H(MacKey) + d_U(modn)$ to the other node V for authentication, where q_U is a static private key of U. But, once V receives the z, it can disclose q_U, because V already got $MacKey$ and d_U in step 1. So, V can easily obtain q_U by computing $q_U = (z - d_U) \cdot H(MacKey)^{-1}$. Thus, the sensor node's private key is exposed to the other node during the key establishment between two nodes. Zhang et al. [10] proposed a distributed deterministic key management scheme based on ECC for dynamic WSNs. It uses the symmetric key approach for sharing the pairwise key for existing nodes and uses an asymmetric key approach to share the pairwise keys for a new node after deployment. However, since the initial key K_I is used to compute the individual keys and the pairwise keys after deployment for all nodes, if an adversary obtains K_I, the adversary has the ability to compute all individual keys and the pairwise keys for all nodes. Thus, such scheme suffers from weak resilience to node compromises. Also, since such scheme uses a simple ECC-based DH key agreement by using each node's long-term public key and private key, the shared pairwise key is static and as a result, is not secure against known-key attacks and cannot provide re-key operation. Du et al. [25] use a ECDSA scheme to verify the identity of a cluster head and a static EC-Diffie-Hellman key agreement scheme to share the pairwise key between the cluster heads. Therefore, the scheme by Du et al. is not secure against known-key attacks, because the pairwise key between the cluster heads is static. On the other hand, Du et al. use a modular arithmetic-based symmetric key approach to share the pairwise key between a sensor node and a cluster head. Thus, a sensor node cannot directly establish a pairwise key with other sensor nodes and, instead, it requires the support of the cluster head. In their scheme, in order to establish a pairwise key between two nodes in the same cluster, the cluster head randomly generates a pairwise key and encrypts it using the shared keys with these two nodes. Then the cluster head transmits the encrypted pairwise key to each node. Thus, if the cluster head is compromised, the pairwise keys between non-compromised sensor nodes in the same cluster will also be compromised. Therefore,

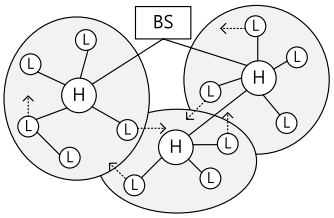


Fig. 1. Heterogeneous dynamic wireless sensor network.

their scheme is not compromise-resilient against cluster head capture, because the cluster head randomly generates a pairwise key between sensor nodes whenever it is requested by the nodes. Moreover, in their scheme, in order to share a pairwise key between two nodes in different clusters, these two nodes must communicate via their respective cluster heads. So, after one cluster head generates the pairwise key for two nodes, the cluster head must securely transmit this key to both its node and the other cluster head. Thus, this pairwise key should be encrypted by using the shared pairwise key with the other cluster head and the shared key with its node, respectively. Therefore, if the pairwise key between the cluster heads is exposed, all pairwise keys of the two nodes in different clusters are disclosed. The scheme by Du et al. supports forward and backward secrecy by using a key update process whenever a new node joins the cluster or if a node is compromised. However, the scheme does not provide a process to protect against clone and impersonation attack.

Most recently, Rahman et al. [4] and Chatterjee et al. [18] have proposed ID-PKC based key management schemes supporting the mobility of nodes in dynamic WSNs which removes the certificate management overhead. However, their schemes require expensive pairing operations. Although many approaches that enable pairing operations for sensor nodes have been proposed, the computational cost required for pairing is still considerably higher than standard operations such as ECC point multiplication. For example, NanoECC, which uses the MIRACL library, takes around 17.93s to compute one pairing operation and around 1.27s to compute one ECC point multiplication on the MICA2(8MHz) mote [17].

III. Network and Adversary Models

A. Network Model

We consider a heterogeneous dynamic wireless sensor network (See Fig. 1). The network consists of a number of stationary or mobile sensor nodes and a BS that manages the network and collects data from the sensors. Sensor nodes can be of two types: (i) nodes with high processing capabilities, referred to as H-sensors, and (ii) nodes with low processing capabilities, referred to as L-sensors. We assume to have N nodes in the network with a number N_1 of H-sensors and a number N_2 of L-sensors, where $N = N_1 + N_2$, and $N_1 \ll N_2$. Nodes may join and leave the network, and thus the network size may dynamically change. The H-sensors act as *cluster heads* while L-sensors act as cluster members. They are connected to the BS directly or by a multi-hop path through other H-sensors. H-sensors and L-sensors can be stationary or mobile. After the network deployment, each H-sensor forms a cluster by discovering the neighboring L-sensors through *beacon* message exchanges. The L-sensors can join a cluster, move to other clusters and also re-join the previous clusters. To maintain the updated list of neighbors and connectivity, the nodes in a cluster periodically exchange very lightweight *beacon* messages. The H-sensors report any changes in their clusters to the BS, for example, when a L-sensor leaves or joins the cluster. The BS creates a list of legitimate nodes, $\mathfrak{M}$, and updates the status of the nodes when an anomaly node or node failure is detected. The BS assigns each node a unique identifier. A L-sensor n_{L_i} is uniquely identified by node ID L_i whereas a H-sensor n_{H_j} is assigned a node ID H_j. A Key Generation Center (KGC), hosted at the BS, generates public system parameters used for key management by the BS and issues certificateless public/private key pairs for each node in the network. In our key management system, a unique *individual key*, shared only between the node and the BS is assigned to each node. The certificateless public/private key of a node is used to establish *pairwise key*s between any two nodes. A *cluster key* is shared among the nodes in a cluster.

B. Adversary Model and Security Requirements

We assume that the adversary can mount a physical attack on a sensor node after the node is deployed and retrieve secret information and data stored in the node. The adversary can also populate the network with the clones of the captured node. Even without capturing a node, an adversary can conduct an *impersonation* attack by injecting an illegitimate node, which attempts to impersonate a legitimate node. Adversaries can conduct passive attacks, such as, eavesdropping, replay attack, etc to compromise data *confidentiality* and *integrity*. Specific to our proposed key management scheme, the adversary can perform a *known-key* attack to learn pairwise master keys if it somehow learns the short-term keys, e.g., pairwise encryption keys. As described in [26] and [8], in order to provide a secure key management scheme for WSNs supporting mobile nodes, the following security properties are critical:

- *Compromise-Resilience:* A compromised node must not affect the security of the keys of other legitimate nodes. In other words, the compromised node must not be able to reveal pairwise keys of non-compromised nodes. The compromise-resilience definition does not mean that a node is resilient against capture attacks or that a captured node is prevented from sending false data to other nodes, BS, or cluster heads.
- *Resistance Against Cloning and Impersonation:* The scheme must support *node authentication* to protect against node replication and impersonation attacks.
- *Forward and Backward Secrecy:* The scheme must assure *forward secrecy* to prevent a node from using an old key to continue decrypting new messages. It must also assure *backward secrecy* to prevent a node with the new key from going backwards in time to decrypt previously exchanged messages encrypted with prior keys. *forward and backward secrecy* are used to protect against node capture attacks.

- *Resilience Against Known-Key Attack:* The scheme must be secure against the known-key attack.

IV. Overview of the Certificateless Effective Key Management Scheme

In this paper, we propose a Certificateless Key Management scheme (CL-EKM) that supports the establishment of four types of keys, namely: a certificateless public/private key pair, an individual key, a pairwise key, and a cluster key. This scheme also utilizes the main algorithms of the CL-HSC scheme [13] in deriving certificateless public/private keys and pairwise keys. We briefly describe the major notations used in the paper (See Table I), the purpose of these keys and how they are setup.

A. Types of Keys

- *Certificateless Public/Private Key:* Before a node is deployed, the KGC at the BS generates a unique certificateless private/public key pair and installs the keys in the node. This key pair is used to generate a mutually authenticated pairwise key.
- *Individual Node Key:* Each node shares a unique individual key with BS. For example, a L-sensor can use the individual key to encrypt an alert message sent to the BS, or if it fails to communicate with the H-sensor. An H-sensor can use its individual key to encrypt the message corresponding to changes in the cluster. The BS can also use this key to encrypt any sensitive data, such as compromised node information or commands. Before a node is deployed, the BS assigns the node the individual key.
- *Pairwise Key:* Each node shares a different pairwise key with each of its neighboring nodes for secure communications and authentication of these nodes. For example, in order to join a cluster, a L-sensor should share a pairwise key with the H-sensor. Then, the H-sensor can securely encrypt and distribute its cluster key to the L-sensor by using the pairwise key. In an aggregation supportive WSN, the L-sensor can use its pairwise key to securely transmit the sensed data to the H-sensor. Each node can dynamically establish the pairwise key between itself and another node using their respective certificateless public/private key pairs.
- *Cluster Key:* All nodes in a cluster share a key, named as cluster key. The cluster key is mainly used for securing broadcast messages in a cluster, e.g., sensitive commands or the change of member status in a cluster. Only the cluster head can update the cluster key when a L-sensor leaves or joins the cluster.

V. The Details of CL-EKM

The CL-EKM is comprised of 7 phases: *system setup, pairwise key generation, cluster formation, key update, node movement, key revocation*, and *addition of a new node*.

TABLE I
List of Notations

L_i	Unique identifier of an L-sensor node n_{L_i}
H_j	Unique identifier of an H-sensor node n_{H_j}
ID_{BS}	Identifier of the Base Station (BS)
q	A k bit primer number
P_{pub}	A system public key of KGC, $P_{pub} = xP$
x	A master private key of KGC
P	Point generator of an additive cyclic group G_q
$\mathbb{Z}^*_q$	The multiplicative group of integers modulo q
E/F_q	The selected elliptic curve over the field F_q: $y^2 = x^3 + ax + b \bmod q$, $a, b, x, y \in F_q$
pk_A	full public key of any node n_A, $pk_A = (P_A, R_A)$
sk_A	full private key of any node n_A, $sk_A = (d_A, x_A)$
K^0_A	Individual key of any node n_A
K_{AB}	Pairwise master key between n_A and n_B
k_{AB}	Pairwise encryption key between n_A and n_B
GK_j	Cluster key shared among the nodes in the j-th cluster
$\mathfrak{M}$	List of all the legitimate nodes in the network, maintained by the BS
$\mathfrak{R}$	List of revoked nodes, maintained by the BS
$HMAC(k, m)$	Message authentication code of m using key k
$E_k(m)$	A symmetric key encryption algorithm to encrypt a message m with a key k.

A. System Setup

Before the network deployment, the BS generates system parameters and registers the node by including it in a member list $\mathfrak{M}$.

1) Generation of System Parameters: The KGC at the BS runs the following steps by taking a security parameter $k \in \mathbb{Z}^+$ as the input, and returns a list of system parameter $\Omega = \{F_q, E/F_q, G_q, P, P_{pub} = xP, h_0, h_1, h_2, h_3\}$ and x.

- Choose a k-bit prime q
- Determine the tuple $\{F_q, E/F_q, G_q, P\}$.
- Choose the master private key $x \in_R \mathbb{Z}^*_q$ and compute the system public key $P_{pub} = xP$.
- Choose cryptographic hash functions $\{h_0, h_1, h_2, h_3\}$ so that $h_0 : \{0,1\}^* \times G^2_q \rightarrow \{0,1\}^*$, $h_1 : G^3_q \times \{0,1\}^* \times G_q \rightarrow \{0,1\}^n$, $h_2 : G_q \times \{0,1\}^* \times G_q \times \{0,1\}^* \times G_q \times \{0,1\}^* \times G_q \rightarrow \mathbb{Z}^*_q$, and $h_3 : G_q \times \{0,1\}^* \times G_q \times \{0,1\}^* \times G_q \times \{0,1\}^* \times G_q \rightarrow \mathbb{Z}^*_q$. Here, n is the length of a symmetric key.

The BS publishes Ω and keeps x secret.

2) Node Registration: The BS assigns a unique identifier, denoted by L_i, to each L-sensor n_{L_i} and a unique identifier, denoted by H_j, to each H-sensor n_{H_j}, where $1 \leq i \leq N_1$, $1 \leq j \leq N_2, N = N_1 + N_2$. Here we describe the certificateless public/private key and individual node key operations for L_i, the same mechanisms apply for H-sensors. During initialization, each node n_{L_i} chooses a secret value $x_{L_i} \in_R \mathbb{Z}^*_q$ and computes $P_{L_i} = x_{L_i} P$. Then, the BS requests the KGC for partial private/public keys of n_{L_i} with the input parameters L_i and P_{L_i}. The KGC chooses $r_{L_i} \in_R \mathbb{Z}^*_q$ and then computes a pair of partial public/private key (R_{L_i}, d_{L_i}) as below:

$$R_{L_i} = r_{L_i} P$$
$$d_{L_i} = r_{L_i} + x \cdot h_0(L_i, R_{L_i}, P_{L_i}) \, mod \, q$$

The L_i can validate its private key by checking whether the condition $d_{L_i} P = R_{L_i} + h_0(L_i, R_{L_i}, P_{L_i}) P_{pub}$ holds.

L_i then sets $sk_{L_i} = (d_{L_i}, x_{L_i})$ as its full private key and $pk_{L_i} = (P_{L_i}, R_{L_i})$ as its full public key. The BS also chooses a uniform random number $x^0 \in \mathbb{Z}_q^*$ to generate the node's individual key $K_{L_i}^0$ ($K_{H_j}^0$ for n_{H_j}). The individual key is computed as an HMAC of x_0, L_i as follows

$$K_{L_i}^0 = HMAC(x^0, L_i)$$

After the key generation for all the nodes, the BS generates a member list $\mathfrak{M}$ consisting of identifiers and public keys of all these nodes. It also initializes a revocation list $\mathfrak{R}$ that enlists the revoked nodes. The public/private key, Ω, and the individual key are installed in the memory of each node.

B. Pairwise Key Generation

After the network deployment, a node may broadcast an advertisement message to its neighborhood to trigger the pairwise key setup with its neighbors. The advertisement message contains its identifier and public key. At first, two nodes set up a long-term pairwise master key between them, which is then used to derive the pairwise encryption key. The pairwise encryption key is short-term and can be used as a session key to encrypt sensed data.

1) Pairwise Master Key Establishment: In this paragraph, we describe the protocol for establishing a pairwise master key between any two nodes n_A and n_B with unique IDs A and B, respectively. We utilize the CL-HSC scheme [13] as a building block. When n_A receives an advertisement message from n_B, it executes the following **encapsulation** process to generate a long-term pairwise master key K_{AB} and the encapsulated key information, $\varphi_A = (U_A, W_A)$.

- Choose $l_A \in_R \mathbb{Z}_q^*$ and compute $U_A = l_A P$.
- Compute

$$T_A = l_A \cdot h_0(B, R_B, P_B) P_{pub} + l_A \cdot R_B \bmod q$$
$$K_{AB} = h_1(U_A, T_A, l_A \cdot P_B, B, P_B)$$

- Compute

$$h = h_2(U_A, \tau_A, T_A, A, P_A, B, P_B)$$
$$h' = h_3(U_A, \tau_A, T_A, A, P_A, B, P_B)$$
$$W_A = d_A + l_A \cdot h + x_A \cdot h'$$

  where τ_A is a random string to give a freshness.
- Output K_{AB} and $\varphi_A = (U_A, W_A)$.

Then, n_A sends A, pk_A, τ_A and φ_A to n_B. n_B then performs **decapsulation** to obtain K_{AB}.

- Compute $T_A = d_B \cdot U_A$.
  Note: Because of $d_B = r_B + x \cdot h_0(B, R_B, P_B)$ *and* $U_A = l_A P \bmod q$, T_A *is computed as* $T_A = (r_B + x \cdot h_0(B, R_B, P_B)) \cdot l_A P \bmod q = l_A \cdot h_0(B, R_B, P_B) P_{pub} + l_A \cdot R_B \bmod q$,
- Compute $h = h_2(U_A, \tau_A, T_A, A, P_A, B, P_B)$ and $h' = h_3(U_A, \tau_A, T_A, A, P_A, B, P_B)$.
- If $W_A \cdot P = R_A + h_0(A, R_A, P_A) \cdot P_{pub} + h \cdot U_A + h' \cdot P_A$, output $K_{AB} = h_1(U_A, T_A, x_B \cdot U_A, B, P_B)$. Otherwise, output invalid.

TABLE II
CLUSTER FORMATION PROCESS

Node Discovery and Authentication
$n_{H_j} \rightarrow *$: $\langle H_j, pk_{H_j} \rangle$
(for $i = 1, \ldots, n$)
$n_{L_i} \leftrightarrow n_{H_j}$: Perform *Pairwise Key Generation* phase
Cluster Key Generation
(for $i = 1, \ldots, n$)
n_{H_j} : Generate GK_j, Compute $C_2 = E_{k_{L_i H_j}}(GK_j, H_j, L_i)$
$n_{H_j} \rightarrow n_{L_i}$: $\langle H_j, C_2 \rangle$
n_{L_i} : Decrypt C_2 to get GK_j and Compute $C_3 = E_{k_{L_i H_j}}(L_i, HMAC(k_{L_i H_j}, GK_j))$
$n_{L_i} \rightarrow n_{H_j}$: $\langle L_i, C_3 \rangle$
n_{H_j} : Decrypt C_3 and Check the validity
Membership Validation
n_{H_j} : Compute $C_4 = E_{K_{H_j}^0}(H_j, \mathfrak{M}_j)$, $C_5 = E_{GK_j}(H_j, \mathfrak{M}_j)$
$n_{H_j} \rightarrow$ BS : $\langle H_j, C_4 \rangle$
BS : Check $\mathfrak{M}_j$
BS $\rightarrow n_{H_j}$: $\langle Acknowledgement \rangle$
$n_{H_j} \rightarrow *$: $\langle C_5 \rangle$

2) Pairwise Encryption Key Establishment: Once n_A and n_B set the pairwise master key K_{AB}, they generate an HMAC of K_{AB} and a nonce $r \in_R \mathbb{Z}_q^*$. The HMAC is then validated by both n_A and n_B. If the validation is successful, the HMAC value is established as the short-term *pairwise encryption key* k_{AB}. The process is summarized below:

- n_B chooses a random nonce $r \in_R \mathbb{Z}_q^*$, computes $k_{AB} = HMAC(K_{AB}, r)$ and $C_1 = E_{k_{AB}}(r, A, B)$. Then, n_B sends r and C_1 to n_A.
- When n_A receives r and C_1, it computes $k_{AB} = HMAC(K_{AB}, r)$ and decrypts C_1. Then it validates r, A and B and if valid confirms that n_B knows K_{AB} and it can compute k_{AB}.

C. Cluster Formation

Once the nodes are deployed, each H-sensor discovers neighboring L-sensors through *beacon* message exchanges and then proceeds to authenticate them. If the authentication is successful, the H-sensor forms a cluster with the authenticated L-sensors and they share a common cluster key. The H-sensor also establishes a pairwise key with each member of the cluster. To simplify the discussion, we focus on the operations within one cluster and consider the jth cluster. We also assume that the cluster head H-sensor is n_{H_j} with $n_{L_i} (1 \leq i \leq n)$ as cluster members. n_{H_j} establishes a cluster key GK_j for secure communication in the cluster. Table II shows the cluster formation process.

1) Node Discovery and Authentication: For node discovery, n_{H_j} broadcasts an advertisement message containing H_j and pk_{H_j}. Once n_{L_i} within H_j's radio range receives the advertisement, it checks H_j and pk_{H_j}, and initiates the *Pairwise Key Generation* procedure. Note that n_{L_i} may receive multiple advertisement messages if it is within the range of more than one H-sensor. However, n_{L_i} must choose one H-sensor, may be by prioritizing over the proximity and signal strength. Additionally, n_{L_i} can record other H-sensor advertisements as backup cluster heads in the event that the primary cluster head is disabled. If n_{L_i} selects multiple cluster heads and sends a response to all of them, it is considered as

a compromised node. n_{L_i} and n_{H_j} perform the *Pairwise Key Generation* procedure to obtain a pairwise master key, $K_{L_iH_j}$ and a pairwise encryption key, $k_{L_iH_j}$.

2) Cluster Key Generation: n_{H_j} chooses $x_j \in_R \mathbb{Z}_q^*$ to generate a cluster key GK_j as follows

$$GK_j = HMAC(x_j, H_j)$$

Then, n_{H_j} computes $C_2 = E_{k_{L_iH_j}}(GK_j, H_j, L_i)$ to distribute the GK_j. Then n_{H_j} sends H_j and C_2 to n_{L_i}. n_{L_i} decrypts C_2 to recover H_j, L_i and GK_j by using $k_{L_iH_j}$. If n_{L_i} fails to check H_j, L_i, it discards the message and reports n_{H_j} to the BS as an illegitimate cluster head. Otherwise, n_{L_i} confirms that n_{H_j} is valid and can compute GK_j. Then, n_{L_i} stores GK_j as a cluster key. Next, n_{L_i} computes $HMAC(k_{L_iH_j}, GK_j)$ and $C_3 = E_{k_{L_iH_j}}(L_i, HMAC(k_{L_iH_j}, GK_j))$. It transmits C_3 and L_i to n_{H_j}. After n_{H_j} receives messages from n_{L_i}, it decrypts C_3 by using $k_{L_iH_j}$. Then it checks L_i and the validity of $HMAC(k_{L_iH_j}, GK_j)$. If the validity check fails, n_{H_j} discards the message. Otherwise, n_{H_j} can confirm that n_{L_i} shares the valid GK_j and $k_{H_jL_i}$. n_{H_j} adds L_i and pk_{L_i} on member list of the jth cluster, $\mathfrak{M}_j$.

3) Membership Validation: After discovering all the neighboring nodes $n_{L_i}(1 \le i \le n)$ in the jth cluster, n_{H_j} computes $C_4 = E_{K^0_{H_j}}(H_j, \mathfrak{M}_j)$ and transmits C_4 and H_j to the BS. After receiving messages from n_{H_j}, the BS checks the validity of the nodes listed in $\mathfrak{M}_j$. If all nodes are legitimate, the BS sends an acknowledgement to n_{H_j}. Otherwise, the BS rejects $\mathfrak{M}_j$ and investigates the identities of invalid nodes (false or duplicate ID). Then, the BS adds the identities of invalid nodes to the revocation list and reports it to n_{H_j}. Upon receiving the acknowledge message, n_{H_j} computes $C_5 = E_{GK_j}(H_j, \mathfrak{M}_j)$ and broadcasts C_5 to all the nodes in jth cluster.

D. Key Update

In order to protect against cryptanalysis and mitigate damage from compromised keys, frequent encryption key updates are commonly required. In this section we provide the pairwise key update and cluster key update operations.

1) Pairwise Key Update: To update a pairwise encryption key, two nodes which shared the pairwise key perform a *Pairwise Encryption Key Establishment* process. On the other hand, the pairwise master key does not require periodical updates, because it is not directly used to encrypt each session message. As long as the nodes are not compromised, the pairwise master keys cannot be exposed. However, if a pairwise master key is modified or needs to be updated according to the policy of the BS, the *Pairwise Master Key Establishment* process must be executed.

2) Cluster Key Update: Only cluster head H-sensors can update their cluster key. If a L-sensor attempts to change the cluster key, the node is considered a malicious node. The operation for any jth cluster is described as follows: 1) n_{H_j} chooses $x'_j \in_R \mathbb{Z}_q^*$ and computes a new cluster key $GK'_j = HMAC(x'_j, H_j)$. n_{H_j} also generates an $Update$ message including $HMAC(GK'_j, Update)$ and computes $C_6 = E_{GK_j}(GK'_j, HMAC(GK'_j, Update))$. Then, n_{H_j} transmits $Update$ and C_6 to its cluster members. 2) Each member n_{L_i} decrypts C_6 using the GK_j, verifies $HMAC(GK'_j, Update)$ and updates a cluster key as GK'_j. Then, each n_{L_i} sends the acknowledgement message to n_{H_j}.

E. Node Movement

When a node moves between clusters, the H-sensors must properly manage the cluster keys to ensure the forward/backward secrecy. Thus, the H-sensor updates the cluster key and notifies the BS of the changed node status. Through this report, the BS can immediately update the node status in the $\mathfrak{M}$. We denote a moving node as n_{L_m}.

1) Node Leave: A node may leave a cluster due to node failure, location change or intermittent communication failure. There are both proactive and reactive ways for the cluster head to detect when a node leaves the cluster. The proactive case occurs when the node n_{L_m} actively decides to leave the cluster and notifies the cluster head n_{H_j} or the cluster head decides to revoke the node. Since in this case n_{H_j} can confirm that the node has left, it transmits a report $E_{K^0_{H_j}}(NodeLeave, L_m)$ to inform the BS that n_{L_m} has left the cluster. After receiving the report, the BS updates the status of n_{L_m} in $\mathfrak{M}$ and sends an acknowledgement to n_{H_j}. The reactive case occurs when the cluster head n_{H_j} fails to communicate with n_{L_m}. It may happen that a node dies out of battery power, fails to connect to n_{H_j} due to interference or obstacles, is captured by the attacker or is moved unintentionally. Since the nodes in a cluster periodically exchange lightweight *beacon* messages, n_{H_j} can detect a disappeared node n_{L_m} when it does not receive the *beacon* message from n_{L_m} for a predetermined time period. So, n_{H_j} reports the status of the node n_{L_m} to the BS by sending $E_{K^0_{H_j}}(NodeDisappear, L_m)$. When the BS receives the report, it updates the status of n_{L_m} in the $\mathfrak{M}$ and acknowledges to n_{H_j}. Once n_{H_j} receives the acknowledgement from the BS, it changes its cluster key with the following operations: 1) n_{H_j} chooses a new cluster key GK'_j and computes $E_{k_{L_iH_j}}(GK'_j, NodeLeave, L_m)$ using pairwise session keys with each node in its cluster, except n_{L_m}. 2) Then, n_{H_j} sends $E_{k_{L_iH_j}}(GK'_j, NodeLeave, L_m)$ to each member node except n_{L_m}. 3) Each n_{L_i} decrypts it using $k_{L_iH_j}$ and updates the cluster key as GK'_j.

2) Node Join: Once the moving node n_{L_m} leaves a cluster, it may join other clusters or return to the previous cluster after some period. For the sake of simplicity, we assume that n_{L_m} wants to join the lth cluster or return to the jth cluster.

(i) *Join a New Cluster:* n_{L_m} sends a join request which contains L_{n+1} and $pk_{L_{n+1}}$ to join a lth cluster. After n_{H_l} receives the join request, n_{L_m} and n_{H_l} perform *Pairwise Key Generation* procedure to generate $K_{L_mH_l}$ and $k_{L_mH_l}$, respectively. Next, n_{H_l} transmits $E_{K^0_{H_l}}(NodeJoin, L_m)$ to the BS. The BS decrypts the message and validates whether n_{L_m} is a legitimate node or not and sends an acknowledgement to n_{H_l} if successful. The BS also updates the node member list, $\mathfrak{M}$. In case of node validation failure at the BS, n_{H_l} stops this process and revokes the pairwise key with n_{L_m}. Once n_{H_l}

receives the acknowledgement, it performs the *Cluster Key Update* process with all other nodes in the cluster. n_{H_l} also computes $E_{k_{L_m H_l}}(GK'_l, H_l, L_m)$, and sends it to the newly joined node n_{L_m}.

(ii) *Return to the Previous Cluster:* n_{L_m} sends a join request which contains L_{n+1} and $pk_{L_{n+1}}$ to join a jth cluster. Once n_{H_j} receives the join request, it checks a timer for n_{L_m} which is initially set to the T_{hold}. T_{hold} indicates the waiting time before discarding the pairwise master key when a L-sensor leaves. If n_{L_m} returns to the jth cluster before the timer expires, n_{L_m} and n_{H_j} perform only the *Pairwise Encryption Key Establishment* procedure to create a new pairwise encryption key, $k'_{L_m H_j}$. Otherwise, they perform the *Pairwise Key Generation* procedure to generate a new $K'_{L_m H_l}$ and $k'_{L_m H_l}$, respectively. Then, the cluster head n_{H_j} also updates the cluster key to protect backward key secrecy. Before updating the cluster key, n_{H_j} transmits $E_{K^0_{H_j}}(NodeReJoin, L_m)$ to the BS. Once the BS decrypts the message and determines that n_{L_m} is a valid node, the BS sends the acknowledgement to n_{H_l}. The BS then updates the member list $\mathfrak{M}$. Once n_{H_l} receives the acknowledgement, it performs the *Cluster Key Update* process with all other nodes in the cluster. Afterwards, n_{H_j} computes $E_{k'_{L_m H_j}}(GK'_j, H_j, L_m)$ and sends it to n_{L_m}.

F. Key Revocation

We assume that the BS can detect compromised *L-sensors* and *H-sensors*. The BS may have an intrusion detection system or mechanism to detect malicious nodes or adversaries [19], [20]. Although we do not cover how the BS can discover a compromised node or cluster head in this paper, the BS can utilize the updated node status information of each cluster to investigate an abnormal node. In our protocol, a cluster head reports the change of its node status to the BS, such as whenever a node joins or leaves a cluster. Thus, the BS can promptly manage the node status in the member list, $\mathfrak{M}$. For instance, the BS can consider a node as compromised if the node disappears for a certain period of time. In that case, the BS must investigate the suspicious node and it can utilize the node fault detection mechanism introduced in [21] and [22]. In this procedure, we provide a key revocation process to be used when the BS discovers a compromised node or a compromised cluster head. We denote a compromised node by n_{L_c} in the jth cluster for *a compromise node case* and a compromised head by n_{H_j} for *a compromise cluster head case*.

1) Compromised Node: The BS generates a $CompNode$ message and a $E_{K^0_{H_j}}(CompNode, L_c)$. Then it sends $E_{K^0_{H_j}}(CompNode, L_c)$ to all $n_{H_j}, (1 \le j \le N_2)$. After all H-sensors decrypt the message, they update the revocation list of their clusters. Then, if related keys with n_{L_c} exist, the related keys are discarded. Other than n_{L_c}, n_{H_j} performs the *Node leave* operations to change the current cluster key with the remaining member nodes.

2) Compromised Cluster Head: After the BS generates a $CompHeader$ message and a $E_{K^0_{L_i}}(CompHeader, H_j)$, it sends the message to all $n_{L_i} (1 \le i \le n)$ in the jth cluster. The BS also computes $E_{K^0_{H_i}}(CompHeader, H_j), (1 \le i \le N_2, i \ne j)$ and transmits it to all H-sensors except n_{H_j}. Once all nodes decrypt the message, they discard the related keys with n_{H_j}. Then, each n_{L_i} attempts to find other neighboring cluster heads and performs the *Join other cluster* steps of the *Node join* process with the neighboring cluster head. If some node n_{L_i} is unable to find another cluster head node, it must notify the BS by sending $E_{K^0_{L_i}}(FindNewCluste L_i)$. The BS proceeds to find the nearest cluster head n_{H_n} for n_{L_i} and connect n_{H_n} with n_{L_i}. Then, they can perform the *Join other cluster* steps.

G. Addition of a New Node

Before adding a new node into an existing networks, the BS must ensure that the node is not compromised. The new node $n_{L_{n+1}}$ establishes a full private/public key through the *node registration* phase. Then, the public system parameters, a full private/public key and individual key $K^0_{L_{n+1}}$ are stored into $n_{L_{n+1}}$. The BS generates $E_{K^0_{H_j}}(NewNode, L_{n+1}, pk_{L_{n+1}})$ and sends it to all n_{H_j}, $(1 \le j \le N_2)$. After $n_{L_{n+1}}$ is deployed in the network, it broadcasts an advertisement message which contains L_{n+1} and $pk_{L_{n+1}}$ to join a neighboring cluster. If multiple H-sensors receive $n_{L_{n+1}}$'s message, they will transmit a Response message to $n_{L_{n+1}}$. $n_{L_{n+1}}$ must choose one H-sensor for a valid registration. If $n_{L_{n+1}}$ selects n_{H_j} according to the distance and the strength of signal, it initiates the *Pairwise Key Generation* procedure. In order to provide backward secrecy, n_{H_j} performs *Cluster Key Update* procedure, where the $Update$ message contains L_{n+1} and $pk_{L_{n+1}}$. Then, n_{H_j} computes $C_7 = E_{k_{L_{n+1} H_j}}(GK'_j, H_j, L_{n+1})$, and sends C_7 and H_j to $n_{L_{n+1}}$. After $n_{L_{n+1}}$'s registration, n_{H_j} transmits $E_{K^0_{H_j}}(NodeJoin, L_{n+1})$ to the BS. Once the BS decrypts the message, it updates the status of the node $n_{L_{n+1}}$ in member list, $\mathfrak{M}$.

VI. Security Analysis

First, we briefly discuss the security of CL-HSC [13] which is utilized as a building block of CL-EKM. Later, we discuss how CL-EKM achieves our security goals. The CL-HSC [13] provides both confidentiality and unforgeability for signcrypted messages based on the intractability of the EC-CDH[1] Moreover, it is not possible to forge or expose the full private key of an entity based on the difficulty of EC-CDH, without the knowledge of both KGC's master private key and an entity's secret value. Here, the confidentiality is defined as *indistinguishability* against adaptive chosen ciphertext and identity attacks (IND-CCA2) while *unforgeability* is defined

[1]The Elliptic Curve Computational Diffie-Hellman problem (EC-CDH) is defined as follows: *Given a random instance* $(P, aP, bP) \in G_q$ *for* $a, b \in_R \mathbb{Z}^*_q$, *compute* abP.

as existential unforgeability against adaptive chosen messages and identity attacks (EUF-CMA). Further details on the CL-HSC scheme and its security proof are provided in [13].

A. Compromise-Resilience of CL-EKM

We assume that an adversary captures a node n_{L_i} in the jth cluster. This adversary can then extract the keys of n_{L_i}, such as the pairwise key shared with the cluster head n_{H_j}, the public/private key pair, the cluster key GK_j, and the individual key. However, the pairwise master/encryption key generation between any two nodes are independent of others, and hence each pair of nodes has different pairwise keys. Therefore, even if the adversary manages to obtain n_{L_i}'s keys, it is unable to extract any information useful to compromise the pairwise keys of other uncompromised nodes. Moreover, due to the intractability of EC-CDH problem, the adversary cannot obtain the KGC's master private key x from n_{L_i}'s public/private keys pk_{L_i}/sk_{L_i}. As a result, the compromise of a sensor does not affect the communication security among other L-sensors or H-sensors. Even though the attacker can read the group communications within the cluster with the cluster key extracted from the compromised node, it cannot get any information about the cluster key of other clusters.

B. Resistance Against Cloning and Impersonation Attack

An adversary can conduct the cloning attack if a node is captured; the key is then extracted and the node is replicated in another neighborhood. However, since the cluster head validates each node with the BS in the *node join* process of our CL-EKM, the BS is able to detect a cloned node when it is placed in an unintended cluster. After the BS investigates the cloned node, it revokes the node and notifies the node revocation to all cluster heads. Thus, although the cloned node may try to join other clusters, the cluster head will abort each attempt. Therefore, our scheme is resistant against the cloning attack.

The adversary may also attempt an impersonation attack by inserting an illegitimate node n_C. Assume that a node n_C poses as n_{L_i}. The node ID L_i and public key, $pk_{L_i} = (P_{L_i}, R_{L_i})$ are publicly known within the network. Hence, n_C can broadcast L_i and pk_{L_i}. When n_{L_j} receives the message, it will compute the pairwise master key $K_{L_iL_j}$, and the encapsulated key information $\varphi_{L_j} = (U_{L_j}, W_{L_j})$ towards establishing the pairwise *Master* key. As the next step, n_{L_j} sends $\langle \varphi_{L_j}, L_j, pk_{L_j} \rangle$ to n_C for *decapsulation*, which requires n_C to compute T_{L_j} as $(d_{L_i} \cdot U_{L_j})$. However, n_C fails to compute T_{L_j} since n_C has no knowledge of n_{L_i}'s partial private key d_{L_i}. Moreover due to the intractability of EC-CDH[1], the adversary cannot forge d_{L_i} without the knowledge of the KGC's master private key. Thus, n_C is unable to generate a legitimate pairwise master key, $K_{L_iL_j}$. However, n_C may try to establish the pairwise encryption with a random key K', rather than generating a legitimate master key. To this end, n_C chooses a random nonce r, computes an encryption key k' as $HMAC(r, K')$ and sends $\langle r, E'_k(r, L_i, L_j) \rangle$ to n_{L_j}. However, n_C cannot successfully pass the validation at n_{L_j}, since n_{L_j} first computes the pairwise encryption key with n_{L_j} as $k_{L_iL_j} = HMAC(r, K_{L_iL_j})$ and then tries to decrypt $E'_k(r, L_i, L_j)$ using $k_{L_iL_j}$. Thus, n_{L_j} fails to decrypt and hence, it does not confirm the pairwise encryption key to n_C, which is then reported to the BS. Thus, CL-EKM is resistant against impersonation attacks.

C. Forward and Backward Secrecy

In CL-EKM, messages exchanged between nodes or within a cluster are encrypted with the pairwise encryption key or cluster key. CL-EKM provides the *key update* and *revocation* processes to ensure forward secrecy when a node leaves or compromised node is detected. Using *key update* process, CL-EKM ensures backward secrecy when a new node joins.

Once a node is revoked from the network, all its keys are invalidated and the associated cluster key is updated. The cluster head sends the new cluster key to each cluster node, except the revoked node, by encrypting the key with the pairwise encryption key between the cluster and each intended node. Thus, the revoked node fails to decrypt any subsequent messages using the old pairwise encryption key or cluster key. When a node joins a cluster, the cluster head generates a new cluster key by choosing a new random value. Since the joined node receives the new cluster key, it cannot decrypt earlier messages encrypted using the older cluster keys.

D. Resistance Against Known-Key Attack

We assume that an adversary obtains the current pairwise encryption key $k_{L_iH_j} = HMAC(K_{L_iH_j}, r)$ between n_{L_i} and n_{H_j} and conducts the known-key attack. The adversary may attempt to extract the long term pairwise master key $K_{L_iH_j}$ using $k_{L_iH_j}$. However, due to the one-way feature of $HMAC(.)$, the adversary fails to learn $K_{L_iH_j}$. Also, when n_{L_i} and n_{H_j} update the pairwise encryption key as $k'_{L_iH_j} = HMAC(K_{L_iH_j}, r')$, the adversary cannot compute the updated pairwise encryption key $k'_{L_iH_j}$, without the knowledge of $K_{L_iH_j}$. Thus, CL-EKM is resistant against known-key attack when the pairwise encryption key is compromised.

VII. Performance Evaluation

We implemented CL-EKM in Contiki OS [29] and used Contiki port [28] of TinyECC [24] for elliptic curve cryptography library. In order to evaluate our scheme, we use the Contiki simulator COOJA. We run emulations on the state-of-the-art sensor platform TI EXP5438 which has 16-bit CPU MSP430F5438A with 256KB flash and 16KB RAM. MSP430F5438A has 25MHz clock frequency and can be lowered for power saving.

A. Performance Analysis of CL-EKM

We measure the individual performance of the three steps in the pairwise master/encryption key establishment process, namely, (i) encapsulation, (ii) decapsulation, and (iii) pairwise encryption key generation. We evaluate each step in terms of (i) computation time, and (ii) energy consumption. In this experiment, we vary the processing power i.e. CPU clock rate of the sensors since we consider heterogeneous

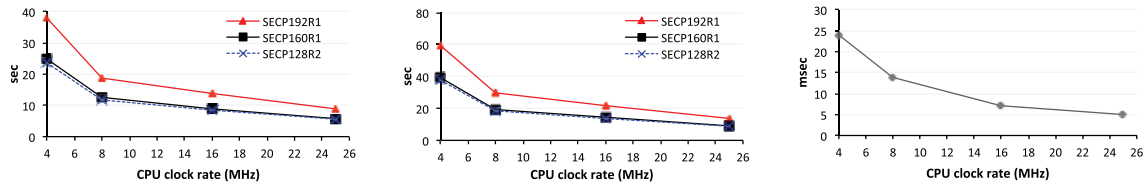


Fig. 2. Computation overhead for pairwise master/encryption key establishment. (a) Encapsulating key information. (b) Decapsulating key information. (c) Pairwise encryption key establishment.

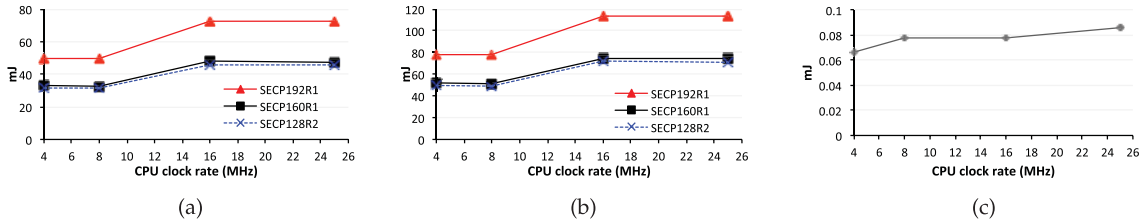


Fig. 3. Energy consumption for pairwise master/encryption key establishment. (a) Encapsulating key information. (b) Decapsulating key information. (c) Pairwise encryption key establishment.

WSNs with *H-sensors* being more powerful. Three different elliptic curves recommended by SECG (Standards for Efficient Cryptography Group) [30], i.e., (i) secp128r2 (128-bit ECC), (ii) secp160r1 (160-bit ECC), and (iii) secp192r1 (192-bit ECC), are used for the experiment.

Fig. 2 shows the time for the pairwise key generation process. As expected, the pairwise master key generation takes most of the time due to the ECC operations (See Fig. 2(a), 2(b)). However, it is important to mention that the pairwise master key is used only to derive the short-term pairwise encryption key. Once two nodes establish the pairwise keys, they do not require further ECC operations. Fig. 2(a) shows the computation times of the *encapsulation* process for various CPU clock rates of the sensor device. The computation time increases with the ECC key bit length. secp192r1 needs almost 1.5 times more time than secp160r1. secp128r2 takes approximately 4% less time than secp160r1. If CPU clock rate is set to 25MHz and secp160r1 is adopted, 5.7 seconds are needed for encapsulation of key. Fig. 2(b) shows the processing time for the *decapsulation*. Decapsulation requires about 1.57 times more CPU computation time than encapsulation. This is because decapsulation has six ECC point multiplications, whereas encapsulation includes only four ECC point multiplications. Finally, the computation time for pairwise encryption key establishment is shown in Fig. 2(c). At 25MHz CPU clock rate, it requires 5 ms, which is negligible compared to the first two steps. This is due to the fact that this step just needs one HMAC and one 128-bit AES operation. Next, we measure the energy consumption. As we can see from Fig. 3, the faster the processing power (i.e. CPU clock rate) is, the more energy is consumed. However, as shown in Fig. 3(a) and Fig. 3(b), there is no difference between 16MHz and 25MHz while 25MHz results in faster computation than 16MHz. In addition, secp160r1 might be a good choice for elliptic curve selection, since it is more secure than secp128r2 and consumes reasonable CPU time and energy for WSNs. In our subsequent experiments, we utilize secp160r1.

B. Performance Comparisons

In this section, we benchmark our scheme with three previous ECC-based key management schemes for dynamic WSNs: HKEP [15], MAKM [25] and EDDK [10]. Due to the variability of every schemes, we chose to compare a performance of the pairwise master key generation step because it is the most time consuming portion in each of the schemes. We measured the total energy consumption of computation and communication to establish a pairwise key between two L-sensors. For the experiment, we implemented four schemes on TI EXP5438 at 25MHz using ECC with secp160r1 parameters and AES-128 symmetric key encryption. EC point is compressed to reduce the packet size and LPL (Low Power Listening) is utilized for power conservation. Thus, sensors wake up for short durations to check for transmissions every second. If no transmission is detected, they revert to a sleep mode. To compute the energy consumption for communication, we utilize the energy consumption data of CC2420 from [27] and IEEE 802.15.4 protocol overhead data from [31]. We consider two scenarios as shown in Fig. 4. In the first scenario, two L-sensors lie within a 1-hop range, but the distance between the H-sensor and the L-sensor varies from 1 to 8 (see Fig. 4 (a)). In the second scenario, two L-sensors and the H-sensor lie in a 1-hop range, but the wireless channel conditions are changed (see Fig. 4 (b)). When a wireless channel condition is poor, a sender may attempt to resend a packet to a destination multiple times. Expected Transmission Count (ETX) is the expected number of packet transmission to be received at the destination without error. Fig. 5 shows the energy consumption of the four schemes for a pairwise key establishment when the number of hops between L-sensors and H-sensor, n, increases. When n is one, HKEP

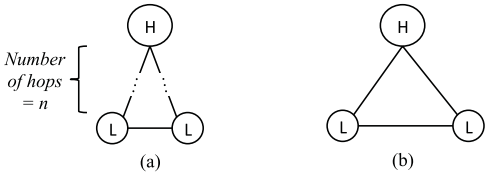


Fig. 4. Network topology.

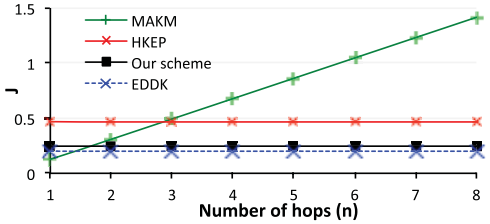


Fig. 5. Energy consumption comparison for pairwise key establishment in scenario (a).

requires more energy than our scheme because it performs six message exchanges to establish a pairwise key between two L-sensors, while our scheme needs just two message exchanges. When n is one, MAKM consumes the least energy because the L-sensor performs a single AES symmetric encryption, but other schemes run expensive ECC operations. However, as n increases, the energy of MAKM increases because the H-sensor is always involved in the generation of a pairwise key between two L-sensors. As a result, MAKM consumes more energy than our scheme when n is larger than one and the gap also widens when n increases. A packet delivery in a wireless sensor network is unreliable due to unexpected obstacles, time-varying wireless channel conditions, and a low-power transceiver. Fig. 6 shows the energy consumption of the four schemes for a pairwise key establishment when ETX varies from 1 to 4. As ETX increases, the energy consumption of HKEP increases more rapidly, because it requires six message exchanges. Also, HKEP is insecure, because the static private key of a node is exposed to the other node while the two nodes establish the session key. Although EDDK and MAKM may show better performance due to low computational overhead, the difference between MAKM and our scheme is only 0.121 J and the difference between EDDK and our scheme is 0.045 J. Both EDDK and MAKM are insecure against the known-key attack and do not provide a re-keying operation for the compromised pairwise key. EDDK also suffers from weak resilience to node compromises. Therefore, this performance evaluation demonstrates that overall, our scheme outperforms the existing schemes in terms of a better trade-off between the desired security properties and energy consumption including computational and communication overhead.

VIII. Simulation of Node Movements

A. Setting

We developed a simulator which counts the key-management-related events and yields total energy consumption for key-management-related computations using the data in Sec. 7.1. We focus on the effects of node movement and

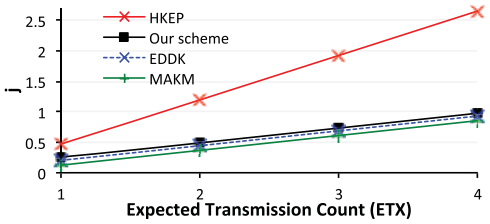


Fig. 6. Energy consumption comparison for pairwise key establishment in scenario (b).

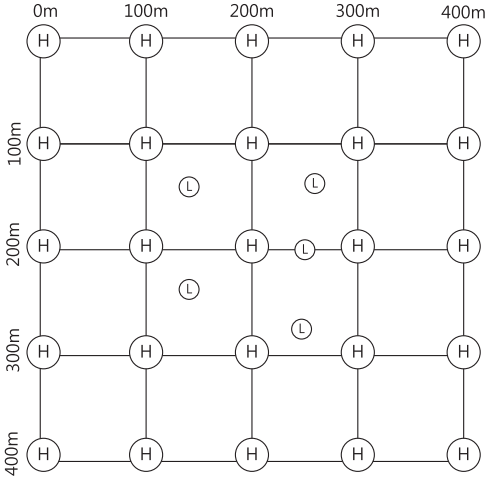
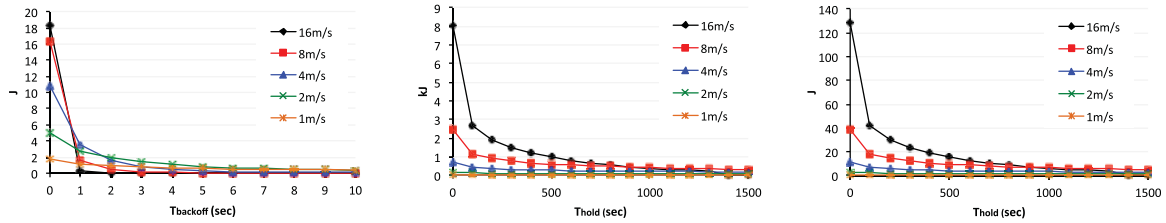


Fig. 7. Network topology for simulation.

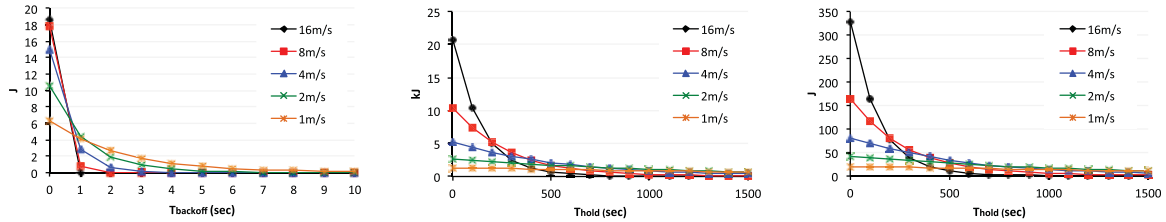
do not consider the impact of lower network layer protocols. We consider a $400 \times 400\ m^2$ space with 25 H-sensors placed on the grid corners (see Fig. 7). In CL-EKM, an H-sensor maintains two timers: $T_{backoff}$ and T_{hold} to efficiently manage the cluster when a node moves. $T_{backoff}$ denotes the cluster key update frequency. If $T_{backoff} = 0$, the cluster key is updated whenever a node joins or leaves. Otherwise, the H-sensor waits a time equal to $T_{backoff}$ after a node joins or leaves to update the cluster key. T_{hold} denotes the waiting time before discarding the pairwise master key when a L-sensor leaves. If $T_{hold} = 0$, the pairwise master key with a L-sensor is revoked right after the node leaves the cluster. Otherwise, the H-sensor stores the pairwise master key with the left L-sensor for a time equal to T_{hold}. For the movement of L-sensor, we adopt two well-known mobility models used for simulation of mobile ad-hoc network: the Random Walk Mobility Model and the Manhattan Mobility Model [32]. H-sensors are set to be stationary since they are usually part of the static infrastructure in real world applications.

1) Random Walk Mobility Model: The Random Walk Mobility Model mimics the unpredictable movements of many objects in nature. In our simulation, 1,000 L-sensors are randomly distributed. Each L-sensor randomly selects an H-sensor among the four H-sensors in its vicinity and establishes the pairwise key and cluster key. After the simulation starts, the L-sensors randomly select a direction and move at a random speed uniformly selected from $[0, 2V_L]$ (i.e., the mean speed $= V_L$). The new direction and speed are randomly selected every second. If a L-sensor crosses a line, it first checks whether it is still connected with its current H-sensor. If not, the node attempts to find an H-sensor which it had



(a) (b) (c)

Fig. 8. Node movement simulation results in random walk mobility model. (a) Energy consumption of one H-sensor for cluster key update for one day ($T_{hold} = 100$ sec). (b) Energy consumption of one H-sensor for pairwise key establishment for one day ($T_{backoff} = 6$ sec). (c) Energy consumption of one L-sensor for pairwise key establishment for one day ($T_{backoff} = 6$ sec).



(a) (b) (c)

Fig. 9. Node movement simulation results in Manhattan mobility model. (a) Energy consumption of one H-sensor for cluster key update for one day ($T_{hold} = 100$ sec). (b) Energy consumption of one H-sensor for pairwise key establishment for one day ($T_{backoff} = 6$ sec). (c) Energy consumption of one L-sensor for pairwise key establishment for one day ($T_{backoff} = 6$ sec).

previously connected to and it still maintains a pairwise master key. In the case of a failure, the node randomly selects an H-sensor among the surrounding H-sensors.

2) Manhattan Mobility Model: The Manhattan Mobility Model mimics the movement patterns in an urban area organized according to streets and roads. In our simulation, 1,000 L-sensors are randomly distributed and move in a grid. They can communicate with two adjacent H-sensors. Each L-sensor randomly selects its direction and chooses an H-sensor within its path as its cluster head. After the simulation starts, the L-sensors move at a random speed uniformly selected from $[0, 2V_L]$. At each intersection, a L-sensor has 0.5 probability of moving straight and a 0.25 probability of turning left or right. If a L-sensor arrives at a new intersection, it first chooses a new direction and checks whether it is still connected with its current H-sensor. If not, it chooses an H-sensor within its new vector as its new cluster head.

B. The Effect of $T_{backoff}$

Fig. 8(a) shows the energy consumption of an H-sensor for a cluster key update during the course of a day in a Random Walk Model. As $T_{backoff}$ increases, the energy consumption decreases since the number of cluster key updates is reduced. The faster V_L is, the more rapidly the energy consumption decreases as $T_{backoff}$ increases since L-sensors frequently cross the border lines. This tendency also shows in the Manhattan Mobility Model (see Fig. 9(a)). However, the H-sensors consume more energy at low speeds than in the Random Walk Mobility Model since the L-sensors do not change directions until they reach an intersection. A larger $T_{backoff}$ means a lower security level. Thus, there is a tradeoff between the security level and the energy consumption of the H-sensor. However, at high speeds, i.e., 16 m/s, $T_{backoff}$ should be less than 1 second since the number of cluster key updates is minimal when $T_{backoff}$ is greater than 1 second. However, at low speeds, 1, 2 or 3 seconds are a reasonable choice for the H-sensors.

C. The Effect of T_{hold}

Fig. 8(b) and Fig. 8(c) show the energy consumption of one H-sensor and one L-sensor for a pairwise key establishment in the Random Walk Mobility Model over the course of a day, respectively. The effect of T_{hold} increases as the node velocity increases. As T_{hold} increases, the energy consumption decreases because in the event that the L-sensors return to the old clusters before the timers expire, no new pairwise master key establishment is necessary. As shown in Table III the energy differences caused by node velocity and T_{hold} is due to the differences in the frequency of pairwise key establishment. Such frequency is linearly proportional to velocity increases. When T_{hold} ranges from 0 to 500 seconds, energy consumption rapidly decreases because several moving nodes may return to their previous clusters within the 500 seconds. However, when T_{hold} ranges from 500 to 1,500 seconds, the energy consumption decreases more slowly since the probability of nodes returning to their previous clusters is dramatically reduced.

In the Manhattan Mobility Model, when T_{hold} is small, more energy is consumed than in the Random Walk Mobility Model during pairwise key establishment since the L-sensors return to their previous clusters with low frequency. (see Fig. 9(b) and Fig. 9(c)). However, when T_{hold} is large, the energy consumed for the pairwise key establishment dramatically decreases. For instance, as shown in Table IV, when the node speed is 16 m/s and T_{hold} is 1,000 seconds, the number of pairwise key establishments is only 24,418 which is 5.4 times smaller

TABLE III
THE FREQUENCY OF PAIRWISE KEY ESTABLISHMENTS FOR ONE DAY IN RANDOM WALK MOBILITY MODEL

T_{hold}	1m/s	2m/s	4m/s	8m/s	16m/s
0	17,017	62,137	230,809	818,958	2,698,879
5,00	11,481	33,931	89,726	202,854	323,296
1,000	10,173	28,594	69,880	139,190	131,201
1,500	9,418	25,384	59,043	104,439	9,572

TABLE IV
THE FREQUENCY OF PAIRWISE KEY ESTABLISHMENTS FOR ONE DAY IN MANHATTAN MOBILITY MODEL

T_{hold}	1m/s	2m/s	4m/s	8m/s	16m/s
0	431,499	863,498	1727,442	3,455,079	6,911,556
5,00	351,965	569,724	715,244	583,129	227,390
1,000	285,616	358,632	294,203	118,162	24,418
1,500	225,772	228,858	130,095	35,159	12,582

than in the Random Walk Mobility Model with the same settings.

Similarly to $T_{backoff}$, a larger T_{hold} means a lower security level. Thus, T_{hold} should be selected according to V_L, energy consumption amount, and the desired security level. Results from Fig. 8(c) and Fig. 9(c) show that our scheme is practical for real-world monitoring applications such as in animal tracking or traffic monitoring. For example, L-sensors moving at 1 m/s in the Random Walk Mobility Model only use at most 0.67 J per day if T_{hold} is greater than 100 seconds. Also, L-sensors moving at 16 m/s in the Manhattan Mobility Model only use at most 1.16 J per day if T_{hold} is greater than 1,000 seconds. Considering that the average energy of one C-size alkaline battery is 34,398 J [23], the energy consumption of the L-sensor for a pairwise key establishment is relatively small.

IX. CONCLUSIONS AND FUTURE WORKS

In this paper, we propose the first certificateless effective key management protocol (CL-EKM) for secure communication in dynamic WSNs. CL-EKM supports efficient communication for key updates and management when a node leaves or joins a cluster and hence ensures forward and backward key secrecy. Our scheme is resilient against node compromise, cloning and impersonation attacks and protects the data confidentiality and integrity. The experimental results demonstrate the efficiency of CL-EKM in resource constrained WSNs. As future work, we plan to formulate a mathematical model for energy consumption, based on CL-EKM with various parameters related to node movements. This mathematical model will be utilized to estimate the proper value for the T_{hold} and $T_{backoff}$ parameters based on the velocity and the desired tradeoff between the energy consumption and the security level.

REFERENCES

[1] H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks," in *Proc. IEEE Symp. SP*, May 2003, pp. 197–213.

[2] W. Du, J. Deng, Y. S. Han, and P. K. Varshney, "A key predistribution scheme for sensor networks using deployment knowledge," *IEEE Trans. Dependable Secure Comput.*, vol. 3, no. 1, pp. 62–77, Jan./Mar. 2006.

[3] W. Du, J. Deng, Y. S. Han, P. Varshney, J. Katz, and A. Khalili, "A pairwise key predistribution scheme for wireless sensor networks," *ACM Trans. Inf. Syst. Secur.*, vol. 8, no. 2, pp. 228–258, 2005.

[4] M. Rahman and K. El-Khatib, "Private key agreement and secure communication for heterogeneous sensor networks," *J. Parallel Distrib. Comput.*, vol. 70, no. 8, pp. 858–870, 2010.

[5] M. R. Alagheband and M. R. Aref, "Dynamic and secure key management model for hierarchical heterogeneous sensor networks," *IET Inf. Secur.*, vol. 6, no. 4, pp. 271–280, Dec. 2012.

[6] D. S. Sanchez and H. Baldus, "A deterministic pairwise key predistribution scheme for mobile sensor networks," in *Proc. 1st Int. Conf. SecureComm*, Sep. 2005, pp. 277–288.

[7] I.-H. Chuang, W.-T. Su, C.-Y. Wu, J.-P. Hsu, and Y.-H. Kuo, "Two-layered dynamic key management in mobile and long-lived cluster-based wireless sensor networks," in *Proc. IEEE WCNC*, Mar. 2007, pp. 4145–4150.

[8] S. Agrawal, R. Roman, M. L. Das, A. Mathuria, and J. Lopez, "A novel key update protocol in mobile sensor networks," in *Proc. 8th Int. Conf. ICISS*, vol. 7671. 2012, pp. 194–207.

[9] S. U. Khan, C. Pastrone, L. Lavagno, and M. A. Spirito, "An energy and memory-efficient key management scheme for mobile heterogeneous sensor networks," in *Proc. 6th Int. Conf. CRiSIS*, Sep. 2011, pp. 1–8.

[10] X. Zhang, J. He, and Q. Wei, "EDDK: Energy-efficient distributed deterministic key management for wireless sensor networks," *EURASIP J. Wireless Commun. Netw.*, vol. 2011, pp. 1–11, Jan. 2011.

[11] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, "Comparing elliptic curve cryptography and RSA on 8-bit CPUs," in *Proc. 6th Int. Workshop Cryptograph. Hardw. Embedded Syst.*, 2004, pp. 119–132.

[12] S. S. Al-Riyami and K. G. Paterson, "Certificateless public key cryptography," in *Proc. 9th Int. Conf. ASIACRYPT*, vol. 2894. 2013, pp. 452–473.

[13] S. Seo and E. Bertino, "Elliptic curve cryptography based certificateless hybrid signcryption scheme without pairing," CERIAS, West Lafayette, IN, USA, Tech. Rep. CERIAS TR 2013-10, 2013. [Online]. Available: https://www.cerias.purdue.edu/apps/reports_and_papers/.Seung-Hyun

[14] S. H. Seo, J. Won, and E. Bertino, "POSTER: A pairing-free certificateless hybrid sign-cryption scheme for advanced metering infrastructures," in *Proc. 4th ACM CODASPY*, 2014, pp. 143–146.

[15] Q. Huang, J. Cukier, H. Kobayashi, B. Liu, and J. Zhang, "Fast authenticated key establishment protocols for self-organizing sensor networks," in *Proc. 2nd ACM Int. Conf. WSNA*, 2003, pp. 141–150.

[16] X.-J. Lin and L. Sun, "Cryptanalysis and improvement of a dynamic and secure key management model for hierarchical heterogeneous sensor networks," in *Proc. IACR Cryptol. ePrint Archive*, 2013, pp. 698–698.

[17] P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab, "NanoECC: Testing the limits of elliptic curve cryptography in sensor networks," in *Proc. 5th Eur. Conf. WSN*, vol. 4913. 2008, pp. 305–320.

[18] K. Chatterjee, A. De, and D. Gupta, "An improved ID-based key management scheme in wireless sensor network," in *Proc. 3rd Int. Conf. ICSI*, vol. 7332. 2012, pp. 351–359.

[19] W. T. Zhu, J. Zhou, R. H. Deng, and F. Bao, "Detecting node replication attacks in mobile sensor networks: Theory and approaches," *Secur. Commun. Netw.*, vol. 5, no. 5, pp. 496–507, 2012.

[20] M. A. Rassam, M. A. Maarof, and A. Zainal, "A survey of intrusion detection schemes in wireless sensor networks," *Amer. J. Appl. Sci.*, vol. 9, no. 10, pp. 1636–1652, 2012.

[21] P. Jiang, "A new method for node fault detection in wireless sensor networks," *Sensors*, vol. 9, no. 2, pp. 1282–1294, 2009.

[22] L. Paradis and Q. Han, "A survey of fault management in wireless sensor networks," *J. Netw. Syst. Manage.*, vol. 15, no. 2, pp. 171–190, 2007.

[23] (2013). *All About Battery*. [Online]. Available: http://www.allaboutbatteries.com/Energy-tables.html, accessed Dec. 2014.

[24] A. Liu and P. Ning, "TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks," in *Proc. Int. Conf. IPSN*, Apr. 2008, pp. 245–256.

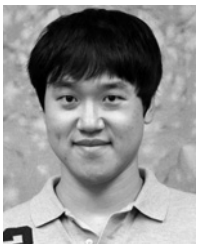
[25] D. Du, H. Xiong, and H. Wang, "An efficient key management scheme for wireless sensor networks," *Int. J. Distrib. Sensor Netw.*, vol. 2012, Sep. 2012, Art. ID 406254.

[26] X. He, M. Niedermeier, and H. de Meer, "Dynamic key management in wireless sensor networks: A survey," *J. Netw. Comput. Appl.*, vol. 36, no. 2, pp. 611–622, 2013.

[27] G. de Meulenaer, F. Gosset, O.-X. Standaert, and O. Pereira, "On the energy cost of communication and cryptography in wireless sensor networks," in *Proc. IEEE Int. Conf. Wireless Mobile Comput.*, Oct. 2008, pp. 580–585.

[28] (2010). *Asymmetric-Key Cryptography for Contiki*, http://publications.lib.chalmers.se/records/fulltext/129176.pdf, accessed Dec. 2014.
[29] (2013). *Contiki: The Open Source OS for the Internet of Things*, http://www.contiki-os.org/download.html, accessed Dec. 2014.
[30] (2000). *SEC 2: Recommended Elliptic Curve Domain Parameters, Certicom Corp*, http://www.secg.org/collateral/sec2_final.pdf, accessed May 2014.
[31] (2013). *Calculating 802.15.4 Data Rates*, http://www.nxp.com/documents/application_note/JN-AN-1035.pdf, accessed Dec. 2014.
[32] (2008). *A Survey of Mobility Models*, http://www.cise.ufl.edu/ helmy/papers/Survey-Mobility-Chapter-1.pdf, accessed Dec. 2014.

Seung-Hyun Seo is currently a Research Professor with Korea University, Seoul, Korea. She has been a Post-Doctoral Researcher of Computer Science with Purdue University, West Lafayette, IN, USA, since 2012, and a Senior Researcher with Korea Internet and Security Agency, Seoul, since 2010. She was a Researcher for three years with Financial Security Agency, Korea. She received the B.S., M.S., and Ph.D. degrees from Ewha Womans University, Seoul, Korea, in 2000, 2002, and 2006, respectively. Her main research interests include cryptography, mobile security, secure cloud computing, and malicious code analysis.



Jongho Won is currently pursuing the Ph.D. degree with the Department of Computer Science, Purdue University, West Lafayette, IN, USA. He received the B.S. and M.S. degrees from Seoul National University, Seoul, Korea, in 2006 and 2008, respectively. He joined the Department of Computer Science at Purdue University in 2011. His main research interests include security in wireless sensor network and smart grid.

Salmin Sultana is currently pursuing the Ph.D. degree in computer engineering with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA. She received the B.S. degree from the Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, in 2007. Her research interests include data provenance and security and fault tolerance of distributed systems, e.g., cloud computing, power grid, and high-performance computing. She is a member of the Center for Education and Research in Information Assurance and Security at Purdue University.

Elisa Bertino (F'02) is currently a Professor of Computer Science with Purdue University, West Lafayette, IN, USA, where she also serves as the Director of the Purdue Cyber Center (Discovery Park) and the Research Director of the Center for Education and Research in Information Assurance and Security. She was a faculty member with the Department of Computer Science and Communication, University of Milan, Milan, Italy. She received the B.S. and Ph.D. degrees in computer science from the University of Pisa, Pisa, Italy, in 1977 and 1980, respectively. Her main research interests include security, privacy, digital identity management systems, database systems, distributed systems, and multimedia systems. She is a fellow of the Association for Computing Machinery. She was a recipient of the 2002 IEEE Computer Society Technical Achievement Award for outstanding contributions to database systems and database security and advanced data management systems, and the 2005 IEEE Computer Society Tsutomu Kanai Award for pioneering and innovative research contributions to secure distributed systems.