Group Rekeying for Filtering False Data in Sensor Networks: A Predistribution and Local Collaboration-Based Approach

Wensheng Zhang and Guohong Cao
Department of Computer Science & Engineering
The Pennsylvania State University
University Park, PA 16802
Email: {wezhang,gcao}@cse.psu.edu

***Abstract*— When a sensor network is deployed in hostile environments, the adversary may compromise some sensor nodes, and use the compromised nodes to inject false sensing reports or modify the reports sent by other nodes. In order to defend against the attacks with low cost, researchers have proposed symmetric group key-based en-route filtering schemes, such as SEF [1] and I-LHAP [2]. However, if the adversary has compromised a large number of nodes, many group keys can be captured, and the filtering schemes may become ineffective or even useless. To deal with node compromise, the compromised nodes should be identified and the innocent nodes should update their group keys. Some existing intruder identification schemes can be used to identify the compromised nodes, but most existing group rekeying schemes are not suitable for sensor networks since they have large overhead and are not scalable. To address the problem, we propose a family of *predistribution and local collaboration-based group rekeying (PCGR)* schemes. These schemes are designed based on the ideas that future group keys can be preloaded to the sensor nodes before deployment, and neighbors can collaborate to protect and appropriately use the preloaded keys. Extensive analyses and simulations are conducted to evaluate the proposed schemes, and the results show that the proposed schemes can achieve a good level of security, outperform most previous group rekeying schemes, and significantly improve the effectiveness of filtering false data.**

Index Terms: System design, simulations, sensor networks, security, group rekeying.

I. Introduction

When a sensor network [3] is deployed in unattended and hostile environments such as battlefield, the adversary may capture and reprogram some sensor nodes, or inject some sensor nodes into the network and make the network accept them as legitimate nodes [4]. After getting control of a few nodes, the adversary can mount various attacks from inside the network. For example, a compromised node (intruder) may inject false sensing reports or maliciously modify reports that go through it. Under such attacks, the sink may receive incorrect sensing data and make wrong decisions, which may be dangerous in scenarios such as battlefield surveillance and environmental monitoring.

To defend against such attacks, the digital signature-based technique can be used to authenticate and filter false messages. However, this technique has high overhead both in terms of computation and bandwidth [5], which makes it unsuitable for sensor networks [6]. Therefore, researchers proposed to adopt symmetric cryptographic techniques such as the *statistical en-route filtering (SEF)* scheme [1] and the *interleaved hop-by-hop authentication (I-LHAP)* scheme [2], to address the problem. The basic idea of these schemes is as follows: Sensor nodes are randomly divided into multiple groups. Nodes in the same group share a symmetric group key and the final receiver (sink) knows all the group keys. Each message is attached with multiple MACs, each is generated using one group key. When such a message is forwarded along a path to the receiver, an en-route node may use its group key to verify the MACs carried in the message. Normally, an en-route node only knows one group key, so it cannot change a passing message or inject a false message without being detected by other en-route nodes who know different group keys. However, if the adversary has compromised a large number of nodes, many group keys may be captured, and the en-router filtering mechanisms may become inefficient or even useless. To deal with node compromises, the compromised nodes should be identified, and the innocent nodes should update their group keys to prevent the adversary from utilizing the captured keys.

To identify the compromised nodes, each node can use the *watchdog* mechanism [7] to monitor its neighbors and identify the compromised nodes when observing misbehaviors. The collaborative intruder identification scheme proposed by Wang

This work was supported in part by the National Science Foundation (CAREER CNS-0092770 and ITR-0219711).

et al. [8] can also be used to improve the accuracy.

The group key updating problem has been extensively studied in the context of secure multicast in wired or wireless networks. Many centralized solutions [9], [10], [11], [12] and a few distributed solutions [13] have been proposed. However, most of them are not suitable for sensor networks. For example, in SKDC [9], each key updating requires N encryptions (N is the number of nodes in the networks) and N key transmissions from the central controller to each individual node, which results in very high communication overhead and rekeying delay. The logic tree-based schemes proposed by Wallner *et al.* [10], Wong *et al.* [11], and Balenson *et al.* [12] can achieve logarithmic broadcast size, storage, and computational cost. However, the communication cost and the rekeying delay are still high when applied to a large scale network. Furthermore, the central controller has to trace the status of all nodes, and maintain a large logic tree connecting all the trusted nodes, which incurs high management overhead. As a distributed solution, Blundo's scheme [13] allows a set of nodes to set up a group key in a distributed way. However, it is still not scalable since the storage cost of each node increases rapidly as the group size increases and each node must know other trusted members in the same group.

To address the drawbacks of the existing group rekeying schemes, we propose a family of distributed and localized group rekeying schemes, called the *predistribution and local collaboration-based group rekeying (PCGR)* schemes. The design of these schemes are motivated by the following ideas: (1) Future keys can be preloaded to individual nodes before deployment to avoid the high overhead of securely and reliably disseminating new keys from a central key server to all trusted nodes at the key updating time. (2) Neighbors can collaborate with each other to effectively protect and appropriately use the preloaded keys; the local collaboration also relieves the high cost of the centralized management. Based on these ideas, we first propose a *basic PCGR (B-PCGR)* scheme. To address some security limitations of B-PCGR, we propose two enhanced PCGR schemes, i.e., the *cascading PCGR (C-PCGR)* scheme and the *random variance-based PCGR (RV-PCGR)*. Extensive analyses are conducted to evaluate the security level and the performance of the proposed schemes, as well as comparing the performance of the proposed schemes with some existing group rekeying schemes. Simulations are used to evaluate the effectiveness of the proposed group rekeying scheme in filtering false data. The analyses and simulation results show that the proposed schemes can achieve a good level of security, outperform most previously proposed schemes, and significantly improve the effectiveness of filtering false data with low overhead.

The rest of the paper is organized as follows: The next section presents the system model. In Section III, we describe and analyze the basic PCGR scheme. The enhanced PCGR schemes are presented in Section IV. Section V reports the performance evaluation results. Section VI discusses some issues related to the proposed schemes. Section VII concludes the paper.

II. The System Model

We consider a large scale wireless sensor network which is deployed in a hostile environment; e.g., a sensor network deployed in a battlefield for tracking enemy tanks [14], [15]. The network is composed of low-complexity sensor nodes, e.g. the Berkeley MICA mote [16], which has a processor running at 4 MHz and a 4KB RAM for data storage. These nodes have limited power supply, storage space, and computation capability. Therefore, public key-based operations cannot be afforded. On the other hand, each node has enough space for storing a few kilobytes of keying information.

Node deployment is managed by a central controller (setup server), which is responsible for assigning group keys and preloading some keying information to a node before it is deployed. We assume that each node is innocent before deployment, and cannot be compromised during the first several minutes after deployment [17] since compromising a node takes some time. Also, each pair of neighboring nodes can establish a pairwise key using some existing techniques [18], [19], [20], [21]. When two neighbors exchange some messages for key updating, the messages must be encrypted using their pairwise key to prevent eavesdropping.

To defend against compromised nodes (or outside intruders) from injecting false reports or modifying the reports generated by other innocent nodes, the statistical en-route filtering (SEF) mechanism [1] is used to detect and drop false messages. We assume that a compromised node can eventually be detected by most of its neighbors within a certain time period. To achieve this, the watchdog mechanism [7] and some collaborative intruder detection and identification schemes [8] can be used. We also assume that nodes are loosely synchronized, and group rekeying is started periodically [22].

III. The Basic Predistribution and Local Collaboration-Based Group Rekeying (B-PCGR) Scheme

In this section, we first present the basic idea of the B-PCGR scheme and then give a detailed description. Finally, we analyze its security property.

A. The Basic Idea

The B-PCGR scheme includes the following three steps.

1) Group Key Predistribution: Before a node is deployed, it is randomly assigned to a group, and is preloaded with the current and all future keys of the group. The keys are represented by a polynomial called *group key polynomial (g-polynomial)*. Compared to most existing group rekeying protocols, in which new group keys are generated and distributed at the key updating time, the B-PCGR scheme can significantly reduce the communication overhead and the key updating delay, since the keys are preloaded.

2) Local Collaboration-Based Key Protection: Since all group keys are preloaded, it is important to protect the keys from being exposed to intruders. For this purpose, nodes should not explicitly keep the future group keys, because the keys can be captured by an adversary when the node is compromised. Based on the assumption that every node is innocent at least during the first few minutes after deployment, we propose a local collaboration-based group key protection technique as follows:

- Each node randomly picks a polynomial, called *encryption polynomial (e-polynomial)*, to encrypt its g-polynomial. We call the encrypted g-polynomial *g'-polynomial*.
- Some shares of the e-polynomials are distributed to its neighbors.
- The node removes its g-polynomial and e-polynomial, but keeps its current key and its g'-polynomial.

After the above steps, a node can not access its future group keys without collaborating with a certain number of neighbors, each of which has a share of its e-polynomial.

3) Local Collaboration-Based Group Key Updating: At the time of group key updating, every innocent node needs to receive a certain number of e-polynomial shares from its trusted neighbors. Also, the received shares can only be used to calculate one instance of the e-polynomial which is necessary for computing the new group key. This group key updating mechanism guarantees that a node can compute its new group key as long as it is trusted by a certain number of neighbors; meanwhile, the node can not derive any group keys that should not be disclosed at this time.

B. Detailed Description of B-PCGR

1) Predistributing g-Polynomials: Initially, the setup server decides the total number of groups. For each group i, a unique t-degree (t is a system parameter) univariate g-polynomial $g_i(x)$ is constructed over a prime finite field $F(q)$ to represent the keys of the group, where $g_i(0)$ is the initial group key, $g_i(j)$ ($j \geq 1$) is the group key of version j, and q is a large prime whose size can accommodate a group key.

Before a node N_u is deployed, the setup server randomly assigns it to a group, and preloads the g-polynomial of the group, denoted as $g(x)$, to it.

2) Encrypting g-Polynomials and Distributing the Shares of the e-Polynomials: After N_u has been deployed and has discovered its neighbors, it randomly picks a bivariate e-polynomial

$$e_u(x,y) = \sum_{0 \leq i \leq t, 0 \leq j \leq \mu} A_{i,j} x^i y^j, \tag{1}$$

where μ is a system parameter.

Using the e-polynomial (i.e., $e_u(x,y)$), as shown in Figure 1 (b), N_u encrypts its g-polynomial (i.e., $g(x)$) to get its g'-polynomial (denoted as $g'(x)$). The encryption is conducted as follows:

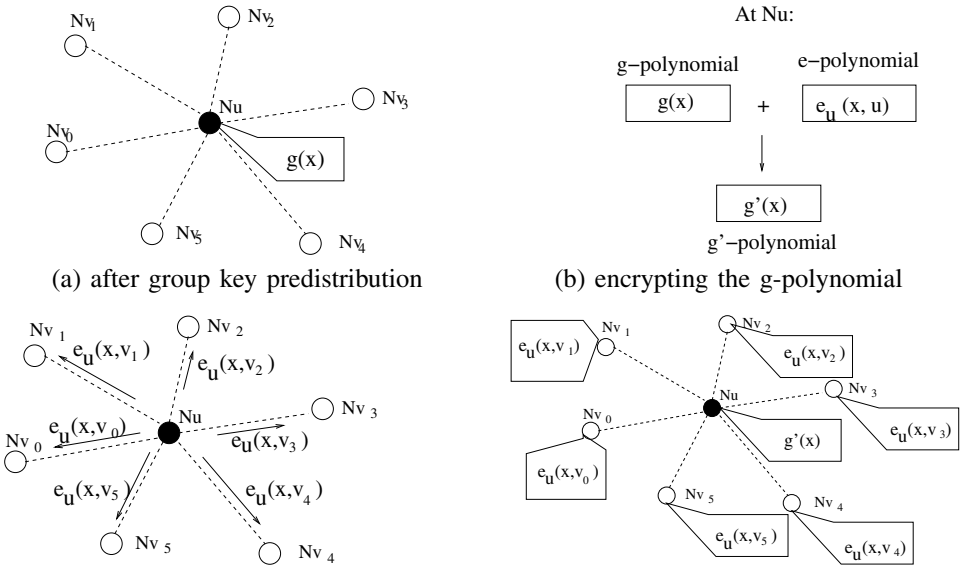
$$g'(x) = g(x) + e_u(x,u). \tag{2}$$

After that, as shown in Figure 1 (c), N_u distributes the shares of $e_u(x,y)$ to its n neighbors N_{v_i} ($i = 0, \cdots, n-1$). Specifically, each neighbor N_{v_i} receives share $e_u(x,v_i)$. At the same time, N_u removes $e_u(x,y)$ and $g(x)$, but keeps $g'(x)$. The final distribution of $g'(x)$ and $e_u(x,v_i)$ is illustrated in Figure 1 (d).

3) Key Updating: Each node maintains a *rekeying timer*, which is used to periodically notify the node to update its group key, and the current version of the group key (denoted as c). Note that c is initialized to 0 when the node is deployed.

To update group keys, each innocent node N_u increases its c by one, and returns share $e_{v_i}(c,u)$ to each trusted neighbor N_{v_i}. Meanwhile, as shown in Figure 2 (a), N_u receives a share $e_u(c,v_i)$ from each trusted neighbor N_{v_i}. Having received $\mu+1$ shares, which are denoted as $\{\langle v_i, e_u(c,v_i)\rangle, i = 0, \cdots, \mu\}$, N_u can reconstruct a unique μ-degree polynomial

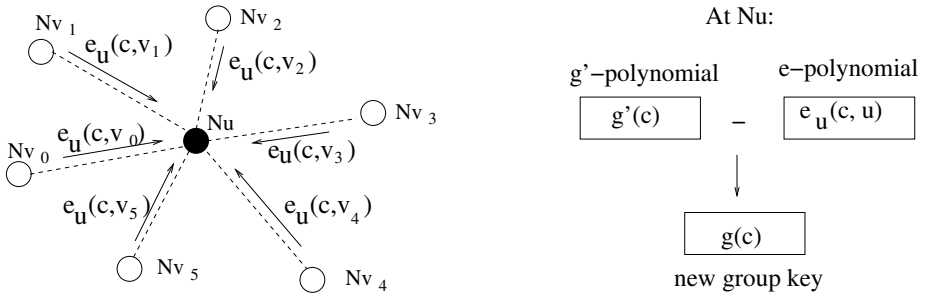
$$e_u(c,y) = \sum_{j=0}^{\mu} B_j y^j, \tag{3}$$



(a) after group key predistribution (b) encrypting the g-polynomial

(c) distributing the shares of the e-polynomial (d) after polynomial encryption and share distribution

Fig. 1. B-PCGR: Polynomial encryption and share distribution



(a) (b)

Fig. 2. B-PCGR: Key updating

by solving the following μ+1 (μ+1)-variable linear equations:

$$\sum_{j=0}^{\mu}(v_i)^j B_j = e_u(c, v_i), \qquad i = 0, \cdots, \mu. \tag{4}$$

Having known $e_u(c, x)$, as shown in Figure 2 (b), N_u can compute its new group key $g(c) = g'(c) - e_u(c, u)$.

C. Security Analysis

The following theorem shows the security property of the B-PCGR scheme.

Theorem 1: For a certain group, its g-polynomial $g(x)$ is compromised if and only if:

(1.1) a node (N_u) of the group is compromised, and

(1.2) at least $\mu + 1$ neighbors of N_u are compromised;

or

(2) at least $t+1$ past keys of the group are compromised.

Proof: (sketch) First, we prove that: if condition (1.1) and (1.2) or (2) is satisfied, $g(x)$ can be compromised.

Assume that N_u and its $\mu + 1$ neighbors, i.e., N_{v_i} ($i = 0, \cdots, \mu$), have been compromised. The adversary can obtain $g'(x)$ and $e_u(x, v_i)$ ($i = 0, \cdots, \mu$). For an arbitrary x', a unique polynomial

$$e_u(x', y) = \sum_{j=0}^{\mu} B'_j y^j \tag{5}$$

can be reconstructed by solving the following $\mu + 1$ ($\mu + 1$)-variable linear equations:

$$\sum_{j=0}^{\mu}(v_i)^j B'_j = e_u(x', v_i), \text{where } i = 0, \cdots, \mu. \tag{6}$$

Knowing $e_u(x', y)$, $g(x')$ can be computed as follows:

$$g(x') = g'(x') - e_u(x', u) \tag{7}$$

Similarly, we can prove that: if condition (2) is satisfied, $g(x)$ can be compromised.

Second, we prove that: if condition (1.1) is satisfied, but condition (1.2) and (2) are not satisfied, $g(x)$ can not be

compromised. Assume that N_u and its $m \leq \mu$ neighbors, i.e., N_{v_i} $(i = 0, \cdots, m-1)$, are compromised. Thus, for an arbitrary x', the adversary can obtain $g'(x')$ and $e_u(x', v_i)$ $(i = 0, \cdots, m-1)$. Since $m \leq \mu$ and $e_u(x', y)$ is a μ degree polynomial of variable y, similar to the proof in [13], we prove in the following that the adversary can not find out $e_u(x', u)$:

We consider the worst case that $m = \mu$. Suppose the adversary guesses $e_u(x', u) = a$. A unique polynomial $e_u(x', y)$ (shown in Eq. (5)), which is a μ-degree polynomial of y, can be constructed by solving the following linear equations

$$\begin{cases} \sum_{j=0}^{\mu} (v_i)^j B'^j = e_u(x', v_i), \ i = 0, \cdots, \mu - 1 \\ \sum_{j=0}^{\mu} u^j B'^j = a. \end{cases} \quad (8)$$

However, since a can be an arbitrary value in $F(q)$, the adversary can construct q different polynomials $e_u(x', y)$. Also, since $e_u(x, y)$ is randomly chosen and hence $e_u(x', y)$ can be an arbitrary polynomial; i.e., the q polynomials the adversary can construct are equally likely to be the actual $e_u(x', y)$. Formally,

$$\begin{aligned} &\forall a, Pr(e_u(x', u) = a \mid \{e_u(x', v_i), 0 \leq i \leq \mu - 1\}) \\ &\equiv Pr(e_u(x', u) = a). \end{aligned} \quad (9)$$

Therefore, the adversary can not derive $e_u(x', u)$.

Without knowing $e_u(x', u)$, the adversary can not find out $g(x)$, which is equal to $g'(x) - e_u(x, u)$. On the other hand, since condition (2) is not satisfied, we assume that the adversary also knows $l \leq t$ keys of the considered group. Without loss of generality, we further assume that the compromised keys are $g(0), g(1), \cdots, g(l-1)$. Because $l \leq t$ and $g(x)$ is a t-degree polynomial of variable x, the adversary can not find out $g(x')$, either.

Similar to the above proof, we can also prove that: if condition (1.2) is satisfied, but condition (1.1) and (2) are not satisfied, $g(x)$ can not be compromised.

■

IV. Enhancements

The B-PCGR scheme is effective on the condition that: (1) no node will be compromised together with $\mu + 1$ or more neighbors, and (2) the adversary can not obtain $t + 1$ or more keys from the same group. In some hostile scenarios, the above conditions can be violated. To deal with these limitations, we propose two enhancements.

A. Cascading PCGR (C-PCGR) Scheme

The C-PCGR scheme is proposed to address the first limitation of B-PCGR. In this scheme, the e-polynomial shares of N_u are distributed to its multi-hop neighbors, instead of only to its one-hop neighbors; at the same time, the e-polynomial shares are distributed/collected in a cascading way, and hence does not introduce much communication/storage overhead.

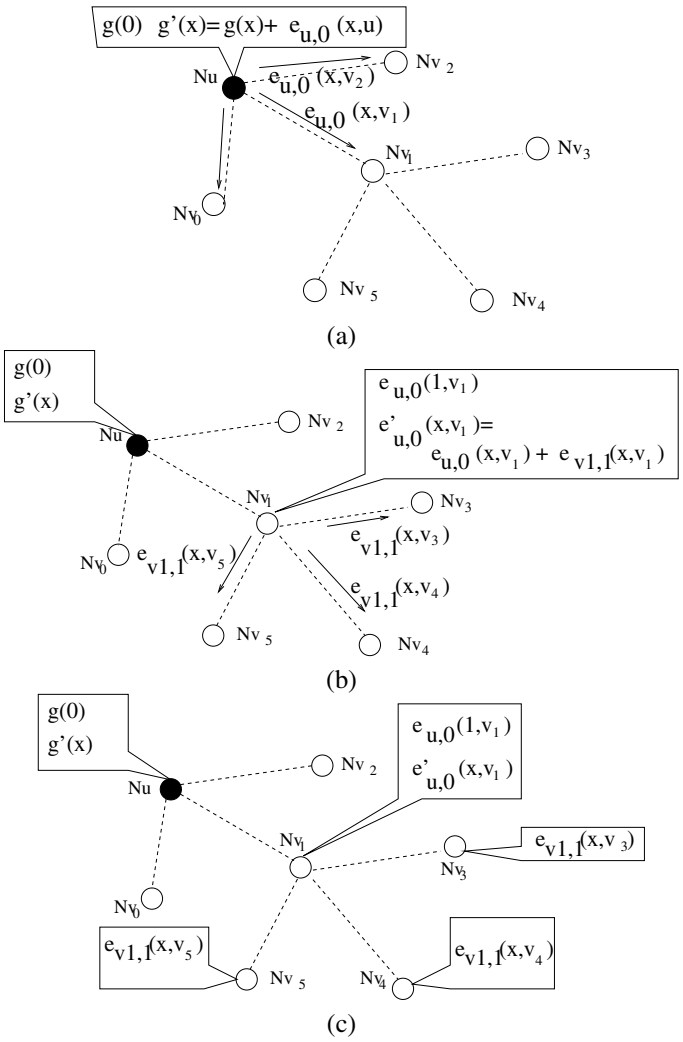


Fig. 3. C-PCGR: Polynomial Decryption and Share Distribution ($c = 0$)

1) The Scheme: The C-PCGR scheme is designed based on the B-PCGR scheme, and it also includes three steps. However, it differs from B-PCGR in the second and the third steps, which are described in the following. To simplify the presentation, we only describe the case where the e-polynomial shares are distributed to its 1- and 2-hop neighbors, while the scheme can be extended to more general cases.

Polynomial Encryption and Share Distribution

After each node (N_u) has been deployed and has discovered its neighbors, it randomly picks two e-polynomials: one is called *0-level e-polynomial* (denoted as $e_{u,0}(x, y)$), and the other is called *1-level e-polynomial* (denoted as $e_{u,1}(x, y)$). In both e-polynomials, the degree of x and y are t and μ,

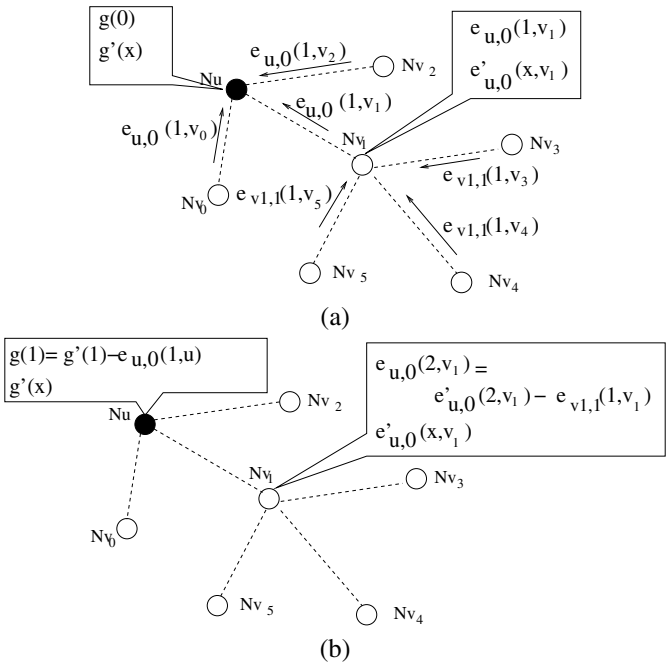


Fig. 4. C-PCGR: Key Updating (c is increased from 0 to 1)

respectively.

Using the 0-level e-polynomial (i.e., $e_{u,0}(x, y)$), each node N_u can encrypt its g-polynomial (i.e., $g(x)$) to get its g'-polynomial (i.e., $g'(x)$). The encryption is conducted as follows:

$$g'(x) = g(x) + e_{u,0}(x, u). \tag{10}$$

After that, as shown in Figure 3 (a), N_u keeps $g(0)$ (i.e., the current group key), removes $g(x)$ and $e_{u,0}(x, u)$, and distributes the shares of $e_{u,0}(x, y)$ to its neighbors. Specifically, each neighbor N_v is given $e_{u,0}(x, v)$.

Having received 0-level e-polynomial shares from its neighbors, as shown in Figure 3 (b), each node (say N_v) uses its 1-level e-polynomial (i.e., $e_{v,1}(x, y)$) to encrypt each received 0-level share (i.e., $e_{u,0}(x, v)$) to obtain

$$e'_{u,0}(x, v) = e_{u,0}(x, v) + e_{v,1}(x - 1, v) \tag{11}$$

After that, N_v keeps $e'_{u,0}(x, v)$ and $e_{u,0}(c+1, v)$, which will be returned to N_u at the next key updating time. It also removes $e_{u,0}(x, v)$, and distributes the shares of its 1-level e-polynomial ($e_{v,1}(x, y)$) to its neighbors. Figure 3 illustrates how the e-polynomial shares of N_u are distributed to its 1-hop and 2-hop neighbors.

Key Updating

To update keys, as shown in Figure 4 (a), each innocent node N_u increases its c by one, and returns shares $e_{v,0}(c, u)$ and $e_{v,1}(c, u)$ to each trusted neighbor N_v (Here, we assume that N_u has received shares $e_{v,0}(x, u)$ and $e_{v,1}(x, u)$ from N_v before.). At the same time, N_u receives its own 0-level and 1-level e-polynomial shares from its neighbors (i.e., $e_{u,0}(c, v)$ and $e_{u,1}(c, v)$ from each trusted neighbor N_v).

Having received $\mu + 1$ 0-level e-polynomial shares, as shown in Figure 4 (a), N_u reconstructs a unique polynomial $e_{u,0}(c, x)$. Knowing $e_{u,0}(c, x)$, N_u can compute its new group key $g(c) = g'(c) - e_{u,0}(c, u)$.

Having received $\mu+1$ 1-level e-polynomial shares, as shown in Figure 4 (a), N_v computes a unique polynomial $e_{v,1}(c, x)$, and then generates a share $e_{u,0}(c + 1, v) = e'_{u,0}(c + 1, v) - e_{v,1}(c, v)$, which will be returned to neighbor N_u at the next key updating time.

2) Security Analysis: The security property of the C-PCGR scheme can be expressed by Theorem 2.

Theorem 2: For a certain group, its g-polynomial $g(x)$ can be compromised if and only if:

(1.1) a node N_u in the group is compromised, and

(1.2) the adversary has compromised at least $\mu + 1$ neighbors of N_u, each of which also has $\mu+1$ neighbors compromised;

or

(2) at least $t + 1$ past keys of group i are compromised.

Proof: Similar to the proof of Theorem 1. ■

B. Random Variance-Based PCGR (RV-PCGR) Scheme

The RV-PCGR scheme aims to address another limitation of B-PCGR. Specifically, as shown in Figure 5 (a), if the adversary has obtained $t + 1$ keys of a certain group, e.g., $g(0), g(1), \cdots, g(t)$, the adversary can break the g-polynomial of the group (i.e., $g(x)$) based on these keys.

1) Basic Idea: The basic idea of the RV-PCGR scheme is illustrated in Figure 5 (b). Let the length of $g(j)$ be $2L$ bits. We can add a L bit random number (called *random variance*) σ_j to $g(j)$ to obtain $g^r(j)$, such that the highest L bits of $g^r(j)$ are the same as those of $g(j)$, but the lowest L bits are different. This can be achieved by constructing the polynomials and conducting the addition/multiplication operations over an extended finite field [23] $F(2^{2L})$, in which the addition operation is defined as *modulo-2 addition*. Using some techniques (presented next), we can guarantee that the adversary can obtain only $g^r(j)$, not knowing the original $g(j)$. Based on $g^r(j)$ ($j = 0, \cdots, t$), the adversary can construct a t-degree polynomial $g^r(x)$, but $g^r(x)$ is different from $g(x)$. That is, the adversary cannot break the future keys of the group.

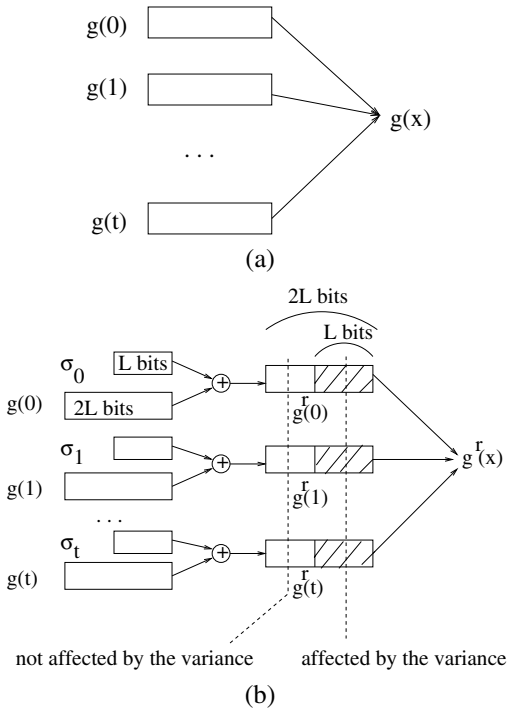


Fig. 5. Basic Idea of RV-PCGR

Certainly, the adversary may guess each σ_j and attempts to find out each original $g(j)$ ($g(j) = g^r(j) \oplus \sigma_j$). However, since each σ_j can be an arbitrary number picked from $\{0, \cdots, 2^L - 1\}$, the probability of guessing correctly is $\frac{1}{2^{(t+1)L}}$. Consider an example, in which $t = 9$ and $L = 64$, the probability is as low as $\frac{1}{2^{640}}$. In the following, the scheme for implementing the above basic idea is described in detail.

2) The Scheme: The RV-PCGR scheme also has three steps.

Predistributing G-Polynomials

Similar to B-PCGR, the setup server decides the total number of groups, and picks a t-degree univariate g-polynomial for each group. Also, each node N_u is given a g-polynomial $g(x)$ when it is deployed.

Different from B-PCGR, each $g(x)$ is constructed over an extended finite field $F(2^{2L})$, where L is the length of a group key (e.g., $64bits$). Also, the group key of any version j is defined as the highest L bits of $g(j)$, instead of $g(j)$ itself.

Encrypting G-Poly. and Distributing Components

Similar to B-PCGR, after N_u has been deployed and has discovered its neighbors, it randomly picks a t-degree e-polynomial $e_u(x)$[1]. Using the e-polynomial ($e_u(x)$), N_u encrypts its g-polynomial ($g(x)$) to get its g'-polynomial ($g'(x) = g(x) \oplus e_u(x)$). After that, N_u randomly decomposes $e_u(x)$ into $\mu + 1$ components, denoted as $e_{u,i}(x)$ ($i = 0, \cdots, \mu$), such that

$$\sum_{i=0}^{\mu} e_{u,i}(x) = e_u(x). \tag{12}$$

These components are evenly distributed to the neighbors, and each neighbor gets only one components.

Key Updating

To update keys, each innocent node N_u increases its key version counter c by one, and returns

$$e^r_{v,j}(c) = e_{v,j}(c) \oplus \sigma'_{c,v} \tag{13}$$

to each trusted neighbor N_v. Here, we assume that N_u has received share $e_{v,j}(x)$ from N_v before, and $\sigma'_{c,v}$ is randomly picked from $\{0, \cdots, 2^L - 1\}$. At the same time, N_u also receives $e^r_{u,i}(c)$ from N_v.

Having received $\mu + 1$ distinct shares $\langle v_i, e^r_{u,i}(c)\rangle$ ($i = 0, \cdots, \mu$), N_u computes $e^r_u(c) = \sum_{i=0}^{\mu} e^r_{u,i}(c)$. Knowing $e^r_u(c)$, N_u can compute $g^r(c) = g'(c) \oplus e^r_u(c)$, and the highest L bits of $g^r(c)$ are used as the new group key.

3) Security Analysis: During the key updating process, the share returned by each node is added a random variance (refer to Eq. (13)). Therefore, the $g^r(c)$ calculated by node N_u has already included a random variance. Without loss of generality, we suppose the adversary has obtained $g^r(i)$, where $i = 0, \cdots, n-1$ and $n \geq t+1$. The adversary can use the following process to guess the original $g(x)$, where

$$g(x) = \sum_{j=0}^{t} D_j x^j. \tag{14}$$

For every distinct $(t+1)$-tuple of variances $\langle \sigma_0, \cdots, \sigma_t \rangle$, where $\sigma_j \in \{0, 1, \cdots, 2^L - 1\}$ ($k = 0, \cdots, t$):

(1) The guessed value of each D_j, denoted as $\hat{D}_j$, is calculated by solving the following $t+1$ $(t+1)$-variable linear equations:

$$\sum_{j=0}^{t} (i)^j \hat{D}_j = g^r(i) \oplus \sigma_i, \text{ where } i = 0, \cdots, t. \tag{15}$$

(2) Let $g(x) = \sum_{j=0}^{t} \hat{D}_j(x)$. If the highest L bits of $g(i)$ are the same as those of $g^r(i)$, where $i = t+1, \cdots, n-1$, this $g(x)$ is recorded as a *candidate polynomial* of the original $g(x)$.

If the total number of the recorded candidate polynomials is 1, the candidate polynomial is the original $g(x)$; otherwise, one of the candidate polynomials is randomly picked as the

[1]In this subsection, the e-polynomial shares distributed to a neighbor is randomly constructed, irrelevant to the identity of the receiver. Thus, we can remove the second parameter y in the polynomial $e_u(x, y)$ as in B-PCGR.

guessed $g(x)$. Obviously, the original $g(x)$ is among the recorded candidate polynomials. However, the complexity to find out the candidates is as high as $o(2^{(t+1)L})$. For example, if $t = 9$ and $L = 64$, the complexity is 2^{640}.

V. Performance Evaluations

In this section, we first analytically compare the performance of the proposed PCGR schemes and some previously proposed group rekeying schemes. Then, we conduct simulations to show that the proposed PCGR scheme can significantly improve the performance of filtering false messages.

A. Performance Analysis

Before analyzing the performance of different schemes, we list some notations that are used in this section as follows:

- N: the total number of nodes in the network.
- n: the average number of trusted neighbors that a node has.
- n_c: the number of (compromised) nodes that should be evicted.
- L: the length (in bits) of a group key.

1) Comparing the Performance of the PCGR Schemes: We compare the performance of the proposed PCGR schemes in terms of communication cost, computation overhead, and storage requirement. The main results are shown in Table I.

Communication Cost

In the B-PCGR scheme, each innocent node needs to send out n messages to its trusted neighbors during each key updating process. Each message includes one share, which has L bits. Therefore, nL bits are sent out by each node.

Similar to B-PCGR, the C-PCGR scheme also requires each innocent node to send out n messages for key updating. However, the size of each message includes two shares, which has $2L$ bits. Therefore, each node has to send out $2nL$ bits.

In RV-PCGR, each innocent node should send out n shares to its trusted neighbors, and each share has $2L$ bits. Therefore, each node needs to sent out $2nL$ bits for each key updating.

Computational Overhead

In all three schemes, each share sent/received by a node should be encrypted/decrypted using pairwise keys to prevent eavesdropping, and the total number of messages sent/received is $2n$ during each key updating process. Therefore, each node needs $2n$ encryptions/decryptions. Next, we discuss other computation overhead of each scheme.

In the B-PCGR scheme, each node N_u first needs to evaluate $n + 1$ t-degree polynomials ($e_{v_i}(c)$ and $g'(x)$) to compute n shares and $g'(c)$, which needs $o((n + 1)t^2)$ multiplications. After receiving $\mu + 1$ or more shares, it also needs to solve a $(\mu + 1)$-variable linear equation system to compute $e_u(c, u)$, and the computational complexity of using Gaussian elimination to solve such an equation system is $o(\mu^3)$ multiplications/divisions. The overall computational complexity is $o((n+1)t^2+\mu^3)$ multiplications/divisions over $F(q)$.

In C-PCGR, each node needs to compute n more shares and solve one more $(\mu + 1)$-variable linear equation system, so the total computational complexity is $o((2n + 1)t^2 + 2\mu^3)$ multiplications/division over $F(q)$

In RV-PCGR, each node also needs to evaluate $n + 1$ t-degree polynomials to compute n shares and $g'(c)$. However, after receiving $\mu + 1$ or more shares, it only needs to add up these shares, instead of solving an equation system. Therefore, the total computational complexity is $o((n + 1)t^2)$ multiplications. However, the multiplications are conducted over field $F(2^{2L})$.

To reduce rekeying delay caused by computations, most of the above computations (i.e., computing and encrypting shares, as well as computing $g'(c)$) can be performed beforehand, and only a few other computations (i.e., decrypting shares and computing $e_u(c, u)$) should be performed during the key updating time. To distinguish these two types of computational overhead, as shown in Table I, we list both the computational overhead at key updating time and the total computational overhead for each key updating.

Storage Requirements

In the B-PCGR scheme, each node N_u needs to store the following information:

- the g'-polynomial $g'(x)$, which needs $(t + 1) \cdot L$ bits to store its coefficients.
- the shares of its neighbors' e-polynomials, i.e., $e_{v_i}(x, u)$ $(i = 0, \cdots, n - 1)$, which need $n(t + 1)L$ bits.

The total storage requirement is $(n + 1)(t + 1)L$ bits.

In C-PCGR, each node N_u needs to store the following information:

- the g'-polynomial $g'(x)$, which needs $(t + 1) \cdot L$ bits to store its coefficients.
- the shares of its neighbors' 0-level e-polynomials, i.e., $e_{v_i,0}(x, u)$ $(i = 0, \cdots, n - 1)$, which need $n \cdot (t + 1) \cdot L$ bits.
- the shares of its neighbors' 1-level e-polynomials, i.e., $e_{v_i,1}(x, u)$ $(i = 0, \cdots, n - 1)$, which need $n \cdot (t + 1) \cdot L$

bits.

The total storage requirement is $(2n+1)\cdot(t+1)\cdot L$ bits.

The storage requirement of the RV-PCGR scheme is similar to B-PCGR, except that each coefficient of the polynomials has a length of $2L$. Thus, the overall storage requirement is $2(n+1)(t+1)L$ bits.

Summary

Table I compares the performance of B-PCGR, C-PCGR and RV-PCGR. From the table, we can see that C-PCGR and RV-PCGR have higher communication, computation, and storage overhead than B-PCGR, but they achieve a higher level of security.

2) Comparison with Other Group Rekeying Schemes: Table II compares B-PCGR with some previous schemes, i.e., SKDC [9], LKH [10] and Blundo's scheme [13]. We only compare the costs of these schemes related to key updating, and n_c represents the number of (compromised) nodes that should be evicted.

In the SKDC scheme, the central controller sends a new key to each trusted node individually. We assume that such a message should go through $\sqrt{N}/2$ hops in average, and each new key has L bits. Therefore, the total traffic introduced by key updating is $\frac{(N-n_c)\cdot L\cdot\sqrt{N}}{2}$, and the average size of the data sent/received by a node is $\frac{(N-n_c)L}{2\sqrt{N}}$. To finish a key updating, each node should receive the new key, which takes $o((N-n_c)+\sqrt{N}/2)$ units of time. The scheme is efficient in terms of computation and storage. Each node needs only one decryption and stores one key.

When analyzing the LKH scheme, we assume that a binary logic key hierarchy is used. Let s_c represent the size of the common ancestor tree (CAT) [24] of the evicted nodes. This scheme requires that each node in CAT should change its key encryption key (KEK) and notify the KEK to its two children (except the evicted nodes). Therefore, $2S_c-n_c$ keys should be transmitted. Each node should receive the keys, which results in a rekeying delay of $o(\sqrt{N})$ units of time. Also, each node in this scheme should keep $logN$ (the height of tree) number of KEKs, and hence the storage requirement is $LlogN$.

If Blundo's scheme is used for distributedly generating a group key for up to N members, each node needs to store and compute a $(N-1)$-variable polynomial. Assume that the degree of the polynomial is t, the total storage requirement is as high as $N(t+1)L$ bits.

Comparing our B-PCGR scheme to the previous schemes, we can find that:

- B-PCGR has smaller rekeying delay than any other schemes.
- B-PCGR generates less traffic than SKDC when the network size (N) is large. Also, it generates less traffic than LHK when $2S_c - n_c > n$.
- As a distributed scheme, B-PCGR requires each node to perform some computations that are performed solely by the central controller in a centralized scheme. Therefore, the computational cost (per node) of B-PCGR is larger than the centralized schemes (SKDC and LHK), but it is smaller than Blundo's scheme. From the table, we can see that only a small fraction of the computations are performed at key updating time, so the rekeying delay should not be increased too much. Furthermore, the key updating process may be initiated once every several hours or even a couple of days, so the computational cost is not significant in the long term.
- The storage requirement of B-PCGR is smaller than Blundo's scheme, but larger than SKDC and LHK. However, the storage requirement is not very large in most cases. For example, if $n = 20$, $t = 30$ and $L = 64bits$, the required storage space is about $5KB$. If the network density is very high and each node has many neighbors, the node may select only a subset of the neighbors to distribute shares. Thus, the storage requirement of each node can be reduced.

B. Simulations

We use simulations to study how the group rekeying scheme (i.e., B-PCGR) can improve the performance of filtering false messages.

1) Simulation Model: In the simulation, 2000 sensor nodes are uniformly distributed to a $1000 \times 800m^2$ field, and the communication range of each node is $40m$. The stationary sink (base station) sits at one corner of the field.

We simulate the behavior of the adversary as follows: The adversary keeps on capturing and compromising sensor nodes. Every certain time interval (denoted as τ_c), the adversary can compromise (reprogram) one node and obtain the keys held by the node. After that, the node is put back to the network (with all the keys already compromised by the adversary). Therefore, every τ_c, the number of compromised nodes is increased by 1, and the number of compromised keys may also be increased if the newly compromised node has keys previously unknown to the adversary. Each compromised node attacks the system by injecting a false report every 10 second (We assume that

TABLE I

COMPARING B-PCGR, C-PCGR AND RV-PCGR

	B-PCGR	C-PCGR	RV-PCGR
Data sent/received by each node (bits)	nL	2nL	2nL
Rekeying delay	$o(1)$	$o(1)$	$o(1)$
Computational overhead at key updating time	n decryptions, $o(\mu^3)$ multiplications/divisions over $F(q)$ $(q > 2^L)$	n decryptions, $o(\mu^3)$ multiplications/divisions over $F(q)$	n decryptions
Total computational overhead per node	$2n$ encryptions/decryptions, $o(\mu^3 + (n+1)t^2)$ multiplications/divisions over $F(q)$	$2n$ encryptions/decryptions and $o(2\mu^3 + (2n+1)t^2)$ multiplications/divisions over $F(q)$	$2n$ encryptions/decryptions and $o((n+1)t^2)$ multiplications/divisions over $F(2^{2L})$
Storage requirement per node (bits)	$(n+1)(t+1)L$	$(2n+1)(t+1)L$	$2(n+1)(t+1)L$

TABLE II

COMPARING B-PCGR WITH PREVIOUS GROUP REKEYING SCHEMES

	SKDC	LKH	Blundo's	B-PCGR
Distributed	No	No	Yes	Yes
Data sent/received by each node (bits)	$\frac{(N-n_c)L}{2\sqrt{N}}$	$(2s_c - n_c)L$	$log(n_c)$	nL
Rekeying delay	$o((N-n_c)+\frac{\sqrt{N}}{2})$	$o(\sqrt{N})$	$o(\sqrt{N})$	$o(1)$
Maximum computational overhead per node (at key updating time)	1 decryption	$logN$ decryptions	evaluating a t-degree $(N-1)$-variable polynomial over $F(q)$ $(q > 2^L)$	n decryptions, $o(\mu^3)$ multiplications/divisions over $F(q)$
Maximum computational overhead per node (overall)	1 decryption	$logN$ decryptions	evaluating a t-degree $(N-1)$-variable polynomial over $F(q)$	$2n$ encryptions/decryptions, $o(\mu^3 + (n+1)t^2)$ multiplications/divisions over $F(q)$
Storage requirement per node (bits)	L	$LlogN$	$N(t+1)L$	$(n+1)(t+1)L$

an intruder will not inject false reports with higher rate, since the intruder is easier to be detected in that case.)

The original SEF scheme [1], the SEF scheme with intruder isolation (*SEF-i*) and the SEF scheme with periodical key updating (called *SEF-u*) are simulated and evaluated in terms of:

- the *injected message overhead,* which is the total number of hops traversed by each injected message (before it is dropped) in one second;
- the *control message overhead,* which is the total number of hops traversed by each control message related to the intruder isolation (identification) or key updating in one second;
- the *total message overhead,* which is the sum of injected message overhead and control message overhead.

2) Simulation Results: We first evaluate the key updating mechanism by comparing the performance of SEF-u to SEF and SEF-i. Figure 6 shows the results when the key updating

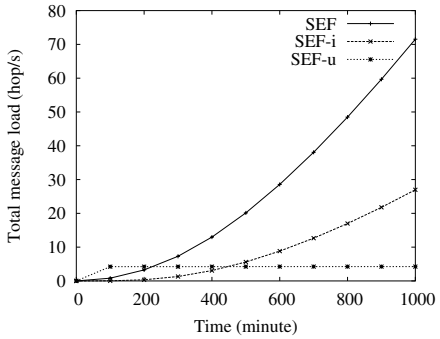


Fig. 6. Performance of the key updating scheme ($\tau_c = 10min$,$\tau_u = 100min$)

interval (τ_u) is 100 minutes and the node compromise interval (τ_c) is 10 minutes. In the figure, a point corresponding to time t refers to the average message overhead during the 10 minute-phase ending at t.

From the figure, we can see that SEF-i and SEF outperform SEF-u at the beginning of the network lifetime. This is due to

the reason that there are very few intruders during that period. The message overhead injected by the intruders is small, and the intruder isolation mechanism can effectively deal with the problem. In this case, if keys are periodically updated, it does not add too much benefit, but increases the message overhead since each node needs to exchange information with its neighbors in order to update its keys.

As the attack continues, the trend is reversed and the SEF-u outperforms the other two. As shown in the figure, the message overhead increases rapidly in SEF and SEF-i. In SEF-u, since the keys are updated periodically, the keys compromised by the adversary become useless after the key updating. Therefore, the adversary can not continuously accumulate its knowledge about the keys to obtain a large portion or all keys. Certainly, some intruders may remain undetected and can renew their keys. However, these nodes have only one key after each key updating, and hence does not have significant impact.

When the periodical key updating mechanism is used, the total message overhead includes two components: the injected message overhead and the control message overhead (i.e., the messages exchanged between neighbors for key updating). Figure 7 shows the tradeoff between these two components. As key updating interval (τ_u) increases, the average control message overhead decreases. At the same time, the average injected message overhead increases, since the adversary can compromise more keys to attack the network during each key phase (i.e., the period between two consecutive key updates). Consequently, there exist an optimal τ_u, at which point the total message overhead is minimized.

From Figure 7 (a), (b) and (c), we can see the impact of the number of groups and parameter τ_c on selecting the optimal τ_u. When the number of groups is 10 and $\tau_c = 10min$, the optimal τ_u is between 150-250 minutes. As the number of groups decreases (e.g., 5), it becomes easier for the adversary to compromise a larger portion of keys and cheat more innocent nodes. Therefore, the injected message overhead increases more quickly and the optimal τ_u becomes smaller (i.e., 100-200 minutes). As τ_c increases, i.e., nodes are compromised more slowly, the injected message overhead also increases more slowly. Consequently, the optimal τ_u becomes larger (i.e., 500-700 minutes).

VI. Discussions

A. Detecting False Shares

In addition to breaking a group polynomial (i.e., $g(x)$), the adversary may prevent a normal node from updating its group key by returning false shares. To defend against this attack, a node (say N_u) can keep some *signature*s of the correct shares, and use them to detect and filter false shares. For example, the key polynomial $e_u(x, y)$ can be constructed as $e_u(x, y) = \alpha(x, y) \times d_u(x, y) + q_u(x, y)$. After distributing shares of $e_u(x, y)$, N_u removes $e_u(x, y)$ and $\alpha(x, y)$, but keeps $d_u(x, y)$ and $q_u(x, y)$. Note that N_u can deliberately select $d_u(x, y)$ and $q_u(x, y)$ such that most of the coefficients of these polynomials are zero and hence the required storage space is small. Based on $d_u(x, y)$ and $q_u(x, y)$, N_u accepts a received share (say $\hat{e}_u(c, v)$) from a neighbor (N_v) only if $\hat{e}_u(c, v) \equiv q_u(c, v) \bmod d_u(c, v)$.

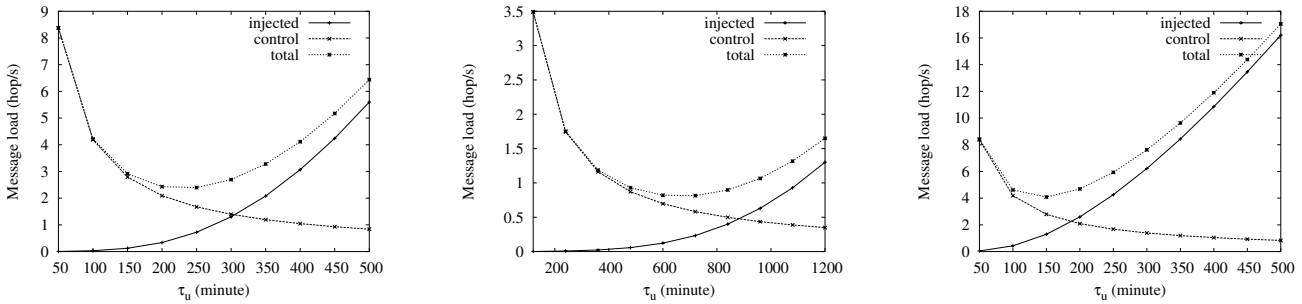
B. Node Isolation and New Node Deployment

If a large fraction of neighbors are compromised, a node may not be able to update its group key and thus be isolated. To deal with this problem, some new nodes may be deployed to the isolated areas. After that, each new node distributes shares only to other new nodes to prevent compromised nodes from obtaining its shares. Also, an isolated innocent node can distribute its shares to these new nodes, which can help the node to update its group key and rejoin the network. To enable these operations, each node should be given an initial master key K_0 before deployment. The key is used only during the first few minutes after deployment and must be removed after that. When a new node distributes a share to another new node, the share will be encrypted with K_0 to prevent old nodes from obtaining it. Also, a node (say N_u) should use K_0 to encrypt its $e_u(x, y)$ before removing $e_u(x, y)$ and K_0, and keep the encrypted polynomial. When N_u is isolated later, it sends the encrypted polynomial to the newly deployed nodes. On receiving the encrypted polynomial, a new node (say N_v) can decrypt it, obtain a share $e_u(x, v)$, and remove $e_u(x, y)$.

C. Other Issues

In the proposed schemes, group keys are updated periodically. When group keys are being updated in the network, nodes may not be able to send information to the sink since the group keys known by the nodes may not be consistent. To address this problem, information sent during this period should be authenticated using the old group keys.

If a node has many neighbors, it is not necessary to send a share to each of them. As future work, we will further investigate how to determine the number of neighbors that should receive a share. If this number is too small, the node may quickly become isolated, since some neighbors are



(a) 10 groups,$\tau_c = 10min$ (b) 10 groups,$\tau_c = 40min$ (c) 5 groups,$\tau_c = 10min$

Fig. 7. Tunning the key updating interval (τ_u)

compromised or failed and it cannot get enough number of shares to update its group key. On the other hand, the security level could be decreased if too many neighbors have received key shares.

VII. Conclusions

In this paper, we proposed a family of *predistribution and local collaboration-based group rekeying (PCGR)* schemes to address the node compromise problem and to improve the effectiveness of filtering false data in sensor networks. These schemes are based on the idea that future group keys can be preloaded before deployment, and neighbors can collaborate to protect and appropriately use the preloaded keys. Extensive analyses and simulations were conducted to evaluate the proposed schemes, and the results show that the proposed schemes can achieve a good level of security, outperform most existing schemes, and significantly improve the effectiveness of filtering false data.

In addition to filtering false data, the proposed PCGR schemes can also be applied to other group rekeying problems, especially for scenarios (e.g., pebblenets [25]) where a group has a large number of widely spread members, the membership changes frequently, or when it is very expensive to maintain a central key manager.

References

[1] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical En-route Filtering of Injected False Data in Sensor Networks," *IEEE Infocom'04*, March 2004.

[2] S. Zhu, S. Setia, S. Jajodia, and P. Ning, "An Interleaved Hop-by-Hop Authentication Scheme for Filtering False Data in Sensor Networks," *IEEE Symposium on Security and Privacy*, 2004.

[3] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E.Cayirci, "Wireless Sensor Networks: A Survey," *Computer Networks*, vol. 38, no. 4, March 2002.

[4] H.Chan and A. Perrig, "Security and Privacy in Sensor Networks," *IEEE Computer*, October 2003.

[5] Y. Hu, A. Perrig, and D. Johnson, "Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks," *IEEE Infocom*, April 2003.

[6] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. Tygar, "SPINS: Security Suite for Sensor networks," *Mobicom'01*, 2001.

[7] S. Marti, T. Giuli, K. Lai, and M. Baker, "Mitigating Routing Misbehavior in Mobile Ad Hoc Networks," *ACM MobiCom*, August 2000.

[8] G. Wang, W. Zhang, G. Cao, and T. La Porta, "On Supporting Distributed Collaboration in Sensor networks," *IEEE Military Communications Conference (MILCOM)*, October 2003.

[9] H. Hugh, C. Muckenhirn, and T. Rivers, "Group Key Management Protocol Architecture," *Request for comments (RFC) 2093, Internet Engineering Task Force*, March 1997.

[10] D. Wallner, E. Harder, and R. Agee, "Key Management for Multicast: Issues and Architectures," .

[11] C. Wong, M. Goudaand, and S. Lam, "Secure Group Communications Using Key Graphs," *ACM SIGCOMM'98*, 1998.

[12] D. Balenson, D. McGrew, and A. Sherman, "Key Management for Large Dynamic Groups: One-Way Function Trees and Amortized Initialization," *IETF Internet draft*, August 2000.

[13] C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro and M. Yung, "Perfectly-Secure Key Distribution for Dynamic Conferences," *Lecture Notes in Computer Science*, vol. 740, pp. 471–486, 1993.

[14] W. Zhang and G. Cao, "Optimizing Tree Reconfiguration for Mobile Target Tracking in Sensor Networks," *IEEE Infocom'04*, March 2004.

[15] W. Zhang and G. Cao, "DCTC: Dynamic Convoy Tree-Based Collaboration for Target Tracking in Sensor Networks," *IEEE Transactions on Wireless Communication*, September 2004.

[16] CROSSBOW TECHNOLOGY INC., "Wireless sensor networks," *http://www.xbow.com/Products/ Wireless_Sensor_Networks.htm*.

[17] S. Zhu, S. Setia, and S. Jajodia, "LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks," *The 10th ACM Conference on Computer and Communications Security*, 2003.

[18] L. Eschenauer and V. Gligor, "A Key-management Scheme for Distributed Sensor Networks," *The 9th ACM Conference on Computer and Communications Security*, pp. 41–47, November 2002.

[19] H. Chan, A. Perrig, and D. Song, "Random Key Predistribution Schemes for Sensor Networks," *IEEE Symposium on Research in Security and Privacy*, 2003.

[20] D. Liu and P. Ning, "Establishing Pairwise Keys in Distributed Sensor Networks," *The 10th ACM Conference on Computer and Communications Security*, 2003.

[21] W. Du and J. Deng, "A Pairwise Key Pre-distribution Schemes for Wireless Sensor Networks," *The 10th ACM Conference on Computer and Communications Security*, 2003.

[22] X. Zhang, S. Lam, D. Lee, and Y. Yang, "Protocol Design for Scalable and Reliable Group Rekeying," *IEEE/ACM Transactions on Networking*, vol. 11, no. 6, pp. 908–922, December 2003.

[23] S. Lin and D. Costello, "Error-Correcting Codes," *Prentice-Hall, Inc.*, 1983.

[24] D. McGrew and A. Sherman, "Key Establishment in Large Dynamic Groups using One-Way Function Trees," *TIS Report No. 0755, TIS Labs at Network Associates, Inc. Glenwood, MD*, May 1998.

[25] S. Basagni, K. Herrin, D. Bruschi, and E. Rosti, "Secure Pebblenets," *ACM MobiHoc '01*, 2001.