Ad Hoc Networks 7 (2009) 1229–1242

Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

Predistribution and local collaboration-based group rekeying for wireless sensor networks ☆

Wensheng Zhang [a,*], Sencun Zhu [b], Guohong Cao [b]

[a] Department of Computer Science, Iowa State University, 226 Atanasoff Hall, Ames, IA 50011, United States
[b] Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 100084, United States

ARTICLE INFO

Article history:
Received 21 January 2008
Received in revised form 19 August 2008
Accepted 19 November 2008
Available online 13 December 2008

Keywords:
Wireless sensor networks
Security
Group key management
False message filtering

ABSTRACT

When a sensor network is deployed in a hostile environment, an adversary may launch such attacks as eavesdropping the communications and compromising sensor nodes. Using the compromised nodes, he may inject false sensing reports or modify the reports sent by other nodes. To defend against these attacks, researchers have proposed symmetric group key-based schemes. In these schemes, however, if a large number of nodes are compromised, many (sub)group keys will be revealed. This greatly endangers the filtering schemes, making them very ineffective or even useless. To address this problem, we propose a family of *predistribution and local collaboration-based group rekeying (PCGR)* schemes, which update the compromised group keys to prevent the compromised nodes from understanding the communications between noncompromised nodes or injecting false data. These schemes are designed based on a simple while controversial idea – preload future group keys into sensor nodes before their deployment. To protect the preloaded keys from being disclosed by compromised nodes, we propose a novel technique that requires neighboring nodes to collaborate to derive the future group keys. To the best of our knowledge, our schemes are the first set of *distributed* group rekeying schemes for sensor networks without involving online key servers. Extensive analysis 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 several previous group rekeying schemes, and significantly improve the effectiveness of false data filtering.

1. Introduction

In many applications, e.g., battlefield surveillance and habitat monitoring, sensor networks are deployed in unattended or hostile environments. If the communications in the networks are not well protected, they can be easily eavesdropped by adversaries. Due to the lack of tamper resistant hardware, adversaries may even capture and reprogram nodes, or inject their own nodes into the network and induce the network to accept them as legitimate nodes [1]. Once in control of a few nodes, the adversaries 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 data sink may accept wrong sensing data and take inappropriate responses, which could cause catastrophic impacts in some strategic scenarios.

To prevent outsiders from eavesdropping messages, legitimate sensor nodes can share one or more group keys [2] for encrypting the messages exchanged among them. To thwart the message injection attacks, messages should be authenticated. Due to high computational and communication overhead, the digital signature-based authentication techniques are not suitable for sensor networks [3].

☆ A preliminary version of this paper appears in the proceedings of IEEE INFOCOM 2005.

* Corresponding author. Tel.: +1 515 294 2821.
E-mail addresses: wzhang@cs.iastate.edu (W. Zhang), szhu@cse.psu.edu (S. Zhu), gcao@cse.psu.edu (G. Cao).

1570-8705/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.adhoc.2008.11.004

Therefore, researchers proposed to adopt symmetric group key-based techniques such as the *statistical en-route filtering (SEF)* scheme [4]. The basic idea of the SEF scheme is as follows: sensor nodes are randomly assigned to multiple groups before deployment, and nodes in the same group are preloaded with a same group key. When a sensor node wants to send a message to a sink, the message is authenticated using multiple MACs contributed by its neighbors and each MAC is generated using one group key. When such a message is forwarded along a path to the sink, each en-route node uses its group key to verify one MAC carried in the message. If an en-route node is compromised, it may modify a passing message or inject a false message. However, it normally knows only one group key, and thus it can only forge one MAC correctly. So the modification or injection will be detected by other en-route nodes who know different group keys.

The group key-based techniques, however, will become ineffective if some nodes are compromised, since an adversary may obtain group keys from the compromised nodes. To deal with node compromise, the compromised nodes should be identified and then the noncompromised nodes should update their group keys to prevent the adversary from making use of the captured keys. In the literature, various techniques have been proposed for identifying compromised nodes. For example, nodes may use the *watchdog* mechanism [5] to monitor its neighbors and identify the compromised nodes when observing misbehavior. The identification accuracy can be further improved by using some collaborative intruder identification schemes [6]. In addition to identifying compromised nodes based on misbehavior observed, Seshadri et al. [7] have proposed a software-based attestation technique to detect nodes that have been reprogrammed.

Although the problem of detecting compromised nodes has attracted lots of interest and been studied extensively, the problem of efficiently updating group keys in sensor networks has not been delved. In the context of secure multicast in wired networks, many centralized schemes [8–11] and a few distributed schemes [12] have been proposed. However, most of them are not suitable for sensor networks. For example, in SKDC [8], each key updating requires N key encryptions and N transmissions (N is the number of nodes in the networks) of keys 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. [9], Wong et al. [10], and Balenson et al. [11] can achieve logarithmic broadcast size, storage, and computational cost. However, the communication cost and the rekeying delay are still high when they are applied to a large scale sensor network with high resource constraints. Furthermore, a central controller has to be online to trace the status of all nodes, and maintain a large logic tree connecting all the trusted nodes, leading to high management overhead. The distributed solutions, e.g., Blundo's scheme [12], allow a set of nodes to set up a group key in a distributed way. However, it is still not scalable or efficient since each node must communicate with other trusted members in the same group, and the storage cost of each node increases rapidly as the group size increases.

To address the group rekeying problem for sensor networks, 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 in securely disseminating new keys 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 avoids the high cost of the centralized management. Based on the above ideas, we first propose a *basic PCGR* (*B-PCGR*) scheme, in which one-hop neighbors collaborate to protect their group key polynomials. In B-PCGR, a group key polynomial is revealed if a node is compromised along with a certain threshold number of its one-hop neighbors. To address this limitation and achieve higher resilience to node compromise, we propose an enhanced PCGR scheme called *cascading PCGR*, in which each node can collaborate with nodes beyond its one-hop neighborhood to protect its group key polynomial. To the best of our knowledge, PCGR schemes are the first set of distributed group rekeying schemes for sensor networks. This distributed property is critical for unattended sensor networks deployed in adversarial environments because the central authority is a single point of failure from security and performance perspectives. Extensive analysis is conducted to evaluate the security level and the performance of the proposed schemes. We also compare the performance of the proposed schemes with several existing group rekeying schemes. Simulations are used to evaluate the effectiveness of the proposed group rekeying scheme in filtering false data. The analysis and simulation results show that the proposed schemes can achieve a good level of security, outperform several previously proposed schemes, and significantly improve the effectiveness and efficiency of false data filtering.

The rest of this paper is organized as follows. The next section presents the system model. In Section 3, we describe and analyze the basic PCGR scheme. An enhanced PCGR scheme is presented in Section 4. Section 5 reports the performance evaluation results. Section 6 discusses a number of issues related to the proposed schemes. Section 7 concludes the paper.

2. System model

We consider a large scale wireless sensor network, which is deployed in an unattended and hostile environment. The network is composed of low-complexity sensor nodes, e.g. the Berkeley MICA mote [13], which has a processor running at 4 MHz and 4KB RAM. These nodes are also limited in power supply, bandwidth, and computational capability. Therefore, public key-based per packet authentication cannot be afforded [14]. On the other hand, each node has enough space for storing a few kilobytes of keying information.

Node deployment is managed by a (offline) central controller (or setup server), which is responsible for picking group keys and preloading keying information to every

node before it is deployed. We assume that each node is innocent before deployment, and cannot be compromised at least during the first several minutes after deployment [2,15] since compromising a node takes some time. Also, each pair of neighboring nodes can establish a pairwise key using an existing technique [16–20]. Later when two neighbors exchange messages for group key updating, they will encrypt the messages using their pairwise key to prevent eavesdropping.

We assume that a compromised node can be detected by most of its neighbors. To achieve this, many existing schemes can be employed. For example, in the physical layer, Hall et al. [21] have studied how to detect the changes of devices through signal changes. In the MAC and network layers, the watchdog mechanism [5] and some collaborative intruder detection scheme [6] can be used. In addition, software attestation techniques, e.g., SWATT [7], have been proposed for verifying the genuinity of the software running in a device, thus finding out nodes whose code has been changed by an attacker. 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 [4] is used to detect and drop false messages. With this mechanism, sensor nodes are randomly assigned to multiple groups, and the nodes in the same group share a unique key. We assume that nodes are loosely time-synchronized. For example, a secure time-synchronization protocol could be applied to provide time synchronization even in the presence of node compromise [22]. We also assume group rekeying is launched by each node periodically [23]. This removes the dependency on an online central server and may also reduce the rekeying frequency. Clearly, the rekeying period is application-dependent and in practice a proper tradeoff between security and performance is necessary. However, the problem of selecting an appropriate rekeying period is out of the scope of this work.

3. Basic predistribution and local collaboration-based group rekeying (B-PCGR) scheme

3.1. Basic idea

The B-PCGR scheme includes the following three steps.

3.1.1. Group key predistribution

Before a node is deployed, it is randomly assigned to a group, and is preloaded with the current and the future keys of the group. The keys are represented by a polynomial called *group key polynomial* (*g-polynomial*). Different from most existing group rekeying protocols, in which new group keys must be generated and distributed reliably at the time for key updating, the B-PCGR scheme can avoid the overhead for network-wide reliable communications since the keys are already preloaded.

3.1.2. Local collaboration-based key protection

Since all group keys have been preloaded, it is necessary to protect the keys from being exposed to intruders. For this purpose, even a node that is currently trusted should not explicitly keep the future group keys because the keys can be captured by an adversary if the node is compromised later. 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, which is briefly described 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*.
- The shares of the e-polynomial, 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 cannot access its future group keys without collaborating with a certain number of neighbors, each of which has a share of its e-polynomial.

3.1.3. Local collaboration-based group key updating

At the time for 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 compute one instance of the e-polynomial that 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 cannot derive any group keys that should not be disclosed at this time on its own.

3.2. Detailed description of B-PCGR

As elaborated in [24,25], following are some design details of the B-PCGR.

3.2.1. Predistributing g-Polynomials

Initially, the setup server decides the total number of groups. For each group, a unique s-degree (s is a system parameter) univariate g-polynomial

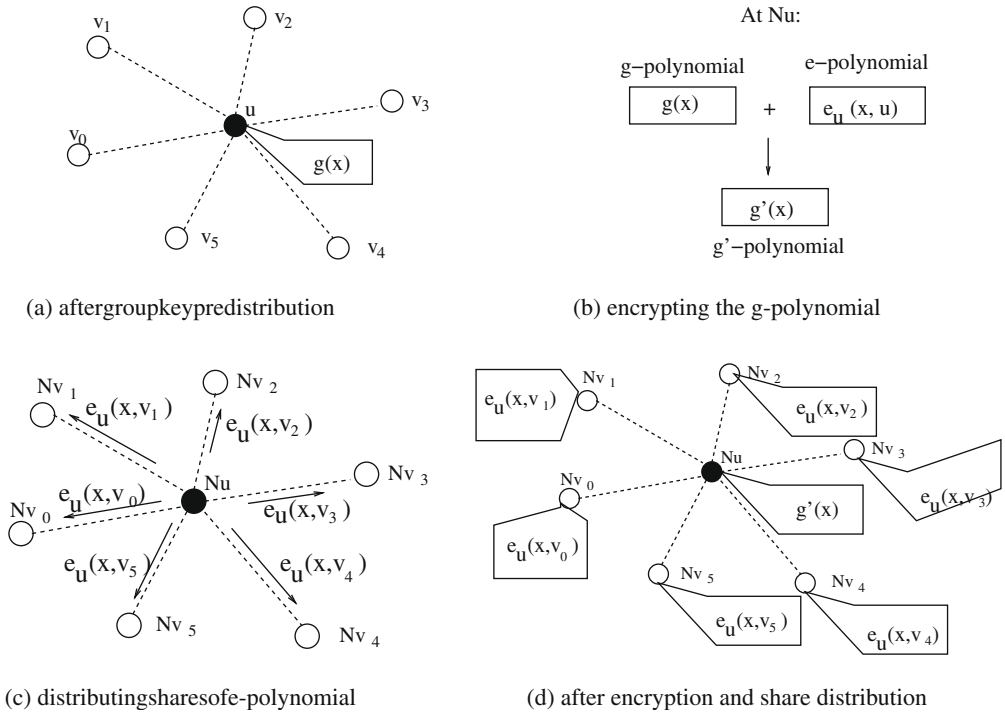
$$g(x) = \sum_{i=0}^{s-1} a_i x^i \tag{1}$$

is constructed over a prime finite field $F(q)$ to represent the keys of the group, where $g(0)$ is the initial group key, $g(i)$ ($i \geqslant 1$) is the group key of version i, and q is a large prime whose size can accommodate a group key.

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

3.2.2. Encrypting g-polynomials and distributing the shares of e-polynomials

After u has been deployed and has discovered its neighbors, it randomly picks a bivariate e-polynomial



(a) aftergroupkeypredistribution

(b) encrypting the g-polynomial

(c) distributingsharesofe-polynomial

(d) after encryption and share distribution

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

$$e_u(x,y) = \sum_{0\leqslant i\leqslant s, 0\leqslant j\leqslant \mu} b_{ij}x^i y^j, \tag{2}$$

where μ is a parameter picked by u itself[1].

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

$$g'(x) = g(x) + e_u(x,u). \tag{3}$$

After that, as shown in Fig. 1c, u distributes the shares of $e_u(x,y)$ to its neighbors v_i ($i = 0,\ldots,n-1$; n is the number of neighbors). Specifically, each neighbor v_i of u receives share $e_u(x,v_i)$. Finally, u removes $e_u(x,y)$ and $g(x)$, but keeps $g'(x)$. The distribution of $g'(x)$ and $e_u(x,v_i)$ is illustrated in Fig. 1d.

3.2.3. Key updating

Each node maintains a *rekeying timer*, which is used to periodically trigger the node to update its group key, and a variable called *current group key version* (denoted as c). Note that c is initialized to 0 when the node is deployed.

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

$$e_u(c,y) = \sum_{j=0}^{\mu} B_j y^j, \tag{4}$$

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

$$\sum_{j=0}^{\mu} (v_i)^j B_j = e_u(c,v_i), \quad i = 0,\ldots,\mu. \tag{5}$$

Next node u evaluates $e_u(c,y)$ at $y = u$, as shown in Fig. 2b, and finally computes the new group key $g(c) = g'(c) - e_u(c,u)$.

3.3. Security analysis

The security property of the B-PCGR scheme can be stated by Theorem 1, which has been proved in [24].

Theorem 1. *The adversary can derive a future group key of a group (whose g-polynomial is $g(x)$) if and only if:*

(1.1) a node (e.g., u) of the group has been compromised, and
(1.2) at least $\mu+1$ neighbors of u (Recall that in the encryption polynomial picked by u, i.e., $e_u(x,y)$, the degree of x is s and the degree of y is μ.) have been compromised;

or

(2) at least $s+1$ past keys of the group have been compromised.

[1] Different nodes may select different value for μ based on the number of its neighbors. The selection approach will be discussed in Section 6.1.

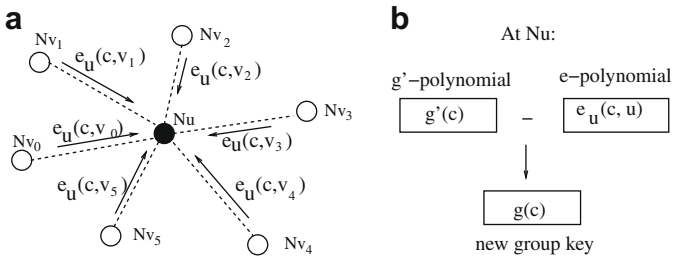


Fig. 2. B-PCGR: key updating.

4. Cascading PCGR (C-PCGR) scheme

When the B-PCGR scheme is applied, a group key polynomial is compromised if a node belonging to this group is compromised along with its $\mu+1$ or more neighbors. Meanwhile, for a node, its value of μ must be smaller than the number of its neighbors. As a result, if there exist some nodes that have very few neighbors, the adversary could selectively compromise these nodes to compromise a group key polynomial with lower complexity. To address this problem, we propose a cascading PCGR scheme, which distributes the shares of a node beyond its direct neighborhood. By applying this scheme, nodes with a small number of neighbors can improve the security level of their group keys.

In this scheme, the e-polynomial shares of 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.

4.1. 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.

4.1.1. Polynomial encryption and share distribution

After each node (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 s and μ, respectively.

Using the 0-level e-polynomial (i.e., $e_{u,0}(x,y)$), each node 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:

$$g'(x) = g(x) + e_{u,0}(x,u). \tag{6}$$

After that, as shown in Fig. 3a, 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 v is given $e_{u,0}(x,v)$.

Having received 0-level e-polynomial shares from its neighbors, as shown in Fig. 3b, each node (say 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

$$e'_{u,0}(x,v) = e_{u,0}(x,v) + e_{v,1}(x-1,v). \tag{7}$$

After that, v keeps $e'_{u,0}(x,v)$ and $e_{u,0}(c+1,v)$, which will be returned to 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. Fig. 3 illustrates how the e-polynomial shares of u are distributed to its 1-hop and 2-hop neighbors.

4.1.2. Key updating

To update keys, as shown in Fig. 4a, each innocent node u increases its c by one, and returns shares $e_{v,0}(c,u)$ and $e_{v,1}(c,u)$ to each trusted neighbor v (here, we assume that u has received shares $e_{v,0}(x,u)$ and $e_{v,1}(x,u)$ from v before). At the same time, 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 v).

Having received $\mu+1$ 0-level e-polynomial shares, as shown in Fig. 4a, u reconstructs a unique polynomial $e_{u,0}(c,x)$. Knowing $e_{u,0}(c,x)$, 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 Fig. 4a, 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 u at the next key updating time.

In fact, the time for exchanging 1-level e-polynomial shares can be postponed to a moment shortly before the next group rekeying phase, as long as the innocent nodes can exchange their 1-level shares and compute their 0-level shares before the phase starts. By this, innocent nodes can still send out 0-level shares to their neighbors at the next group rekeying phase. Furthermore, the shares are not computed too earlier, which reduces the risk of being revealed.

4.2. 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:*

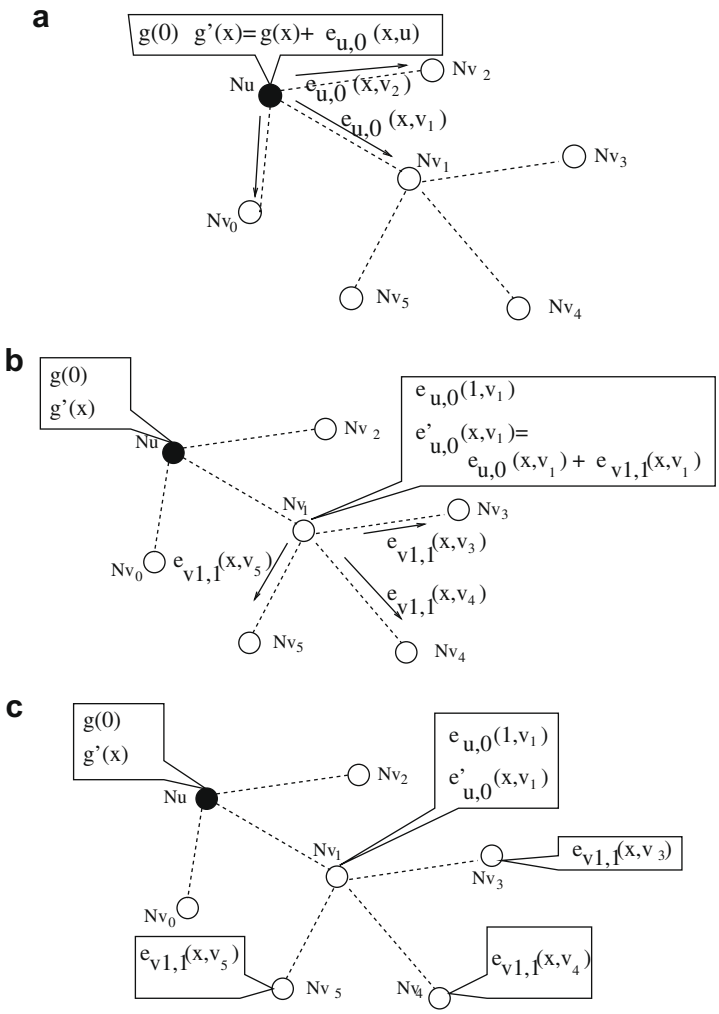


Fig. 3. C-PCGR: polynomial encryption and share distribution.

(1.1) a node u in the group is compromised, and
(1.2) the adversary has compromised at least $\mu+1$ neighbors of u, each of which also has $\mu+1$ neighbors compromised;

or

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

Proof. Similar to the proof of Theorem 1. □

5. Performance evaluations

In this section, we first use the analytic approach to compare the performance of the proposed PCGR schemes and some previously proposed group rekeying schemes. Then, we conduct simulations to study the communication overhead of B-PCGR and how B-PCGR can improve the effectiveness of filtering false messages.

5.1. 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.

5.1.1. Comparing the performance of PCGR schemes

We compare the performance of the proposed PCGR schemes in terms of communication cost, computation overhead, and storage requirement (Table 1). The main results are shown in Table 3.

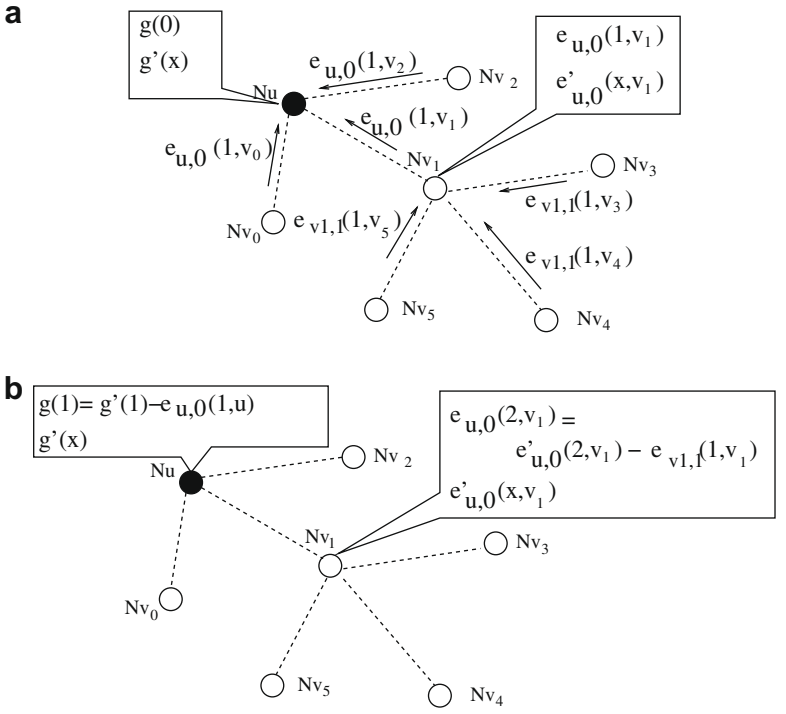


Fig. 4. C-PCGR: key updating (C is increased from 0 to 1).

Table 1
Comparing B-PCGR, C-PCGR and RV-PCGR.

	B-PCGR	C-PCGR
Data sent/received by each node (bits)	nL	2nL
Rekeying delay	$o(1)$	$o(1)$
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 $F(q)$
Total computational overhead per node	$2n$ encryptions/ decryptions, $o(\mu^3 + (n+1)s^2)$ multiplications/ divisions over $F(q)$	$2n$ encryptions/ decryptions and $o(2\mu^3 + (2n+1)s^2)$ multiplications/divisions over $F(q)$
Storage requirement per node (bits)	$(n+1)(s+1)L$	$(2n+1)(s+1)L$

5.1.1.1. 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. Each message includes two shares, which have $2L$ bits. Therefore, each node has to send out $2nL$ bits.

5.1.1.2. 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 u first needs to evaluate $n+1$ s-degree polynomials ($e_{v_i}(c)$ and $g'(x)$) to compute n shares and $g'(c)$, which needs $o((n+1)s^2)$ multiplications. After receiving $\mu+1$ or more shares, it also needs to solve a $(\mu+1)$-variable linear equation group to compute $e_u(c,u)$, and the computational complexity for solving such an equation group is $o(\mu^3)$ multiplications/divisions. The total computational complexity is $o((n+1)s^2+\mu^3)$ 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 group, so the total computational complexity is $o((2n+1)s^2 + 2\mu^3)$ multiplications/division over $F(q)$.

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 3, we list both the computational overhead at key updating time and the total computational overhead for each key updating.

5.1.1.3. Storage requirements. In the B-PCGR scheme, each node u needs to store the following information:

- the g′-polynomial $g'(x)$, which needs $(s+1)\cdot L$ bits to store its coefficients.
- the shares of its neighbors' e-polynomials, i.e., $e_{v_i}(x,u)$ $(i=0,\ldots,n-1)$, which need $n(s+1)L$ bits.

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

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

- The g′-polynomial $g'(x)$, which needs $(s+1)\cdot L$ bits to store its coefficients.
- The shares of its neighbors' 0-level e-polynomials, i.e., $e_{v_i,0}(x,u)$ $(i=0,\ldots,n-1)$, which need $n\cdot(s+1)\cdot L$ bits.
- The shares of its neighbors' 1-level e-polynomials, i.e., $e_{v_i,1}(x,u)$ $(i=0,\ldots,n-1)$, which need $n\cdot(s+1)\cdot L$ bits.

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

5.1.1.4. Summary. Table 3 compares the performance of B-PCGR and C-PCGR. From the table, we can see that C-PCGR has higher communication, computation, and storage overhead than B-PCGR, but achieves a higher level of security.

5.1.2. Comparison with other group rekeying schemes

Table 2 compares B-PCGR with some previous schemes, i.e., SKDC [8], LKH [9] and the Blundo's scheme [12]. 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 $o(\sqrt{N})$ 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) [26] of the evicted nodes. This scheme requires that each node on 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 $\log N$ (the height of tree) number of KEKs, and hence the storage requirement is $L\log N$.

If the 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 s, the total storage requirement is as high as $N(s+1)L$ 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 the centralized schemes (SKDC and LHK).
- 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 the Blundo's scheme. From the table, we can see that only a small fraction of the computations should be 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 the Blundo's scheme, but larger than SKDC and LHK. However, the storage requirement is not very large in most cases. For example, if $n=20$, $s=30$ and $L=64bits$, the required storage space is about 5KB. Note that, 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.

5.2. Simulations

5.2.1. Communication overhead of B-PCGR

5.2.1.1. Simulation model. In this ns2-based simulation, 2000 nodes are uniformly distributed to a $1000\times800\ m^2$ field. The communication range of each node varies between 40 m and 50 m. Note that, when the range is 40 m, each node has 12 neighbors in average, and the average number of neighbors becomes 20 when the range is

Table 2
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	$o(\sqrt{N})$	$o(\sqrt{N})$	$o(\sqrt{N})$	$o(1)$
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	$\log N$ decryptions	evaluating a s-degree $(N-1)$-variable polynomial over $F(q)$	$2n$ encryptions/decryptions, $o(\mu^3+(n+1)s^2)$ multiplications/divisions over $F(q)$
Storage requirement per node (bits)	L	$L\log N$	$N(s+1)L$	$(n+1)(s+1)L$

Table 3
Lookup table for selecting μ ($N = 2000$, $r_f = 3.9e-7$, $r_c = 1.4e-7$, $t = 6$ days; $P_s(\mu,t) \geqslant 0.9$, $P_c(\mu,t) \leqslant 0.01$).

Number of neighbors	12	14	16	18	20	22	24
μ	6	6–8	7–9	7–10	7–12	8–13	8–14

50 m. Parameter μ picked by the nodes varies between 6 and 12. In the case that the number of neighbors of a node (n) is smaller than $\mu + 1$, μ is re-selected as $min(\mu, n-1)$.

We simulate two implementations of B-PCGR, namely, the *conservative B-PCGR* and the *optimistic B-PCGR*. In the conservative B-PCGR, each node returns a share to every trusted neighbor at the time for key updating. This may cause high redundancy in share exchanges. For example, some nodes may have a large number of neighbors (say, 24), and the number could be much larger than its μ (say, 12). A certain level of redundancy enables nodes to receive enough number of shares even if some shares are lost in transmission. But too much redundancy may significantly increase overhead due to the increased amount of traffic and the congestions introduced by the traffic. To address this issue, we propose another implementation called the optimistic B-PCGR. In this implementation, a node returns a share to its neighbor u with the probability of $p_r = \frac{\mu_u+1}{n_u} + \alpha$, where n_u is the number of neighbors of u, μ_u is the value μ selected by u, and α represents a redundancy level. Here, α should be carefully selected: if it is too small (e.g., $\alpha = 0$), many nodes may not get enough number of shares due to communication errors; on the other hand, if α is too large, many shares are exchanged unnecessarily, and therefore cannot reduce the overhead. We have run the simulations with various choices of α, and the results show that the performance is optimized when $\alpha = 0.1$. So, we let $\alpha = 0.1$ in the simulations of which the results are shown and discussed in this section. In both implementations, the conservative B-PCGR and the optimistic B-PCGR, if a node does not obtain enough number of shares from its neighbors, it will broadcast a request to its neighbors which it has not received shares from. On receiving the request, the neighbors will send their shares to the requesting node.

5.2.1.2. Simulation results. Fig. 5 shows the message complexity of the conservative B-PCGR and the optimal B-PCGR as n and μ vary. Here, μ varies between 6 and 12 when $n = 20$, and it varies between 6 and 10 when $n = 12$. Comparing Figs. 5a and b, we can find that the optimistic B-PCGR has significantly lower message complexity than the conservative B-PCGR. This is because many shares are not needed in the optimistic implementation, especially when μ is much smaller than n. From the figure, we can also find that, in the conservative B-PCGR, the message complexity increases rapidly as n increases, but it does not change much as μ varies. This is due to the fact that every pair of neighbors must exchange a share, and thus n becomes the dominant factor for message complexity. In the optimistic B-PCGR, however, n does not affect much the message complexity, while μ has a significant impact.

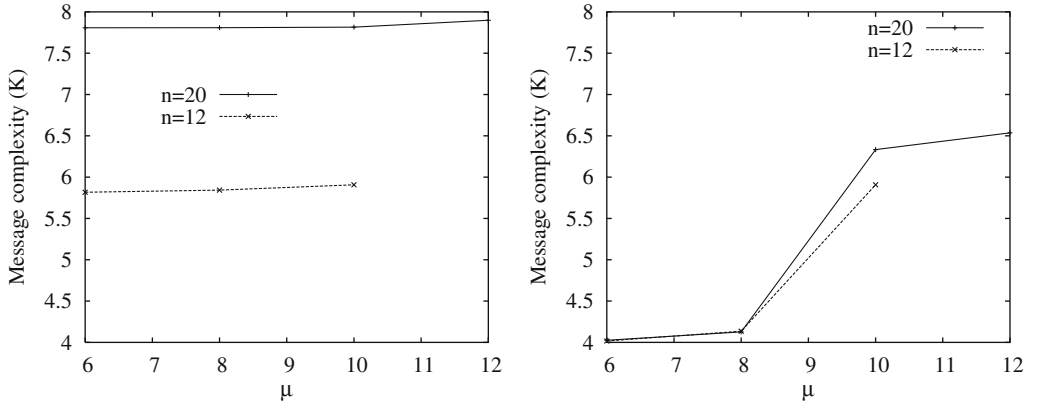
Fig. 6 shows the average delay for one key updating. As we can see, the optimistic B-PCGR has lower delay than the conservative scheme. Furthermore, the optimistic B-PCGR performs better when n is much larger than μ.

5.2.2. Effectiveness of filtering injected messages

5.2.2.1. Simulation model. To simulate the procedure of node compromise, fake message injection by compromised nodes, and the detection of injected fake messages, we develop a customer simulator using C++.

In the simulation, we deploy 2000 sensor nodes uniformly to a $1000 \times 800\ m^2$ field, and the communication range of each node is fixed as 40 m. The stationary sink (base station) sits at one corner of the field.

We simulate the behavior of node compromise as follows: the adversary keeps on randomly 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.



(a) Conservative B-PCGR (b) Optimistic B-PCGR

Fig. 5. Message complexity for one key updating.

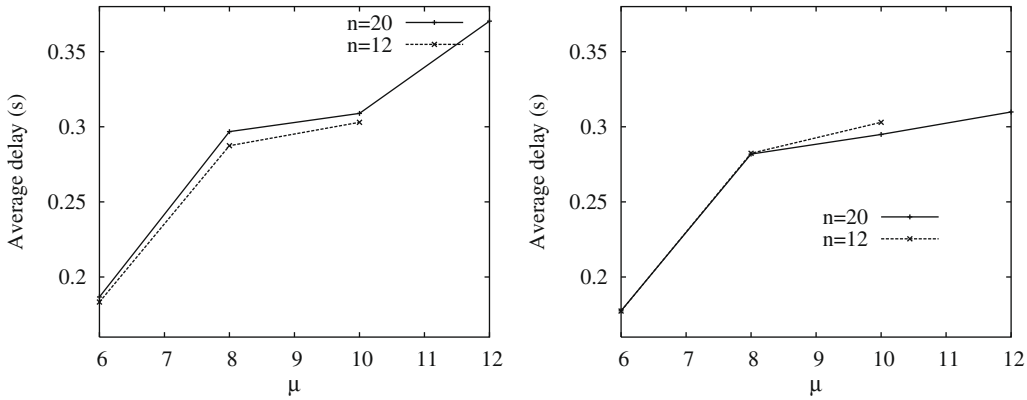


Fig. 6. Average delay for one key updating.

Each compromised node attacks the system by injecting a fake message every 10 s. (We assume that an intruder will not inject false reports with higher rate, since the intruder is easier to be detected in that case.)

We simulate three schemes. The first one is the original SEF scheme [4]. The second one is the SEF scheme with intruder isolation (*SEF-i*), which is proposed for the purpose of comparison. In this scheme, innocent nodes do not send messages to or forward messages from compromised nodes once they have been identified. The third one is the SEF scheme with our periodical key updating (called *SEF-u*). These schemes are evaluated in terms of:

- the *injected message load*, which is the total number of hops traversed by each injected message (before it is dropped) in 1 s;
- the *control message load*, which is the total number of hops traversed by each control message related to the intruder isolation (identification) or key updating in 1 s;
- the *total message load*, which is the sum of injected message load and control message load.

The simulator collects the number of fake messages injected by compromised nodes and the number of such messages that can be detected by innocent nodes. In the following figures, the performance of the above three schemes are depicted as the injected message load (i.e., the total number of hops that injected messages are transmitted before being detected or being received by the sink). the control message load (i.e., the total number of hops that key updating messages are transmitted), and the total message load which is the sum of the above two.

5.2.2.2. Simulation results. We first evaluate the key updating mechanism by comparing the performance of SEF-u to SEF and SEF-i. Fig. 7 shows the result when the key updating interval (τ_u) is 10 h and the node compromise interval (τ_c) is 1 h. In the figure, a point corresponding to time t refers to the average message load during the 1 h-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

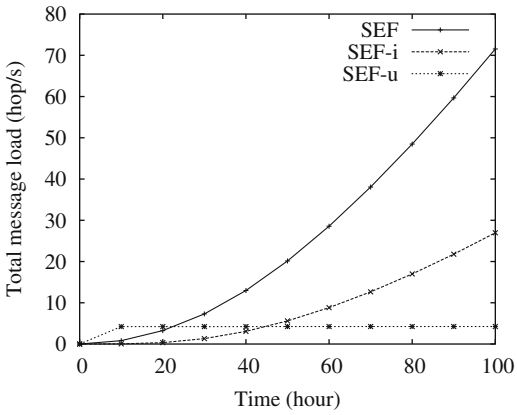


Fig. 7. Performance of the key updating scheme ($\tau_c = 1$ h, $\tau_u = 10$ h).

is due to the reason that there are very few intruders during that period. The message load 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 bring too much benefit, but increases the message load 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 load 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 cannot continuously accumulate its knowledge about the keys to obtain a large portion or all keys. Certainly, some intruders may remain undetected and can renew its 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 load includes two components: the injected message load and the control message load (i.e., the messages exchanged between neighbors for key updating). Fig. 8 shows the tradeoff between these two components. As the key updating interval (τ_u) increases, the average control message overhead decreases. At the same

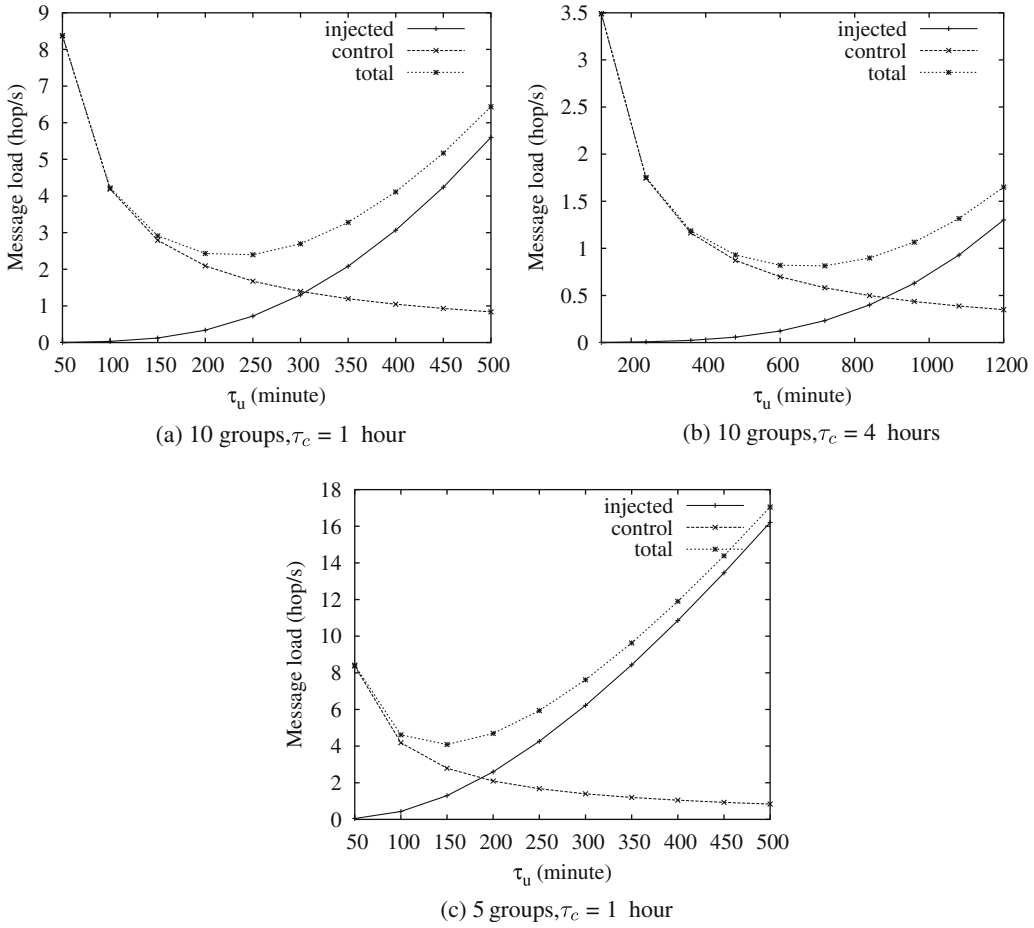


Fig. 8. Tuning key updating interval (τ_u).

time, the average injected message load increases, since the adversary can compromise more keys to attack the network during each key phase (i.e., the period between two consecutive key update operations). Consequently, there exists an optimal τ_u, at which point the total message load is minimized.

From Fig. 8a–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 $\tau_c = 1$ hour, the optimal τ_u is between 15 and 25 h. As the number of groups decreases (e.g., 5), it becomes easier for the adversary to compromise a larger portion of keys to cheat more innocent nodes. Therefore, the injected message load increases more quickly, and the optimal τ_u becomes smaller (i.e., 10–20 h). As τ_c increases, i.e., nodes are compromised more slowly, the injected message load also increases more slowly. Consequently, the optimal τ_u becomes larger (i.e., 50–70 h).

6. Discussions

6.1. Parameter selection

When applying the B-PCGR scheme in a sensor network, each node in the network must select parameter μ appropriately to achieve a desired level of system security and reliability. In this paper, the level of system security and reliability is measured by two metrics:

- $P_s(\mu, t)$: the probability that an arbitrary node, if not compromised or failed, can update its group key successfully at time t; i.e., it has $\mu + 1$ or more neighbors that are not compromised or failed at time t.
- $P_c(\mu, t)$: the probability that at least one group key polynomial is compromised; i.e., there exists a node that is compromised along with its $\mu + 1$ or more neighbors.

In the following, we develop a theoretic model for analyzing the relationship between parameter μ, the number of neighbors (denoted as n), $P_s(\mu, t)$, and $P_c(\mu, t)$. Based on the model, we show how each node determines its value of μ.

We assume that a node fails (or is compromised) following a Poisson process. Specifically, a node fails before time t with the probability of $1 - e^{-\lambda_f t}$, where λ_f denotes the failure rate. For example, if the expected life time of a node is 30 days (i.e., 2,592,000 s) and t is represented in the unit of second, $\lambda_f = 1/2,592,000 = 3.9 \times 10^{-7}$. Similarly, a node, if not fails, is compromised before time t with

the probability of $1 - e^{-\lambda_c t}$, where λ_c denotes the compromising rate. Considering a scenario where the number of nodes in the network, denoted as N, is 2000, and the adversary can compromise one node per hour in average, $\lambda_c = \frac{1}{1\ \mathrm{h} \times 2000} = 1.4 \times 10^{-7}$.

Based on the above assumption, we can derive the probability that a node is not compromised or failed before t, denoted as $p_s(t)$, to be

$$p_s(t) = e^{-(\lambda_f + \lambda_c)t}. \tag{8}$$

Therefore,

$$P_s(\mu, t) = \sum_{i=\mu+1}^{n} \binom{n}{i} p_s(t)^i [1 - p_s(t)]^{n-i}. \tag{9}$$

We can also derive the probability that a node is compromised before t, denoted as $p_c(t)$, to be

$$p_c(t) = \frac{\lambda_c}{\lambda_c + \lambda_f}(1 - e^{-(\lambda_c + \lambda_f)t}). \tag{10}$$

So the probability that a node and its $\mu + 1$ or more neighbors are compromised is

$$p_c(t) \times \sum_{i=\mu+1}^{n} \binom{n}{i} p_c(t)^i [1 - p_c(t)]^{n-i}. \tag{11}$$

Further, the probability for the adversary to compromise at least one group key polynomial, i.e., $P_c(\mu, t)$, is no larger than

$$1 - \{1 - p_c(t) \times \sum_{i=\mu+1}^{n} \binom{n}{i} p_c(t)^i [1 - p_c(t)]^{n-i}\}^N. \tag{12}$$

we now describe how a node determines its value of μ to achieve a desired level of system security. The approach will be explained through an example.

Suppose we want to deploy a sensor network that is composed of 2000 nodes. The expected life time of each node is 30 days; i.e., $r_f = 3.9e-7$. The rate for the adversary to compromise the sensor node is no larger than 1 node/hour; i.e., $r_c \leqslant 1.4e-7$. We aim to achieve the goal that, during the 6 days after deployment, at least 90% nodes, if alive, can update their group keys successfully, and the probability for the adversary to compromise one group key is lower than 0.01. That is, $P_s(\mu, t) \geqslant 0.9$ and $P_c(\mu, t) \leqslant 0.01$ for $t = 518400$ s.

For each node to select its value of μ appropriately, the following steps are executed:

(1) The setup server computes $P_s(\mu, t)$ and $P_c(\mu, t)$ based on the known values of t, r_f and r_c.
(2) The server constructs a lookup table as shown in 3. This table shows the range from which the value of μ is selected if the number of neighbors (i.e., n) is known. For example, if a node has 16 neighbors, μ must be 7, 8 or 9 such that $P_s(\mu, t) \geqslant 0.9$ and $P_c(\mu, t) \leqslant 0.01$ for $t = 518400$ s.
(3) Before a node is deployed, the server preloads the lookup table to it.
(4) After a node has discovered its neighbors, it selects a value for μ according to the preloaded lookup table.

6.2. Detecting false shares

In addition to compromising nodes and breaking group key polynomials, the adversary may prevent a normal node from updating its group key by injecting false shares to the network or letting an undetected compromised return false shares to its neighbors. To thwart this type of attacks, a node must be able to authenticate its shares. Specifically, a node (say u) can keep some *signatures* of the correct shares, and use them to detect and filter false shares. A naive approach for generating signatures is to use hash functions. That is, for each share s, u keeps a hashed value $h(s)$ for authentication. This method, however, is not scalable. For example, if a node has 20 neighbors and it is expected to update its group keys for 100 times, it has to store 2000 hashed values. Note that a Mote of current generation has only limited storage capacity. So this poses a high storage requirement.

To achieve scalability, we propose a polynomial-based technique for signature generation and authentication. Specifically, the key polynomial of u, i.e., $e_u(x, y)$, can be constructed as $e_u(x, y) = \alpha(x, y) \times d_u(x, y) + q_u(x, y)$, where $\alpha(x, y)$, $d_u(x, y)$ and $q_u(x, y)$ are all polynomials constructed over finite field $F(q)$. When distributing the shares of $e_u(x, y)$, u sends $e_u(x, v)$ and $\alpha(x, v)$ to each neighbor v. Then, u removes $e_u(x, y)$ and $\alpha(x, y)$, but keeps $d_u(x, y)$ and $q_u(x, y)$. Note that, 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. At the time for updating group keys from version $c - 1$ to c, v returns both $e_u(c, v)$ and $\alpha(c, v)$ to u. Based on $d_u(x, y)$ and $q_u(x, y)$, u accepts a received share (say $\hat{e}_u(c, v)$ and $\hat{\alpha}(c, v)$) from a neighbor (v) only if $\hat{e}_u(c, v) = \hat{\alpha}(c, v) * d_u(c, v) + q_u(c, v)$.

6.3. Node isolation and addition

If a large fraction of its neighbors are compromised, a node may not be able to update its group key and thus be isolated. To address this problem, 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 realize the above 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 distrib-

utes 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 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 u is isolated later, it will send the encrypted polynomial to the newly deployed neighboring nodes. On receiving the encrypted polynomial, a new node (say v) can decrypt it with K_0, obtain a share $e_u(x,v)$, and finally remove $e_u(x,y)$.

6.4. Other issues

In the PCGR schemes, the keys of an arbitrary group i are generated by group key polynomial $g_i(x)$, where the degree of x is s. If $s+1$ or more distinct keys of the same group are exposed, $g_i(x)$ could be reconstructed by the adversary. To thwart this attack, instead of using $g_i(c)$ directly as the group key for the c^{th} period, we can use $H(g_i(c))$ as the group key, where $H(.)$ is a one-way hash function. By this, only the hashed values of $g_i(x)$ are exposed. The one-wayness property of function H prevents the adversary from deriving the shares of $g_i(x)$ that were used in the past rekeying operations.

In the proposed schemes, group keys are updated periodically. While 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, data reports forwarded during a threshold time interval after the rekeying operation should still be authenticated using the previous group key. This threshold time interval could be determined by the maximal time synchronization error in the sensor network.

7. Conclusions

In this paper, we have proposed a family of *predistribution and local collaboration-based group rekeying* (*PCGR*) schemes to address the node compromise problem. These schemes are based on the idea that future group keys can be preloaded to nodes before deployment, and neighbors can collaborate to protect and appropriately use the preloaded keys. Extensive analysis 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 several previously proposed schemes, and significantly improve the effectiveness of filtering false data.

References

[1] H. Chan, A. Perrig, Security and privacy in sensor networks, in: IEEE Computer, October 2003.
[2] S. Zhu, S. Setia, S. Jajodia, LEAP: efficient security mechanisms for large-scale distributed sensor networks, in: The 10th ACM Conference on Computer and Communications Security, 2003.
[3] A. Perrig, R. Szewczyk, V. Wen, D. Culler, J. Tygar, SPINS: security suite for sensor networks, in: Mobicom'01, 2001.
[4] F. Ye, H. Luo, S. Lu, L. Zhang, Statistical en-route filtering of injected false data in sensor networks, in: IEEE Infocom'04, March 2004.
[5] S. Marti, T. Giuli, K. Lai, M. Baker, Mitigating routing misbehavior in mobile ad hoc networks, in: ACM MobiCom, August 2000.
[6] G. Wang, W. Zhang, G. Cao, T. La Porta, On supporting distributed collaboration in sensor networks, in: IEEE Military Communications Conference (MILCOM), October 2003.
[7] A. Seshadri, A. Perrig, L. van Doorn, P. Khosla, SWATT: software-based attestation for embedded devices, in: IEEE Symposium on Security and Privacy, 2004.
[8] H. Hugh, C. Muckenhirn, T. Rivers, Group key management protocol architecture, Request for comments (RFC) 2093, Internet Engineering Task Force, March 1997.
[9] D. Wallner, E. Harder, R. Agee, Key management for multicast: issues and architectures.
[10] C. Wong, M. Goudaand, S. Lam, Secure group communications using key graphs, in: ACM SIGCOMM'98, 1998.
[11] D. Balenson, D. McGrew, A. Sherman, Key management for large dynamic groups: one-way function trees and amortized initialization, IETF Internet Draft, August 2000.
[12] C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro, M. Yung, Perfectly-secure key distribution for dynamic conferences, Lecture Notes in Computer Science 740 (1993) 471–486.
[13] Crossbow Technology Inc., Wireless sensor networks, <http://www.xbow.com/Products/Wireless_Sensor_Networks.htm>.
[14] A. Perrig, R. Szewczyk, V. Wen, D. Culler, J. Tygar, Spins: security protocols for sensor networks, Wireless Networks 8 (5) (2002) 521–534.
[15] H. Yang, F. Ye, Yuan Yuan, S. Lu, W. Arbaugh, Toward resilient security in wireless sensor networks, in: ACM MOBIHOC'05, May 2005.
[16] L. Eschenauer, V. Gligor, A key-management scheme for distributed sensor networks, in: The 9th ACM Conference on Computer and Communications Security, November 2002, pp. 41–47.
[17] H. Chan, A. Perrig, D. Song, Random key predistribution schemes for sensor networks, in: IEEE Symposium on Research in Security and Privacy, 2003.
[18] D. Liu, P. Ning, Establishing pairwise keys in distributed sensor networks, in: The 10th ACM Conference on Computer and Communications Security, 2003.
[19] W. Du, J. Deng, A pairwise key pre-distribution schemes for wireless sensor networks, in: The 10th ACM Conference on Computer and Communications Security, 2003.
[20] Haowen Chan, Adrian Perrig, PIKE: peer intermediaries for key establishment in sensor networks, in: The 24th Conference of the IEEE Communications Society (Infocom 2005).
[21] J. Hall, M. Barbeau, E. Kranakis, Detection of transient in radio frequency fingerprinting using phase characteristics of signals, in: Proceedings of the 3rd IASTED International Conference on Wireless and Optical Communications (WOC), 2003.
[22] K. Sun, P. Ning, C. Wang, Fault-tolerant cluster-wise clock synchronization for wireless sensor networks, IEEE Transactions on Dependable and Secure Computing (TDSC) (2005).
[23] X. Zhang, S. Lam, D. Lee, Y. Yang, Protocol design for scalable and reliable group rekeying, IEEE/ACM Transactions on Networking 11 (6) (2003) 908–922.
[24] W. Zhang, G. Cao, Group rekeying for filtering false data in sensor networks: a predistribution and local collaboration-based approach, in: IEEE INFOCOM , March 2005.
[25] S. Zhu, W. Zhang, Group key management, in: Yang Xiao (Ed.), Security in Sensor Networks, CRC Press, 2006.
[26] D. McGrew, 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.

Wensheng Zhang received his BS degree from Tongji University, Shanghai, China, his MS degree from The Institute of Computing Technology, Chinese Academy of Science, and his PhD degree from the Pennsylvania State University. He has been an assistant professor at the department of Computer Science, Iowa State University, since August 2005. His research interests are wireless networks, mobile computing, and network security.

Sencun Zhu received the B.S. degree in Precision Instruments from Tsinghua University, Beijing, China, in 1996 and the M.S. degree in Signal Processing from University of Science and Technology of China, Graduate School at Beijing, in 1999. He received the PhD degree in Information Technology from George Mason University in 2004. His research interests include network and systems security, ad hoc and sensor networks, performance evaluation, peer-to-peer computing. Currently he is working on issues related to ad hoc and sensor network security, DDoS attack prevention, and Worm detection. His research is funded by NSF and ARO. He is also a member of the Networking and Security Research Center, the Systems and Internet Infrastructure Security Lab, and the Cyber Security Lab.

Guohong Cao received his BS degree from Xian Jiaotong University, Xian, China. He received the MS degree and PhD degree in computer science from the Ohio State University in 1997 and 1999, respectively. Since then, he has been with the Department of Computer Science and Engineering at the Pennsylvania State University, where he is currently a Full Professor. His research interests are wireless networks and mobile computing. He has published over 100 papers in the areas of sensor networks, wireless network security, data dissemination, resource management, and distributed fault-tolerant computing. He has served on the editorial board of the IEEE Transactions on Mobile Computing and IEEE Transactions on Wireless Communications, and has served on the program committee of many conferences. He was a recipient of the NSF CAREER award in 2001.