A Cluster-Based Random Key Revocation Protocol for Wireless Sensor Networks

Yi Jiang and Hao-Shan Shi

***Abstract*—In recent years, several random key pre-distribution schemes have been proposed to bootstrap keys for encryption, but the problem of key and node revocation has received relatively little attention. In this paper, based on a random key pre-distribution scheme using clustering, we present a novel random key revocation protocol, which is suitable for large scale networks greatly and removes compromised information efficiently. The revocation protocol can guarantee network security by using less memory consumption and communication load, and combined by centralized and distributed revocation, having virtues of timeliness and veracity for revocation at the same time.**

***Index Terms*—Cluster-based, key pre-distribution, revocation, wireless sensor networks.**

1. Introduction

Because sensor nodes have many limits[1],[2] in the process of communication, we use symmetric key cryptosystems and one-way function instead of asymmetric key cryptosystems. To ensure security for information sent among nodes, key management in sensor networks is an important issue.

Key management includes two primary aspects: key distribution and key revocation. To generate keys for encryption successfully, several key pre-distribution schemes[3]-[7] have been proposed, however, key revocation protocol which refers to securely removing keys and nodes that have been compromised has received relatively little attention. Eschenauer and Gligor[3] proposed a centralized revocation scheme, which needs base station to broadcast a revocation message to all the sensor nodes. The scheme induces low speed revocation and large communication load for base station. A distributed revocation scheme is put forward by Chan *et al*.[4], which is suitable for the random pairwise scheme solely. Dini *et al*.[8] described a group key revocation protocol, which is unsuitable for random key pre-distribution schemes.

In this paper, we present a novel random key revocation protocol based on a random key pre-distribution scheme using clustering. The revocation protocol is suitable for large scale networks greatly and removes compromised information efficiently. Because the network topology is a two-level hierarchical, we consider the revocation protocol from two aspects: in a cluster and between clusters. In a cluster, the revocation protocol is combined by centralized and distributed revocation with virtues of timeliness and veracity. Between clusters, the revocation protocol can guarantee network security by using less memory consumption and communication load. In addition, we also propose a special method to set up secure links for any two clusters, which is a high point in the paper, except for the revocation protocol.

The remainder of the paper is organized as follows. A random key pre-distribution scheme is described in Section 2. Section 3 details the cluster-based random key revocation protocol from two aspects: in a cluster and between clusters. Finally, our concluding remark is exposed in Section 4.

2. Cluster-Based Random Key Pre-Distribution Scheme

In this Section, we will briefly introduce a novel random key pre-distribution scheme based on cluster that we have presented previously.

2.1 Structure of Cluster-Based Sensor Networks

To insure effective communication, we divide all the sensor nodes into several clusters. The network topology is compartmentalized into two-level hierarchical, which are clusters and corresponding cluster head nodes. Each cluster corresponds to a region in the deployment field. Fig. 1 depicts the clustering structure.

Since the clusters reside in different areas, they may have different probabilities to be attacked. The performance of the scheme (such as connection probability and resilience against node capture) can be dealt with respectively in different clusters.

Nodes in different clusters can communicate with each other by their head nodes. Cluster nodes share a cluster key

Manuscript received July 3, 2007; revised September 27, 2007. This work was supported by the Ministry of Education Doctor Foundation in China under Grant No. 20050699037.

Y. Jiang and H.-S. Shi are with School of Electronics and Information, Northwestern Polytechnical University, Xi'an, 710072, China (e-mail: jiangyiv88@nwpu.edu.cn)

with its head node, and use this key encrypt communication with its head node. The identifiers of all nodes in a cluster must be stored in the memory of the head node, so the head node can determine whether the message belongs to its member nodes or not, when it receives a message from other cluster head node. The method of setting up communication between any two cluster head nodes will be expatiated in Section 3.2.

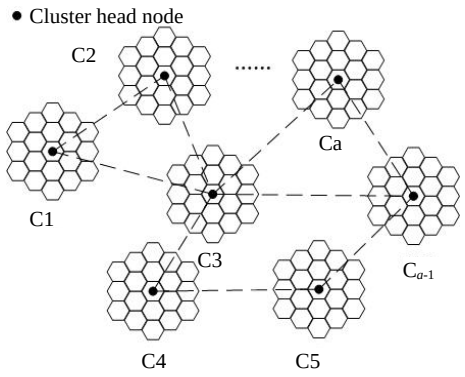

Fig. 1. Clustering structure.

2.2 Key Pre-Distribution Scheme in a Cluster

In our scheme, the deployment region of each cluster is divided into multiple hexagonal sub-regions, and cluster nodes are evenly deployed. If they belong to the different sub-regions, the key rings are chosen from the different key pools. The probabilities of sharing common keys between each sub-region and its neighboring sub-regions are equal, and the key for encryption of each link can be generated efficiently. The process of this scheme is similar to that of the previous scheme[5], only differing in the initialization and the method of setting up link-keys which is the same as the *q*-composite keys scheme[4], so we discuss only the initialization phase here.

This phase is conducted offline before the sensor nodes are deployed. First, the deployment field of cluster C_i is divided into multiple hexagonal sub-regions $Z_{i(\eta)}$ (for η=1,2,⋯,*v*). Then we need to divide the key pool S_i into *v* sub-key pools $S_{i(\eta)}$ (for η=1,2,⋯,*v*), with $S_{i(\eta)}$ corresponding to the deployment sub-region $Z_{i(\eta)}$. We assume the two sub-key pools share a few keys, only when their corresponding deployment sub-regions are neighbors. Otherwise they have no keys to share. To do this, the effect on any secure communication links among uncompromised nodes is reduced, when some nodes are compromised by adversaries. After the sub-key pools are set up, we select m_i random keys from the sub-key pool $S_{i(\eta)}$ (where m_i is the number of keys, each node can carry in its key ring) and store them into the memory of the nodes which will deploy in the corresponding sub-region $Z_{i(\eta)}$.

3. Cluster-Based Random Key Revocation Protocol

The centralized approach and the distributed approach are the mainly two key revocation approaches in recent years. In order to coordinate the random key pre-distribution scheme that has been depicted in Section 2, we propose a new key revocation protocol, which improves the two approaches and is applied to the complex hierarchical structure. Because the network topology is two-level hierarchical, the protocol can be divided into two parts: in a cluster and between clusters.

3.1 Revocation Protocol in a Cluster

A. Preparation to Revocation

Before deployed in the sub-region $Z_{i(\eta)}$, every node has selected m_i keys to store in its key ring from the sub-key pool $S_{i(\eta)}$. To revoke nodes and keys which are compromised, a node must store some additive information.

We assume that each node in the sub-region $Z_{i(\eta)}$ possesses u_i nodes has the right to vote against it, in which only *w* nodes are needed to remove a node from the network. Degree of a node in sub-regions $Z_{i(\eta)}$ is denoted by d_i. In ideal condition, only d_i nodes which are neighbors and have direct connection ability with the node can vote against it. If the actual degree of the node is below *w*, the node will be revoked.

1) Each node must store IDs of u_i nodes which have the right to vote against it. When two neighboring nodes set up a secure link with each other, they exchange their IDs. If one node ID belongs to the set of IDs stored in another node, it becomes a voting member to that node with real voting right.

2) Each vote is a special code. u_i votes which vote against the same node are calculated by hash function, the outcome of which generates a Merkle tree[9]. Each of u_i nodes uses the root hash value and hash values of the $\log_2 u_i$ brother path nodes from the root to a leaf to authenticate a vote corresponding to the leaf node. From the above discussion, each node must store u_i root hash values and $u_i \log_2 u_i$ brother path nodes hash values and u_i votes, because it possibly becomes a voting member to u_i nodes. (Brother path node means the node who has the same father as the corresponding path node.)

3) In order to send votes to other voting members, which have voting right to the same node, each node must also store IDs of other voting members.

From the above depiction, the memory consumption to every node for voting is denoted by

$$S_{mer} = \{u_i^2(ID) + u_i(root) + u_i \log_2 u_i(path) + u_i(vote)\} \quad (1)$$

and the space complexity is $O(u_i \log_2 u_i)$.

B. Vote Mechanism

To describe the vote mechanism, we take Fig. 2 for instance.

1) To prevent widespread release of revocation keys by compromised nodes, we require that only voting members (see from clause 1) of A) of node A have the right to vote against A. Each vote is encrypted by A, and can be used only by decrypting of A, which insures to connect directly with A.

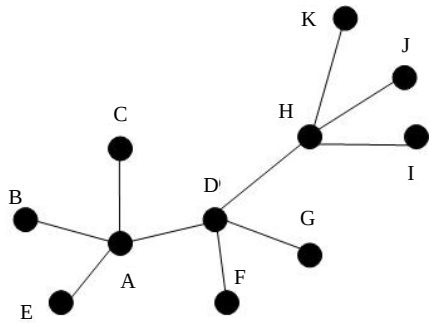

Fig. 2. Voting relationship among nodes.

After the shared-key discovery phase, nodes B, C, D, and E become voting members of node A, and they have the right to vote. When vote begins, A sends k_A to B, C, D, and E to decrypt their votes. We suppose that node D wants to vote against A and D broadcasts its vote k_D and path hash values (see from clause 2) of A), which is depicted in Fig. 3.

2) When voting members of A receive the message, they will calculate the information from the message, and the result will be compared with the root hash value stored in their memory. If they are equal, voting members amend flags to indicate that A is voted once. If a flag notes *w* votes, the voting member will send this information to its cluster head node.

For example, B, C, and E receive the message from D, they find D is a voting member of A like them (see from clause 3) of A), and then they compute the message to prove validity of vote and amend their flags.

$$\text{root}_A = H(H(\cdots H(H(k_D) \mid H(k_E))\cdots)) \quad (2)$$

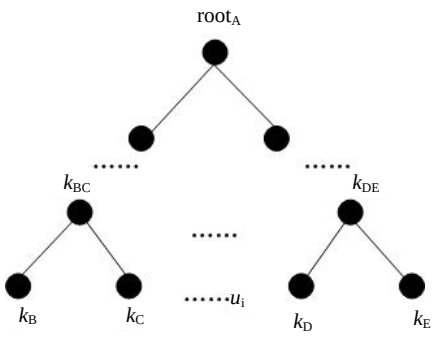

Fig. 3. Voting Merkle tree.

3) A voting member will broadcast a vote message every Δt time to ensure that the vote message could be sent to destination. If other voting members receive the vote message, they will reply it, so broadcast is over.

To ensure the revocation decision is completed in a timely fashion, we prescribe the revocation decision and execution occur within a bounded time period Δd ($\Delta d >> \Delta t$). The whole life of the network is divided into a large number of time periods, within every period Δd, if there is an insufficient quorum of sensor nodes agreeing that a node is to be revoked, then the revocation decision returns to a negative result, which avoids the erroneous votes that can accumulate over the network's lifetime and result in the revocation of a legitimate node.

If a voting member votes in the current time period which is close to finish, it will also vote in the next time period, because the vote maybe not be counted in the current revocation decision.

C. Revocation Mechanism

The revocation mechanism is the last approach for our scheme to revoke nodes in a cluster. If a voting member finds its vote flag of node A comes up to *w*, it will cut off the link with A, and inform the cluster head node.

A node in one hexagonal sub-region is revoked, which can only affect nodes in six hexagonal sub-regions around the sub-region where the captured node resides, because only two neighboring sub-regions share the same keys, and no key is shared by more than two neighboring sub-regions. If the cluster head node confirming node A is compromised, it will broadcast a single revocation message containing a signed list of m_i key IDs of the key ring to be revoked and the compromised node ID to the nodes in seven hexagonal sub-regions related to node A. Each node there can decrypt the message by using the public cluster key, removing the link with node A, and checking whether its key ring includes the revoked keys, then removing the compromised keys.

Once the keys are removed from key rings, some links may disappear, and the affected nodes need to reconfigure those links by restarting the shared-key discovery and path-key establishment phase.

D. Performance Analysis

In a cluster, the revocation protocol is compound with centralized revocation and distributed revocation. At first, nodes distribute vote and deal with the compromised links by themselves with virtue of timeliness, then depend on cluster head node to remove captured keys with virtue of veracity, preventing adversary from extending invasion.

3.2 Revocation Protocol between Clusters

The cluster head node is a special node, which usually has larger computation and communication capabilities than other nodes within the cluster. We can assume that all the

head nodes are fully connected and the communication links are not easily compromised by the adversaries.

A. Setting up Communication between Any Two Cluster Head Nodes

To connect all cluster head nodes efficiently, we set up a hierarchical structure of symmetric keys, which is not managed by key center (KC), called key management tree. Fig. 4 depicts the tree, and it has 2 dimensions. We assume that dimension of the tree can be changed at will.

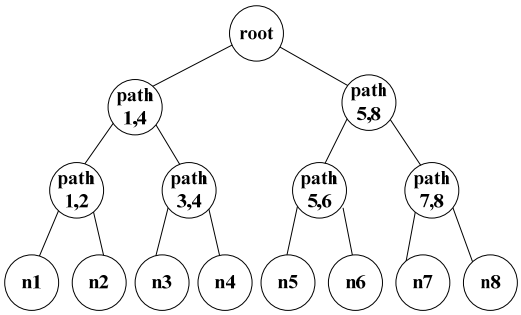

Fig. 4. Key management tree.

Each leaf in the tree is cluster head node, which is managed by its father node. The father node can be also managed by its father, so a key management tree is established. We call all father nodes path nodes, except for root node. We suppose that each sensor node shares a private-key with *KC*, denoted by k_{nx}. *KC* will remember all path nodes keys and manage them by the root node key. We denote the path nodes keys and root node key by k_{pathx} and k_{root} respectively.

If there are too many cluster head nodes need to communicate with each other, and all connections must be established by *KC*, burden of *KC* will become too large. The key management tree can reduce the communication load of *KC*. All cluster head nodes can set up communication links by themselves. Each leaf node stores a set of keys of path nodes lying on the path from the root to itself and all keys of path nodes possess their own ID. When the setting up communication phase begins, each cluster head node broadcasts IDs of the stored keys which are called ID-Ring to other nodes. The process to negotiate about communication key between two neighboring head nodes can be considered into the following several steps.

1) If two neighboring nodes have the same stair father node, they can communicate with each other. They will encrypt a link key by the stair father node key and exchange it within the scope of transmission radius.

2) After all nodes which have the same stair father have established the communication links, there are some neighboring nodes with different stair father did not set up links with each other. We will search the same secondary father nodes for them. They will encrypt a link key by the second level father nodes key and exchange it within the radius of transmission to communicate each other.

3) We can use this method to the root node by analogy, until all neighboring nodes can connect with others.

Taking Fig. 4 for instance, we depict the concrete process of setting up communication links as follows.

a) n_3 and n_4 are neighbors: to connect with each other, they broadcast their ID-Ring within range of transmission firstly.

$$\mathrm{ID-Ring}_3 = \{n_3, \mathrm{ID}_{\mathrm{path3,4}}, \mathrm{ID}_{\mathrm{path1,4}}, \mathrm{ID}_{\mathrm{root}}\}$$

$$\mathrm{ID-Ring}_4 = \{n_4, \mathrm{ID}_{\mathrm{path3,4}}, \mathrm{ID}_{\mathrm{path1,4}}, \mathrm{ID}_{\mathrm{root}}\}$$

We can see n_3 and n_4 have the same stair father node $\mathrm{ID}_{\mathrm{path3,4}}$, so n_3 can use $k_{\mathrm{path3,4}}$ to encrypt their communication key $k_{3,4}$ and transmit it to n_4. n_4 has the $k_{\mathrm{path3,4}}$, so it can decrypt the $k_{3,4}$. Then they can use $k_{3,4}$ to encrypt the following date.

$$\mathrm{Key}_{\mathrm{pack}} = E_{k_{\mathrm{path3,4}}}\{k_{3,4}\}$$

b) If n_1 and n_4 are neighbors. To connect with each other, they broadcast their ID-Ring within range of transmission.

$$\mathrm{ID-Ring}_1 = \{n_1, \mathrm{ID}_{\mathrm{path1,2}}, \mathrm{ID}_{\mathrm{path1,4}}, \mathrm{ID}_{\mathrm{root}}\}$$

$$\mathrm{ID-Ring}_4 = \{n_4, \mathrm{ID}_{\mathrm{path3,4}}, \mathrm{ID}_{\mathrm{path1,4}}, \mathrm{ID}_{\mathrm{root}}\}$$

We can see n_1 and n_4 have the same secondary father node $\mathrm{ID}_{\mathrm{path1,4}}$, so n_1 can use $k_{\mathrm{path1,4}}$ to encrypt their communication key $k_{1,4}$ and transmit it to n_4. n_4 has the $k_{\mathrm{path1,4}}$, so it can decrypt the $k_{1,4}$. Then they can use $k_{1,4}$ to encrypt the following date.

$$\mathrm{Key}_{\mathrm{pack}} = E_{k_{\mathrm{path1,4}}}\{k_{1,4}\}$$

c) If n_2 and n_5 are neighbors. To connect with each other, they broadcast their ID-Ring within range of transmission.

$$\mathrm{ID-Ring}_2 = \{n_2, \mathrm{ID}_{\mathrm{path1,2}}, \mathrm{ID}_{\mathrm{path1,4}}, \mathrm{ID}_{\mathrm{root}}\}$$

$$\mathrm{ID-Ring}_5 = \{n_5, \mathrm{ID}_{\mathrm{path5,6}}, \mathrm{ID}_{\mathrm{path5,8}}, \mathrm{ID}_{\mathrm{root}}\}$$

We can see n_2 and n_5 have different father node except root, so n_2 can use k_{root} to encrypt their communication key $k_{2,5}$ and transmit it to n_5. n_5 has the k_{root}, so it can decrypt the $k_{2,5}$. Then they can use $k_{2,5}$ to encrypt the following date.

$$\mathrm{Key}_{\mathrm{pack}} = E_{\mathrm{root}}\{k_{2,5}\}$$

All the neighboring nodes can establish connections with each other by themselves. This method increase the memory consumption and communication load for each node, but it can reduce the burden of *KC* greatly, and can be used to renew the compromised keys in the next section.

B. Renewing Keys Stored in Cluster Head Node

In this section, we will discuss how to insure security when some head nodes are captured.

1) Key chain

To avoid being captured by adversaries, we assume that every node key is composed of a key chain. We must change the compromised keys after the capture action happened. The nodes in the tree are able to authenticate a renew message and use it to their new node key. We use a hash function $H(\ )$ to form a node key chain, which is denoted by (3). k_n is generated by pseudorandom sequence generator, and k_0 has been distributed to corresponding cluster head nodes before they are deployed. k_i will be dispensed to the corresponding nodes when every capture action happens, then nodes can prove it by the hash function $H(\ )$. The key that nodes receive is the correct one, if the result of the key computed by $H(\ )$ equals to the former one. n is the length of a key chain corresponding to the life cycle of the network.

$$k_i = H(k_{i+1}), \quad 0 \le i \le n-1 \tag{3}$$

2) Renewing compromised keys

The method of renewal in this section is similar to [4]. Every cluster head node store a set of current path node keys from root to itself, called Set-Key. If a head node is compromised, other connected nodes with it will remove the links. But all keys belong to the head node's Set-Key are captured, we must renew them to other nodes which can use the keys to set up new links. These keys must be securely sent to related nodes, expect for the compromised node. We assume that *patha* is a path node lying on the path from root to the compromised node, and *pathb* is its child node. The principle of renewing keys as follows:

a) If *pathb* is a leaf node, *KC* will use a private key with *pathb* to encrypt the next key of *patha*, then send it to *pathb*. The method is the same to other child nodes of *patha*, except for the compromised node.

b) If *pathb* is an internal tree node and belongs to the compromised path, *KC* will use a next key of *pathb* to encrypt the next key of *patha*, then send it to the corresponding child leaf nodes of *pathb*, except for the compromised node.

c) If *pathb* is a internal tree node but dose not belong to the compromised path, *KC* will use a current key of *pathb* to encrypt the next key of *patha*, then send it to the corresponding child leaf nodes of *pathb*.

Taking Fig. 4 for instance, we depict the detailed process of renewing compromised keys. If n_5 has been captured, its ID-Ring includes $\mathrm{ID}_{\mathrm{path}5,6}$, $ID_{\mathrm{path}5,8}$ and ID_{root}, so *KC* must renew these path nodes keys to each corresponding head node.

- For *path5,6*, its child node is leaf n_5 and leaf n_6, and n_5 is compromised, so *KC* will send the next key of *path 5,6* to n_6 ($E_{k_6}\{k_{\mathrm{next}5,6}\}$).
- For *path5,8*, its child node is *path5,6* and *path7,8*. *path5,6* accords with (2), so *KC* will send the message ($E_{k_{\mathrm{next}5,6}}\{k_{\mathrm{next}5,8}\}$) to *path5,6*'s child node n_6. *path7,8* accords with (3), so *KC* will send the message ($E_{k_{\mathrm{cur}7,8}}\{k_{\mathrm{next}5,8}\}$) to *path7,8*'s child nodes n_7 and n_8.
- For root, its child node is *path1,4* and *path5,8*. *path1,4* accords with (3), so *KC* will send the message ($E_{k_{\mathrm{cur}1,4}}\{k_{\mathrm{root(next)}}\}$) to *path1,4*'s child leaf nodes $n_1 \sim n_4$. *path5,8* accords with (2), so *KC* will send the message ($E_{k_{\mathrm{next}5,8}}\{k_{\mathrm{root(next)}}\}$) to *path5,8*'s child leaf nodes $n_6 \sim n_8$.

Every leaf node can use its corresponding key to decrypt the received message, and use $H(\ )$ to authenticate whether the new key is a correct one to key chain.

3) Revoking communication to compromised nodes

When a compromised node is removed from the whole tree, all head nodes previously connected with it must detach links with it. Because revocation can induce some nodes to have no affiliation to their neighboring nodes, and the setting up communication phase will restart. They use new path keys to establish new communication links, and generate the new link-keys. This process is accomplished automatically, not related to *KC*. When a head node is captured, it can not participate in receiving any message. How to judge whether a head node is compromised can refer to section3.1.

C. Performance Analysis

Among clusters, all the head nodes can establish connections by storing a set of keys of path nodes, do not depend on *KC*, and avoid remembering all node keys to set up connections, so the storage consumption is low. The scheme of renewing keys stored in cluster node head can improve performance by reducing the complexity and the number of re-keying messages. As to the complexity of re-keying messages, we use symmetry encryption algorithm which has lower complexity than digital signature, and our key management tree has better secure performance to reach the same purpose to dissymmetrical encryption algorithm. As to the number of re-keying messages, *KC* needs to broadcast $(\alpha/2)\log_{\dim}\alpha + (\alpha/2)$ messages, not to send $2\alpha(d-1)$ messages to set up a connected network, which can reduce communication load greatly (dim is the dimension of tree, α is the number of clusters).

4. Conclusions

We describe a novel random key revocation protocol, which is suitable for large scale networks greatly and compromised secrets information removing. From the analysis of performance, in a cluster, the revocation protocol is combined by centralized and distributed revocation with

virtues of timeliness and veracity, and between clusters, the revocation protocol can guarantee network security with less storage consumption and communication load.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor network," *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102-114, Aug. 2002.

[2] D. W. Carman, P. S. Kruus, and B. J. Matt, "Constraints and approaches for distributed sensor network security," *NAI Labs Technical Report #00-010*, Sep. 1, 2000.

[3] L. Eschenauer and V. D. Gligor. "A key-management scheme for distributed sensor networks," in *Proc. 9th ACM Conference on Computer and Communications Security*, Washington, DC, 2002, pp. 41-47.

[4] H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks," in *Proc. IEEE Symposium on Security and Privacy*, Berkeley, California, 2003, pp. 197-213.

[5] W. Du, J. Deng, and Y.-S. Han, "A key management scheme for wireless sensor networks using deployment knowledge," in *Proc. IEEE INFOCOM*, Hong Kong, 2004, pp. 172-183.

[6] W. Du, J. Deng, and Y.-S. Han, "A pairwise key pre-distribution scheme for wireless sensor networks," in *Proc. 10th ACM Conference on Computer and Communications Security*, Washington DC, 2003, pp. 42-51.

[7] D. Liu and P. Ning, "Establishing pairwise keys in distributed sensor networks," in *Proc. 10th ACM Conference on Computer and Communications Security*, Washington DC, 2003, pp. 52-61.

[8] G..Dini and I. M. Savino, "An efficient key revocation protocol for wireless sensor networks," in *Proc. International Symposium on a World of Wireless, Mobile and Multimedia Networks*, 2006, pp. 3-5.

[9] R. Merkle, "Protocols for public key cryptosystems," presented at the IEEE Symposium on Research in Security and Privacy, 1980.

Yi Jiang was born in Heilongjiang Province, China, in 1980. She received the B.S. and M.S. degrees in communication engineering from Northwestern Polytechnical University, China, in 2002 and 2005, respectively. She is currently a Ph.D. student and a lecturer with School of Electronics and Information, Northwestern Polytechnical University. Her research interests include wireless communication and wireless sensor network security.

Hao-Shan Shi was born in Shaanxi Province, China, in 1946. He received the M.S. degree in electronic engineering from Northwestern Polytechnical University, China, in 1982. Currently, he is a professor with School of Electronics and Information, Northwestern Polytechnical University, and the Vice Director of the Institute of Multimedia Communication and Software Engineering, Northwestern Polytechnical University. His research interests include wireless communication, software defined radio, and wireless multimedia signal processing.