SecRout: A Secure Routing Protocol for Sensor Networks

Jian Yin

Sanjay Kumar Madria
Missouri University of Science and Technology, madrias@mst.edu

SecRout: A Secure Routing Protocol for Sensor Networks

Jian Yin Sanjay Madria
Department of Computer Science, University of Missouri-Rolla, MO 65401, USA
{jian, madrias}@umr.edu

Abstract

In this paper, we present a ***Sec****ure* ***Rout****ing Protocol for Sensor Networks (SecRout) to safeguard sensor networks under different types of attacks. The SecRout protocol uses the symmetric cryptography to secure messages, and uses a small cache in sensor nodes to record the partial routing path (previous and next nodes) to the destination. It guarantees that the destination will be able to identify and discard the tampered messages and ensure that the messages received are not tampered. Comparing the performance with non-secure routing protocol AODV [1] (Ad hoc on Demand Distance Vector Routing), the SecRout protocol only has a small byte overhead (less than 6%), but packet delivery ratio is almost same as AODV and packet latency is better than AODV after the route discovery.*

Keywords: Routing, Security, Sensor Networks

1. Introduction

For many sensor network applications, security is one of the most important aspects [2,3]. Secure routing protocols in sensor networks present challenges due to resource limitations, the ad hoc nature, and no centrally administered secure routers. An existing route could be broken down or a new route could be prevented from being established because of the attacks [4-6]. There are several examples of attacks against routing in sensor networks. For instance, the routing packet is dropped or tampered; the attacker inserts spurious messages in the sensor network.

In this paper, we propose a **Sec**ure **Rout**ing Protocol for Sensor Networks (*SecRout*) to safeguard sensor networks using the two-level architecture. The sensor network is divided into clusters with one cluster head for each cluster. The nodes in the cluster send the data to the cluster head instead of sending data directly to the sink node (the destination node). Then, the cluster head aggregates the data from the sensor nodes in the cluster and sends it to the sink node. The two-level architecture can greatly lower the message overhead. It can greatly save the energy, and decrease the usage of memory and bandwidth.

The *SecRout* protocol guarantees that the sink node will be able to identify and discard the tampered messages and ensure that the accepted messages are not tempered with. The asymmetric cryptography algorithms [7-9] for cryptography and authentication are not suitable for sensor networks due to the large computation and communication overhead. The *SecRout* protocol uses symmetric cryptography to secure the messages, and uses a small cache in sensor nodes to record the partial routing path (previous and next nodes) to the sink node. Due to the frequent communications in the cluster, we use the cluster key in the cluster to secure the messages. The cluster head uses the preloaded key to secure aggregated messages to send them to the sink node.

In the remainder of the paper, the detail protocol description is given in Section 2. Section 3 describes security analysis. Section 4 provides performance evaluation of the *SecRout* protocol. Section 5 addresses the related works. We conclude in Section 6.

2. Secure Routing Protocol

In the proposed *SecRout* protocol, we assume that every node has a unique identity (*ID*) and a preloaded key [4]. The network is divided into clusters after self-organization [10] and each cluster has a cluster head, which knows the *ID* of the sensor node in its cluster. The sensor node knows the *ID* of its cluster head. The sink node knows the topology of the sensor network after self-organization. We assume that the sink node is a super node, which has more power and memory. It stores a table containing (*ID, Key*) pairs of all the sensor nodes. In the proposed *SecRout* protocol, we have one sink node, which is trusted. All other sensor nodes can be compromised.

We build the secure route from the source node to the sink node using *SecRout*. We guarantee that a packet reaches the sink node even if malicious nodes exist on the route. It guarantees that the message is originated from the authenticated source node and is not tampered on the route. In our secure route discovery, the malicious node on the route can be detected. Therefore, the route created using our route discovery protocol is secure. The *SecRout* protocol has the following four features:

1) The routing packet and data packet are very small because they only include the partial path information. We do not use the source routing because in source routing the identities of the traversed intermediate nodes are accumulated in the route request [11].
2) We create the route first, and then we forward the data to the sink node along the route. We do not flood the data packet to the sink node because the data packet has much bigger size than the routing packet.
3) It uses the two-level architecture, where the cluster head aggregates the data, then sends it to the sink node along the route. It lowers the communication overhead.
4) It only uses high efficient symmetric cryptographic operations to secure messages.

Table 1 provides the notation description which will be used in this paper.

Table 1. Notation description

Notation	Description
A, B	Principles, such as sensor nodes
$M_1 \| M_2$	The concatenation of messages M_1 and M_2
K_A	The secret key held by *A*
MAC(*K,M*)	The message authentication code of message *M* using a symmetric key *K* [12]
$E_K(M)$	Encryption of message M with key *K* [13]
N_A	A nonce generated by node *A*, which is a random number
ID_A	The identity of node *A*

2.1. Secure Route Discovery

The source node initiates the route discovery process through sending the route request (*RREQ*) to the sink node. When the sink node receives the *RREQ*, it creates the route reply (*RREP*) to the source node. After the route discovery, every node along the route has established the appropriate routing table.

2.1.1. Secure Route Request. The Source node initiates the route discovery by broadcasting the *RREQ* to its neighbors, which is constructed as follows:

$$\begin{Bmatrix} ID_{source} \mid ID_{\sin k} \mid ID_{RREQ} \mid N_{source} \mid \\ MAC(K_{source}, ID_{source} \mid ID_{\sin k} \mid ID_{RREQ} \mid N_{source}) \end{Bmatrix}$$

Where ID_{RREQ} is a random number.

The intermediate node only accepts the first *RREQ* according to ID_{RREQ}. The routing table is updated through adding the *ID* of the previous two hops, ID_{source}, and ID_{RREQ}. Then, it updates the *RREQ*. It adds its *ID* to the *RREQ* which is directly from the source node. Or it replaces ID_{this}, ID_{pre} embedded in the *RREQ* with its previous and current *ID* if the *RREQ* is from other nodes. Finally, it broadcasts the updated *RREQ*, which is constructed as follows:

$$\begin{Bmatrix} ID_{this} \mid ID_{pre} \mid ID_{source} \mid ID_{\sin k} \mid ID_{RREQ} \mid N_{source} \mid \\ MAC(K_{source}, ID_{source} \mid ID_{\sin k} \mid ID_{RREQ} \mid N_{source}) \end{Bmatrix}$$

Where ID_{pre} and ID_{this} are the *IDs* of previous and current node.

When the sink node receives the *RREQ*, it gets the key of the source node from the (*ID, Key*) pair table, which is used to verify the *MAC*. It only accepts the first *RREQ* with the valid *MAC* according to the source node and ID_{RREQ}. Then it stores ID_{source}, ID_{RREQ} and previous two hops in the routing table.

2.1.2. Secure Route Reply. When the sink node accepts the *RREQ*, it constructs the *RREP* as follows:

$$\begin{Bmatrix} ID_{pre} \mid ID_{this} \mid ID_{next} \mid ID_{source} \mid ID_{\sin k} \mid ID_{RREQ} \mid N_{\sin k} \mid \\ MAC(K_{source}, ID_{source} \mid ID_{\sin k} \mid ID_{RREQ} \mid N_{\sin k}) \end{Bmatrix}$$

Where ID_{next} is the *ID* of the next node. Then it broadcasts it.

The intermediate node with the same *ID* as the ID_{pre} embedded in the *RREP* updates the ID_{pre}, ID_{this} and ID_{next} in the *RREP* with the *IDs* of its previous, current, and next nodes. Then it broadcasts the updated *RREP* packet. Simultaneously it updates its routing table to add *IDs* of the next two hops. If it cannot get acknowledge from its next hop within the specified time, the previous node may be a malicious node. It drops any other *RREQs* during the next route discovery. If it detects the *RREP* broadcasted by its previous hop with the wrong ID_{pre} since it stores two previous hops in its routing table, it blocks its previous hop which may be a malicious node.

The source node verifies the *MAC* after it receives the *RREP*. If the verification succeeds, it inserts the *IDs* of the next two hops on the route to its routing table. After the route discovery, the intermediate node has the routing table as shown in Table 2.

Table 2. Routing table

Source Node	Previous Two Hops	Next Two Hops
$ID_{source1}$	ID_{pre2}, ID_{pre1}	ID_{next1}, ID_{next2}
$ID_{source2}$	ID_{pre2}, ID_{pre1}	ID_{next1}, ID_{next2}
...	...	...
$ID_{sourceN}$	ID_{pre2}, ID_{pre1}	ID_{next1}, ID_{next2}

2.1.3. Secure Route Maintenance. If a sensor node has no route in its routing table when it starts to send the data, it initiates the route discovery. If the source node gets the error message after it sends data or routing packet, or it is out of the specified time, it triggers a new route discovery.

2.2. Secure Data Forwarding

The *SecRout* protocol is based on the two-level architecture. The cluster heads are at the higher level, and the other nodes are at the lower level. The cluster head aggregates the data from the nodes in its cluster. Then it sends it to the sink node. We use the cluster key to provide the secure communication in the cluster. The cluster key can be established during the self-organization [4]. We use the preloaded key to secure the message which is sent to the sink node.

2.2.1. Secure Data Forwarding in the Cluster. A sensor node sends the data packet to its cluster head using the cluster key, which is constructed as follows:

$$\{ID \mid E_{CK}(data) \mid MAC[CK, ID \mid E_{CK}(data)]\}$$

Where *ID* is the *ID* of the cluster head, and *CK* is the cluster key shared by the nodes within the cluster. The node with the same *ID* as the *ID* embedded in the data packet such as the cluster head verifies the *MAC*. If the verification succeeds, it decrypts the data using the cluster key. Then it aggregates the data from the sensor nodes in the cluster, and constructs the data packet which will be sent to the sink node.

2.2.2. Secure Data Forwarding among the Clusters. The cluster head becomes the source node after it aggregates the data. If there is a route in the routing table, it constructs the following data packet:

$$\begin{Bmatrix} ID_{this} \mid ID_{next} \mid ID_{source} \mid ID_{\sin k} \mid Q_{ID} \mid E_{K_{source}}(data) \mid \\ MAC[K_{source}, ID_{source} \mid ID_{\sin k} \mid Q_{ID} \mid E_{K_{source}}(data)] \end{Bmatrix}$$

Where Q_{ID} is the *Query ID* from the sink node, which is a random number. Then it broadcasts it.

The intermediate node with the same *ID* as the ID_{next} embedded in the packet will rebroadcast the updated packet, where the ID_{this} and ID_{next} are replaced by its *ID* and the *ID* of the next hop in the routing table. Other intermediate nodes drop the packet.

If the source node can not receive the packet again that the next hop rebroadcasts, it triggers a new route discovery. If the intermediate node cannot get the packet broadcasted by the next hop within a certain time, it reports the error message to the source node. Simultaneously it adds the next hop in its blacklist. It will not accept the *RREP* from the node in its blacklist.

After the sink node receives the packet, it verifies the *MAC* using the key of the source node from the {*ID, Key*) pair table. If the verification succeeds, it gets the result by decrypting the encrypted data.

2.2.3. Further Data Verification. After the sink node gets the data packet, two operations need to be executed: 1) the sink node compares the replies from other cluster heads which are residing the neighboring clusters of the source node; 2) the sink node will compare the replies with the history record.

If the data packet is abnormal, the sink node must verify it by sending the further request query to the nodes in the cluster as shown in Figure 1. They broadcast the result of the request query to the sink node, which is constructed as follows:

$$\begin{Bmatrix} ID_{source} \mid ID_{\sin k} \mid Q_{ID} \mid E_{K_{source}}(data) \mid \\ MAC[K_{source}, ID_{source} \mid ID_{\sin k} \mid Q_{ID} \mid E_{K_{source}}(data)] \end{Bmatrix}$$

The sink node continues the further analysis after it receives the further result from these nodes.

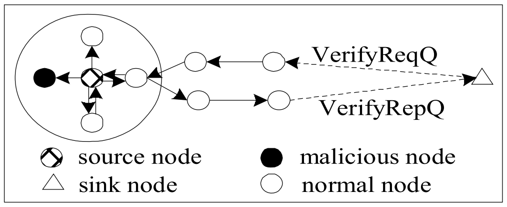

Figure 1. Reply verification

2.3. Secure Query Dissemination

The sink node wants to get the information from a particular area or from the whole sensor network. According to these two scenarios, the request query (*reqQ*) is respectively sent to a particular area or the whole sensor network.

2.3.1. Query a Particular Area. The sink node checks its routing table whether the route to the cluster head of that area exists. If yes, it sends the *reqQ* packet to the cluster head of that area, which is constructed as follows:

$$\begin{Bmatrix} ID_{this} \mid ID_{pre} \mid ID_{\sin k} \mid ID_{source} \mid Q_{ID} \mid E_{K_{source}}(reqQ) \mid \\ MAC[K_{source}, ID_{\sin k} \mid ID_{source} \mid Q_{ID} \mid E_{K_{source}}(reqQ)] \end{Bmatrix}$$

Otherwise, it broadcasts the *reqQ* packet to the cluster head, which is constructed as follows:

$$\begin{Bmatrix} ID_{\sin k} \mid ID_{source} \mid Q_{ID} \mid E_{K_{source}}(reqQ) \mid \\ MAC[K_{source}, ID_{\sin k} \mid ID_{source} \mid Q_{ID} \mid E_{K_{source}}(reqQ)] \end{Bmatrix}$$

When the cluster head receives the query request packet, it verifies the *MAC*. If the verification succeeds, it decrypts the *reqQ* using its secret key. Then it sends the *reqQ* to the nodes in that cluster using its cluster key, which is constructed as follows:

$$\{ID \mid E_{CK}(reqQ) \mid MAC[CK, ID \mid E_{GK}(reqQ)]\}$$

Where *ID* is the *ID* of the cluster head, and CK is the cluster key. The nodes in the cluster receive *reqQ* after they decrypt the packet using the cluster key.

2.3.2. Query the Whole Sensor Network. The sink node broadcasts the plaintext *reqQ*, which can be eavesdropped, modified or dropped by the malicious node. However, the *reqQ* can reach any sensor node even if a malicious node drops it because of broadcast. The sensor nodes get the query result according to the *reqQ*. The query result is the most important. In our approach, either the sink node gets the correct information or it detects that the tampered query reply. The query reply *ReplyQ* is constructed as follows:

$$\begin{Bmatrix} ID_{this} \mid ID_{next} \mid ID_{source} \mid ID_{\sin k} \mid Q_{ID} \mid reqQ \mid E_{K_{source}}(data) \mid \\ MAC[K_{source}, ID_{source} \mid ID_{\sin k} \mid Q_{ID} \mid reqQ \mid E_{K_{source}}(data)] \end{Bmatrix}$$

Then the sensor node applies the same technique as data forwarding process. But unlike data forwarding, the query reply includes the *reqQ*, which is also authenticated using *MAC*. If a malicious node modifies the *reqQ*, the sink node can verify it using *MAC*.

3. Security Analysis

We divide the attacks into two categories according to the compromised nodes. First, the source node is a normal node, but the intermediate node is a malicious node; the second, the source node is a malicious node. If the intermediate node is a malicious node, it can perform the following three actions: 1) Broadcast; 2) Drop; 3) Modify. (See Figure 2)

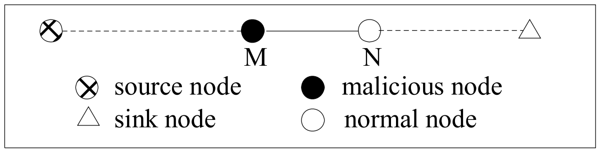

Figure 2. Sensor networks embedded malicious nodes

3.1. Intermediate Node Broadcasts Messages

In the *RREQ* process, the intermediate node broadcasts the updated *RREQ* and creates the routing table. The malicious node (Node M) may have three choices to attack this process: Case 1, it updates the *RREQ* by inserting the wrong *ID* of the current node; Case 2, it does not create the routing table, or it creates it with the wrong information; Case 3, it updates the *RREQ* by inserting the wrong *ID* of the previous node.

Case 1: The next hop (Node N) records the wrong *ID* of the previous node in its routing table. When the *RREP* reaches Node N, it is updated using the tampered *ID* of the previous node. Other nodes drop this *RREP* since their *ID* does not match the tampered *ID*. The node N detects that its previous hop is a malicious node since it cannot hear its rebroadcast. Then it refuses to broadcast other *RREQs* and the sink node selects other routes during next route discovery.

Case 2: When the *RREP* packet reaches Node M, it broadcasts it with a wrong *ID* of the previous node since it has incorrect information in its routing table. Node N can detect the tampered *RREP* broadcasted by Node M since its routing table stores two previous hops. It blacklists Node M.

Case 3: If Node M broadcasts the *RREQ* with the wrong ID of its previous node, the previous node can detect it through comparing its *ID* with the *ID* of the previous node embedded in the *RREQ*. It blacklists the malicious node M and informs the source node. The source node blocks the malicious node M. It triggers another route discovery with the *RREQ* as follows:

$$\begin{Bmatrix} ID_M \mid ID_{source} \mid ID_{\sin k} \mid ID_{RREQ} \mid N_{source} \mid \\ MAC(K_{source}, ID_M \mid ID_{source} \mid ID_{\sin k} \mid ID_{RREQ} \mid N_{source}) \end{Bmatrix}$$

If the malicious node M receives the *RREQ*, it broadcasts the updated *RREQ* as follows:

$$\begin{Bmatrix} ID_{this} \mid ID_{pre} \mid ID_M \mid ID_{source} \mid ID_{\sin k} \mid ID_{RREQ} \mid N_{source} \mid \\ MAC(K_{source}, ID_M \mid ID_{source} \mid ID_{\sin k} \mid ID_{RREQ} \mid N_{source}) \end{Bmatrix}$$

Its neighbors ignore it since the ID_M is the same as the ID_{this} embedded in the *RREQ*. If Node M tampers ID_{this} and rebroadcasts it, it's similar to Case 1 of the *RREQ* process. If Node M tampers ID_M, the sink node can detect it when it verifies the *MAC*.

In the *RREP* process, a malicious node (Node M) broadcasts a tampered *ID* of the previous node, the current node or the next node. This is the same as case 2 in the *RREQ* process. The next hop of Node M, Node *N*, can detect it.

In the data forwarding process, if the node along the route tampers ID_{next}, other nodes drop it since their *IDs* cannot match the ID_{next} embedded in the data forwarding packet. The sender can detect the tampered ID_{next} since every node records its next two hops. It blacklists the next hop as the malicious node and reports it to the source node.

3.2. Intermediate Node Drops Messages

In the route discovery, if a malicious node drops the *RREQ* packet, it blocks itself. The sink node can receive the *RREQ* packet through other routes. If the malicious node (Node M) drops the *RREP* packet, the next hop (Node N) can detect it since it cannot receive the packet again. In a data forwarding process, if a malicious node drops the data packet, the sender can detect it since it can not get acknowledgement from the next hop. It will inform it to the source node.

3.3. Intermediate Node Modifies Messages

In the route discovery, if a malicious node modifies the *RREQ* core content, such as ID_{source}, ID_{sink}, ID_{RREQ}, or N_{source}, the sink node can detect it through verifying the *MAC*. It drops the corrupted *RREQ* packet, and receives it from other routes. If a malicious node modifies the *RREP* core content, such as ID_{source} , $ID_{\sin k}$ or ID_{RREQ} , the source node can detect it through the *MAC*. In the data forwarding, we use the same solution as the case of the modified *RREQ* core content through verifying the *MAC*. In the query dissemination in the whole sensor network, the intermediate nodes can be compromised as Figure 3.

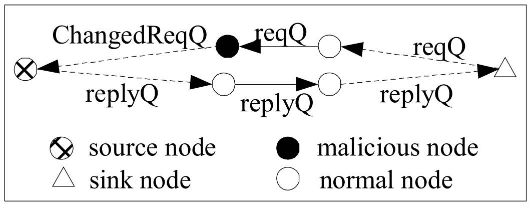

Figure 3. Query dissemination

The intermediate malicious node modifies the request query (*reqQ*) and broadcasts the changed request query (*ChangedReqQ*). The other sensor node receives the *ChangedReqQ*, and gets the query result which is packed up in the reply Query (*replyQ*). The query result contains the wrong information since the *reqQ* is tampered. The sink node receives the *replyQ*, and gets the *reqQ* from the *replyQ* packet. Here, the *reqQ* actually is the *ChangedReqQ*. The sink node also can get the correct *reqQ* according to the Q_{ID}. Then, it compares the correct *reqQ* with *ChangedReqQ*. If they match, the sink node can trust the query result from the reply packet. Otherwise, it drops it.

3.4. Source Node is a Malicious Node

If the source node is a malicious node, it tries to send abnormal messages to the sink node. The sink node can detect this after it compares the data with the record in the history and the neighboring node's report. Then it sends the further request to the nodes in the cluster, and waits for the replies from these nodes for the further analysis.

4. Performance Evaluation

We use the *NS2* [14] to evaluate the performance of the *SecRout*. In our simulation study, we use one source-destination pair. The source node sends a Constant Bit Rate (*CBR*) flow of 50 data packets per second. Each data packet is 50 bytes in size. We measure the performance using the following metrics [9]:

- *Packet Delivery Ratio*: The total number of packets received is divided by the total number of application–level packets originated.
- *Byte Overhead*: The total number of overhead bytes transmitted.
- *Packet Latency*: The elapsed time between the application layer passing a packet to the routing layer and that packet first being received at the destination.

The packet delivery ratios for *SecRout* and *AODV* are shown in Figure 4. From Figure 4, we observe that at the beginning the packet delivery ratio for *SecRout* is lower than that for *AODV*. This is because *AODV* builds the route faster than *SecRout*. But after *SecRout* creates the route to the sink node, the packet delivery rate is very close to *AODV*.

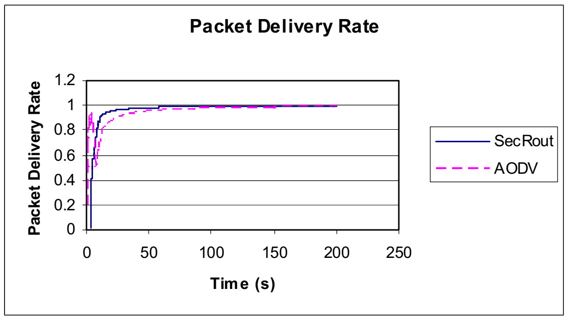

Figure 4. Packet delivery rate

Byte overhead for *SecRout* and *AODV* is shown in Figure 5. From Figure 5, we observe that at the beginning *SecRout* has less byte overhead than *AODV*. This is because *AODV* build the route faster, and then sends data packets faster. Since data packet has bigger size than the routing packet, *SecRout* has less byte overhead when it is still doing the route discovery while *AODV* has already sent the data packets. But on the average, *SecRout* is a little more byte overhead than *AODV* (less than 6%) since *SecRout* needs the security parameters, which are embedded in the data packets as well as in the route discovery packets.

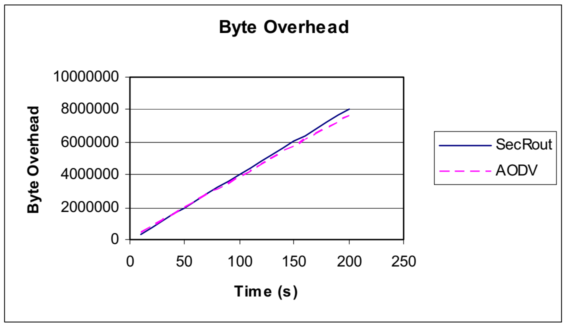

Figure 5. Byte overhead

Packet latency for *SecRout* and *AODV* is shown in Figure 6. From Figure 6, at the beginning *SecRout* has higher packet latency than *AODV* since *AODV* can build the route faster and therefore data can be sent faster. However, after *SecRout* builds the route, *SecRout* performs better.

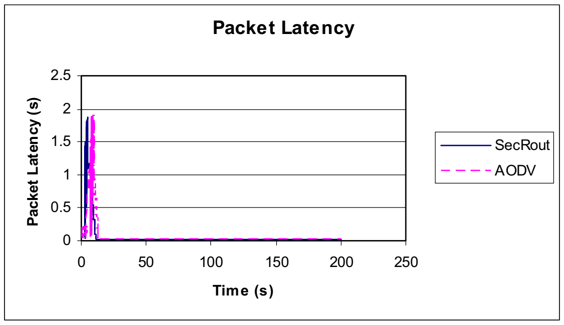

Figure 6. Packet latency

5. Related work

Perrig et al. [2] presented two security protocols, *SNEP* and *µTESLA*, for sensor networks using symmetric schemes. This paper only considers peer to peer architecture, has not considered the sensor network as a hierarchical structure. Failure of the nodes has not been considered in this paper. Yi et al. [15] proposed a security-aware routing protocol (*SAR*) for wireless ad hoc networks. In this protocol, every node is set by one security level. The routing packet with the security parameters will make routing decisions according to the security parameters and the security level of the node. Papadimitratos and Haas [11] proposed a secure routing protocol (*SRP*) in *MANET*. The route request packet is composed of *SRP* header in addition to the basic routing protocol packet and *IP* header. This routing discovery protocol can provide the correct connectivity information despite some compromised nodes in ad hoc networks. Hu et al. [9] proposed a secure efficient ad hoc distance vector routing protocol (*SEAD*) based on the design of the destination-Sequenced Distance-Vector routing protocol (*DSDV*). In this protocol, efficient one-way hash functions are used.

6. Conclusions

In this paper, we proposed the secure routing protocol for sensor networks (*SecRout*). The *SecRout* can guarantee that the sink node gets the correct query result from the sensor network. In the *SecRout* protocol, only high efficient symmetric cryptography is used. The two-level architecture is used, which dramatically lowers message overhead. Within the cluster, we secure messages using the cluster key. Among clusters, we encrypt the message using the shared key. In this paper, we also gave the security analysis for our protocol. We analyzed the different kinds of attacks, which may present in sensor networks. Our protocol is robust in presence of these attacks.

7. References

[1] C.E. Perkins and E.M. Royer, "Ad hoc On-Demand Distance Vector Routing", *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, LA, Feb 1999, pp. 90-100.

[2] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J.D. Tygar, "SPINS: Security Protocols For Sensor Networks", In *Proceedings of Mobicom*, 2001

[3] C. Karlof and D. Wagner, "Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures", First *IEEE International Workshop on Sensor Network Protocols and Application*, May 2003.

[4] S. Zhu, S. Setia and S. Jajodia, "LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks", In *Proceedings of 10th ACM Conference on Computer and Communications Security (CCS '03)*, Washington D.C., October, 2003.

[5] P. Ning and K. Sun, "How to Misuse AODV: A Case Study of Insider Attacks against Mobile Ad-hoc Routing Protocols", *Proceedings of the 2003 IEEE Workshop on Information Assurance United States Military Academy*, West Point, NY June 2003.

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

[7] M.G. Zapata, "Secure Ad hoc On-demand Distance Vector Routing", *ACM SIGMOBILE Mobile Computing and Communications Review*, Volume 6, Issue 3, July 2002.

[8] M. Tubaishat, J. Yin, B. Panja, and S. Madria, "A Secure Hierarchical Model for Sensor Network", *ACM SIGMOD Record*, Vol. 33, No. 1, March, 2004.

[9] Yih-Chun Hu, David B. Johnson, and Adrian Perrig, "SEAD: Secure Efficient Distance Vector Routing for Mobile Wireless Ad Hoc Networks", *Fourth IEEE Workshop on Mobile Computing Systems and Applications*, WMCSA'02

[10] Budhaditya Deb, Sudeept Bhatnagar and Badri Nath, "A Topology Discovery Algorithm for Sensor Networks with Applications to Network Management", In *IEEE CAS workshop*, September 2002

[11] P. Papadimitratos and Z.J. Haas, "Secure Routing for Mobile Ad Hoc Networks", *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, San Antonio, TX, January 27-31, 2002

[12] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", *RFC 2104*, February 1997

[13] R. L. Rivest, "The RC5 Encryption Algorithm", *Proc. 1st Workshop on Fast Software Encryption*, pages 86–96, 1995

[14] NS2 web site, http://www.isi.edu/nsnam/ns

[15] Seung Yi, Prasad Naldurg, And Robin Kravets, "A Security-Aware Routing Protocol for Wireless Ad Hoc Networks", Poster presentation, *ACM Symposium on Mobile Ad Hoc Networking & Computing (Mobihoc 2001)*, Long Beach, California, October, 2001