CHAPTER 5

SECURE KEY-TREE ARCHITECTURE FOR DOUBLE CLUSTER BASED ROUTING IN WSN

5.1 OVERVIEW

In this chapter, secure key-tree architecture for cluster based routing is proposed in WSN. In this technique, two cluster heads namely main and sub-ordinate cluster heads are selected based on the parameters such as residual energy, minimum average distance from the member, nodes timer and node degree using particle swarm optimization technique. When a source node wants to transmit data, it is performed using a secured communication methodology. It involves inter and intra-cluster key generation. Then all the data are gathered in clusters and multicast communication between base stations and cluster heads by hop by hop encryption.

5.2 HASH AND ENCRYPTION MECHANISM

5.2.1 One-Way Hash Function

A one-way hash function is also called a message digest, fingerprint or compression function. One-way hash function is a mathematical function that takes a variable-length input string and converts it into a fixed-length binary sequence. In addition, a one-way hash function is designed in such a way that it is hard to reverse the process i.e. to find a string that hashes to a given value. A good hash function also makes it hard to find two strings that would produce the
same hash value. Almost all the modern hash algorithms produce hash values of 128 bits and higher.

Even a slight change in an input string should cause the hash value to change drastically. Even if 1 bit is flipped in the input string, at least half of the bits in the hash value will flip as a result. This is called an avalanche effect.

As it is computationally infeasible to produce a document that would hash to a given value or find two documents that hash to the same value, a document's hash can serve as a cryptographic equivalent of the document. This makes a one-way hash function a central notion in public-key cryptography. When producing a digital signature for a document, we no longer need to encrypt the entire document with a sender's private key. It is sufficient to encrypt the document's hash value instead. One-way hash function is used mostly for generating digital signatures. It can also be used in case of secure password storage, file identification and MAC.

5.2.2 SHA-1

In cryptography, Secure Hash Algorithm, SHA-1 is a cryptographic hash function that produces a 160-bit (20-byte) hash value. SHA-1 hash value is typically rendered as a hexadecimal number, 40 digits long. The four SHA algorithms that are structured differently are SHA-0, SHA-1, SHA-2, and SHA-3. SHA-1 is the most widely used of the existing SHA hash functions, and is employed in
several widely used applications and protocols. SHA-1 uses a 512 bit block size and has a maximum message size of $2^{64}-1$ bits.

The algorithm for SHA-1 is briefly discussed below:

**Step 1:** The message is padding with a single one followed by zeros until the final block has 448 bits.

**Step 2:** The size of the original message is appended as an unsigned 64 bit integer.

**Step 3:** The 5 hash blocks ($h_0$, $h_1$, $h_2$, $h_3$, and $h_4$) to the specific constants are defined in SHA-1 standard.

**Step 4:** For each 512-bit block, hashing is performed. Then, an 80 word array is allocated for the message schedule. The first 16 words are set to be 512-bit block split into 16 words.

**Step 5:** The rest of the words are generated using the following algorithm. $\text{word}[i-3] \oplus \text{word}[i-8] \oplus \text{word}[i-14] \oplus \text{word}[i-16]$ then rotated 1 bit to the left.

**Step 6:** Loop 80 times. SHAfunction() and the constant K are calculated based on the current round number.

\[
\begin{align*}
  e &= d, \\
  d &= c, \\
  c &= b \text{ (rotated left 30)}, \\
  b &= a, \\
  a &= a \text{ (rotated left 5)} + \text{SHAfunction()} + e + k + \text{word}[i]
\end{align*}
\]

$a$, $b$, $c$, $d$ and $e$ are added to the hash output.
Step 7: The concatenation \((h_0, h_1, h_2, h_3, h_4)\) is obtained as output, which is the message digest.

![SHA-1 Algorithm Diagram](image)

**Figure 5.1 SHA-1 Algorithm**

In Figure 5.1, A, B, C, D and E are 32-bit words of the state, F is a nonlinear function that varies, \(\lll_n\) denotes a left bit rotation by \(n\) places, \(n\) varies for each operation, \(W_t\) is the expanded message word of round \(t\) and \(K_t\) is the round constant of round \(t\).

**5.2.3 RC-6**

RC6 refers to the Rivest Cipher 6. In cryptography, RC6 is a symmetric key block cipher derived from RC5. RC6 can satisfy the requirements of the Advanced Encryption Standard (AES) competition. RC6 proper has a block size of 128 bits and supports key sizes of 128,
192, and 256 bits, but, like RC5, it may be parameterized to support a wide variety of word-lengths, key sizes, and number of rounds. RC6 is very similar to RC5 in structure, using data-dependent rotations, modular addition, and XOR operations; in fact, RC6 could be viewed as interweaving two parallel RC5 encryption processes; however, RC6 does use an extra multiplication operation not present in RC5 in order to make the rotation dependent on every bit in a word, and not just the least significant few bits.

The pseudo code of the encryption algorithm of RC6 is

**Input:**

Plain text stored in four w-bit input registers, A, B, C, D

Number of r rounds

w-bits round keys S[0, ...., 2r + 3]

**Output:**

Cipher text stored in A, B, C, D.

Procedure:

```plaintext
B = B + S[0];
D = D + S[1];
For (i=1; i<r; i++)
{
    T = (B * (2B + 1)) <<< log w;
    u = (D * (2D + 1)) <<< log w;
    A = ((A ⊕ t) <<< u) + S[2i];
    C = ((C ⊕ u) <<< t) + S[2i+1];
    (A, B, C, D) = (B, C, D, A);
}
```
The pseudo code of the encryption algorithm of RC6 is

**Input:**

Cipher text stored in four w-bit input registers, A, B, C, D

Number of r rounds

w-bits round keys S[0, ..., 2r + 3]

**Output:**

Plain text stored in A, B, C, D.

**Procedure:**

```plaintext
C = C + S[2r+3];
A = A + S[2r+2];
for (i=r; i>=1; i--)
{
    (A, B, C, D) = (D, A, B, C);
    u = (D * (2D + 1)) <<< log w;
    t = (B * (2B + 1)) <<< log w;
    C = ((C - S[2i+1]) >>> t) ⊕ u;
    A = ((A - S[2i]) >>> u) ⊕ t;
}
D = D - S[1];
B = B - S[0];
```

The encryption and decryption of RC6 makes cipher text and plain text after carrying out twenty rounds repeatedly with cipher text and plain text in the four storages (A, B, C, and D) per 32bit word.
After doing four words round function, it operates left / right rotate per word with parallel operation as shown in the above pseudo code. Before and after executing the round functions, it executes round key and add / subtract operations.

The major security in the round functions is kept by data dependent rotate operation, and the amount of this rotate operation is produced by the fixed 5bit left rotate operation of the quadric, \( f(x) = x(2x+1) \). In decryption operation, the round key of encryption is used with the inverse order. Thus, RC6 has a Feistel structure, but the operation between encryption and decryption is different. RC6 does not have a non-linear transformation s-box [78].

5.3 SECURE COMMUNICATION

After performing clustering, the following two keys are defined to encrypt messages during cluster communication:

[1] Inter-cluster keys – for communicating among CHs
[2] Intra-cluster keys – for communicating among cluster members \((CM_i)\)

5.3.1 Inter-cluster Key Generation

The steps involved are as follows:

[1] BS broadcasts the initial message \((K_{mes})\) to CHs.

\[ BS \xrightarrow{K_{mes}} CH_i \]

\( K_{mes} : \{KM_i \mid \mid SN_i \mid \mid H_L \} \)

where \(KM_i = \) initial message
SN<sub>i</sub> = sequence number estimated using a one way hash function with SHA-1 algorithm.

H<sub>L</sub><sub>i</sub> = last hop to store the route information.

If H(SN<sub>i</sub>) = SN<sub>i-1</sub>,

Then

K<sub>mes</sub> is authorized

Else

CH<sub>i</sub> drops K<sub>mes</sub>

End if

The CHs updates the identity in the H<sub>L</sub><sub>i</sub>, which helps in routing the information from CH to BS.

[2] After receiving K<sub>mes</sub>, CH sends back a reply message R<sub>mes</sub> to BS join the multicast tree.

\[ \text{BS} \leftarrow \text{R<sub>mes</sub>} \rightarrow \text{CH}<sub>i</sub> \]

R<sub>mes</sub> : \{ID(CH<sub>i</sub>)||E(K<sub>ri</sub>, K<sub>pri</sub>(CH<sub>i</sub>))||MAC(Re||ID(CH<sub>i</sub>)||E(K<sub>ri</sub>, K<sub>pri</sub>(CH<sub>i</sub>))\%

where Re = reply message with a random key K<sub>ri</sub> encrypted with private key K<sub>pri</sub>.

\[ \text{MAC(Re||ID(CH<sub>i</sub>)||E(K<sub>ri</sub>, K<sub>pri</sub>(CH<sub>i</sub>)))} = \]

message authentication code encrypted with K<sub>pri</sub>

[3] BS after receiving R<sub>mes</sub> from CH validates the message and generates an authorization message (Auth<sub>mess</sub>) to every CHs.
[4] BS decrypts $K_{ri}$ from $R_{mes}$ and adds it to $Auth_{mes}$ and unicasts the message to all CHs in sequence.

$$BS \xrightarrow{Auth_{mes}} CH_i$$

where $Auth_{mes}$: \{ID(CH_i)||E(K_{ri}, K_{pri}(CH_i))\}

MAC(Re ||ID(CH_i) ||E(K_{ri}, K_{pri}(CH_i)))

[5] CH upon receiving the $Auth_{mes}$ verifies and decrypts it and generates level key ($K_{Li}$) for its child cluster heads. Then it forwards the cluster head authorization message (CH AUTH_mes) to child cluster heads.

$$CH_i \xrightarrow{CH(AUTH_{mes})} child \ CH_{i, i}$$

$CH (AUTH_{mes})$: \{ID (CH_i) || E(K_{Li}, K_{ri})\}
Figure 5.2 Cluster Tree Construction

Figure 5.2 illustrates that BS shares the first level key $K_{L1}$ to child CH$_4$ and CH$_{18}$. CH$_4$ shares the next level key $K_{L2}$ with CH$_{17}$ and CH$_{18}$ shares the next level key $K_{L3}$ with CH$_{19}$.

5.3.2 Intra-cluster Key Generation

The steps involved in the intra-cluster key generation are given below:

[1] CH broadcasts routing message (Route_req) to its CM$_i$. 

110
[2] Each CM$_i$ broadcasts the Route_req message to its neighbors until the message reaches all the nodes within the cluster.

[3] Each CM$_i$ upon receiving Route_req message compares the hop count to CH$_i$ and selects the CH with minimum hop as its parent node. If more than one CH$_i$ contains the similar hop count, then the CH with maximum residual energy is choosen.

[4] Then CM$_i$ sends a route reply message (Route_rep) to CH$_i$ in the reverse path traversed by the request message.

[5] CH$_i$ then constructs a route based key tree.

[6] CH then divides its members into branches. Each branch node is the root of branch tree. Following tree construction, the following keys are generated for secure communication within the cluster:

- Cluster key $K_c$ – to be shared with entire cluster
- Branch key $K_b$ – to be shared with branch nodes
- Fellow key $K_f$ – to be shared with nodes with same parent node.
- Secret key $K_s$ – generated when a new node joins the cluster

**Note:** If a node in the branch tree has no child nodes, $K_f$ will not be generated and the node contains only $K_c$, $K_b$ and $K_s$.

For the generation of the above keys, CH requests a source (z) from BS. Using the source, CH estimates the respective keys as follows:
\[ K_c = (z)^{RK_c} \]
\[ K_b = (z)^{RK_b} \]
\[ K_f = (z)^{RK_f} \]

where \( z \) = generating element of cyclic multiplicative group

\( RK_c, RK_b, RK_f = \) random numbers generated by CH

[7] CH unicasts the \( K_c, K_b \) and \( K_f \) with respective \( K_s \) for each node. Then it removes \( z \) and all \( K_b \) and \( K_f \).

Note: CH is cracked, the attacker will be unable to retrieve all keys.

[8] If the cluster wants to update keys, then CH fetches the old \( z \) and sends a request to BS for new \( z \).

[9] Using new \( z \), CH can perform previous keys estimation, new key generation and respective key updation.

[10] CH then removes all source entries (\( z \)), keys and old random number.

Note: This mitigates the effects of compromised node.

Consider Figure 5.1. All the nodes share the cluster key \( K_{ci} \). All the leaf nodes \((1, 2, 3, 6, 8, 9, 10, 11, 13, 14, 15, 20)\) share the \( K_{Bi} \). Nodes with same parent share \( K_{Fi} \). The nodes with no child nodes contain \( K_{bi}, K_s \) and \( K_c \).
5.3.3 Multicasting Data

In order to multicast data between CHs and BS, we utilize the hop-by-hop encryption methodology.

It involves the following steps:

i. Initially, BS encrypts the message using $K_{Li}$ and broadcasts it to CH$_i$.

ii. CH$_i$ then decrypts the message and re-encrypts it with the level key shared with child CH$_i$.

iii. CH$_i$ broadcasts the re-encrypted message to its child CH$_i$.

iv. The above hop by hop encryption is repeated until all CHs receive the message.

v. CH encrypts the message with $K_c$ and broadcasts it to all the cluster members. The message without $K_c$ cannot be decrypted. Thus, message transmission is secured.

The overall flowchart of the proposed work is given in fig. 5.3.
Perform clustering

If it is inter cluster key generation

Yes

CH joins the multicast tree by sending $K_{mes}$

No

BS unicasts Auth_mes message with $K_{ri}$ to CH

CH sends CH_Auth_mes to child CH

CH sends CH_Auth_mes with level key $K_{Li}$ to child CH

CH construct route based key tree

Generate $K_c, K_b, K_f$ and $K_s$ keys

For each node, CH unicasts $K_c, K_b$ and $K_f$ with respective $K_s$

CH request BS for new z to update keys

Multicast data using hop-by-hop encryption

End

Fig 5.3 Overall Flow Diagram
5.4 SIMULATION RESULTS

5.4.1 Simulation Model and Parameters

The Network Simulator (NS2) is used to simulate the proposed architecture. In the simulation, 50 mobile nodes move in a 1000 meter x 1000 meter region for 50 seconds of simulation time. All nodes have the same transmission range of 250 meters. The simulated traffic is Constant Bit Rate (CBR). In our simulation, 4 source nodes send their sensor data to the sink. There are two jamming attack nodes along the same channel.

The simulation settings and parameters are summarized in Table 5.1.

<table>
<thead>
<tr>
<th>No. of Nodes</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area Size</td>
<td>1000 X 1000</td>
</tr>
<tr>
<td>Mac</td>
<td>IEEE 802.11</td>
</tr>
<tr>
<td>Transmission Range</td>
<td>250m</td>
</tr>
<tr>
<td>Simulation Time</td>
<td>50 sec</td>
</tr>
<tr>
<td>Traffic Source</td>
<td>CBR</td>
</tr>
<tr>
<td>Packet Size</td>
<td>512</td>
</tr>
<tr>
<td>Rate</td>
<td>100,200,300,400 and 500kb</td>
</tr>
<tr>
<td>Initial Energy</td>
<td>20.1J</td>
</tr>
<tr>
<td>Transmission Power</td>
<td>0.660</td>
</tr>
<tr>
<td>Receiving Power</td>
<td>0.395</td>
</tr>
<tr>
<td>Antenna</td>
<td>Omni-Antenna</td>
</tr>
</tbody>
</table>

Table 5.1 Simulation Parameters
5.4.2 Performance Metrics

The proposed Secure Key-Tree Architecture for Double Cluster Based Routing (SKADC) is compared with the Double Cluster-Heads algorithm (DCA). The performance is evaluated mainly according to the following metrics.

- **Packet Delivery Ratio**: It is the ratio between the number of packets received and the number of packets sent.

- **Packet Drop**: It refers to the average number of packets dropped during the transmission.

- **Delay**: It is the time taken by the nodes to transmit the data packets to the receiver.

- **Energy Consumption**: It is the amount of energy consumed by the nodes to transmit the data packets to the receiver.

5.4.3 Results

In the experiment, the transmission rate is varied as 100, 200, 300, 400 and 500Kb. The results of SKADC and DCA are given in table 5.2.

<table>
<thead>
<tr>
<th>Rate</th>
<th>Delay</th>
<th>Delivery Ratio</th>
<th>Drop</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SKADC</td>
<td>DCA</td>
<td>SKADC</td>
<td>DCA</td>
</tr>
<tr>
<td>100</td>
<td>15.305</td>
<td>18.264</td>
<td>0.31878</td>
<td>11.8544</td>
</tr>
<tr>
<td>200</td>
<td>16.756</td>
<td>20.191</td>
<td>0.132064</td>
<td>11.29157</td>
</tr>
<tr>
<td>300</td>
<td>18.307</td>
<td>21.040</td>
<td>0.090232</td>
<td>11.55586</td>
</tr>
<tr>
<td>400</td>
<td>19.963</td>
<td>20.718</td>
<td>0.073599</td>
<td>11.21994</td>
</tr>
<tr>
<td>500</td>
<td>20.435</td>
<td>22.263</td>
<td>0.063443</td>
<td>11.34409</td>
</tr>
</tbody>
</table>

*Table 5.2: Results for Varying Rate*
**Fig 5.4: Rate Vs Delay**

**Fig 5.5: Rate Vs Delivery Ratio**
Figure 5.4 shows the end-to-end delay of SKADC and DCA techniques for different transmission rate scenario. It is concluded that the end-to-end delay of our proposed SKADC approach has 11% of less than DCA approach.

Figure 5.5 shows the delivery ratio of SKADC and DCA techniques for different transmission rate scenario. It is concluded
that the delivery ratio of our proposed SKADC approach has 24% of higher than DCA approach.

Figure 5.6 shows the packet drop of SKADC and DCA techniques for different transmission rate scenario. It is concluded that the packet drop of our proposed SKADC approach has 35% of less than DCA approach.

Figure 5.7 shows the energy consumption of SKADC and DCA techniques for different transmission rate scenario. It is concluded that the energy consumption of our proposed SKADC approach has 14% of less than DCA approach.

The percentage improvement of SKADC over DCA is given in table 5.3.

<table>
<thead>
<tr>
<th>Rate</th>
<th>Delay (%)</th>
<th>Delivery Ratio (%)</th>
<th>Drop (%)</th>
<th>Energy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>16.19638</td>
<td>42.14097</td>
<td>50.9806</td>
<td>12.65997</td>
</tr>
<tr>
<td>200</td>
<td>17.01464</td>
<td>13.01793</td>
<td>34.97789</td>
<td>14.79208</td>
</tr>
<tr>
<td>300</td>
<td>12.98979</td>
<td>19.0919</td>
<td>33.26144</td>
<td>14.83768</td>
</tr>
<tr>
<td>400</td>
<td>3.644641</td>
<td>23.60222</td>
<td>29.68151</td>
<td>15.72753</td>
</tr>
<tr>
<td>500</td>
<td>8.20992</td>
<td>23.40999</td>
<td>26.16282</td>
<td>14.22607</td>
</tr>
</tbody>
</table>

**Table 5.3: Percentage Improvement of SKADC**

**5.5 CONCLUSION**

In this chapter, secure key-tree architecture for cluster based routing is proposed in WSN. In this technique, two cluster heads namely main and sub-ordinate cluster heads are selected based on the
parameters such as residual energy, minimum average distance from the member, nodes timer and node degree using particle swarm optimization technique. When a source node wants to transmit data, it is performed using a secured communication methodology. It involves inter and intra-cluster key generation. The data gathered in clusters are multicast between base stations and cluster heads by hop by hop encryption technique. By simulation results, it is shown that the proposed technique reduces latency and increases data protection.