CHAPTER 6

SECURE CONTIKIRPL

6.1 INTRODUCTION

This chapter discusses the performance and evaluation of the proposed Secure ContikiRPL architecture and protocol. In chapter 4, attempts have been made to address the IPv6 implementation for LLN’s and the performance study of Secure Routing Protocol with TinyOS operating system. Difficulties to build a Secure TinyRPL protocol with both Confidentiality and Authentication established in the routing layer of the sensor motes has been found, due to the memory constraint of the devices. Also the programming model with TinyOS resembles the hardware organization following a hierarchical and component based design. The component based design also increases the steepness of the learning curve to interface the components and modules in all layers. The necessity of implementing the Secure Routing Protocols in Contiki OS with COOJA simulator is discussed clearly in the summary of chapter 4.

During these years, the academicians and Industrialists have diverted their concentration towards simple light weight mechanism and abstractions that can provide rich execution experience adhering to the limitations of the resource constrained devices. The Operating System supporting these devices with a flexible architecture is Contiki OS. It also has a facility of dynamically loading and unloading of programs and services. It has an event driven kernel supporting a pre-emptive multithreading that can be linked on a per process basis. This facilitates the utilization of multithreading as a library for programs linked at application layer on top of event driven kernel avoiding overhead due to multiple stacks and re-entrancy of all
parts of the system. Hence provides low memory usage and context switching by stack rewinding.

6.2 PROPOSED ARCHITECTURE

The existing RPL does not have any security. The proposed method is to implement security in the network layer. The confidentiality and authentication is provided by using RC5 and Skipjack along with CBC-MAC. The routing security is provided by creating secure RPL messages. The location of secure routing in network layer as Secure ContikiRPL protocol is shown in the Figure 6.1.

![Figure 6.1 Position of secure RPL messages](image)

The Secure ContikiRPL protocol manages the formation of DODAG, maintains a set of preferred parents and their associated choice of
metrics for information, communicates based on choice of metrics for objective functions and validates the communication of secure control messages such as SDIS, SDIO, SDAO and SDAO-ACK satisfying the logical module of RPL specifications.

- **Application Layer:** The application layer can have protocols like HTTP, CoAP etc and has flexibility to adapt to any real time application.

- **Transport Layer:** It decides whether the communication is UDP or TCP. Contiki has the option to completely disable TCP for the purpose of saving memory. In this work UDP application is used.

- **Network Layer:** The Network layer consists of uIPv6, 6LoWPAN adaptation layer and Secure Contiki RPL for routing. This enables the IPv6 packets from the internet to communicate directly with the wireless devices. The proposed work involves implementing security for routing protocol (RPL) that can be used in 6LoWPAN. In Secure ContikiRPL, packet forwarding activity is the sole function of uIPv6 based on its forwarding table. The outgoing packets from uIPv6 reach the 6LoWPAN layer for header compression and fragmentation which in turn reaches the MAC layer.

- **MAC Layer:** It enables transmission of MAC frames using physical channel. CSMA is generally used in the MAC layer. Contiki uses ContikiMAC mechanism to save power. The MAC layer uses a default contikiMAC protocol for operating in power saving mode. The MAC layer is also responsible for placing the outgoing packets in a queue using CSMA/CA mechanism of ContikiMAC. Orderly transmission from the
queue is monitored by the radio duty cycling layer. If a collision occurs, retransmission of the packets is possible with a linear back off present in this layer. The MAC layer retransmits until an acknowledgement for packet reception is obtained from the receiver.

- Physical Layer: It is used to provide data transmission and as well as interface to physical layer management utility. Tmote sky uses 250kbps 2.4GHz IEEE 802.15.4 Chipcon Wireless Transceiver which enables it to interoperate with other IEEE 802.15.4 devices. The onboard antenna has range of 50m indoor and 125m outdoor. The radio used is CC2420.

The unsecured control messages easily disclose traffic information and enable an adversary node to join the network. In order to address this issue, the proposed work provides security to the control messages by the parameters that are defined in the security header. The adversary node on receiving the control messages will verify for compatibility of the security parameters with its parameters. If it’s compatible then the node joins the network. Otherwise it is banned from joining.

### 6.3 RPL PACKET STRUCTURE WITH AND WITHOUT SECURITY IMPLEMENTATION

The proposed work uses secure variants of RPL control messages. The confidentiality and authentication of the routing messages is provided by cryptographic algorithms that are found to be suitable for the resource constrained environment such as RC5 and skipjack algorithms and CBC-MAC algorithm as shown in Figure 6.2.
6.4 SECURITY HEADER

The security header fields give flexibility to implement any cryptographic algorithm through an 8-bit algorithm field in the security header section. The security level field LVL defines varying levels of data authenticity and optionally data confidentiality. The Key Identifier Mode (KIM) determines whether the key used for confidentiality and authentication is determined implicitly or explicitly. A sample security header format with values is shown in Figure 6.3. In the implementation RC5 is given value 2 and Skipjack is given value 3 as value 1 has been allocated to the AES algorithm as default option in the IETF draft. The sample value for KIM is 0, LVL is 1 which implies that the key is determined by the key index field and security level includes both encryption and authentication. The counter field T is set to 0. Thus the counter field can have any non-repeating random values. Since the KIM field is zero, only the key index field exists. The sample value is 3, but it can take any value based on the implementation. The sample format for the security header is shown for better understanding.
6.5 SECURING NODES IN THE SAME NETWORK FROM EACH OTHER

The work uses the key index field to announce to the receiving mote the key used by the sender mote. If the receiving mote supports the key index specified by the sender then it processes the message. A mote can support more than one key index. The root mote is designed to support the entire key index supported by its subordinate motes. The communication between the motes supporting different key indices can be made through the root node. The above feature facilitates the implementation of filters in the root mote there by creating a secure path for a particular purpose which the nodes with other key index cannot use without the knowledge of the root node.

The advantage of the using the key index is that the routes for a specific application can be secured because the mote can take as a parent only which supports its key index. Figure 6.4 clearly depicts the scenario in which there exists three networks with root mote acting as a router for connecting these networks.
6.6 IMPLEMENTATION OF SECURE ICMPv6 RPL CONTROL MESSAGES

The first step is to add a security header with fields described in Figure 4.3. The confidentiality and authentication is defined by the values in the security header. Authentication is mandatory for any security implementation and it is carried out using CBC MAC (Karlof). Confidentiality depends on the value of security field in the security header. The confidentiality is provided starting from the security header to the end of the packet whereas MAC is calculated for the entire packet and appended to the routing message.
The steps for constructing secure RPL packets are as shown in figure 6.5:

1. Attach the security header on top of the base object.
2. Encrypt only the base object. (using cryptographic algorithms)
3. Generate MAC for the entire packet.
4. Append the generated MAC at the end of the packet.

6.7 ENCRYPTION ALGORITHM

6.7.1 Block Cipher

Block ciphers are known to be most efficient in terms of energy consumption and latency when compared to other security algorithms. This is the reason why they are preferred for use in WSN environments. The block ciphers studied are RC5 and Skipjack. The choice of algorithm is due to the memory constraint nature of Tmote sky. Other standard algorithms like AES did not fit in to the available memory space. Also Karlof et al (2004)
recommended the use of RC5 and Skipjack as encryption algorithms for wireless sensor mote using TinyOS.

### 6.7.2 RC5

An advantage of RC5 is its flexibility. Unlike other encryption algorithms, RC5 has a variable block size (32, 64, or 128 bits), number of rounds (0...255), and key size (0...255 bits). The values of those parameters determine the level of security of the algorithm.

### 6.7.3 SKIPJACK

SKIPJACK is a block cipher developed by the U.S. National Security Agency (NSA). The algorithm is an unbalanced Feistel network with 32 rounds. Skipjack uses an 80-bit key for encryption and decryption and the size of the data blocks is 64 bits. Since its parameters such as Key size and Block size are fixed, SKIPJACK is not as flexible as AES or RC5.

### 6.7.4 Message Authentication Code

Without authentication, the receiver cannot know who the source of the message is or if the data has been manipulated. Attackers can take advantage of this weakness and flip bits in the message, which results in predictable changes to the plaintext. These changes allow the attackers to infer information about the original message. Authentication code is used to address this vulnerability.

### 6.7.5 CBC-MAC

Cipher block chaining MAC (CBC-MAC) is the customary way to construct a MAC from a block cipher. CBC-MAC carries the encryption as CBC, and takes the last result as a MAC. CBC and CBC-MAC must use
different keys for encryption and MAC generation, otherwise even if changes are made to the message the MAC tag could still not change. In addition, CBC-MAC is not secure for arbitrary-length messages. The draft (Winter et al 2010) recommends the use of CBC-MAC because the mote could not afford to have different algorithms for encryption and message authentication.

The advantage of using CBC-MAC is that it can use the existing cryptographic algorithm to generate message authentication codes, thus saving significant memory.

6.8 IMPLEMENTATION

6.8.1 Analysis of Secure ContikiRPL Variants

The analysis is done using Tmote sky in COOJA simulator. The analysis will discuss the following implementation based on the choice of algorithms:

1. Unsecured
2. RC5
   1. MAC
   2. ENC-MAC
3. SKIPJACK
   1. MAC
   2. ENC-MAC

The data for analysis is obtained from the mote outputs which are saved in a log file. Perl script is used to analyze the log file for calculating
power consumption and convergence time of the deployed network. The steps involved in the analysis and implementation of Secure ContikiRPL are illustrated in Figure 6.6.

![Diagram](image)

**Figure 6.6 Steps for analyzing log file**

### 6.8.2 Technologies used

The programming and scripting languages used are:

- C - language: It is a widely used computer programming language.

- Make files: They are text files following a particular syntax. A software can be built from scratch using Make Utility.

- PERL: It is a scripting language used to manipulate and process text files.
- Apache ant: It is a Java based build tool. It is written in XML and has the advantages of being portable, open standard and very simple to understand.

The work environment set up is as follows:

- Operating system: Ubunto Linux 14.04.
- System Processor: Intel core i5 @ 3.20GHz
- System RAM capacity: 8GB.

Contiki based optimized parameters obtained from Table 5.14 are furnished in Table 6.1 to establish Optimized and Secure ContikiRPL.

**Table 6.1 Contiki based parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Unsecured</th>
<th>RC5-MAC</th>
<th>RC5-ENC-MAC</th>
<th>SKIPJACK-MAC</th>
<th>SKIPJACK-ENC-MAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contiki Version</td>
<td>2.7</td>
<td>2.7</td>
<td>2.7</td>
<td>2.7</td>
<td>2.7</td>
</tr>
<tr>
<td>DIO Interval Minimum (Secs)</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>DIO Doubling rate (Seconds)</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>DIO Redundancy Rate (Secs)</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Algorithm</td>
<td>-</td>
<td>RC5 (2)</td>
<td>RC5 (2)</td>
<td>SKIPJACK (3)</td>
<td>SKIPJACK (3)</td>
</tr>
<tr>
<td>Security Level (LVL)</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Counter</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
COOJA based parameter are

- Radio medium: Unit Disk Graph Medium (UDGM): Distance Loss
- Mote startup delay (ms): 1000
- Number of Motes: 50

6.8.3 Contiki-OS Structure

The ContikiOS is written in basic C language. The Figure 6.7 shows different folders that come with Contiki 2.7. The apps folder consists of application that can be run on Contiki. The core consists of source code for the IP stack. The CPU folder has source codes for processors that can run Contiki. The platform consists of readily available motes that can be used to build Contiki. The Makefile.include compiles and builds the entire Contiki system.

![Figure 6.7 Contiki 2.7](image-url)
A security algorithm folder created for this project shown in Figure 6.8 portrays many algorithms of which only RC5 and Skipjack are supported.

![Security Algorithm Folder](image)

**Figure 6.8 Security Algorithm**

The RPL folder shown in Figure 6.9 contains the source code for RPL functions. The files modified are

- Makefile.rpl
- rpl-icmp6.c
- rpl-conf.h
- rpl-private.h

The new files created for secure RPL are

- rpl-security.c
- rpl-security.h
The rpl-security.c provides security header creation, encryption and MAC mechanisms. The rpl-conf.h file should have the definition in Figure 6.10 added to it. The definitions can be changed based on the security requirement.

/*
 * Default security configuration
 */

#define RPL_SECURITY_COUNTER_SET_INCREMENTING_COUNTER
#define RPL_ALGORITHM RC5
#define RPL_KIM KIM_MODE_0
#define RPL_LVL_LVL1
#define RPL_COUNTER_0
#define RPL_KEY_INDEX 0x00
#define RPL_KEY_SOURCE 0x00

When changes are made to the RPL_ALGORITHM field, relevant changes have to be made in the Makefile.rpl file shown in Figure 6.11. The '#' symbol denotes the statements that are commented. Depending on the RPL_ALGORITHM the Make file is changed.
After all the changes are made in RPL, a rpl-udp application is run using a server and many clients motes. The udp-server.c and udp-client.c can be used to run rpl-udp. After compilation a .sky file will be formed which is a complete build. A simulation environment called COOJA provided by Contiki makes things simple. The udp application can be compiled and build on simulation motes for testing purposes. The rpl-udp files are depicted in Figure 6.12.
In order to save memory, project-conf.h file should have the definition statements shown in Figure 6.13 which disables the TCP configuration (UIP_CONF_TCP) and limits the maximum number of neighbour (NBR_TABLE_CONF_MAX_NEIGHBOURS) and route (UIP_CONF_MAX苦苦

```c
/* Free some code and RAM space */
#define UIP_CONF_TCP 0
#undef NBR_TABLE_CONF_MAX_NEIGHBORS
#define NBR_TABLE_CONF_MAX_NEIGHBORS 8
#undef UIP_CONF_MAX/routes
#define UIP_CONF_MAX/routes 8
#undef PROCESS_CONF_NO_PROCESS_NAMES 1
#undef NETSTACK_CONF_RDC
#ifndef NETSTACK_CONF_RDC
#ifndef NETSTACK_CONF_RDC nullrdc_driver
```

**Figure 6.13 Project configuration file to save memory**

### 6.9 COOJA SIMULATOR

COOJA is a java based simulator. To start the simulator 'ant' command has to be run from the terminal by navigating to the Contiki (source path)/tools/COOJA. When new simulation is created the window shown in Figure 6.14 will be displayed. The radio medium can be used to change the medium as lossy or lossless. The default is Unit Disk Graph Medium (UDGM): Distance loss which denotes that the medium will have loss if the motes are far apart. There is also an option to include constant loss medium or multipath ray tracer medium. Changing the simulation parameters as per requirements and clicking the create button creates a simulation.
Creating new simulation in COOJA

When a new simulation is created the window shown in Figure 6.15 appears. It shows multiple windows such as

- **Network**: An area used to create and position motes. The view menu in this window has options that can be used for better understanding of the motes being simulated.

- **Simulation control**: It is used to start, stop, pause or reload the simulation when relevant changes are made in the source code. The speed of the simulation can also be controlled in this window.

- **Mote output**: The motes running in the Network window will send outputs which are generally print statements given in the source code. These outputs are captured in the Mote output window.

- **Timeline**: This window displays the active time of each mote in which it sends or receives data.
The motes can be added using the motes menu as shown in Figure 6.16. The motes used in the simulation are sky motes. There are also other motes that can be used. A COOJA mote is also available to test any application specific implementations.
When sky mote is selected the window similar to Figure 6.17 appears. The required firmware can be selected using browse button. Then the firmware has to be compiled and create button has to be selected.

![Figure 6.17 Choosing firmware for sky mote](image)

Then an ‘Add motes’ new window shown in Figure 6.18 appears. The number of motes and positions can be specified in this dialog box.

![Figure 6.18 Number and positioning of motes](image)
When a network is to be formed with 20 motes and they are added, the screen shot of the network window resembles the one as shown in Figure 6.19. When one mote is clicked a green circle is formed around it to indicate range of communication of that mote.

![Image of network window showing 20 motes](image1)

**Figure 6.19 Demo setup consisting of 20 motes**

Now start the simulation and it can be observed that the mote outputs are displayed in the relevant mote output window with the time and mote ID as shown in Figure 6.20. This can be saved and later processed for required analysis.

![Image of mote output after simulation started](image2)

**Figure 6.20 Mote output after simulation started**
6.10 IMPLEMENTATION OF COOJA SETUP

The Figure 6.21 shows the setup environment simulated for 50 motes. The mote output window displays the output of each mote along with its mote ID. The time is also displayed along with the corresponding output. This makes it very compatible for analysis purposes.

![COOJA Screenshot](image)

Figure 6.21 COOJA Screenshot

The mote output in the Figure 6.22 shows the successful encryption and decryption of RC5 implementation. The message is encrypted during transmission and MAC is added. At reception MAC has to be verified and then messages have to be decrypted. The mote output also has a filter facility to display only required outputs. The pattern matching is done using any word that may come in all the required outputs.

Otherwise the simulation script editor in COOJA can be used to display, manipulate and process the mote outputs.
6.11 RESULTS AND DISCUSSION

6.11.1 Impact on Convergence Time

The simulation is carried out with variable number of nodes (20, 50 and 100) and the convergence time is estimated. It can be seen from Figure 6.23 that the secured versions take more time for convergence compared to the unsecured version. The convergence time for RC5 is higher than that of SKIPJACK implementation. The convergence time vividly shows a foremost dependence on the distribution of motes. For a fixed Network setup, the convergence time is evaluated for various security implementations under consideration. Modification in setup ends up exhibiting slightly varying results towards the convergence of the network.
6.11.2 Impact on Memory

In any wireless sensor network memory plays a vital role. When the memory of the mote is increased, the power consumption also increases. All microcontrollers generally possess two types of memory, ROM and RAM. The ROM size is very limited and it usually has compiled codes stored in its memory. The RAM is a volatile memory and its size is even more limited than the ROM size.

The size can be found by using the size command followed by the file name. The file name usually ends with .sky (it varies depending on the platform). The resulting output of the size command obtained for comparing the memory occupation of the five different algorithms is shown in Figures 6.24, 6.25, 6.26, 6.27 and 6.28. The text column in the figures denote the ROM size; the bss column denotes the RAM size; the dec and hex column represents the total size in decimal and hexadecimal formats.

![Figure 6.23 Convergence Time](image-url)
### Unsecured

![Unsecured UDP Application](image1)

**Figure 6.24 Unsecured UDP Application**

### RC5-MAC

![RC5 Secure ContikiRPL with only MAC](image2)

**Figure 6.25 RC5 Secure ContikiRPL with only MAC**

### RC5-ENC-MAC

![RC5 Secure ContikiRPL with encryption and MAC](image3)

**Figure 6.26 RC5 Secure ContikiRPL with encryption and MAC**

### SKIPJACK-MAC

![SKIPJACK Secure ContikiRPL with only MAC](image4)

**Figure 6.27 SKIPJACK Secure ContikiRPL with only MAC**

### SKIPJACK-ENC-MAC

![SKIPJACK Secure ContikiRPL with encryption and MAC](image5)

**Figure 6.28 SKIPJACK Secure ContikiRPL with encryption and MAC**
In wireless networks, nodes may need to relay messages from others to reach their destination. With Contiki, even relay nodes, so-called routers, can be battery-operated because of ContikiMAC radio duty cycling mechanism which allows them to sleep between each relayed message. Contiki can be compiled even without Contiki-MAC. In order to illustrate the importance of the work, the graph describing the memory consumption with and without Contiki-MAC and RPL is shown in Figure 6.29. It illustrates the memory constrained nature of the work and justifies the reason for choosing RC5 and SKIPJACK algorithms for providing security.

![Comparison of Memory Occupation](image)

**Figure 6.29 Memory Occupation**

Implementing Contiki-MAC consumes a lot of memory, but reduces power consumption to approximately 1mW from around 60mW. The memory consumption with both RPL and Contiki-MAC consumes 41.06KB of ROM thus leaving only 7KB for security and application oriented implementations. Different algorithms can be implemented for application layer security when contikiMAC and RPL are not included. Since the current
work revolves around battery operated motes and securing RPL, both Contiki-MAC and RPL has to be implemented.

![Figure 6.30 Memory Occupation of Secure ContikiRPL](image)

**Figure 6.30 Memory Occupation of Secure ContikiRPL**

Based on the results from the size command the ROM and RAM consumption is charted in Figure 6.30. The percentage increase in memory size, while implementing RC5 with only CBC-MAC is 4.5% for ROM and 13% for RAM. When confidentiality is entrenched with MAC, there is an increase of 9% ROM and 15.82% RAM size. Similar calculation for SKIPJACK with only CBC-MAC yields 9.7% for ROM and 63% for RAM. With added confidentiality, there is an increase of 15% ROM and 63% RAM size. The increased memory consumption is calculated against the unsecured contikiRPL version. With respect to memory it can be clearly seen that RC5 is a better contender. Also SKIPJACK uses almost all the ROM size defined for Tmote sky, which abruptly shrinks the scope of implementation of any UDP socket application.
6.11.3 Impact on Power Consumption

Wireless Sensor Motes are mostly designed to be battery operated which limits their flexibility in power and reduces the life time of the network. COOJA uses an accurate power profiling tool, which is used to estimate the power consumption of motes in the simulated environment. The powertrace.app will provide the time spent by the mote in CPU processing, LPM, transmission and reception. With this time information the power is calculated. The comparison plot for power is depicted clearly in Figure 6.31. Naturally the secure implementations consume more power than the unsecured implementation. When comparing the two algorithms, SKIPJACK Secure ContikiRPL appears to be more power efficient than RC5 Secure ContikiRPL.

![Figure 6.31 Power Comparison](image-url)
The pattern in power consumption levels after one hour observation of network communication can be vividly seen from the graph. The total power consumption of each mote is categorized under four heads namely CPU, Low Power Mode (LPM), Transmission and reception power. It can be seen that a major variation in power results from the increased CPU and reception power. It is obvious from the fact that the confidentiality and authentication requires more CPU processing time.

![Graph showing power consumption analysis](image)

**Figure 6.32 Power Consumption analyzed for 1 hour**

It is evidently shown from Figure 6.32 that the power consumption is high before the convergence of the deployed network. High power consumption in the initial stage can be attributed to the mote startup and increased overhead to setup a network. After convergence, the power slowly stabilizes to values between 0.90 to 1.5 mV which creates a hope of building a secure ContikiRPL for eminent applications. The major difference is seen in reception power though encryption and authentication consumes some amount of extra CPU power.
There is not much change in the transmission and LPM power. When Contiki-MAC is not used the power consumption is close to 60mW. Thus some memory is sacrificed to drastically reduce the power consumed which is mandatory for a resource constrained environment.

6.12 SUMMARY

The task of implementing secure ContikiRPL ready for portability into any hardware motes has been made successful using COOJA simulator with two cryptographic primitives combined with message authentication. The performance of secure ContikiRPL in terms of convergence, memory and power consumption profiling has been estimated with RC5 and SKIPJACK along with CBC-MAC and is compared against the unsecured version of RPL. Based on the simulation results SKIPJACK shows better performance in terms of convergence time and power consumption. However SKIPJACK’S memory occupation reaches the brim of the ROM size which makes it unsuitable for any other implementation of real world applications. In spite of its convergence time and power consumption, RC5 occupies considerably less memory providing enough space confidentiality, authentication and further application oriented implementation. Hence it is concluded that RC5 with CBC-MAC could be the right choice for secure ContikiRPL implementation which can further be exploited for any real world application.