12-11-2006

Practical implementation and performance analysis on security of sensor networks

Nidhi Verma

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by the Thesis/Dissertation Collections at RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
Practical Implementation and Performance Analysis
On Security of Sensor Networks

By

Nidhi Verma

A Thesis submitted In Partial Fulfillment of the requirements for
Degree of Master of Science in Computer Engineering

Supervised by

Dr. Fei Hu
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, NY
November 2006

Approved by:

Dr Fei Hu
Primary Advisor – R.I.T. Dept. of Computer Engineering

Dr Marcin Lukowiak
Secondary Advisor – R.I.T. Dept. of Computer Engineering

Dr Pratapa Reddy
Secondary Advisor – R.I.T. Dept. of Computer Engineering
Thesis Release Permission Form

Rochester Institute of Technology

Kate Gleason College of Engineering

TITLE: Practical Implementation and Performance Analysis on Security of Sensor Networks

I, Nidhi Verma, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or part.

________________________________________
Nidhi Verma

________________________________________
Date
Dedication

This thesis is dedicated to my parents, Chandra Shekhar and Meera Verma, my sister Shilpi Verma for their never ending love, faith and motivation. I appreciate the time given by my fiancé Abhishek Sinha to proof read the first version of this text and providing motivational guidance throughout writing the thesis I would also take the opportunity to thank my in-laws for their constant support and encouragement. I would also like to thank my close friend Juil Cha for his faith in me and help during the research phase.

Their encouragement and love is greatly appreciated.
Acknowledgements

I would like to sincerely thank my Advisor Dr Fei Hu for his time, inspiration and help. Without his support I would not be able to finish my thesis. I would also like to thank my committee members Dr Pratapa Reddy and Dr Marcin Lukowiak for their constructive criticism and timely review which has resulted in tremendous improvements.
Abstract

A wireless sensor network (WSN) is a network made of thousands of sensing elements called as nodes with wireless capabilities. Their application is varied and diverse ranging from military to domestic and household. As the world of self-organizing sensor networks tip to the edge of maximum utilization, their wider deployment is adding pressure on the security front. Powerful laptops and workstations make it more challenging for small sensors. In addition, there are many security challenges in WSN, e.g.- confidentiality, authentication, freshness, integrity etc. Contributions of this work are as follows:

- **“Symmetric” security implementation**: This thesis work designs a symmetric-key based security in sensor hardware in the Link layer of sensor network protocols. Link Layer security can protect a wireless network by denying access to the network itself before a user is successfully authenticated. This prevents attacks against the network infrastructure and protects the network from devastating attacks.

- **“Public key” implementation in sensor hardware**: Asymmetric key techniques are attractive for authentication data or session keys. Traditional schemes like RSA require considerable amounts of resources which in the past has limited their use. This thesis has implemented Elliptic Curve Cryptography (ECC) in Mica2 hardware, which is an approach to public-key cryptography based on the mathematics of elliptic curves.
• **Quantitative overhead analysis:** This thesis work analyzes the wireless communication overhead (No. of packets transmitted) vs the (transmit and receive) energy consumed in mJoules and memory storage overhead (bytes) for ECC as compared to the symmetric counterpart for the implemented WSN security protocols.
# Table of Contents

Chapter 1: Introduction 13

Chapter 2: Background 17

2.1 Sensor Networks 17

2.2 Kind of Attacks 18

2.3 Security goals for Wireless Sensor Networks 21

2.3.1 Access Control 21

2.3.2 Message Authenticity and Integrity 22

2.3.3 Confidentiality 22

2.3.4 Explicit Omission 22

2.4 Platform Motes 23

2.5 Environment 24

Chapter 3: TinyOS 26

3.1 Does TinyOS have a ISO model 26

3.2 Architecture 27

3.3 Memory Model of TinyOS 28

3.4 Modularity of TinyOS 28

Chapter 4: TinySec 33

4.1 Why Link Layer Security is needed 33

4.2 TinySec Design 34

4.3 TinySec Packet Format 34

4.4 Using TinySec in TinyOS applications 36
4.5 Different modes of operation in TinySec

4.5.1 Enabling TinySec Mode

4.5.2 Key Management

4.5.3 Updating Keys

4.5.4 TinySec Files

Chapter 5: Encryption and Security Primitive

5.1 MAC

5.2 Initialization Vector

5.3 Encryption Scheme

5.4 Skipjack

5.4.1 Notation and Terminology

5.4.2 Basic Structure

5.4.3 Stepping Rules

5.4.4 Stepping Sequence

5.4.5 G Permutation

5.4.6 Crypto variable Schedule

5.4.7 F Table

Chapter 6: Elliptic Curve Cryptography

6.1 Introduction

6.2 Motivation for using Elliptic Curve Cryptography

6.3 Mathematics for RSA Vs ECC

6.4 Asymmetric Cryptography – fine balance

6.5 Why Asymmetric Cryptography
6.6 Authentication with Asymmetric Cryptography 56
6.7 How Elliptic Curve are used 57
6.8 Diffie Hellman/ DSA cryptosystems 58
6.9 Elliptic Curve Discrete Logarithm Problem 60
6.10 Elliptic Curve Groups 61
6.11 Implementation of ECC in NesC 62
6.12 Conclusion for ECC 70

Chapter 7: Overhead analysis 72
7.1 Memory Overhead 74
7.2 Comparison with RSA (asymmetric cryptography) 74
7.3 Comparison with Symmetric Counter Part 75
7.4 Comparison with One Time Sensors 75
7.5 Analysis of models 78

Chapter 8: Conclusion 82
Chapter 9: References 84
List of Figures

Figure 1 Sensor Networks and Adversaries in action 17
Figure 2 Block Diagram of Mica2 platform 22
Figure 3 Mica2 Sensor 23
Figure 4 Mica2 sensor board 23
Figure 5 Block Diagram of Mica2 dot 24
Figure 6 TinyOs version of ISO Model 26
Figure 7 TinyOs Layer Structure 27
Figure 8 Configuration Wiring 28
Figure 9 NesC component 29
Figure 10 Configuration 29
Figure 11 Configuration and Module 30
Figure 12 Blink Text 31
Figure 13 Blink Diagram 31
Figure 14 TinySec Data Packet 34
Figure 15 Cipher-Block Chaining Mode Diagram 43
Figure 16 SKIPJACK Stepping Rules 45
Figure 17 G-permutation diagram 47
Figure 18 Key Length Vs Increased Security Error! Bookmark not defined.
Figure 19 EccM Directory in Cygwin 59
Figure 20 Installing Ecc on mica2dot (id # 1) 61
Figure 21 Installing Ecc on mica2dot (id # 2) 62
Figure 22 Ecc result

Figure 23 Energy consumption for CBC and variants of DES
List of Tables

Table 1: NesC Jargon 31
Table 2: List of Libraries 32
Table 3: Encryptor and Decryptor 46
Table 4: Sample F Table 49
Table 5: Comparison chart for RSA and ECC 52
Table 6: Comparison chart for RSA and ECC (Overhead) 75
Table 7: Message Format for Energy Overhead 89
Table 8: TinySec Operating Modes for Transmitting 90
Table 9: TinySec Operating Modes for Receiving 90
Glossary

DSA: Digital Signature Algorithm
ECC: Elliptic Curve Cryptography
IV: Initialization Vector
MAC: Message Authentication code
WSN: Wireless Sensor Network
Chapter 1

Introduction

“It has become appallingly obvious that our technology has exceeded our humanity”

~Albert Einstein

Recently technology has started to play a very important role in our lives. There is an ocean of devices we use daily. Focusing on new applications enabled by wireless networks we can see considerable excitement on the broader deployment of its applications. The study of wireless sensor networks is challenging in that it requires an enormous breadth of knowledge from a wide variety of disciplines.

A wireless sensor network (WSN) is a network made of thousands of nano-computers with onboard sensor boards. They are called nodes. The nodes self-organize their networks, rather than having a pre-programmed network topology. One of the primary purposes of WSN is to gather information and send it to the base station for accumulation and/or analysis. However as is it obvious due to the nature of wireless network that the data is freely available in air. The loss of confidentiality and integrity and the threat of denial of service (DoS) attacks are risks typically associated with wireless communications. Unauthorized users may gain access to agency systems and information, corrupt the agency’s data, consume network bandwidth, degrade network performance, and launch attacks that prevent authorized users from accessing the network, or use agency resources to launch attacks on other networks.
There are many security challenges in WSN [12]. Some of them are as follows: Confidentiality, Authentication, Freshness, and Integrity etc. Because of the limitations due to battery life, nodes are built with power conservation in mind, and generally spend large amounts of time in a low-power "sleep" mode or processing the sensor data.

This work reveals the issues and challenging problems involved with the security dimension/parameter for the sensor nodes. It also advocates a robust solution approach-use of public key cryptography considering the small memory, energy overhead and strong security mesh provided. Overall, this thesis ponders on several on-demand sensor node issues and makes several important contributions to an ever-growing active research and branch of wireless sensor network security.
Chapter 2

Background

2.1 Sensor Networks:

The terminology sensor network is referred to a heterogeneous system combining tiny sensors and actuators with general-purpose computing elements [1]. The important characteristics are as follows:

1. This huge sensor sea consists of innumerable low-power and low-cost wireless nodes. Their primary job is to monitor and report the conditions in the deployed environment.

2. The sensor networks have limited space and hence these hefty cryptographic protocols cannot reside on them. On the contrary these limitations can work in our favor as the adversaries have to also match the capabilities of the sensors, for e.g. even they are limited by the bandwidth restrictions (number of packets per second), hence they can’t eavesdrop by bombarding it with fake messages.

3. Energy is the most important element of concern, it can be truly said that energy saved is energy earned. Every iota of energy used brings the wireless nodes closer to their end.

4. The sensor network communicates in a broadcast medium; hence this wireless media is very unsafe. Also the adversaries can use this to flood the network and drain the
sensors energy. It can be typified by the term pursuit evasion game [6] as there is a group of pursuers who attempt to capture the evaders.

A simple diagram will be as follows: [4]

![Diagram of Sensor Networks and Adversaries in action](image)

**Figure 1: Sensor Networks and Adversaries in action**

2.2 Kind of Attacks [6] [15] [30] [31]:

The various kinds of attacks are as follows:

- Jamming: interferences of radio frequencies
- Spoofed, altered, or replayed routing information (flooding: Bombardment of messages by the invaders)
- Drop, forge, modify, deletion of message packets
- Selective forwarding
- Sinkhole attacks
- Sybil attacks
- Wormholes
- HELLO flood attacks
- Acknowledgement spoofing.

Some of them we will see in detail as follows:

**2.2.1 Replay Attack:**

Under this attack the adversary targets the routing information exchanged between nodes. It alters/ replays several times to create a loop to repel the network traffic. It may also generate false error messages, partition the network, increase end-to-end latency, etc.

**2.2.2 Selective forwarding:**

In a selective forwarding attack, malicious nodes may refuse to forward certain messages and simply drop them, ensuring that they are not propagated any further. An adversary interested in suppressing or modifying packets originating from a few select nodes can reliably forward the remaining traffic and limit suspicion of their wrongdoing.

**2.2.3 Sink Hole Attack:**

Sinkhole attacks typically work by making a compromised node look especially attractive to surrounding nodes with respect to the routing algorithm. For instance, an adversary could spoof or replay an advertisement for an extremely high quality route to a base station. Some protocols might actually try to verify the quality of route with end-to-end acknowledgements containing reliability or latency information. In this case a
laptop class device can act as a very good sink hole. It should be noted that the reason sensor networks are particularly susceptible to sinkhole attacks is due to their specialized communication pattern.

2.2.4 Acknowledgement spoofing:

Several sensor network routing algorithms rely on implicit or explicit link layer acknowledgements. Due to the inherent broadcast medium, an adversary can spoof link layer acknowledgments for ‘‘overheard’’ packets addressed to neighboring nodes. Goals include convincing the sender that a weak link is strong or that a dead or disabled node is alive.

2.2.5 HELLO flood attack:

Many protocols require nodes to broadcast HELLO packets to announce themselves to their neighbors, and a node receiving such a packet may assume that it is within (normal) radio range of the sender. This assumption may be false: a laptop-class attacker broadcasting routing or other information with large enough transmission power could convince every node in the network that the adversary is its neighbor.

2.2.6 The Sybil attack:

In a Sybil attack, a single node presents multiple identities to other nodes in the network. The Sybil attack can significantly reduce the effectiveness of fault-tolerant schemes such as distributed storage, dispersity and multi path routing, and topology
maintenance. Replicas, storage partitions, or routes believed to be using disjoint nodes could in actuality be using a single adversary presenting multiple identities.

2.2.7 **Worm Hole Attack:**

An adversary situated close to a base station may be able to completely disrupt routing by creating a well-placed wormhole. An adversary could convince nodes who would normally be multiple hops from a base station that they are only one or two hops away via the wormhole. This can create a sinkhole: since the adversary on the other side of the wormhole can artificially provide a high quality route to the base station, potentially all traffic in the surrounding area will be drawn through it if alternate routes are significantly less attractive. This will most likely always be the case when the endpoint of the wormhole is relatively far from a base station.

2.3 **Security goals for Wireless Sensor Networks:**

There are a number of important goals for a link layer security mechanism in sensor networks as described below:

2.3.1 **Access control:**

Access control means there should be a control over access to the wireless network. The link layer protocol should prevent unauthorized/unknown entities from participating in the network communication. The pre deployment process should make the sensors capable of detecting messages from unauthorized nodes and reject them.
2.3.2 *Message authenticity and integrity:*

These are very closely related terms and should be understood clearly. Message authenticity means that the receiver node should be able to distinguish if the message packet is from the brother node or if it has been modified by an adversary. This modification is usually done when the message is in transit. Message integrity means that the message packet should be received the way it was sent and not modified. Both these concerns can be resolved by including a message authentication code with each packet.

2.3.3 *Confidentiality:*

Confidentiality means keeping information secret from unauthorized parties. It is typically achieved with encryption. Preferably, an encryption scheme should not only prevent message recovery, but also prevent adversaries from learning even partial information about the messages that have been encrypted. This strong property is known as semantic security.

2.3.4 *Explicit omission:*

The enemy node / station often wants to take advantage of the fact that sensor nodes have limited space to store messages and hence they keep replaying the same message to drain energy and prevent later messages from reaching the designated destination. Since the message originated from an authorized sender, the same receiver will accept it again. Replay protection is a difficult problem when there is a limited
amount of state that each recipient keeps. A common defense solution is to include an increasing counter with every message and reject messages with old counter values.

2.4 Platform (motes):

In this thesis we will see a representative sensor structure called mica2 and mica2dots. It is from the Crossbow Company [2]. The operating system is TinyOS about which we will discuss in detail later on.

The mica2 processor is an 8 bit, 7.3 mhz Atmel CPU with

- 128 Kb of instruction memory,
- 4 Kb of data ram and
- 512 kb of flash memory
- 1850 mAh capacity
- Low powered radio from Chipcon which has a range of 100 meters.

![Figure 2: Block Diagram of Mica2 platform](image)

Here is the actual sensor device:
The mica2dot has the following specifications:

1. Atmel processor
2. 4.0 MHz clock
3. 128 kB of Flash for program memory
4. 512 KB flash memory
5. 1 programmable LED
6. 560 mAh capacity
The block diagram of mica2 dot is as follows:

![Block Diagram of Mica2 dot](image)

**Figure 5: Block Diagram of Mica2 dot**

2.5 Environment:

We used Cygwin which is Linux-like environment for Windows [3]. It is similar to dual boot operating systems except for the easy switch between windows and Linux emulation. It consists of two parts:

1) A DLL library (cygwin1.dll) which acts as a Linux API emulation layer providing the necessary Linux API functionality.

2) A collection of tools, which provides a Linux look and feel in terms of directory structure and scripting.
Chapter 3

TinyOS (Operating System for tiny devices)

TinyOS is the operating system for low power, scalable wireless embedded sensor networks. Two of its eminent features are open-source and component based framework. It is open source as the programmers can read, redistribute, and modify the source code. They can also improve and are welcome to discover and fix bugs. It features a component-based framework for modular architecture and providing opportunity to improve the connectivity. Components are statically linked hence it enables rapid innovation and implementation. Also leads to code reusability. More on modularity will come in further sections. Needless to say minuscule code size is preferred as severe memory constraints are prevalent in sensor networks. TinyOS supports this and hence it is said to be light weight. The key elements are Networking / communication and power management.

3.1 Does TinyOs have an ISO model?

Tinyos is an operating system as stated above so the next question should be does it have an ISO-OSI model? Well the answer is yes and no. Yes because it has a similar structure and no because it doesn’t follow it that strictly. These tiny devices don’t have enough memory to accommodate the lavish 7 layer design. Following depicts it better:
3.2 Architecture:

TinyOS has an Event-driven architecture. Some of the other important features are:

1. Single shared stack
2. No kernel in the traditional sense
3. No process/memory management
4. No virtual memory
5. No dynamic run time allocation of memory
6. It uses global variables to preserve unity
7. Also uses pointers to save code space

The layer structure is as follows:

Figure 6: TinyOs version of ISO Model
3.3 Main Attraction – Memory Model of TinyOs:

In TinyOs the memory is allocated at run time, also the components are statically linked at compile time to determine the size requirement. The use of local variable is limited and when used is saved on stack. Use of global variable is more prevalent along with pointers to save memory.

3.4 Modularity of TinyOs:

As mentioned before TinyOs has modular structure. There are two kinds of components:

1. Module: It is the component written with code
2. Configuration: It is the components wired together

In short components wire together as a configuration to form any application.
The green interfaces provide the link and the red interfaces uses the links. Every TinyOs component has a frame, function and interfaces. The frame keeps a track of the internal/ initial state. It consists of global variables and other fixed entities. The function carries the code; it has the commands, events and tasks which are written in simple C code called NesC. The commands are the entry point and the events act as callback function. At the entry point the parameters are passed on the stack with the return status. The interfaces provide the link. It provides the interface boundary for the components. The following diagram explains better:
3.4.1 More on NesC:

The code is written in NesC. The NesC output is a ‘C’ program file that is compiled and linked using gnu-gcctools for a specific Mote. The preprocessor does the conversion. The NesC file extension is “.nc”. C stands for configuration. C distinguishes between the interfaces and components that provide it.

M is for module and it indicates that a single component contains both module and configuration.
3.4.2 NesC Jargon:

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interfaces</td>
<td>Collection of events and commands</td>
</tr>
<tr>
<td>Module</td>
<td>Basic component implemented in NesC</td>
</tr>
<tr>
<td>Configuration</td>
<td>Component made from wiring other components</td>
</tr>
<tr>
<td>Implementation</td>
<td>Contains codes and variables for modules/ configuration</td>
</tr>
<tr>
<td>Provides/ uses</td>
<td>Defines /uses interfaces provided by component</td>
</tr>
<tr>
<td>As</td>
<td>Alias interface to another name</td>
</tr>
<tr>
<td>Call</td>
<td>Execute the command</td>
</tr>
<tr>
<td>Signal</td>
<td>Execute the event</td>
</tr>
<tr>
<td>Post</td>
<td>Place task on execution queue</td>
</tr>
<tr>
<td>Task</td>
<td>Computationally intensive low level job</td>
</tr>
<tr>
<td>Async</td>
<td>Commands/ events executed asynchronously</td>
</tr>
<tr>
<td>Atomic</td>
<td>Atomic (can’t stop) execution of statement</td>
</tr>
<tr>
<td>Norace</td>
<td>Eliminates race conditions</td>
</tr>
</tbody>
</table>

Table 1: NesC Jargon
configuration Blink {
}

implementation {
    components Main, BlinkM, ClockC, LedsC;
    Main.StdControl->BlinkM.StdControl;
    BlinkM.Clock->ClockC;
    BlinkM.Leds->LedsC;
}

Figure 12: Blink Text

Correlation:

Figure 13: Blink Diagram

List of Libraries:

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broadcast</td>
<td>Basic Broadcast command</td>
</tr>
<tr>
<td>Deluge</td>
<td>Over the air programming</td>
</tr>
<tr>
<td>X mesh</td>
<td>State of art mesh networking stack</td>
</tr>
<tr>
<td>Queue</td>
<td>Handles send/receive radio packets</td>
</tr>
<tr>
<td>ReliableRoute</td>
<td>Original Cross bow mesh networking stack</td>
</tr>
</tbody>
</table>

Table 2: List of Libraries
Chapter 4

TinySec: a link layer security implementation in hardware

TinySec[1] is a link layer architecture designed by a team of 3 dedicated individuals from the University of California, Berkeley. It is said to be the first fully-implemented link layer security architecture for wireless sensor networks. Some of the eminent features are:

1. It is called lightweight because it doesn’t add much overhead unlike 802.11, GSM, Bluetooth etc.
2. It is identified as a generic security package because it can be paired with other high level algorithms and be used as an open research platform for evaluating them. Also developers can easily integrate into other sensor network applications.
3. TinySec is portable to a variety of hardware and radio platforms. It also has a software cryptographic module and doesn’t need any hardware assistance like Bluetooth, GSM etc.
4. TinySec can be used for secure pair wise communication in public key cryptography.

4.1 Why Link Layer security is needed?

Link Layer security is needed, primarily because it is transparent and covers message integrity, authenticity and confidentiality. Also the end to end security will not be possible since the communication frequency is dense and at times many sensors may note
the same change in environment parameters and send similar data to the base station. It will also increase the data length and use more energy.

4.2 TinySec design:

The following are the different operation modes of TinySec. It supports two different security options:

1) Authenticated encryption (TinySec-AE) and
2) Authentication only (TinySec-Auth).

With the first one: authenticated encryption, TinySec encrypts the data packet and authenticates the packet with a MAC. The MAC is computed after the data has been encrypted along with the packet header. In authentication only mode, the data is not encrypted but TinySec authenticates the entire packet with a MAC.

4.3 TinySec Packet Format:

After seeing the different modes of Tinysec let us see the Tinysec and TinyOS data packets in the figure below. We will later compare these two data packets and observe interesting facts.

(a) TinySec-AE packet format
The common fields are as follows:

Dest: Destination address,

AM: active message type, it is similar to port numbers in TCP/IP. The AM type specifies the appropriate handler function to decrypt, extract and interpret the message on the receiver side.

Len: Length.

As observed these fields are unencrypted because it can be sent in that fashion and also it may lead to saving some save power as, a sensor node may go back to sleep mode after determining that the message is not addressed to it.

If the address and AM type are encrypted, early rejection cannot be invoked until after these fields are decrypted. This wastes power. Encrypting the length field adds little to security since the length of message can be inferred regardless.

To detect transmission errors, TinyOS senders compute a 16-bit cycle redundancy check (CRC) over the packet. The receiver re-computes the CRC during reception and
verifies it with the received CRC field. If they are equal, the receiver accepts the packet and rejects it otherwise. However, CRCs provide no security against malicious modification or forgery of packets.

To guarantee message integrity and authenticity, TinySec replaces the CRC with a MAC. The MAC protects the entire packet, including the destination address, AM type, length, source address and counter (if present), and the data (whether encrypted or not). This protects the data from tampering. It also prevents attackers from redirecting a packet intended for one node to another node, and prevents packet truncation and other attacks. Since MACs detect malicious changes, they also detect transmission errors, and TinySec does not require a CRC. The TinyOS packet format contains a group field to prevent different sensor networks from interfering with each other. It can be thought of as a kind of weak access control mechanism for non-malicious environments.

4.4 Using TinySec in TinyOS applications:

TinySec is fairly easy to apply to the traditional non-security communication stack. In order to enable TinySec, include to Makefile, “TINYSEC=true”, or in command line type:

```
make platform_name_pc_mica_mica2 TINYSEC=true
```

By default TinySec will only authenticate all messages, without encryption. *TinySecMode* has two commands to setup transmit and receive mode.
Example: Enabling TinySec default settings.

```makefile
# add the following line in the Makefile
TINYSEC=true
COMPONENT=Blink
include ../Makerules
```

**4.5 Different modes of operation in TinySec:**

There are different modes of operation for transmission and receiving in TinySec.

Please refer to Appendix.

**4.5.1 Enabling TinySecMode:**

In order to change the modes of operations, `TinySecMode` interface must be used which can be exported using the `TinySecC` component. Here are the steps to wire and setup.

a. Add “TINYSEC=true” in `Makefile`

b. In the interface file (*.nc), export from `TinySecC` component the `TinySecMode` interface.

c. Wire the module component to the `TinySecC`.

d. In the module file (*M.nc), add in “uses”, the TinySecMode interface.

e. Call the commands to change the transmission/receive mode
Example: Setting modes of operation in TinySec to transmission encrypt and authenticate and receiving authenticated only using the example program Blink. The Blink program just blinks the red led, and does not transmit or receive anything. This example demonstrates how to wire and use the interface needed to enable TinySec.

*Blink.nc configuration file:*

```plaintext
configuration Blink { //1 Blink program
}
implementation {
    components Main, BlinkM, SingleTimer, LedsC, // 2) add TinySecC
    TinySecC;

    Main.StdControl -> SingleTimer.StdControl;
    Main.StdControl -> BlinkM.StdControl;
    BlinkM.Timer -> SingleTimer.Timer;
    BlinkM.Leds -> LedsC;

    // 3) Wiring
    BlinkM.TinySecMode -> TinySecC.TinySecMode;
} // Blink
```

*BlinkM.nc module file:*

```plaintext
module BlinkM {
    provides {
```
interface StdControl;
}
uses {
    interface Timer;
    interface Leds;
    // 4) Add TinySecMode interface
    interface TinySecMode;
}
}

implementation {
    /**Initialize the component. @return Always returns <code>SUCCESS</code> **/
    command result_t StdControl.init() {
        call Leds.init();

        // 5) Setting modes of operation
        call TinySecMode.setTransmitMode(TINYSEC_ENCRYPT_AND_AUTH);
        call TinySecMode.setReceiveMode(TINYSEC_RECEIVE_AUTHENTICATED);

        return SUCCESS;
    }
}
4.5.2 Key Management:

TinySec uses symmetric keys to encrypt and authenticate messages. When TinySec is first used a default key is created in a file called:

~/.tinyos_keyfile, which looks like this:

```
# TinySec Keyfile. By default, the first key will be used.
# You can import other keys by appending them to the file.

default A9FB5D03138166ABD1639B360E4263D4
```

When programming devices, from different computers, they must all have the same key-file.

When TinySec is enabled, the concept of “groups” no longer exists. The motes will only receive messages from motes that share the same key, which is enforcedcryptographically.

Notice that the key stored in the “.tinyos_keyfile” is composed of 32 hex values.

TinySec uses key size of 8 bytes or 16 hex values. The first 16 hex values are used as the key for encryption, and the last 16 hex values are used for authentication.
4.5.3 Updating Keys:

*TinySecControl* enables the user to update TinySec keys and reset the initialization vector (IV), with these following commands:

- `command result_t updateMACKey(uint8_t * MACKey);`
- `command result_t getMACKey(uint8_t * result);`
- `command result_t updateEncryptionKey(uint8_t * encryptionKey);`
- `command result_t getEncryptionKey(uint8_t * result);`
- `command result_t resetIV();`
- `command result_t getIV(uint8_t * result)`

To enable these commands *TinySecControl* interface must be used.

4.5.4 TinySec File:

TinySec files can be located under the directory:

/opt/tinyos-1.x/tos/lib/TinySec.

Example files using TinySec for communication can be found under the directory:

/opt/tinyos-1.x/apps/TestTinySec.
Chapter 5

Encryption and Security Primitive

This chapter is the most important as we will see the security modules.

5.1 MAC:

Earlier we had studied the security primitives for wireless security network such as message authenticity, integrity and confidentiality. Previous work has indicated that Message Authentication Code (MAC) can suffice the entire above said network needs. It is a secure checksum calculated with a cryptographic process. Under this the sender and receiver share a secret key. The sender node computes MAC over the data and appends to it. The receiver re-computes the MAC (as it already knows the key) and compares the two MAC’s. The data is accepted if both the MAC’s are equal. The benefit is that the enemy node doesn’t know the secret MAC key so even if they can forge the data they can’t compute the MAC and append to the data payload.

5.2 Initialization Vector:

The most common application in wireless sensor network for the nodes is to report the change in the surrounding conditions. For e.g. if the node is sending YES or NO, it will be obvious for the enemy to decode this as the encrypted part of YES and NO will be same all the time. Hence encryption of same plaintext twice should yield different cipher text. This is where Initialization vector comes is the scene as it adds variation. It is
also sent clearly with the data payload as it is used in the decryption process. The structure of IV is as follows:

```
Dst || AM|| L|| src || ctr
```

Obviously:

Dst (2) : is the destination address

AM (1): Active Message handler

L (1): Length of data

Src (2): Source address

Ctr (2): Counter, it goes from 0 to 16

Tinysec uses 8 byte IV structure which is pre encrypted for better security. IV reuse is not a problem unless key updates are not made.

5.3 Encryption scheme:

Tinysec uses Cipher block chaining (CBC) as the encryption scheme. Firstly because it is designed to be used with random IV’s. Also encryption of two plaintexts P, P’ with the same IV under CBC mode, will reveal only the length (in blocks) of the longest shared prefix of P and P’. That means CBC leaks only a small amount of information in the presence of repeated IVs. CBC is a kind of block cipher which operates on blocks typical 8 or 16 bytes. Also MAC protocol algorithms uses block structure hence blends well with CBC. Tinysec uses Skipjack for software implementation of block cipher.
5.4 Skipjack:

As mentioned before Skipjack [12] is the heart of operation for TinySec. We shall see the operational details and algorithm in detail now.

Skipjack uses 80 bit crypto variable and has 4 modes of operation. One of those is cipher Block Chaining, which we shall see next. Here is the figure:

![Cipher-Block Chaining Mode Diagram](image)

**Figure 15: Cipher-Block Chaining Mode Diagram**

Following sections defines the basic notational terminology:

5.4.1 Notation and terminology:

- $V^n$: the set of all $n$-bit values.
- word: an element of $V^{16}$; a 16-bit value.
- byte: an element of $V^8$; an 8-bit value.
- permutation on $V^n$: an invertible (one-to-one and onto) function from $V^n$ to $V^n$.

That is, the values are permuted within $V^n$, not the bits within the value.
\( X \oplus Y \): the bitwise exclusive-or of \( X \) and \( Y \).

\( X \| Y \): \( X \) concatenated with \( Y \). Let \( X \), \( Y \) be bytes, then 
\[
X \| Y = X \times 2^8 + Y
\]
is a word. Furthermore, \( X \) is the high-order byte, and \( Y \) is the low-order byte.

### 5.4.2 Basic structure:

SKIPJACK encrypts 4-word (i.e., 8-byte) data blocks by alternating between the two stepping rule (A and B) show below. A step of rule A does the following:

- G permutes \( w_1 \),
- The new \( w_1 \) is the xor of the G output, the counter, and \( w_4 \)
- Word \( w_2 \) and \( w_3 \) shift one register to the right; i.e., become \( w_3 \) and \( w_4 \) respectively,
- The new \( w_2 \) is the G output,
- The counter is incremented by one.

Rule B works similarly.
5.4.3 Stepping rules:

In the equations below, the superscript is the step number.

ENCRIPT

\[ w_1^{k+1} = G^k (w_1^k) \oplus w_4^k \oplus \text{counter}^k \]
\[ w_2^{k+1} = G^k (w_1^k) \]
\[ w_3^{k+1} = (w_2^k) \]
\[ w_4^{k+1} = (w_3^k) \]

\[ w_1^{k+1} = w_4^k \]
\[ w_2^{k+1} = G^k (w_1^k) \]
\[ w_3^{k+1} = w_1^k \oplus w_2^k \oplus \text{counter}^k \]
\[ w_4^{k+1} = w_3^k \]

DECRIPT

\[ w_1^{k-1} = [G^{k-1}]^{-1} (w_2^k) \]
\[ w_2^{k-1} = w_3^k \]
\[ w_3^{k-1} = w_4^k \]
\[ w_4^{k-1} = w_1^k \oplus w_2^k \oplus \text{counter}^{k-1} \]

\[ w_1^{k-1} = [G^{k-1}]^{-1} (w_2^k) \]
\[ w_2^{k-1} = [G^{k-1}]^{-1} (w_2^k) \oplus w_3^k \oplus \text{counter}^{k-1} \]
\[ w_3^{k-1} = w_4^k \]
\[ w_4^{k-1} = w_1^k \]
5.4.4 Stepping sequence:

The algorithm requires a total of 32 steps.

- To encrypt: The input is $w^0_i$, $1 \leq i \leq 4$, (i.e., $k = 0$ for the beginning step). Start the counter at 1. Step according to Rule A for 8 steps, then switch to Rule B and step 8 more times. Return to Rule A for the next 8 steps, then complete the encryption with 8 steps in Rule B. The counter increments by one after each step. The output is $w^{32}_i$, $1 \leq i \leq 4$.

- To decrypt: The input is $w^{32}_i$, $1 \leq i \leq 4$, (i.e., $k = 32$ for the beginning step). Start the counter at 32. Step according to Rule $B^1$ for 8 steps, then switch to Rule $A^1$ and step 8 more times. Return to Rule $B^1$ for the next 9 steps, then complete the decryption with 8 steps in Rule $A^1$. The counter decrements by one after every step. The output is $w^0_i$, $1 \leq i \leq 4$.

5.4.5 G-permutation:

The cryptovariable-dependent permutation $G$ on $V^{16}$ is a four-round Feistel structure. The round function is a fixed byte-substitution table (permutation on $V^8$), which will be called the $F$-table. Each round of $G$ also incorporates a byte of cryptovariable. We give two characterizations of the function below:
Recursively (mathematically): \( G^k (w = g_1 || g_2) = g_3 || g_6 \) where

\[ g_i = F(g_{i-1} \oplus cv_{4k+i-3}) \oplus g_{i-2} \] and where \( k \) is the step number (the first step is 0), \( F \) is the substitution table, and \( cv_{4k+i-3} \) is the \((4k+i-3)\)th byte in the cryptovariable schedule. Thus,

\[ g_3 = F(g_2 \oplus cv_{4k}) \oplus g_1 \]
\[ g_4 = F(g_3 \oplus cv_{4k+1}) \oplus g_2 \]
\[ g_5 = F(g_4 \oplus cv_{4k+2}) \oplus g_3 \]
\[ g_6 = F(g_5 \oplus cv_{4k+3}) \oplus g_4 \]

Similarly, for the inverse, \([G^k]^{-1} (w = g_5 || g_6) = g_1 || g_2\) where

\[ g_{i-2} = F(g_{i-1} \oplus cv_{4k+i-3}) \oplus g_i \].

\[ G^k \]

\[ [G^k]^{-1} \]

Figure 17: G-permutation diagram
5.4.6 Cryptovariable schedule:

The cryptovariable is 10 bytes long (labeled 0 through 9) and used in its natural order.

So the schedule subscripts given in the definition of the G-permutation are to be interpreted mod-10.

5.4.7 F Table:

The SKIPJACK F-table is given below in hexadecimal notation. The high order 4 bits of the input index the row and the low order 4 bits index the column. For example, F(7a) = d6.

<table>
<thead>
<tr>
<th></th>
<th>x0</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
<th>x5</th>
<th>x6</th>
<th>x7</th>
<th>x8</th>
<th>x9</th>
<th>xA</th>
<th>xB</th>
<th>xC</th>
<th>xD</th>
<th>xE</th>
<th>xF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x</td>
<td>a3</td>
<td>d7</td>
<td>09</td>
<td>83</td>
<td>f8</td>
<td>48</td>
<td>f6</td>
<td>f4</td>
<td>b3</td>
<td>21</td>
<td>15</td>
<td>78</td>
<td>99</td>
<td>b1</td>
<td>af</td>
<td>f9</td>
</tr>
<tr>
<td>1x</td>
<td>e7</td>
<td>3d</td>
<td>4d</td>
<td>8a</td>
<td>ce</td>
<td>4c</td>
<td>ca</td>
<td>2e</td>
<td>52</td>
<td>95</td>
<td>d9</td>
<td>1e</td>
<td>4e</td>
<td>38</td>
<td>44</td>
<td>28</td>
</tr>
<tr>
<td>2x</td>
<td>0a</td>
<td>df</td>
<td>02</td>
<td>a0</td>
<td>17</td>
<td>f1</td>
<td>60</td>
<td>68</td>
<td>12</td>
<td>b7</td>
<td>7a</td>
<td>c3</td>
<td>e9</td>
<td>fa</td>
<td>3d</td>
<td>53</td>
</tr>
<tr>
<td>3x</td>
<td>96</td>
<td>84</td>
<td>6b</td>
<td>Ba</td>
<td>f2</td>
<td>63</td>
<td>9a</td>
<td>19</td>
<td>7c</td>
<td>ae</td>
<td>e5</td>
<td>f5</td>
<td>f7</td>
<td>16</td>
<td>6a</td>
<td>a2</td>
</tr>
<tr>
<td>4x</td>
<td>39</td>
<td>b6</td>
<td>7b</td>
<td>0f</td>
<td>c1</td>
<td>93</td>
<td>81</td>
<td>1b</td>
<td>ee</td>
<td>b4</td>
<td>1a</td>
<td>ea</td>
<td>d0</td>
<td>91</td>
<td>2f</td>
<td>b8</td>
</tr>
<tr>
<td>5x</td>
<td>55</td>
<td>b9</td>
<td>Da</td>
<td>85</td>
<td>3f</td>
<td>41</td>
<td>bf</td>
<td>e0</td>
<td>5a</td>
<td>58</td>
<td>80</td>
<td>5f</td>
<td>77</td>
<td>0b</td>
<td>d8</td>
<td>90</td>
</tr>
<tr>
<td>6x</td>
<td>35</td>
<td>d5</td>
<td>c0</td>
<td>a7</td>
<td>33</td>
<td>06</td>
<td>65</td>
<td>69</td>
<td>45</td>
<td>00</td>
<td>94</td>
<td>56</td>
<td>6d</td>
<td>98</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>7x</td>
<td>c9</td>
<td>7f</td>
<td>0c</td>
<td>b2</td>
<td>c2</td>
<td>b0</td>
<td>fe</td>
<td>db</td>
<td>200</td>
<td>e1</td>
<td>eb</td>
<td>d6</td>
<td>e4</td>
<td>dd</td>
<td>47</td>
<td>4a</td>
</tr>
<tr>
<td>8x</td>
<td>42</td>
<td>ed</td>
<td>9e</td>
<td>6e</td>
<td>49</td>
<td>3c</td>
<td>cd</td>
<td>43</td>
<td>27</td>
<td>d2</td>
<td>07</td>
<td>d3</td>
<td>de</td>
<td>c7</td>
<td>67</td>
<td>18</td>
</tr>
<tr>
<td>9x</td>
<td>89</td>
<td>cb</td>
<td>30</td>
<td>1f</td>
<td>8d</td>
<td>c6</td>
<td>8f</td>
<td>aa</td>
<td>c8</td>
<td>74</td>
<td>dc</td>
<td>9c</td>
<td>5d</td>
<td>5c</td>
<td>31</td>
<td>a4</td>
</tr>
<tr>
<td>Ax</td>
<td>06</td>
<td>7e</td>
<td>88</td>
<td>61</td>
<td>2c</td>
<td>9f</td>
<td>0d</td>
<td>2b</td>
<td>87</td>
<td>50</td>
<td>82</td>
<td>54</td>
<td>64</td>
<td>26</td>
<td>17</td>
<td>40</td>
</tr>
<tr>
<td>Bx</td>
<td>34</td>
<td>4b</td>
<td>1c</td>
<td>73</td>
<td>d1</td>
<td>c4</td>
<td>fd</td>
<td>3b</td>
<td>cc</td>
<td>fb</td>
<td>7f</td>
<td>ab</td>
<td>e6</td>
<td>3e</td>
<td>5b</td>
<td>a5</td>
</tr>
<tr>
<td>Cx</td>
<td>Ad</td>
<td>04</td>
<td>23</td>
<td>9c</td>
<td>14</td>
<td>51</td>
<td>22</td>
<td>f0</td>
<td>29</td>
<td>79</td>
<td>71</td>
<td>7e</td>
<td>ff</td>
<td>8c</td>
<td>0e</td>
<td>e2</td>
</tr>
<tr>
<td>Dx</td>
<td>0c</td>
<td>ef</td>
<td>Bc</td>
<td>72</td>
<td>75</td>
<td>6f</td>
<td>37</td>
<td>a1</td>
<td>ec</td>
<td>d3</td>
<td>8e</td>
<td>62</td>
<td>8b</td>
<td>86</td>
<td>10</td>
<td>e8</td>
</tr>
<tr>
<td>Ex</td>
<td>08</td>
<td>77</td>
<td>11</td>
<td>Be</td>
<td>92</td>
<td>4f</td>
<td>24</td>
<td>c5</td>
<td>32</td>
<td>36</td>
<td>9d</td>
<td>cf</td>
<td>f3</td>
<td>a6</td>
<td>bb</td>
<td>ac</td>
</tr>
<tr>
<td>Fx</td>
<td>5e</td>
<td>6c</td>
<td>a9</td>
<td>13</td>
<td>57</td>
<td>25</td>
<td>b5</td>
<td>e3</td>
<td>bd</td>
<td>a8</td>
<td>3a</td>
<td>01</td>
<td>05</td>
<td>59</td>
<td>2a</td>
<td>46</td>
</tr>
</tbody>
</table>

Table 4: Sample F Table
Chapter 6

Elliptic Curve Cryptography

6.1 Introduction:

Elliptic curve systems as applied to cryptography were first proposed in 1985 independently by Neal Koblitz from the University of Washington, and Victor Miller, who was then at IBM, Yorktown Heights. Elliptic curve cryptography (ECC) is an approach to public-key cryptography based on the mathematics of elliptic curves. It provides fast decryption and digital signature processing. The main benefit of ECC is that under certain situations it uses smaller keys than other methods — such as RSA — while providing an equivalent or higher level of security. ECC uses points on an elliptic curve to derive a 163-bit public key that is equivalent in strength to a 1024-bit RSA key. Hence smaller number of keys lead to faster key operation and less memory need. It is said to be ideal for resource-constrained systems because it provides more "security per bit" than other types of asymmetric cryptography. One drawback, however, is that the implementation of encryption and decryption operations may take longer than in other schemes.

6.2 Motivation for using Elliptic Curve Cryptography:

As previously mentioned before RSA and ECC are both public key algorithms. RSA [8] is the most used public key algorithm but ECC [9] can provide the same security at much smaller key size. RSA-1024 is equivalent to ECC-160 bit key [10]. It has also been also calculated that by the year 2010 we have to use RSA-
2048 but in case of ECC: ECC-224 will be needed! In this section we will see the comparison between both and learn why ECC is better for the embedded sensor network as a public key algorithm. The energy analysis has been done on the Mica2dots[2], which is one of the popular research platforms for WSN. [10] It has been noted that the power required to transmit 1 bit is equivalent to roughly 2090 clock cycles of execution on the Mica2dot microcontroller. The cost of receiving one byte (28.6µJ) is roughly half of that required to transmit a byte (59.2 µJ). During transmit and receive, the microcontroller is powered on along with the wireless transceiver. The usual packet size is 32 for the payload and 9 bytes for the header. The header, ensuing a 8-byte preamble, consists of source, destination, length, packet ID, CRC, and a control byte. Receiving one 41-byte packet (including 8-byte preamble) costs 49*28.6uJ=1.40mJ and transmitting one 41-byte packet costs 49*59.2uJ=2.90mJ. To summarize:

| Effective data rate 12.4 kbps |
| Energy to transmit 59.2 µJ/byte |
| Energy to receive 28.6 µJ/byte |

### 6.3 Mathematics for RSA Vs ECC:

Verification of RSA is cheap but what makes it expensive is the signature for authentication. This is due to the fact that RSA-based key exchange protocol relies on party A to encrypt a randomly generated secret key with party B's public key, and party B decrypting the key using its private key. Now for ECC the signature
algorithm is called: Elliptic Curve Digital Signature Algorithm [ECDSA][11]. For ECDSA the signature and verification both are inexpensive. The transition from RSA-1024 to RSA-2048 the energy cost of signing increases by a factor of more than seven, while ECDSA-224 signing is less than three times as expensive as ECDSA-160 signing. With ECC, both parties perform a single ECDH operation to derive the secret key. From [10]:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Signature</th>
<th>Key Exchange</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sign</td>
<td>Verify</td>
</tr>
<tr>
<td>RSA-1024</td>
<td>304</td>
<td>11.9</td>
</tr>
<tr>
<td>ECDSA-160</td>
<td>22.82</td>
<td>45.09$^3$</td>
</tr>
<tr>
<td>RSA-2048</td>
<td>2302.7</td>
<td>53.7</td>
</tr>
<tr>
<td>ECDSA-224</td>
<td>61.54</td>
<td>121.98$^3$</td>
</tr>
</tbody>
</table>

Table 5: Comparison chart for RSA and ECC

The above is measured in mille Joules.

Hence with the facts, figures and numbers we can clearly say that ECC is the better choice than the most popularly used encryption algorithm for WSN.

6.4 Asymmetric Cryptography as fine balance:

It has been claimed that public key cryptosystems are not feasible to implement in these tiny devices because they are constrained with less resources but this is not true [33]. Asymmetric cryptography is a marvelous technology. Its uses are many and varied.

For many situations in distributed network environments, asymmetric cryptography is a must during communications. For instance key distribution or any
kind of network protocol or application requiring secure communications such as buying something on the Internet, when the vendor is using a secure server.

But asymmetric cryptography is demanding and complex, by nature. The hard problems in number theory — the key to the algorithms' functionality — are all intrinsically difficult.

Let’s see some of the features of elliptic curve cryptography (ECC) and later see the justification so as to need this in the sensor networks.

1) ECC offers considerably greater security for a given key size

2) The smaller key size also makes possible much more compact implementations for a given level of security, which means faster cryptographic operations, running on smaller chips or more compact software. This means less heat production and less power consumption — all of which is of particular advantage in constrained devices, but of some advantage anywhere else.

3) There are extremely efficient, compact hardware implementations available for ECC exponentiation operations, offering potential reductions in implementation footprint even beyond those due to the smaller key length alone.

In short: asymmetric cryptography is demanding but looking at the cryptosystem for more security per bit, ECC is a better choice.
6.5 Why Asymmetric Cryptography?

Asymmetric cryptographic algorithms do not use a single key as in symmetric cryptographic algorithms such as AES, but a key pair. One of the keys (the public key) is used for encryption, and its corresponding private key must be used for decryption. The critical feature of asymmetric cryptography, which makes it useful, is this key pair — and more specifically, a particular feature of the key pair: the fact that one of the keys cannot be obtained from the other. In all asymmetric cryptosystems, as mentioned above, the key length is the parameter that determines how difficult both the forward and inverse algorithms are.

Hence, the inverse operation rapidly becomes more difficult as key length increases than does the forward operation.

6.6 Authentication with Asymmetric Cryptography:

In the case of asymmetric authentication methods there are two main entities: private key (in the possession of the entity wishing to prove its identity) and the public key (in the possession of anyone who wishes to verify the identity of the entity possessing the private key).

The Public key can verify the knowledge of the private key but the private key cannot be derived from the public key. This is the critical feature of asymmetric cryptographic schemes that makes them so useful.

This property is useful for a number of reasons: it greatly simplifies key exchange, as one example, and it solves one critical problem symmetric cryptography
cannot solve — the problem of guaranteeing unique authentication and non-repudiation. Symmetric hashing/authentication methods — ones for which there is only one key, and both parties in the exchange use it both for authentication and for signature generation — have the distinct disadvantage that they do not, on their own, offer any way to distinguish which party to the exchange signed a given message. If both or all parties must know the key, based on cryptography alone, you cannot distinguish which signed any given message, because any of them could have. In asymmetric authentication schemes, only one party knows the private key, with which the message is signed. Any number may know the public key. Since the private key cannot be derived from the public, the signature serves as a unique identifier. If the message verifies as having been signed by the person with knowledge of the private key, we can narrow down who sent the message to one. But any number of people may have knowledge of the public key, and all of them can therefore verify the identity of the sender.

6.7 How are elliptic curves used?

The crucial property of an elliptic curve is that we can define a rule for "adding" two points which are on the curve, to obtain a 3rd point which is also on the curve. Having defined addition of two points, we can also define multiplication k*P where k is a positive integer and P is a point as the sum of k copies of P.

Thus 2*P = P+P
3*P = P+P+P
Alice, Bob, Cathy, David... agree on a (non-secret) elliptic curve and a (non-secret) fixed curve point F. Alice chooses a secret random integer Ak which is her secret key, and publishes the curve point AP = Ak*F as her public key. Bob, Cathy and David do the same. Now suppose Alice wishes to send a message to Bob. One method is for Alice to simply compute Ak*BP and use the result as the secret key for a conventional symmetric block cipher (say DES).

Bob can compute the same number by calculating Bk * AP, since Bk*AP = Bk*(Ak*F) = (Bk*Ak)*F = Ak*(Bk*F) = Ak*BP.

Although the starting point and public key are known, it is extremely difficult to backtrack and derive the private key. An essential property for cryptography is that a group has a finite number of points.

6.8 The Diffie Hellman/DSA Cryptosystems and the Discrete Logarithm

Problem:

Diffie Hellman along with the Digital Signature Algorithm (DSA) based on it is another of the asymmetric cryptosystems in general use.

ECC, in a sense, is an evolved form of Diffie Hellman. So to understand how ECC works, it helps to understand how Diffie Hellman works first.
Diffie Hellman uses a problem known as the discrete logarithm problem as its central, asymmetric operation. The discrete log problem concerns finding a logarithm of a number within a finite field arithmetic system.

Prime fields are fields whose sets are prime that is, they have a prime number of members. These are of particular interest in asymmetric cryptography because, over a prime field, exponentiation turns out to be a relatively easy operation, while the inverse computing the logarithm is very difficult.

To generate a key pair in the discrete logarithm (DL) system, therefore, the calculation is as follows:

\[ y = (gx) \mod p \]

where p is a large prime — the field size. x and g are smaller than p. y is the public key. x is used as the private key. In Diffie Hellman, again, the operations we wish to make 'easy', or tractable, we harness to the operation in the field which is (relatively) easy — exponentiation. So encryption using the public key is an exponentiation operation. Decryption using the private key is as well. Decryption using the public key, however, would require performing the difficult inverse operation — solving the discrete logarithm problem.
The discrete logarithm problem, using the values in the equation above, is simply finding \( x \) given only \( y \), \( g \) and \( p \).

Expanding that thought slightly: someone has multiplied \( g \) by itself \( x \) times, and reduced the result into the field (performed the modulo operation) as often as necessary to keep the result smaller than \( p \). Now, knowing \( y \), \( g \) and \( p \), you're trying to find out what value of \( x \) they used.

It turns out that for large enough values of \( p \), where \( p \) is prime, this is extraordinarily difficult to do — much more difficult than just finding \( y \) from \( g \), \( x \) and \( p \).

ECC defines its group differently. And it is, in fact, the difference in how the group is defined and particularly how the mathematical operations within the group are defined that give ECC its greater security for a given key size.

### 6.9 The Elliptic Curve Discrete Logarithm Problem:

The elliptic curve discrete logarithm problem [27] is the cornerstone of much of present-day elliptic curve cryptography. It relies on the natural group law on a non-singular elliptic curve which allows one to add points on the curve together. Given an elliptic curve \( E \) over a finite field \( F \), a point on that curve, \( P \), and another point you know to be an integer multiple of that point, \( Q \), the "problem" is to find the integer \( n \) such that: \( nP=Q \)
The problem is computationally difficult unless the curve has a "bad" number of points over the given field, where the term "bad" encompasses various collections of numbers of points which make the elliptic curve discrete logarithm problem breakable. For example, if the number of points on E over F is the same as the number of elements of F, then the curve is vulnerable to attack. It is because of these issues that point-counting on elliptic curves is such a hot topic in elliptic curve cryptography.

The inverse operation to point multiplication finding a log in a group defined on an elliptic curve over a prime field is defined as follows: given points Q and P, find the integer k such that Q = kP.

This is the elliptic curve discrete logarithm problem and this is the inverse operation in the cryptosystem the one we effectively have to perform to get the plaintext back from the cipher text, given only the public key.

Now naively the obvious, certain way of finding k would be to perform repeated addition operations stepping through P, 2P, 3P, and so on, until we find kP. Begin by doubling P, then adding P to 2P finding 3P, then 3P to P finding 4P and so on. This is the brute force method. The difficulty with this approach is by using a large enough prime field, the number of possible values for k becomes inconveniently large. So inconveniently large that it's quite practical to create a sufficiently large prime field that searching through the possible values of k would take all the processor time currently available on the planet thousands of years.
6.10 Elliptic curve groups [13]:

Many cryptosystems require the use of algebraic groups. Elliptic curves may be used to form elliptic curve groups. A group is a set of elements with custom-defined arithmetic operations on those elements. For elliptic curve groups, these specific operations are defined geometrically. Introducing more stringent properties to the elements of a group, such as limiting the number of points on an elliptic curve, creates an underlying field for an elliptic curve group.

6.11 Implementation of ECC in NesC for Mica2: [14]

Up until now Tinysec implements symmetric key cryptography. In this section this thesis work depicts how asymmetric key cryptography can be implemented in these tiny devices.

The code is in the following directory:

/opt/tinyos-1.x/apps/EccM-2.0

![EccM Directory in Cygwin](image)

**Figure 19: EccM Directory in Cygwin**

The original code has errors and gives the error
EccM.nc:968: interface has no command or event named 'get'

For higher version of TinyOs (greater than tinyos 1.1.0) the code has to be modified as follows:

```c
// this is old SysTime
// call SysTime.get(&before);
// This line has to be replaced as follows
  before = call SysTime.getTime32();

// this is old SysTime
// call SysTime.get(&after);
// This line has to be replaced as follows
  after = call SysTime.getTime32();
```

To fix the “init” problem, it needs to be commented out, since it gets automatically initiated.

```c
// initialize clock
// result = rcombine(call SysTime.init(), result);
```

After these fixes, it should compile successfully on mica2 platform. The code then can be downloaded and tested on the mica2 hardware.

Install the ECC demo code in each of the motes giving them a unique ID, with the command: `make mica2dot install1 MIB510=COM5`
Figure 20: Installing Ecc on mica2dot (id # 1)

Similarly for the other mote:
Figure 21: Installing Ecc on mica2dot (id # 2)

To run the demo on the two motes:

One mote should be left on the programming board, to be connected to the computer with a serial cable/ USB. Execute a serial forwarder on your computer, as with the command below, adjusting as needed for your particular configuration.
With the SerialForwarder backgrounded (or running in its own terminal), also the following has to be executed:

```
java Ecc
```

which will simply report to standard output any messages delivered to the UART on the programming board.

Turn both motes on, roughly simultaneously. The mote will turn on the yellow LED when the private/public keys are generated.

Here are the results for the ECC code from the window that was running serialforwarder, which is a program written in java to the packets sent by the motes through the serial port:

```
$ java net.tinyos.sf.SerialForwarder -comm serial@COM1:57600 -no-gui
```
Figure 22: Ecc result

In text format it is as follows:

$ java Ecc
privKeyTime: 0.23898398981416266
pubKeyTime: 35.265216990843584
secKeyTime: 35.308831879503714

privKeyTime: 0.23686663596467467
pubKeyTime: 35.028514655686195
secKeyTime: 35.134459554640515

privKeyTime: 0.23603036788210435
pubKeyTime: 35.922323237795965
secKeyTime: 35.723772010619285

privKeyTime: 0.23516728070650703
pubKeyTime: 34.74242645066912
secKeyTime: 34.78137468169238

privKeyTime: 0.23742286937205397
pubKeyTime: 35.36152652110311
secKeyTime: 35.37715527983963

privKeyTime: 0.23735059868884437
pubKeyTime: 35.95089667876687
secKeyTime: 35.0038926965379

privKeyTime: 0.23550495746871108
pubKeyTime: 35.81251259684673
secKeyTime: 35.66207062361163

privKeyTime: 0.23474266240450777
pubKeyTime: 34.27488121038089
secKeyTime: 34.16137928699139

privKeyTime: 0.2344175615143308
pubKeyTime: 36.48261838326922
secKeyTime: 36.577420355420706

privKeyTime: 0.23760592187246032
pubKeyTime: 33.955729398060356
secKeyTime: 33.88390718968413

privKeyTime: 0.23957495801051093
pubKeyTime: 33.809408219103865
secKeyTime: 33.82116175434794
secKeyTime: 35.308831879503714
privKeyTime: 0.23686663596467467
pubKeyTime: 35.028514655686195
secKeyTime: 35.134459554640515

privKeyTime: 0.23603036788210435
pubKeyTime: 35.922323237795965
secKeyTime: 35.723772010619285

privKeyTime: 0.23516728070650703
pubKeyTime: 34.74242645066912
secKeyTime: 34.78137468169258

privKeyTime: 0.23472286937205397
pubKeyTime: 35.36152652110311
secKeyTime: 35.7715527983963

privKeyTime: 0.23735059868884437
pubKeyTime: 34.95089667876687
secKeyTime: 35.0038926965379

privKeyTime: 0.23550495746871108
pubKeyTime: 35.81251259684673
secKeyTime: 35.66207062361163

privKeyTime: 0.23748266240450777
pubKeyTime: 34.27488121038089
secKeyTime: 34.16137928699139

privKeyTime: 0.23441756515143308
pubKeyTime: 36.48261838326922
secKeyTime: 36.577420355420706

privKeyTime: 0.23760592187246032
pubKeyTime: 33.955729398060356
secKeyTime: 33.88390718968413

privKeyTime: 0.23957495801051093
pubKeyTime: 33.809408219103865
secKeyTime: 33.82116175434794

privKeyTime: 0.23542328113994693
pubKeyTime: 37.43303611096061
secKeyTime: 37.327121552798395
The ECC code is known to be CPU intensive, and thus take a long time to generate the keys. During this time, events posted by other asynchronous events might fill up the TinyOS’s queue. During this demo, timers were used to avoid this kind of problem and it is sufficient for demo purposes. As for real usage of this code, it is recommended by the author to use a cooperative threading mechanism for TinyOS such as one developed by Geoffrey Mainlands Fibers module.

CPU intensive also means it uses a lot of power. This was observed on the mote that was connected to the PC. After running it for about 40 minutes, the power LED on the MIB510 programming board started to blink, which signals low power.
6.12 Conclusion: ECC for Portable Devices, Applications …and for the Future

And this, in the end, is the reason ECC is a stronger option than the RSA and discrete logarithm systems for the future. The research shows that ECC is an excellent choice for doing asymmetric cryptography in portable, necessarily constrained devices.

As an example: as of this writing, a popular, recommended RSA key size for most applications is 2,048 bits. For equivalent security using ECC, a key of only 224 bits is needed. ECC also provides more security per bit and meets the constrained power demands of sensor devices [32].

The difference becomes more and more pronounced as security levels increase (and, as a corollary, as hardware gets faster, and the recommended key sizes must be increased). A 384-bit ECC key matches a 7680-bit RSA key for security.

The smaller ECC keys mean the cryptographic operations that must be performed by the communicating devices can be squeezed into considerably smaller hardware, that software applications may complete cryptographic operations with fewer processor cycles, and operations can be performed that much faster, while still guaranteeing equivalent security.
This results, in turn, less heat, less power consumption, less real estate consumed on the printed circuit board, and software applications that run more rapidly and make lower memory demands. Leading in turn to more portable devices which run longer, and produce less heat.
Chapter 7

Energy analysis:

To refresh Mica2/ Mica2dots have TinyOS as the operating system and TinySec as the security protocol. The security has been enhanced by implementing ECC.

Tinysec operates in two modes: ‘AE’ which has 42 bytes and ‘A’ which has 38 bytes to send.

The energy equation for communicating over the distance r is as follows:

\[ e(r) = e_t(r) + e_r(r) \]  \[29\]  \text{(from main thesis)}

i.e. the energy spent is the sum total of energy spent is transmitting and receiving the data packets.

Note: in this document tx is used for transmitting and rx refers to receiving data.

Even when the network is idle a certain non-zero level of power is radiated. Energy is spent in encrypting, decrypting, transmitting, receiving, buffering and processing.

The most typical case where the node that is receiving and at the same time transmitting has been discussed below. The power for receiving is usually constant because it buffers the data. Therefore the total energy/power (denoted by \( p \)- function of \( r= p(r) \)) required for communicating between the base station and / or nodes over distance ‘r’ is as follows \[28\]:

\[ p(r) = \text{max/ min transmitting power + receiving power}. \]
Total \( p(r) = \max \{ p_{\min}, c.r^\alpha \} + p_{rx} \)

Here

\( \alpha = \) power loss in the path which multiplies by the distance

\( p_{\min} = \) minimum transmitter power

\( p_{rx} = \) fixed overhead for receiving data and

\( c = \) constant, function of transmitter distance from the base station

The current work has been implemented with ECC for 160 bit keys; the case scenario is as follows:

Laptop= base station \hspace{1cm} \text{Mica2= nodes}

Here are the following formula has been used:

\( \text{Watt} = \text{Joules/ sec} \)

\( = \text{Volt} \times \text{Ampere} \)

\( = \text{Volt} \times \text{Coulomb/Sec} \)

\( \text{Joules} = \text{Volt} \times \text{Ampere} \times \text{sec} \)

From specification sheet ([2] – thesis page #80) the following is the current consumption:

Sensor board: (transmitting): 5mA

Microprocessor: (transmitting): 6 mA

Radio: (transmitting): 12 mA

Total= 23 mA
Also after running the set up for ECC- 162 (result at page 66) time consumed is on an average= 35 sec for key generation for AE data packets. This experiment is for worst case scenario therefore, doesn’t include the pre deployment or hard coding of the key.

Hence, the energy calculation after running ECC (from page 66) is as follows:

At transmitting side (for Authentication encryption mode):

Energy consumed (Joules) = Key generation + energy used to transmit 42 bytes + signature

= 226.65 mJ [Ref: 36] + volt * (23 mA) * 35 secs + 134.2 mJ (constant for signing, irrespective of byte size)

By observation the volt (battery power) varies as a function of the distance between the transmitting and receiving entities. The mica2 uses 2 AA battery which provides 3v.

For calculation purpose and from experimental observation the volt is assumed to be 0.01 constant then the energy consumed during transmitting AE type of Tinysec data packet will be:

= .22665 + 0.01*.805 + .1342 -- Joules

= .22665 + .00805 +.134

= .3687 J

Similar calculation can be done for receiver side:
Energy consumed (Joules) = Key Verify + energy used to receive 42 bytes

= 196.23 mJ [Ref: 36] + volt * [(6+5) mA] * 35 secs

= .19623 + 0.01 * .385 – Joules

= .1962 + .00385

= .2 J

There is no use of radio as the data is received in this case.

Also for the sleep mode in which the node may be residing is not considered.

Considering it adds an overhead of the following [2]:

Microprocessor = 8 mA

Radio = 2 mA

Sensor Board = 5 uA

Now for Authentication mode the data packet size reduces to 38 bytes.

For Mica2 dots the numbers differ by the following [35]:

Energy to transmit: 59.2 microJ/byte

Energy to receive: 28.6 microJ/byte

For transmitting:

Energy consumed (Joules) = Key generation + energy used to transmit 42 bytes +

signature

= 226.65 mJ + 59.2 uJ * 42 + 134.2 mJ

= 363.336 mJ
For Receiving:

Energy consumed (Joules) = Key Verify + energy used to receive 42 bytes

= 196.23 mJ + 28.6 uJ * 42

= 196.23 mJ + 1.201 mJ

= 197.43 mJ

Now for Authentication mode the data packet size reduces to 38 bytes.

7.1 Memory Overhead:

Typically nodes are deployed in thousands. In the current set up there are only 2 mica2 and 2 mica2 dots. Under pragmatic scenario for 10,000 nodes deployed each node will have 30 neighbors (for dense connection) with whom they have to share the key. If every node stores 30 keys of 192 bits, which will be way less than 1 Kbytes of RAM memory. This is the overhead for each node.

7.2 Comparison with RSA (asymmetric cryptography): [36]

Below is a comparison of energy spent (in mJ) for ECC with RSA for two different key lengths. As mentioned before RSA-1024 is equivalent to ECDSA -160 and RSA-2048 is equivalent to ECDSA -224.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Key Generation</th>
<th>Signature - mJ</th>
<th>Verify - mJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA – 1024</td>
<td>270.13</td>
<td>304</td>
<td>11.9</td>
</tr>
<tr>
<td>ECDSA -160</td>
<td>226.65</td>
<td>134.2</td>
<td>196.23</td>
</tr>
</tbody>
</table>
Table 6: Comparison Table for ECC and RSA (Overhead)

7.3 Comparison with Symmetric Counter Part:

Although Symmetric key cryptographic systems are 2-4 times faster they require pre – distribution of keys and storage of n-1 keys. For e.g. for small network of say 1024 nodes the scheme with 128 keys will require $1024 \times 128 = 16$ Kbytes of Ram memory.

Even the random pair wise distribution where the secret keys stored are $n \times p$ where $n$ is number of node and $p$ is the probability of sharing key with other node. It is not scalable for big network system hence it gets worse.

7.4 Comparison with One Time Sensors: [35]

According to the commonly used scheme for one time sensors the nodes need to be pre loaded before deployment. It is loaded with single cryptographic key, node id and list of hashed values. For e.g for a big network of 10,000 nodes with 128 bit key value, 64 bit hash value the total Ram value will be around 80 Kbytes. For small network with ~ 1000 nodes the Ram value will be little over 8 Kbytes. Combining with another scheme even with storing the least significant bits of the Id the total ram value is 8 Kbytes.

As mentioned before in great detail about different modes of skipjack, the simplest is the ECB. Other modes (cipher block chaining (CBC), cipher-feedback
mode (CFB), and output-feedback mode (OFB)) employ a feedback mechanism so that the encryption of a plain-text block is made dependent on the results of encryption of previous plain-text blocks. Due to the feedback mechanism, even for the same key, a given plain-text will not always map to the same ciphertext. In addition to variants due to operating mode considerations, there are variants of the basic DES algorithm such as DES-X. The corresponding energy consumption profile is plotted in figure below: [36]

![Energy consumption for CBC and variants of DES](image)

**Figure 23: Energy consumption for CBC and variants of DES**

Proposed methods to reduce energy overhead are:

1. Alternate between sleep and idle modes
2. Synchronization using Virtual clustering of nodes on same sleep schedule.
3. Use of in-network signaling so that the neighbor nodes can go to sleep when the nodes are transmitting the data.
4. Transmission of data in store-forward format. Hence the data is sent in chunks and the nodes that have large chunks of data to be sent can access the channel.

Under this practice the nodes first listen for a certain amount of time. If there is no change in the environment and if none of the neighbors want to talk then it goes to sleep and puts a timer on itself to wake up later. It also switches off its radio. Also the nodes can synchronize with its neighbors so that they sleep and wake up at the same time. Nodes also exchange their schedule by broadcasting to its immediate neighbors. If a node receives the schedule from its neighbor before broadcasting its schedule then it adjusts its schedule accordingly. It is called a follower. In order to have collision avoidance the nodes have to contend for the medium. There is a duration field in each transmitted packet that indicates how long the transmission will be, hence it knows how long to sleep for. This can also help in the overhearing problem, so that the nodes know that the data is not meant for it.

[21] also suggests that unequal clustering can lead to better energy management. In the wireless networks (multi hop scenario) the nodes organize themselves to form clusters. The cluster has a cluster head which aggregates the data and sends it to the base station. The clusters have inter and intra communication. The energy consumed on intra-cluster communication changes proportionally with the number of nodes within a cluster, while the energy spent on inter-cluster communication (i.e., forwarding data from other clusters) is a function of the expected load from the clusters further away. Therefore, by changing the number of nodes in every cluster
with respect to the expected relay load, the network can maintain more uniform energy consumption. The cluster formation can either be static or dynamic. This can depend on various factors like energy remaining, distance from cluster head etc.

7.5 Analysis of models: [22]

Various models have been studied and devised for analyzing the networks. Two major categories are the Time based and Packet based models. Time based models analyzes the network in terms of time spent doing calculation, processing and how it can be reduced. The Packet based model analyzes how to make the packet size accommodate all necessary data and control information yet not over burden the nodes or for that matter the cluster head, this may also help in solving the hot-spot problem.

Analysis has also been done with [23] Network Models. They have been categorized into: Homogenous and Hybrid. Homogenous networks have similar kinds of nodes and they take turns in becoming the cluster head. Also they have similar processing power and sleep schedules. The deployment of these nodes can be either pre determined or random.

In heterogeneous networks nodes with higher processing power and better energy saving protocols are the cluster heads. They are deployed as another layer. The other nodes have to determine the closest such nodes and affiliate to its cluster.
Much work has been done with traffic models. Two kinds of traffic models have been identified: Under Ad Hoc model: where nodes can be sender or receiver and they communicate one-to-one. Second kind: Sensor Traffic Model, in which all nodes are senders and they communicate to sink (cluster head).

Other models [24] suggest that the duty of being the cluster head should be rotated among the nodes which have relatively highly energy. [25] Suggests that when the nodes communicate they send information to the neighboring nodes to forward to cluster head. They take turns to do so hence by alternating their chance with sleep they save energy. This works well for even as well as odd number of nodes.
Conclusion

This thesis work gives a better understanding of the sensor networks in terms of their working environment, threats, attacks and security protocols. This work emphasized and brought out in light the security algorithm called Skipjack and enhanced it with Elliptic Curve Cryptography. The idea for ECC has been borrowed from Mathematics where the unique property of Elliptic curves have been used and proved that Asymmetric Cryptography can be implemented in these minuscule sensor devices. As observed it drains the battery supply but there has to be a trade off between the energy utilized and level of security threshold. This thesis work also described the energy consumption for all flavors and modes of tinysec with ECC and its overhead analysis when compared to symmetric techniques. Most of the modern sensors today operate on renewable energy source hence public key cryptography can be implemented in embedded sensor devices.

The future work will include a good support from routing and clustering schemes which can further enhance better security systems to keep away adversaries and eaves droppers. Perhaps ECDSA can be implemented in hardware and its energy consumption can be monitored on a bigger scale and more practical platform. Also the current working platform was mica2 and mica2 dots which are research platforms; they should be tried on the more modern industrial level sensors available in the market.
This work has seen the codes in NesC in linux environment which is not very user friendly, it can be further extended to windows based application and software which will create more awareness and harness it at research level. This will help in eliminating the various security threats and improve secure data transmission protocol.
References


www.crossbow.com


Conference on
8-12 March 2005

Appendix

1) The system specifications for message format are as follows:

<table>
<thead>
<tr>
<th>Specifications</th>
<th>Value</th>
<th>units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Msg Size</td>
<td>40</td>
<td>bytes</td>
</tr>
<tr>
<td>Msg Preamble</td>
<td>16</td>
<td>bytes</td>
</tr>
<tr>
<td>Baud Rate</td>
<td>38400</td>
<td>baud</td>
</tr>
<tr>
<td>Duty Cycle</td>
<td>0.5</td>
<td>%</td>
</tr>
<tr>
<td># of msgs to rcv/re-xmit</td>
<td>5</td>
<td>msg</td>
</tr>
<tr>
<td>during wake time</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Message Format for Energy Overhead

2) Transmit mode:

```c
command result_t setTransmitMode( argument );
```

Different modes of operation for transmit

<table>
<thead>
<tr>
<th>Mode</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>TINYSEC_AUTH_ONLY</td>
<td>Sends authenticated message</td>
</tr>
<tr>
<td>Encryption</td>
<td>TINYSEC_ENCRPT_AND_AUTH</td>
<td>Sends encrypted and</td>
</tr>
</tbody>
</table>
Table 8: TinySec Operating Modes for transmitting

Receive mode:

```
command result_t setReceiveMode( argument );
```

Different modes of operation for receiving

<table>
<thead>
<tr>
<th>Mode</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>TINYSEC_RECEIVE_AUTHENTICATED</td>
<td>Receives only authenticated or encrypted /authenticated modes</td>
</tr>
<tr>
<td>Disabled</td>
<td>TINYSEC_RECEIVE_CRC</td>
<td>Receives only from disabled mode</td>
</tr>
<tr>
<td>All</td>
<td>TINYSEC_RECEIVE_ANY</td>
<td>Accepts from all modes</td>
</tr>
</tbody>
</table>

Table 9: TinySec Operating Modes for receiving
3) [23] (*Reference from main thesis*) the energy equation gets modified for the heterogeneous networks. This has not been implemented in this work but in a heterogeneous network, there are cluster heads which communicate to base station after collecting data from the ordinary low power sensors. The energy model for the simplest heterogeneous network can be written as follows:

Energy spent in transmission: \( e_d b d^a + e_t b \) and

Energy spent in reception: \( e_t b \)

Energy spent sensing: \( e_s b \)

Definition of terms:

- \( e_d \): Energy spent in reception
- \( e_t \): Energy spent in transmission
- \( e_s \): Energy spent by reception circuitry
- \( e_t b \): Energy spent sensing each bit
- \( b \): Number of bits to be transmitted/ received
- \( d \): Distance of receiver from transmitter
- \( a \): Constant

The model changes for the networks which communicate by hop by hop mode [23].

The above model is limited to communication aspect of the model. The heterogeneous network has overlay sensors which act as cluster heads as per energy available. In every round \( R \) they get to become cluster head once. The expected energy spent every \( R \) round is as follows:
\[ W_o = K_1 + (K_2 + A_1 E[d^a]) C_o(q) \]

\( W_o \) = Energy consumed by overlay sensor

\( q \) = average number of clusters in network

\( K_1, A_1 \) and \( K_2 \) = Constants

\( C_o(q) \) = Expected number of transmission in order to have a successful transmission

Similarly energy spent by normal sensor will be as follows:

\[ W_s = K_3 + (K_4 + A_2 E[s^a]) C_s(n/q) \]

\( W_s \) = Energy consumed by sensor

\( s \) = random variable denoting distance from sensor to cluster head

\( K_3, A_2 \) and \( K_4 \) = Constants

\( C_s(n/q) \) = Expected number of transmission in order to have a successful transmission within the cluster. This depends on the MAC function used in the network.

Let \( Z = d^2 \) then:

\[ E[Z] = E[d^2] \]

In heterogeneous system the energy has to be balanced between the ordinary sensors and the overlay sensors. One way is to balance the energy. Let \( \alpha \) be the energy
allocated to the overlay sensors and $\beta$ be the energy allocated to the normal sensors then:

$$\alpha \quad \beta$$

$$\frac{K_1 + (K_2 + A_1 E[d]) C_0(q)}{K_3 + (K_4 + A_2 E[s]) C_s(n/q)}$$

Hence the number of rounds the overlay and normal sensors should last is as follows:

$$R(\beta)$$

$$R_s = \frac{K_3 + (K_4 + A_2 E[s]) C_s(n/q)}{K_3 + (K_4 + A_2 E[s]) C_s(n/q)}$$