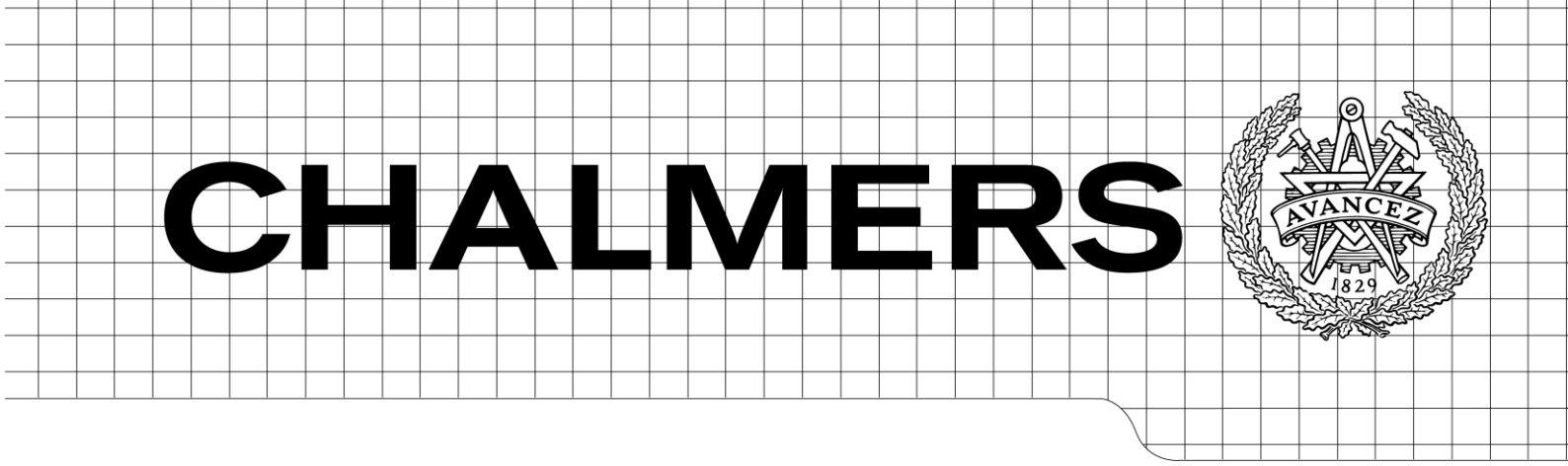


# Asymmetric-Key Cryptography for Contiki

*Master of Science Thesis in the Programme Networks and Distributed Systems*

QAMAR TOHEED
HASSAN RAZI

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, July 2010

Asymmetric-Key Cryptography for Contiki

QAMAR TOHEED
HASSAN RAZI



Examiner: PHILIPPAS TSIGAS

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2010

# Abstract

Wireless Sensor Networks have become a core component in much diverse application range which extends from just a forest temperature monitoring to monitoring in many power plants. With this increased dependency on WSN and its association to current internet, hardened security primitives are required to ensure the correct behaviour of sensor nodes on its own and as a whole network. Public-Key Cryptography was considered too expensive for WSN but it all changed due to the advancements in software and hardware prototypes.

In this thesis different pubic key cryptographic approaches have been analysed, that can be used with Contiki. Contiki is a new but popular operating system used in WSN. Using public key cryptography in wireless sensor networks has its own negative aspects like more energy utilization, requirement of more RAM and ROM space. To investigate the feasibility of public key cryptography, different cryptographic libraries were analysed, out of them two libraries, LibtomCrypt & Relic were selected for evaluation. After a methodical review and code reduction these libraries were ported to Contiki, the code reduction was carried out to minimize the use of different resources by these libraries while running on top of Contiki. For evaluation of these libraries with Contiki, MSB430 mote and simulation based Tmote Sky is used. Results have shown that public key cryptography is possible, and fewer efforts are required to use it with Contiki. Factors like processor speed and RAM size lead to better results in case of such integrations. It is also observed that use of many mathematical optimizations provided with these algorithms can significantly help in performance increase.

# ACKNOWLEDGEMENTS

We would like to express our deepest gratitude to supervisor Prof. Philippas Tsigas at Chalmers University for offering us such an interesting thesis. The comments and suggestions given by him have helped a lot to improve this thesis work.

Many special thanks to friends especially Shahid Nawaz who kept us motivated and provided a moral support.

We are also grateful to contributors of Contiki development forum who have helped us in understanding many new concepts and overcoming implementation problems faced.

Finally we both are deeply grateful to our parents and families for their continuous support, love, guidance and encouragement throughout all the thesis work. We would like to dedicate our work to them.

# Table of Contents

List of Figures



List of Tables

Chapter 1

# Introduction

In this chapter the basic description about the master's thesis project "Asymmetric-Key Cryptography for Contiki" is presented. This includes project overview, scope, limitations and basic details about implementation technique used throughout this thesis.

## 1.1. Background

Wireless Sensor Networks are nowadays becoming a regular component in our daily life for ease of many operations. It can be seen that they are now deployed in nearly every field of computer science research. Many areas in science and technology are adapting to use wireless sensor networks for analysis of data. This includes its usage in large areas as well in small vicinities for a real time gathering of data. Due to these advancements in the use of WSN, more and more critical and sensitive operations of industry and military are becoming part of sensor networks. With such high dependency of large WSN for a complete picture of any sort of operation, security becomes a major concern. If some intruder can access the data and/or tamper with it, then it can result in any undesirable results.

Security has been a major concern in all modern communications. Now with the increasing use of wireless sensor networks in many real time applications for data gathering, security has become a vital ingredient and its continual need in this technology is emerging as a big concern. Embedded Systems like WSN need specailly designed & written operating systems due to their limited resources. There are some famous OS specially written with WSN in perspective, these include TinyOS [1], Contiki [2] and MANTIS [3]. Contiki is a new operating system as compared to most of the others operating systems in this field. During its six years of development it has become the second major operating system by the contribution of large community of developers. Due to the use of C language for its development, many advanced and light components are now becoming part of it such as *uIP*v4 [4] & *uIP*v6 [5] network stacks.

Although the speed of development in Contiki is very rapid and up-to-date, but it still lacks the use of security primitives for secure communication between sensor nodes. Till now there is only one Security architecture "ContikiSec" [6] proposed for this operating system. ContikiSec has presented a very detailed analysis with regard to use of different symmetric ciphers and their effect on scarce resources of sensor nodes. A framework is presented in ContikiSec which can be used by Contiki to communicate securely using different security primitive requirements.

The symmetric cipher based security cannot be used alone as it cannot solve some security problems like inability to handle node compromise and time synchronization issues.These issues are resolved by use of two major security primitives, key management and authentication. The major reason for not promoting such additions in Contiki till now, is its extensive use of resources mostly battery power and volatile memory. However with

growing use of Contiki in WSN such requirements need more attention now. Once Contiki is able to integrate ContikiSec along with key management and authentication, then it can harden the security of data communication between sensor nodes.

## 1.2. Objectives

The main objective of this thesis work is to explore the feasibility of public-key cryptography implemetation in Contiki. This includes the integration of public key based cryptographic operaions along with regular desig of Contiki. It should be able to provide a proof of concept for embedding additional security parameters in security layer of Contik, which can be a part of its future designs.

The proposed solution should be able to provide developers and administrators about the details and suggestions, relevant to addition of security primitives from asymmetric cryptography. The resultant solutions should be tested on some sensor network for their concreteness in order to access for their future deployment. There should be a discussion with sound results about the consequences if such integration is adopted by Contiki. All complications associated in any integration should be listed and explained in detail.

## 1.3. Problem Description

Once these objectives are analyzed, some questions become very prominent, these can be then formulated into problems. By solving these problems project goals can be acheived.

**Security:** Is it required to have such security features to be added into Contiki? What kind of scenario will require such usage?

**Implementation:** Is it possible to implement such advanced cryptographic functions for Contiki? Can such implementation be power efficient at reasonable cost of resources? How Contiki will cope with such integration with reference to its share of resources?

## 1.4. Limitations

a. The only platform available for this project is based on 16-bit RISC MSP430 [7] processor. It cannot be tested on any 8-bit or 32-bit sensor nodes due to non-availability. However it is assured that this work will be applicable on all other platforms as well.
b. The implementations used in this project are not specifically designed for MSB430 [8] or Tmote Sky. One of these libraries has assembly code for MSP430 based processors but it is not yet released in free version.
c. Manually optimizing cryptographic code is a complex process and it requires in-depth knowledge of programming and mathematics involved. Only GCC [9] based compiler optimizations were used in this project.
d. The hardness of any cryptographic protocol is based on a strong pseudorandom number generator. To reduce the load on Contiki system built-in random function is used.

e. In this project an important limitation is unavailability and no support of debugger. Contiki has developed a debugger called MSPSim [10] for Tmote Sky [11] but it is not available for MSB430 at this time.
f. The performance of programs is only evaluated on two platforms. Firstly on real MSB430 sensor node and secondly Cooja simulated Tmote Sky.
g. Network utilization and bandwidth related issues were not focused in this project. The main task is to analyze cryptographic operations on a single sensor node.

## 1.5. Method

The project work is organized in several steps. It started from theoretical studies of Contiki and cryptographic algorithms. A search for suitable cryptographic libraries present in open source community was also started in parallel. As many things were new and under development in case of Contiki, therefore scope was reviewed and redesigned after acquiring many intermediate goals. Similarly theoretical studies were reused in the intermediate and final stage of the project for verification and review respectively. Implementation phase consisted of the following steps.

1) Understanding whole library and thorough analysis of PK cryptographic parts included in it.
2) Take out minimum code and test it for suitable usage.
3) Port the program to Contiki (Cooja [12], MSB430).
4) Redesign program to remove any additional functionality or resource usage.

The above stated algorithm was used for both cryptographic libraries as well as for mathematical portions, that are used with them. Once it was assured that the code is now in most optimal form then different tests were performed for detail analysis. In final stage these results were analyzed and compared for performance merits suitable expecially for Contiki. This included suggestions and improvements for a balanced cryptosystem intended for Contiki.

## 1.6. Structure

This report is organized in such a way that the reader will understand every detail which can help to understand efficiently about subsequent chapters. Next chapter describes the basic details about wireless sensor networks, their growth and different security frameworks. In addition security properties required in diverse scenarios of WSN and attacks are explored. After this an overview of public key cryptography techniques that are commonly used is provided. Till this point reader has acquired all necessary information required to understand this project, then all the details and outcomes of the project are presented. In chapter 5 all the relevant information and analysis about libraries that are used or reviewed for this project are provided. In Chapter 6 measurement methodology used to evaluate and analyze security primitives are elaborated. The results and the motivation behind such results are included in Chapter 7. After the complete analysis a clear picture about the public key cryptography recommendations suitable for Contiki are documented in Chapter 8. Finally Chapter 9 includes conclusion and some future work recommendations.

Chapter 2

# Background

## 2.1. Wireless Sensor Networks (WSN)

Wireless sensor networks consist of small nodes also called motes that monitor physical or environmental conditions around them such as temperature, sound, vibration etc , process data, and communicate through wireless links [13]. A wireless sensor network (WSN) generally consists of a basestation (or "gateway"), which holds the ability to communicate with a number of wireless sensors present nearby by use of a radio link. Once the data is collected by some intermediate node, it is then compressed, and transmitted to the gateway directly or, if not directly connected then uses other wireless sensor nodes to forward data to the gateway. Once this data reaches the base-station then it is presented to the system by the gateway connection [14].
Wireless Sensor Networks are widely used these days and are very popular in research for use of embedded systems in our daily life. WSN's are used in applications involving monitoring, tracking, or controlling such as habitat monitoring, robotic toys, battlefield monitoring, packet insertion [15], traffic monitoring, object tracking and nuclear reactor control.

## 2.2. Security Principles for WSN:

The basic security primitives required for wireless sensor networks are listed below [16].

### Data Confidentiality:

In WSN, data confidentiality is very important because it ensures that only authorized nodes can get access to the data. The basic purpose of confidentiality is to ensure that data transferred from one node to other node is not understandable from any intermediate node or unauthorized parties. This is acheived by use of symmetric key cryptography. The sender node and receiver node use a predefined secret key, or negotiate for a shared secret key. The data is then encrypted/decrypted by use of such key.

### Data Authentication:

It is important in WSN to ensure that the data is originated from the right source, in other words an intruder cannot insert false messages or data into the network. Asymmetric cryptography has the ability to create signatures, which can only be generated by a sender by use of some special information. In order to acheive this property signatures are prepared and used by sender. The receiver verifies the signature and hence confirms that the data is sent from the authorized node.

### Data Integrity:

Data integrity ensures that the data is not altered by unauthorized parties during transmission. It can be acheived by use of message integrity code (MIC) or a checksum added to each packet. MIC can detect message altering caused by accidental transmission errors as well as malicious altering. Checksum on the other hand can only detect accidental transmission errors.

### Data freshness:

The messages transmitted should not repeat at receiving end, likewise a node should not receive two identical messages in certain time range. The main goal is to ensure that data is recent, fresh and, it is not used by any intruder who wants to learn some symmetric key used in the communication. There are two kinds of freshness techniques. Weak Freshness is typically achieved by use of partial message ordering, but without delay information. While strong freshness is achieved by use of complete ordering and delay estimation.

### Availability:

The whole system or even a single node in a WSN should be available, and capable to provide its services when-ever required. When modifications are applied to WSN, then it can affect the availibility of the node to a great extent. It can be said that when security proprties are applied to WSN then it doesn't only effects the operations of the network, but it is also important that the availability of the whole network should be preserved as well.

## 2.3. Security Threats in WSN

In WSN, traditional security techniques cannot be applied directly. WSNs are growing rapidly in many aspects, therefore security mechanism for WSN should be updated as well. WSN operates in outdoor environment where they are unattended, so security should be considered in the design phase [17]. Security in WSN can be categorized into two types, operational and information security. Operational security means that the network should be able to provide services even if some of its components fail or become compromised. Information security means that the network should not disclose any secret information, plus it should guarantee integrity and authenticity of the messages. Security is a critical requirement for WSN, the data and node should be protected against attacks like eavesdropping, tempering, DOS attack [15] etc. When designing a security model for WSN, all these type of attacks should be considered and security should be defined on different layers to ensure high degree to reliability.

## 2.4. WSN Adversary

An adversary is a person or another entity that attempts to cause harm to the network. The reason to do such can vary from just a denial of service attack to larger extent of unauthorized access to the WSN, resulting in loss of many security properties related to important data [18].

There are two general types of adversary attacks on any network.

a) Active adversary technique involves with the tampering to data while it is being under process in the network. This includes deletion, alteration and addition to the data. This result in violation of core security primitive's like data integrity, confidentiality and authentication.
b) Passive attacks only monitor the communication link and listens to every piece of information that passes through. After this the adversary use different techniques in offline mode, to either decrypt a saved encrypted data or moves to active mode to guess secret key. Here the main aim of adversary is to attack the confidentiality thus he can view the original data from encrypted data.

In WSN the adversary classifications are broadened to four more classes.

i) Mote-Class Attacker: Such adversary has access to a few nodes which have capabilities similar to the nodes that are deployed in the network.
ii) Laptop-Class Attacker: This type of adversary has access to a much powerful device like laptop as compared to sensor nodes. This provides gain to the adversary as he has access to larger resources at his disposal, and hence he can use larger set of techniques for his purpose.
iii) Insider: In some cases adversary is able to compromise or capture some internal nodes therefore he can become a part of network easily. Once it is done then he can exploit whole network for his own purpose.
iv) Outsider: Such adversary has no special access to this network.

## 2.5. Types of attacks

In this section an overview about some simple and common attacks on different layers is presented.

### Physical Layer

**Jamming.** Jamming is a type of physical layer attack in which the radio frequencies of WSN are disturbed by use of interference. This can result in altering some important radio parameters like collision rate, bad frame rate and RSSI level [19].

**Tampering.** In such attack the node is physically compromised. This is mostly done by capturing a node from the WSN field. The attacker can collect all information from the node and try to recover beneficial infromation. An advanced attacker can recover, reprogram and redeploy it in the field to attack the whole WSN [20].

### Data-link Layer

**Collision.** The packets can be disrupted by altering the transmission octet, this can result in checksum mismatch or back-off in some mac protocols. Attacker listens on the communcation medium and try to guess the expected time of message transmission. Then he notices the time and send a message at the same time when a proper message is started. So these two messages collide in the wirelss medium and results in an incorrect message for the receiver.

**Exhaustion.** The batteries of sensor nodes can be exhausted if the network face continuous collisions and back-off in MAC protocols. This results in degradation of avaialability on a large scale in the WSN.

### Network Layer

**Selective forwarding.** Certain malicious nodes can refuse to forward some messages and just drop them. This can result in delay and bandwidth degradation in the whole WSN.

**Sinkhole.** In such attack the routing information can be altered in different nodes connected to the compromised node. An adversary attracts a node and provides it with wrong routing information. Senosor nodes will expect that this node provides an efficient route to the sink node but in reality compromised node in most cases drop, or alter the received messages [21].

**Sybil attack.** A single node creates its own multiple identities and presents it to other nodes in the network. This will result in removal of all original neighbours from the table of active sensor nodes in the routing table. In some cases if the transmission quality of compromised node(s) is good, then it can also remove the original sink node from node's routing table as well [22].

**Hello flood.** A laptop class attackers broadcasts messages with powerful signals that the nodes in the network think that the attacker is in the network and sending normal startup packets used in routing protocols [21].

### Transport Layer

**Flooding.** The attacker can exhaust important resources like battery of the victim by sending the victim many connection establishment requests. This will ultimaltely result in denial of service attack on the node.

**Desynchronization.** Request for retransmission of missed frames can be made by repeatedly forcing messages into the network which carry sequence numbers to one or both end points.

## 2.6. Existing Security Frameworks in WSN

Security was not considered as a main focus in WSN till few years back. As WSN have started to mark their presence in more cirtical applications, hence more and more focus is transferred to securing them in every possible way so that they cannot be exploited further. Traditional Security techniques cannot be applied in WSN due to two main reasons.

a) In-network processing of large data to reduced aggregated information.
b) Important resources incase of WSN are very limited.

There are only few security frameworks present in WSN till now. These include SPINS [23], TinySec, SenSec, MiniSec, TinyECC and ContikiSec. An overview of some important frameworks is presented below.

## TinySec [24]:

TinySec is the first fully functional link layer security architecture presented for wireless sensor networks and it has been a part of TinyOS. The biggest motivation for developing link-layer security architecture was to detect unauthorized packets when they are first injected into the network to save energy and bandwidth. The security goals acheived by use of TinySec are confidentiality, message integrity & access control.

## TinyECC [25]:

Tiny ECC is an ECC based PKC software package that can be configured and easily integrated into sensor network applications. It is specifically designed for use in TinyOS. TinyECC provides special optimization switches which can be turned on or off by the programmer according to the programmer's need. By configuring different optimization switches execution time and resource consumption can be controlled. TinyECC has support of digital signature scheme (ECDSA), a key exchange protocol (ECDH), and public key encryption scheme (ECIES) which are different ECC schemes.

## ContikiSec [16]:

ContikiSec provides some necessary security primitives on link layer of Contiki. WSN supports many different types of applications therefore ContikiSec is designed to be more flexible. Three different security levels confidentiality-only (ContikiSec-Enc), authentication-only (ContikiSec-Auth), and authentication with encryption (ContikiSec-AE) are provided in ContikiSec. This provides the programmer/administrator a choice to select between these three types of security level according to the application or needs of the WSN.

”ContikiSec has been designed to balance low energy consumption and security while conforming to a small memory footprint” [16]. ContikiSec uses symmetric key encryption algorithms AES, RC5, Skipjack, Triple-DES, Twofish, XTEA. According to the ContikiSec developer, AES is the most appropriate block cipher for the WSNs taking into account memory usage (RAM and ROM), time (Encryption, decryption, and key expansion time), energy consumption and considering trade-off between security and resource consumption.

An overview of all the important frameworks alongwith their relevant information concerning this project discussion is shown below.

| | year | Language used | Security Properites implemented | Algorithms |
|---|---|---|---|---|
| **SPINS** | 2002 | N/A | Data Confidentiality, Integrity, authentication and freshness, authentication | |
| **TinySec** | 2004 | NesC | Access Control, Integrity, Confidentiality, Replay Protection | Skipjack CBC-CS mode |
| **SenSec** | 2005 | NesC | Access Control, Integrity, Confidentiality, Key Management | Skipjack-X CBC-CS mode |
| **MiniSec** | 2007 | NesC | Pre-Deployed symmetric keys, Confidentiality, Replay Protection, authentication | Skipjack OCB mode |
| **TinyECC** | 2007 | NesC | Key Exchange, Public key encryption, Digital signature | ECC SECG-160 |
| **ContikiSec** | 2009 | C | Authentication, Integrity & Confidentiality | AES CBC-CS mode |

**Table 1: Overview of Security Frameworks present in WSN**

## 2.7. Contiki

Contiki is developed by Swedish Institute of computer science and it is a lightweight, open source, highly portable and multitasking operating system used for embedded systems which are highly memory efficient. The memory usage for contiki is about 2kb of RAM and 40Kb of ROM [2].
The first version of contiki was released in 2004 and the latest version 2.4 is released in feb 2010. Contiki is developed in C and is currently used in many microcontrollers like MSP430, AVR, HC 12 and Z80. The biggest advantage of Contiki is that it provides dynamic loading and unloading of applications and services which enhances many resource utilization in sensor networks and the kernel is event driven. Contiki also provides multithreading and is implemented in a separate library, which can be used by an application when needed. By combining multi-threading and event driven kernel contiki is the best operating system for sensor networks to implement asymmetric cryptography.

## 2.8. MSB430

MSB 430 is a hardware platform for sensor networks created by ScaterWeb with its own operating system and it has a microcontroller, external storage, two sensors, radio and LED. MSP430F1612 microcontroller is used in MSB 430 which has 16bit RISC architecture, 5Kb RAM and 55Kb of flash ROM. The clock of MSB430 can be configured dynamically between 1 and 11 MHz by accessing it's software. MSB430 is equipped with Chipcon CC1020 transceiver which has 8.6 dBm max. transmission power and an external LNA (low noise amplifier). It can be operated in 402-470 and 804-940 MHz frequency range and the frequency can be selected differently for transmitting and receiving. MSB430 has two sensors onboard, humidity sensor (Sensirion SHT11) and temperature sensor (Sensirion SHT11). This sensor is

very important for temperature compensation which results in accurate clock frequency and measurements plus it can additionally use a sensor for movement detection, which is three-axis accelerometer MMA7260Q [8].

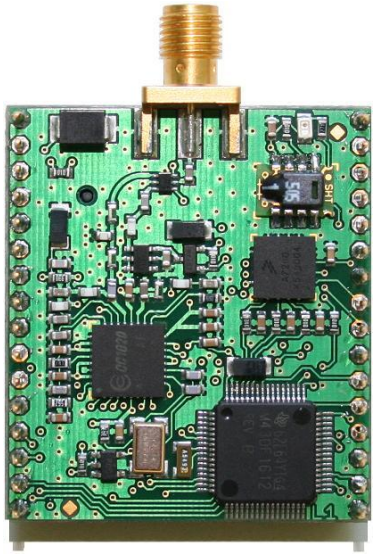

**Figure 1: MSB430**

## 2.9. Tmote Sky

Tmote Sky is an ultra low power IEEE 802.15.4 compliant wireless module for use in sensor networks used in monitoring based applications, and rapid application prototyping. Tmote Sky gets its power by AA batteries but it has a usb port which can be connected to usb port of a computer to utilize power from the computer. The low power usage is of Tmote Sky is possible due to MSP430 F1611 microcontroller which consists of 10Kilobytes RAM and 48Kilobytes flash ROM. This mote also includes the ability of fast wake up from sleep which takes less than 6µs. Its 16 bit RSIC processor enables it to use less power while active and in sleep mode enabling the node to run for years on single pair of AA batteries [26].

Tmote Sky is equipped with Integrated onboard antenna with 50m range indoors / 125m range outdoors, the radio is Chipcon CC2420 which provides reliable wireless communication. There are three types of integrated sensors in Tmote Sky including Humidity, Temperature, and Light sensors. It also provides hardware link-layer encryption and authentication for better security. For external code and data storage Tmote Sky provides ST M25P80 40MHz serial code flash that can hold 1024 kilobytes of data. The flash is decomposed into 16 segments, each 64kB in size.

In this project a simulator COOJA [27] which is sepcially built for Contiki is used to simulate the Tmote Sky.

**Figure 2: Tmote Sky**

Chapter 3:

# Overview of Asymmetric Cryptography

## 3.1. Cryptography

Cryptography is a subject in the field of mathematics that is applied in computer science to ensure the security primitives. The term cryptography which is also referred to cryptology is derived from a combination of two Greek Words, *kryptos* which means ”hidden” and *grafo* stands for ”write”. Cryptography is used for lots of purposes like encryption, data integrity, authentication, asymmetric encryption and digital signatures. There are two types of cryptography techniques symmetric cryptography and asymmetric cryptography. Symmetric cryptography is not used in this project hence it is not discussed here in detail. In this chapter basic introduction about asymmetric cryptography is presented.

## 3.2. Asymmetric Cryptography

Data transferred from one system to another is protected by means of encryption, a shared secret key is used to encrypt and decrypt the data by the sender and receiver respectively. Such encryption is called symmetric key cryptography. There are many symmetric key algorithms which are tested to be very secure, but the biggest problem with this kind of system is that the key has to be shared over a public network and in most cases it is predeployed on each sensor node. Asymmetric key cryptography solves problem which cannot be resolved by use of symmetric-key cryptography. In asymmetric-key cryptography two keys are generated, one private part of the key and the other is called public key part. The private key is kept secret while the public key is made public, the message is encrypted using the public key and the private key is used to decrypt the message. Another wide usage of such technique is utilization of private key to sign the message, while the public key is used to verify the signature at other end. The biggest problem associated with use of asymmetric cryptographic system is that it is slow and expensive. However it adds key management scheme & digital signatures to any network to ensure hardened security.

## 3.3. Asymmetric Cryptography Techniques

In recent years it has been a major challenge for the researchers in the field of WSN to reduce the computational complexity, and minimize memory usage of the traditional asymmetric cryptographic algorithms like Al-Gamal, RSA, DSA and ECC. Among all these algorithms ECC is considered to be most suitable for wireless environments because its memory and resource consumption is least as compared to all others [28].

## RSA [29]:

RSA stands for Rivest, Shamir and Adleman, which are the names of the authors of RSA. RSA is an asymmetric/public key cryptographic algorithm. It is one of the first algorithms (presented in 1977) to be suitable for signing as well as encryption. It is believed to be secure and still is widely used in e-commerce. RSA is considered to be secure due to factorization problem which states that it is very difficult to factorize large number.

RSA consists of three operations namely key generation, encryption and decryption.

***Key generation.***

Key generation process is used to generate a pair of keys i.e. private key, and its public key part. Public key is available for all users while private key part is secret and not provided to any other user.If the RSA keys are not built already then they need to be created when any user wants to communicate with it. Key generation process has a drawback of being slow but this operation is only required at first time or when the keys need to be regenerated. TheKey genartion is composed of following five steps [30].

1) First of all, two large distinct prime numbers *p* and *q* must be generated. Make sure that p!=q
2) The product of above selected prime numbers is computed, let's call this component *n*. It must be large enough so that the numbers *p* and *q* cannot be extracted from it like at least 512 bits i.e. numbers should be greater than $10^{154}$
   $$n=pq$$
3) Compute phi
   $$\Phi = (p-1)(q-1)$$
4) Choose a public exponent e such that $1 < e < \Phi$ . Make sure that gcd $(e, \Phi) = 1$
5) Finally decryption key *d* can be made in such a way that
   $$de \bmod m = 1.$$

Now public key {n,e} and private keys {d} are generated and are ready to be used for encryption and decryption.

***Encryption***

RSA encryption is always done by use of public key. Any person or computer who wants to communicate with the target machine using RSA encryption must first obtain its public key. It can be done by direct communicatuon or by use of any certification authority ”CA”. If the message is greater than ”n” present inside public key then it should be broken down into blocks of messages such that each block is less than n. Now to encrypt message ”m” into ciphertext ”c” following computation is done where ”e” and ”n” are public key components.

$$c = m^e \bmod n$$

***Decryption***

This operation can only be performed by the host of RSA key using its private key. Original message will be extracted from ciphertext by use of following computation.

$$m = c^d \bmod n.$$

***Signing messages***

RSA encryption and decryption are not so much used as compared to RSA digital signatures. Digital Sinatures are always computed by use of private key. This signature can later be verified by any receiving party just by knowledge of host public key.

RSA Digital sinature is never applied to the whole message as it is very slow, several techniques are present to forge such signature. RSA based signature is always computed over the hash of the original message. In order to make a signature the sender node will produce a hash of the message and then raise it to the power of "***dmod n"***, and then attach this to the origianl message. Once the receiving node receives the signed message then same hash function is used to calculate the hash of the message like sender did. After this it will be raised to the power of "***emod n"***. Finallt the resultant value is compared to the received signature appended to the received message. If they are same then this means that the sender of this message holds its private key, and hence the message is not tampered during its transmission.

## 3.4. ECC

Elliptic Curve Cryptography "ECC" was proposed by Niel Koblitz and Victor Miller in 1985. ECC is a public key cryptography approach based on algebraic structure of elliptic curves over finite fields. ECC is emerging as a strong public key crypto system compared to other public key cryptosystems like RSA. RSA is considered to be secure on the basis of assumption, that it is very difficult to find factor of very large prime numbers (Integer Factorization Problem). While ECC is considered secure on basis that it is infeasible to find discrete logarithm of a random elliptic curve element with the knowledge of base point present in public "ECDLP" [31. In conclusion ECC is more secure and provides equivalent security with smaller key sizes which results in faster computation, lower power, memory and bandwidth usage. ECC has the highest strength-per-bit compared to other public key cryptosystems [32], so ECC is considered to be very useful for mobile devices.

The mathematical formula of ECC over the elliptic curve is

$$y^2 = x^3 + ax + b$$

Where x,y,a and b are real numbers and with the condition

$$4a^3 + 27\,b^2 \neq 0.$$

By changing the values of 'a' and 'b' different eliptic curves can be generated. All the points which satisfy the above equation lie on the eliptic curve. Private key is genrated by random number generations while public key is obtained by multiplying the private key with a constant base point G (Scalar multiplication) in the curve. Public key is obtained as a point in this curve. ECC biggest advantage is its small key size, a 160 bit key size of ECC is equivalent in securit to 1024 bit key size of RSA.

There are two basic operations performed in ECC.

a) Point Addition
b) Point Doubling

A simple graphical explanation provided by SANS is shown below [33].

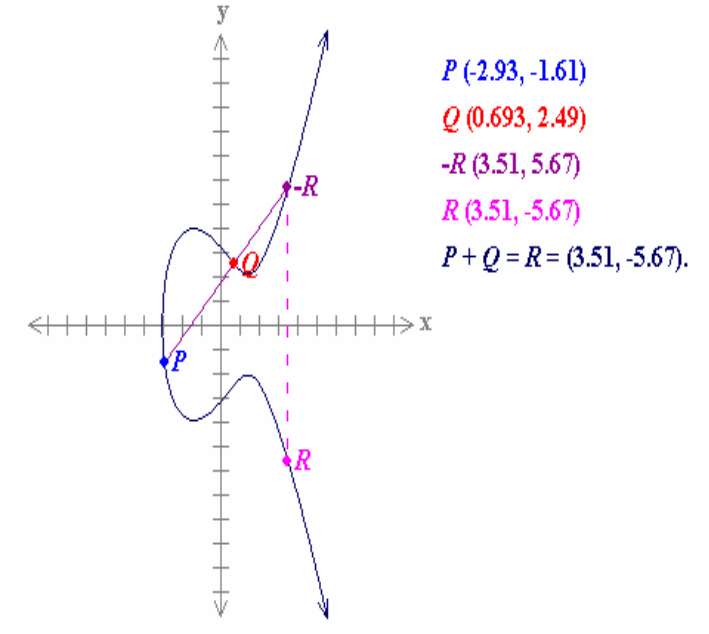


**Figure 3: ECC Point Addition**

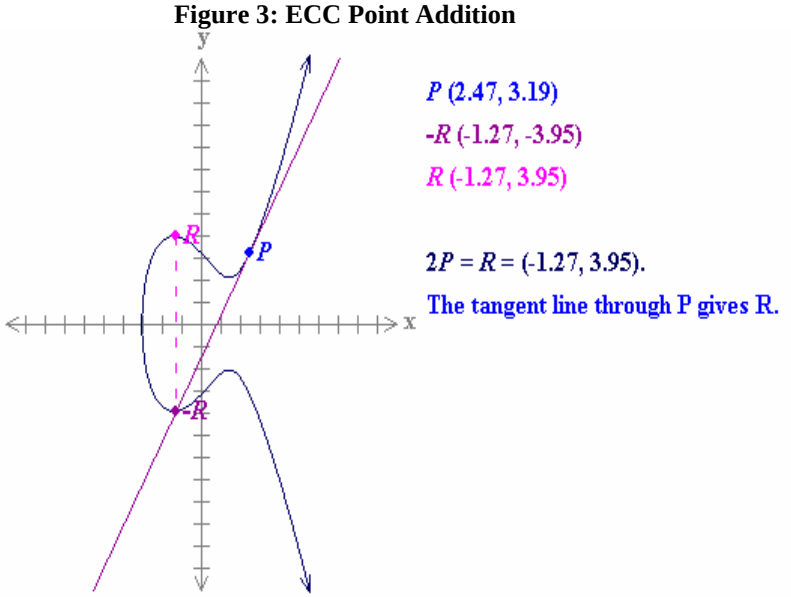


**Figure 4: ECC Point Doubling**

Listed below are some of the popular schemes that use ECC.

## 3.5. ECDH

The Elliptic Curve Diffie-Hellman (ECDH) key agreement protocol provides ability to two users, each holding a public-private key pair to create a shared secret agreement over a public channel. The biggest advantage of such technique is its ability to be used in non-secure and public medium [34].

It is a version of the Diffie–Hellman protocol to be used with elliptic curve cryptography. The shared agreement between two nodes can be directly used as a key or can be used to make another key, the key can be used to encrypt messages by a symmetric algorithm.

***ECDH Mathematics***

ECDH relies on two public parameters, 'p' is a large prime number and parameter 'g' which is an integer that is less than p. These parameters can be exchanged over an unsecure channel, after receiving the two parameters 'p' and 'g', both parties select private integers 'a' and 'b' respectively. These values are called the private keys of both sides. Then both parties create public keys based upon the public parameter and their corresponding private keys, the private keys are asymmetric because they do not match. One private key is created based upon $g^a \bmod p$ and other is created based upon $g^b \bmod p$. In all this process the secret part only consists of a, b and it is known that $(g^b)^a = (g^a)^b$ , all the other values are public [34].

Anyone who eavesdrops will get 'p', 'g' but it is computationally infeasible to generate shared secret agreement from the public values without knowing both parties private keys. So this protocol is secure because no one can drive private key of the both parties unless the Elliptic Curve Discrete Logarithm Problem is solved. ECDH uses elliptic curves instead of logarithmic curves, due to this ECDH is able to achieve same degree of security by using shorter keys.

***ECDH limitation***

The ECDH key exchange protocol is not secure because it does not prevent man in the middle attack. The main reason for such inability in ECDH is no pre-authentication of both nodes. An eavesdropper can easily get a public key of one and send its own instead of the original to the recipient. By doins so, the attacker can decrypt messages, change the messages, re-encrypt them with its own keys, and then send them to the recipients. This problem can be solved if both parties use digital signatures to sign public keys before exchanging them.

## 3.6. ECDSA

Eliptic curve digital signature "ECDSA" is a variant of Digital Signature Algorithm "DSA", a process of calculating digital signature but using elliptic curves. The signature has the property that it can only be created by one (one who owns the private key) but can be verified by many (public key of sender is know to them) . A security level of 80 bits, means that the attacker needs to generate $2^{80}$ signatures so that he can find the key, which is equivalent to a DSA keysize of 1024, but in the case of ECDSA the key is only 160 bits long. The process of ECDSA is consists of three steps:

***ECDSA key generation***

Key generation is required to make a key pair of public and private key. Key generation for ECC based ECDSA is done in 5 steps [33].

1. Select an elliptic curve *E* over a finite field, say *GF(p)*. The number of points on *E*

should be divisible by a large prime *n*.
2. Select a point $P = (x,y) \in GF(p)$ of order *n*.
3. Select an unpredictable integer *d* in the range [*1, n-1*]. *d* will act as the private key
4. Compute *Q=dP*
5. The user's public key is *(E, P, n, Q)*.

***ECDSA signature generation***

To compute a signature on message "m" following procedure is used [33].

1) Choose a random integer *k* in the range 1 < k < *n-1*].
2) Compute $(x_1, y_1) = kP = k(x,y)$, and set
$$r = x_1 (\text{mod } n)$$
If *r=0* then go back to step 1.

3) Compute $s = k^{-1}(h(m) + dr) \text{ mod } n$, where *h* is the hash value obtained from a suitable hash algorithm (for example, the Secure Hash Algorithm, SHA-1).

4) If *s=0* go to step 1.

5) The signature to be included in the message *m* is the pair of integers (*r, s)*.

***ECDSA signature verification***

Once the message is received by the other side then the receiver will first parse out r and s from the trailing part of the message. Following computations are performed on the rcceived r and s to verify that message sender's identity [33].

1) Obtain an authentic copy of the public key *(E, P, n, Q)*.
2) Verify that *r* and *s* are integers in the range [*1, n-1*].
3) Compute
$w = s^{-1} \text{ mod } n$ and $h(m)$.

where h is the same hash function used by sender to calculate the hash of the message.

4) Compute
$u_1 = h(m).w \text{ mod } n$
and
$u_2 = r.w \text{ mod } n.$
5) Compute
$u_1 P + u_2 Q = (x0 , y0)$
and
$v = x0 \text{ mod } n.$
6) Accept the signature if and only if *v=r.*

## 3.7. Summary

The common perception about public key cryptography is that it is complex, slow and consumes lot of energy and memory, so it is not suitable for wireless sensor networks as they have access to low power and low memory. But it is possible to design a public key encryption architecture with low energy and memory consumption by selecting right algorithms and parameters . To prove this someresults provided by [31] are shown below in Table 2. They have tested both techniques on mobile processors and proved by results that ECC operations are much more feasible and efficient as compared to RSA in limited resources. RSA seems to be more efficient but it is only in the case of decryption, which is also reduced to large degree is key size is increase. Secondly RSA consumes a large amount of memory for its operations, while on ther hand ECC provides same level of security with much less memory consumption.

| Level of Security | Key Size | Decryption Time (seconds) | Verification Time (seconds) |
|---|---|---|---|
| 80 | RSA-1024 | 2.694 | 0.191 |
| | ECC-160 | 0.765 | 1.042 |
| 112 | RSA-2048 | 14.734 | 0.665 |
| | ECC-224 | 1.187 | 1.626 |
| 128 | RSA-3096 | 44.274 | 1.378 |
| | ECC-256 | 1.375 | 1.905 |

Table 2: RSA and ECC performance comparison

Chapter 4

# Library Details

In this chapter detailed analysis and review of different libraries is presented. This chapter can help future developers to understand the merits that need attention for understanding, or implementing cryptographic functions with context to Contiki development. At the end of this chapter experiences with each library are presented, that determined whether library suits our requirements or not concerning this project.

## 4.1. Selection Criteria

After completion of initial research study the next step was to look for cryptographic libraries present in open source domain. Many factors were important for selection of library that can be used in our project. The main concentration from the start of project emphasized on finding a library that is freely available, easy to use and holds a small memory footprint. A cryptographic library especially prepared with embedded system's perspective will be more suitable for selection. Another most important selection criteria is programming language, the library should be implemented in C language.

Contiki is completely written in C language hence cryptographic library written in C language can only be integrated with it. It is found that there are very less cryptographic libraries written in C language as compared to C++ and Java. Currently only 2 open source libraries specifically written for embedded systems were located in free domain. Nearly all of the specially designed cryptographic implementations are for commercial use and hence their source code is not freely available.

Symmetric Cryptography is much simpler to implement and it does not require any complex mathematical computation code to be added for its execution. However Asymmetric cryptography is much more compound when it comes to its implementation. In the case of asymmetric cryptography very large mathematical computations needed to be performed which cannot be done by simple mathematical code included with compilers. Due to this reason developers implement mathematical parts separately from the cryptography based algorithmic details. Details are presented below in two different categories so that it is easy to understand different parts with their context.

## 4.2. Cryptographic libraries

### LibTomCrypt

LibTomCrypt [35] is a famous and renowned cryptographic library when it comes to open source community and C language based implementation. It has been ported to many Linux platforms like Redhat, Debian and Gentoo due to its simple and reliable design. The biggest

advantage concerning this project is the flexibility of this library. LibTomCrypt gives developers an independence to select from three mathematical libraries (TomFastMath [36], LibTomMath [37] & GnuMP). Before minimizing code of this library in this case, mathematical program need to be prepared from any of these mathematical libraries. This will be discussed below in mathematical library details.

After understanding the library behavior in detail, library code is reduced and prepared to be used on a normal PC. Once it is assured that previous procedure is done rightly and program is performing in a correct behavior, then again code is optimized to become minimal so that it can easily reside in a sensor node. By the end of the project the program reduced nearly all the unnecessary code from the library. This could not have been done at early stages because the nature of the change in requirements were not sure as they can be modified later in the project.

Pros: Easy to understand and modify
Much more flexible to add or remove different parts.
Mathematical library can be easily changed.

Cons: Less functionality provided as compared to Relic.
Library is old and not so regularly updated.
Basic co-ordinate system and Barret Reduction are not present in current versions.

## Relic

Relic [38] is a cryptographic toolkit specially written for use in sensor networks. It is very flexible library and provides a large set of modern cryptographic functions. There is no documentation resources available on this library for new users as it is very new and under continuous development. This resulted in problems to understand it at the start, but once any user understands all the details then it is very easy to use. The to work on this library was initiated with the same procedure as was followed in LibTomCrypt. In first phase all mathematical computation code was extracted, prepared a simple program of it and performed the tests provided with it. Then same work was cariied out for cryptographic library part and its integration with mathematical program. An important difference compared to LibTomCrypt is Relic had much more functionality and features when it comes to algorithmic details.

Pros: Support for 2 mathematic libraries (Relic-Easy and GnuMP)
New library with many new algorithms included.
Support for 3 Memory allocation methods (Static, Stack & Dynamic)
Multithreading support included.
Support for testing and benchmarking of different components.

Cons: Documentation details are very less.
Some configuration details are tricky and difficult to understand.

## 4.3. Mathematical libraries

### TomFastMath

TomFastMath library is also developed by the same developer who wrote LibTomCrypt and LibTomMath. The main aim of developer was to write a mathematical library that can increase performance, speed and can compute results faster. It is not flexible like LibTomMath but it includes inline assembly fragments for achieving quicker results. The biggest difference between TomFastMath and LibTomMath is use of fix precision integers by TomFastMath while LibTomMath uses multi precision integers.

Montgomery modular reduction is preferred over Barret Reduction technique for implementation in this library. Most importantly use of Barret Reduction is also not favorable for this project as nearly all project work was carried out by use of Projective co-ordinate System. Barret reduction provides better results only for Basic Co-ordinate system in ECC. The developer has compared and showed that TomFastMath outperforms LibTomMath if it is used by LibTomCrypt.

### LibTomMath

LibTomMath is a very comprehensive and detailed library written with the focus to understand the mathematical algorithms to be used with asymmetric cryptography. It was prepared to compete with GnuMP on many grounds. It is not designed to be more efficident and dynamic but not as fast as TomFastMath, however it provides much larger mathematical library to support many additional functions.

### Relic

Relic toolkit provides users flexibility to choose between Easy C implementation written by its developer and GnuMP as well. Currently GnuMP cannot be used with Contiki as it is not yet build or developed for Contiki, so this bounds to use only developer's written mathematical library. The developer of this library has stated that the results can be much better if GnuMP is used. Relic math library offers a large set of functions which provides user independence to choose from any of them depending upon different requirements.

## 4.4. Experiences learned with Cryptographic libraries

### LibTomCrypt

LibTomCrypt uses dynamic memory allocation for all ecc-point variables. Complete stack based implementations were built as well. Still heap allocation is recommended in this project due to memory limitations, especially in case of low memory based sensor nodes like MSB430 it can perform its operations properly. If malloc is unable to allocate further memory then it can terminate properly after prompting an error. In the finalized version a heterogeneous approach is used for the performance evaluation . An important point in such

implementation is use of dynamic memory for ecc points only, while all other variables are stack based.

In this project TomFastMath is preferred over LibTomMath due to two main reasons. Firstly it is much more efficient and secondly due to fragmentation issues related to use of LibTomMath. LibTomMath has the ability to grow and shrink the size of multi precision integers which is a big advantage when it comes to resource management but in case of sensor networks it cannot be handled very efficiently. Contiki has not included use of reallocation function ”realloc” in memory allocation functions. In our view it is done rightly as in case of sensor nodes with very low RAM like MSB430, it is not possible to efficiently use same memory space. An alternative used in this project to overcome problem was to free memory space manually, and then reallocating from free space from the end. This solution still resulted in memory fragmentation issue, this eventually leads to reduced RAM utilization and not able to complete whole operations. Hence TomFastMath was selected which uses fixed size for multi precision integer. This looks to have some wastage in ram resource but it is very less memory wastage as compared to LibTomMath, similarly no fragmentation issues arrive in such case. In order to eliminate this small wastage in RAM resources using TomFastMath, it is configured to use only the estimated required size of its fix point. Hence for different keysize of ECC algorithms it uses different fix point variable size required by the ecc points.

## Relic

Relic uses three different types of memory allocation techniques. In this library stack allocation is preferred as Cooja simulator can be used to see the stack usage during all operations. Secondly stack allocation also requires very less code size as compared to dynamic memory allocation in case of Relic. Relic provides different set of techniques for all functions required to perform cryptographic operations. As the code documentation was not available so the basic settings of all the functions were used initially. The results obtained were not so comparable to LibTomCrypt results. After discussions with developer of Relic it was noticed that no modular reduction was used by us in early stages. Once code was modified to use Montgomery reduction then the execution time was dramatically reduced. The results prepared by use of Montgomery reduction are showed in chapter 7. SHA1 implementation provided with Relic was also not easy to integrate but after code review it was noticed that it is using the same 8-bit SHA1 for its hashing mechanism so additional layer of configuration was reduced,and simple 8-bit SHA1 code was plugged with Relic. This reduced flexibility but reduced the ROM utilization by this part. However as SHA1 is becoming less secure SHA 256 can be plugged in future for increased security.

## PolarSSL

PolarSSL is a simple and elegant library especially designed for embedded systems. It is written in C language which suits our integration requirements. ECC is not implemented in this library so it cannot be used in our project. In the start of project this library was used for RSA behavior learning and it was found to be very easy to work with. In future if they provide ECC with their library then it can be very good experience and will benefit Contiki.

## Bitint

BitInt [39] is large number mathematical library which is especially written for Contiki. The biggest advantage of this library is the use of memb. Memb [40] is memory block management system completely written for use in Contiki. The developer of this library has written all basic mathematical library functions that are required by any public key cryptography algorithm. It has been tested and verified by developer for its concreteness and use in Contiki. If any developer want to write a complete new ECC library especially designed for Contiki then BitInt can reduce a lot of load and developer just needs to implement cryptographic functions.

Below a simple table is presented which states the options which were used by the libraries. There are some optimizations that are not present or provided in the library which effects on the results as well.

| | LTCTFM | Relic |
|---|---|---|
| *Simple Multiplication* | Included | Included |
| *Sliding Window Multiplication* | Included | Not Available |
| *Basic Co-ordinates* | Not Available | Included |
| *Projective Co-ordinates* | Included | Included |
| *Simple Verification* | Included | Included |
| *Shamir's Trick based ver.* | Included | Included |
| *SECG-160 tested on* | MSB430 , Tmote | MSB430 , Tmote |
| *NIST-192 tested on* | MSB430 , Tmote | Tmote |

**Table 3: ECC optimization details in each library**

Chapter 5

# Measurement Methodology

Wireless Sensor networks have very limited resources, due to these constraints many important factors need to be analysed before deploying in a field. In this section software based methods, used to measure different performance merits of the implementation phase are presented.

## 5.1. Execution Time

There are two techniques used to measure the time consumed by software to perform various operations. The most accurate method is to use high precision oscilloscope to check the data output pin. This method seems to be more complex and many external factors need like margin error of measuring tool needed to be considered. The second method is to use real-time timers present in Contiki. By using these real-time timers higher accuracy in execution time is visible.

Contiki supports use of real-time timers and event based timers. “A real-time timer does not post an event when it expires, but invokes a function (from within the hardware timer’s interrupt service routine).This provides better control over scheduling than an event timer” [41].

Real-time timer in Contiki is hardware dependent. This gives a choice to select any of two clocks present in the sensor node. Main Clock (MCLK) provides a maximum resolution of 0.4069 microseconds if the processor cycle is executed at 2.4576 MHz. This is very accurate but it cannot be used as this clock is disabled in low power modes, while Contiki runs in low power mode 1out of 5 low power modes available. The second clock option is to use auxillary clock ACLK which runs by use of a crystal oscillator with frequency rate of 32768 Hz. This configuration was preferred and selected for all the measurements in the project.

In the initial phase RTIMER was not performing proper and the results showed abnormal values. After analysis it was known that it is a 16-bit counter and hence it overflows so it is not usable here to get the proper execution time. So decision was made to use the clock library present inside Contiki. An advantage of using this library was its hardware specific utilization. clock_init() is initialized at the start-up and it configures the hardware timers and interrupts. clock_time() is a function that returns the increasing tick counter. So to measure the performance speed initial clock value can be stored before code execution and final clock value once code finishes execution.

```
start=clock_time();
```

```
ecc_make_key (); // Any target function for time consumption

diff=clock_time() - start;

num_seconds=(double)diff/CLOCK_SECOND;
```

The above written code shows the method for time measurements. Before RTIMER was patched to be used as 32-bit counter clock library was used to measure the execution time. Once RTIMER was corrected to handle long time durations by the Contiki developers then real-timer "RTIMER" was used for speed measurements. After comparison it was seen that clock_time_t based variable shows a little more time e.g. for near 9 seconds computation it delayed a difference of 200 milliseconds, however the results using clock_time_t were very consistent on MSB430 mote and Cooja simulation for Tmote Sky.

```
t1=RTIMER_NOW();
    ecc_make_key ();
t2=RTIMER_NOW();
printf("Ticks=  %u\n", t2-t1);
```

The above written code is used for performance analysis after RTIMER was patched to work at 32-bit values.

## 5.2. ROM

ROM is a very limited resource in WSN and hence a very important parameter that needs to be fulfilled for any application to be able to reside on the node. Most of the freely available cryptographic libraries have not taken this constraint into consideration. Additionally there are only few free libraries present that are designed to be used in embedded devices. Once the library is chosen then it cannot be ported as a whole package into Contiki so it needs to be reduced and create a minimal application that can reside in the node with a code range of around 15 to 30 bytes. Public Key cryptography use very large numbers for computation, such computations cannot be done with the simple mathematical operations. Due to this reason all public key cryptographic libraries include or support some big number libraries. So this makes an additional load on the code size, but it is mandatory and cannot be skipped.

In order to know the ROM requirements of the code msp430-size utility [9] is used. In first step this utility is used to find out the code size used by Contiki. Then PKC code is integrated with it, now again the compiled code size is measured. In this way the code size required by the cryptographic functions is estimated. Mathematically it can be written as;

```
ROM required= PKC included compiled code size - Normal compiled code size
```

## 5.3. RAM

RAM is the most important constraint that needs the most attention. WSNs are equipped with only a few kilobytes of RAM. MSB430 is equipped with only 5 KB of RAM while Tmote Sky has 10KB of RAM. As the stack has variable size during the process execution hence it was not easy to measure ram. A mixed approach was chosen for Ram estimations. Initially msp430-ram-usage [9] program is used to see the memory footprint of Contiki on the sensor node. Cooja simulator provides a Stack viewer which can show the stack usage during all the process execution. Therefore Cooja is used to view the stack usage during all of the operations to see their stack utilization. The maximum stack usage in Cooja is the estimated Ram consumption for the PKC operations. To correctly estimate the Ram usage in case of Relic library these results were enough as static variables and dynamic memory allocation were not used. For LibTomCrypt heap allocation allotted for ecc-points for final results was added to stack usage for final results.

## 5.4. Energy

As WSN are equipped with a limited power source hence energy is a very scarce resource. Normal applications consume use energy in the range of micro-joules but as in public key cryptography complex and long mathematical equations are solved so the consumption increases from micro-joules to mill-joules as is shown in the case of TinyECC [25]. In addition, the cost of a hardware-based mechanism for energy measurement is too high; the cost per-hardware-unit is similar to the price of the sensor node [42].Contiki provides a built in tool ENERGEST for energy estimations. “Energest is a software-based on-line energy estimation mechanism that estimates the energy consumption of a sensor node” [43].

Energest: It is a mechanism provided inside Contiki to be used by sensor node to provide energy estimations of all components such as radio transceiver and CPU. The core of this process is to use timers. When a component is turned on, a counter starts to measure the estimated energy consumption. When the component is turned off then the current value is added to the entry table of the component. The difference between the two values is a resultant which is then multiplied by the components power to show the energy consumed by it during its usage. This is called denormalization. To denormalize the values of the table, the specific characteristics of the hardware must be known. Following table shows the parameters used for denormalization. These values are obtained from the MSP4301612 and CC1020 datasheets.

| Component | Current Consumption ( mA ) |
|---|---|
| CPU | 1.8 |
| LPM | 0.0545 |
| Radio TX | 20.5 |
| Radio RX | 19.9 |

**Table 4: Component Current Comsumption**

Once the table entry of each component was derived, then these components were denormalized and finally they needed to be multiplied with the voltage of sensor node to estimate energy. Till now the energy estimations are not normalized for average results. Finally it is divided with RTIMER_SECOND component of Contiki to provide energy estimates for time used by the process. Following formula shows it in the code format.

Energy_Consumed = ((1.8 * diff.cpu + 0.545 * diff.lpm + 20.0 * diff.listen + 17.7 * diff.transmit) * 3 / RTIMER_SECOND));

Vcc is used at value of 3Volts as sensor node was connected to a USB interface and it is running to full power mode. Similarly RTIMER_SECOND represents “Number of Ticks per second” variable used in Contiki.

Chapter 6

# Performance Evaluation

In this chapter results acquired from the tests performed using public key cryptography with Contiki are presented. Initially a background about certain factors needs to be known that can help to understand & analyze these results more effectively.

## 6.1. Platform Selection

For testing different parameters two scenarios were opted due to different restrictions.

1) **Cooja Simulator** [27]: Cooja is a java based simulator provided with Contiki to test the applications in simulator prepared for different motes. For this project Tmote Sky mote based simulation provided in Cooja was selected.
2) **MSB430:** As discussed earlier in 2.8, MSB430 is a wireless sensor node which belongs to MSP430 based processor family.

An important reason to select Tmote Sky based simulation along with MSB430 is their use of same processor family. Tmote Sky uses MSP430F1611 microcontroller while MSB430 uses MSP430F1612 microcontroller. Hence Tmote sky was chosen for simulation, as in some cases MSB430 fails to fulfill some operations due to limited resources mainly RAM.

## 6.2. ECC Optimizations

Elliptic Curve Cryptography is a simple but time consuming process so people have worked on many mathematical methods to reduce the operational time taken to perform such calculations. There are many ECC optimizations that can lead to better performance and resource utilization. TinyECC has provided a brief description of these optimizations which can be beneficial especially for WSN. They have presented a complete ECC framework for resource optimized as well as efficiency optimized implementation by use of available optimizations. In this project all optimizations available cannot be focused as the two libraries used do not provide all these optimizations. However some optimizations that are present in these libraries are analyzed, that can be used to make an efficient implementation of ECC over Contiki. These optimizations are selected such that a simple notion can be viewed, afterwards resource based optimizations and performance based optimizations will be understable for different kinds of sensor nodes. Some basic information about optimizations used in this project are provided below.

a) Projective Co-ordinates: Projective Co-ordinate takes little more RAM and ROM as compared to basic co-ordinates but they reduce the execution time very effectively.
b) Sliding Window: Sliding Window consumes more RAM but they help to reduce the time taken for scalar multiplication.

c) Shamir's Trick [44]: This technique reduces signature time to a large extent but it requires more RAM as well.
d) Curve Specific Optimization: NIST and SECG curves use pseudo-Messene prime which reduces the execution time as reduction modulo is performed by less modulo multiplications and modulo additions. Similarly no division operation is required if such curves are used.
e) Montgomery Reduction: Montgomery Reduction modulo was preferred over Barret Reduction. TinyECC results have showed that Barret reduction requires more RAM and ROM as compared to Montgomery Reduction and still it has not a very effective difference in results.

## 6.3. Benchmarking Issues

Before going into the details of results it will be a good practice to share the constraints faced during the testing phase. Contiki is only stable at low power mode and it is not been tested to run on full speed of sensor node's clock rate. Just few weeks back a developer has patched the kernel of Contiki so that it can run on maximum speed but currently it cannot synchronize with serial interface to give the output on console. The main architects of Contiki were contacted and they have told that there are many fix value adjustments in the core of Contiki and due to this reason it is not so stable at high speeds.

Due to this major problem it was not possible to test properly the code on full speed of sensor node. However LEDs present on MSB430 were used to get the feedback of estimated time in seconds, this in-turn provides an idea of time reduction. Tmote Sky was also patched to run on full speed but as it was running inside Cooja simulator hence it didn't show any output after the half of its full clock rate i.e after 4MHz.

Energy estimation has the same problem as well. As energy estimation requires ENERGEST and its result needed to be viewed on serial interface so it was not possible to take any energy estimations on clock rate higher than 4.7MHz.

## 6.4. Speed

Time taken to execute different parameters of ECDSA is very vital in this project. In ECDSA there are three main parts that are analyzed for speed. Key Generation assuming there is no public key defined before this point, Signature on a message or symmetric key, and verification of this signature.

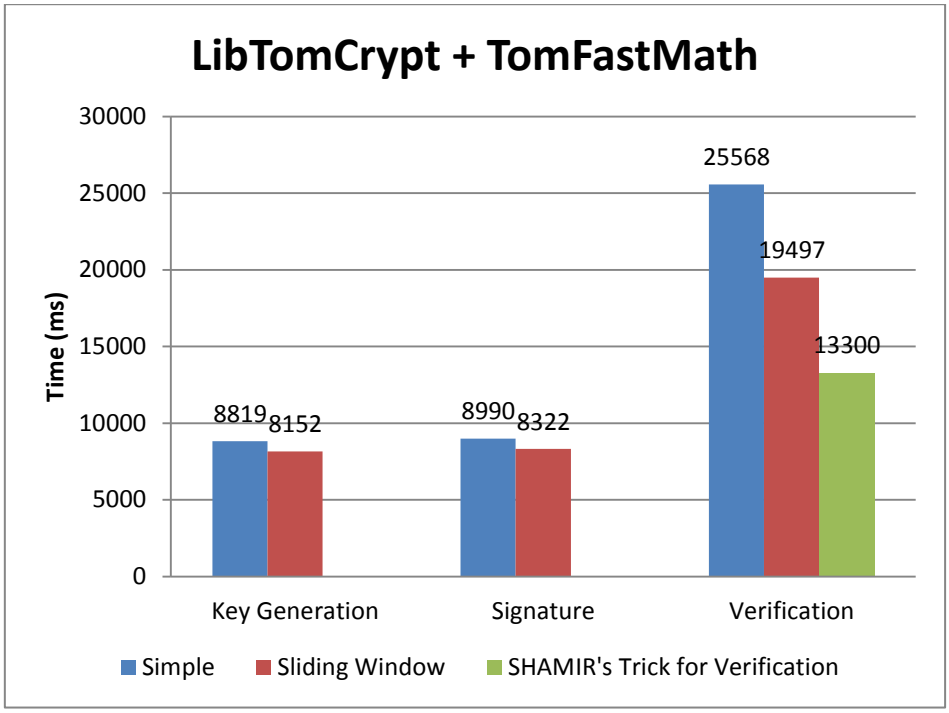


**Figure 5: LTCTFM perfromance**

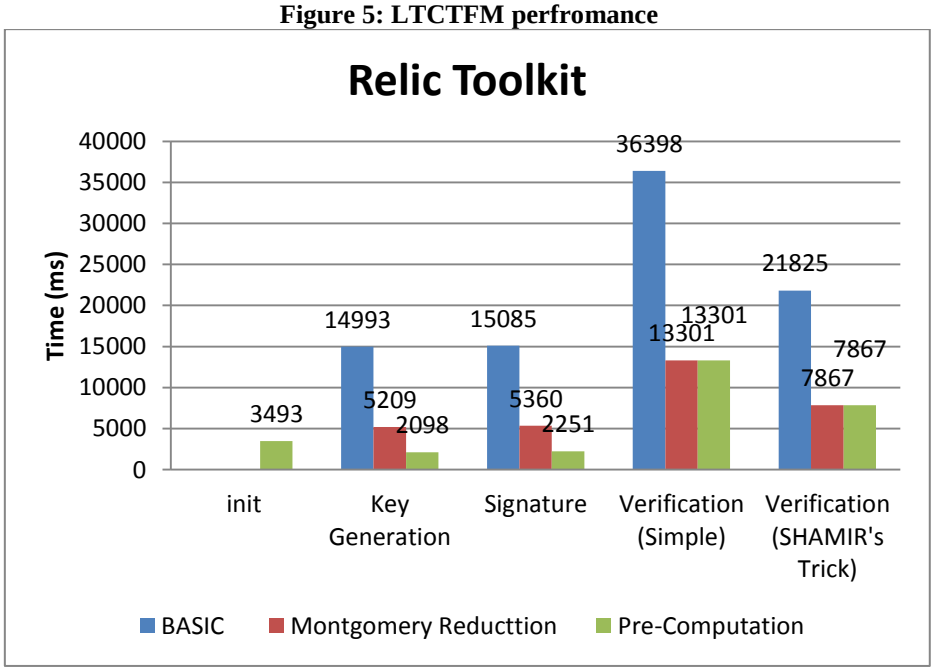


**Figure 6: Relic perfromance**

Here both results are presented independently so the time consumption by each process using different libraries can be elaborated more clearly. In overall comparison review Relic takes less time as compared to LibTomCrypt. Signature time is nearly equivalent to key generation time as it goes almost through the same process of key generation. It just adds some small mathematical functions after scalar multiplication but they take very less time.

Verification is the most time consuming task in the whole ECDSA process. It takes around 3 times more duration than the signature procedure in the case of LibTomCrypt minimal usage. However it can be dramatically reduced to 1.5 times if SHAMIR's trick is used. The only drawback in using Shamir's trick is that it consumes a lot of volatile memory, especially it cannot be used in case of MSB430 which is very low on RAM. To test SHAMIR's trick

simulator based on Tmote sky mote was used. The details about RAM usage by verification are written in RAM analysis section.

Contiki is not able to support UART (Universal asynchronous receiver/transmitter) protocol properly if it is run at full CPU speed of sensor node so the code was tested with the use of led to find the estimated time. Following graph shows the performance improvement as the MSB430 sensor node is configuring to run at maximum clock rate using Contiki.

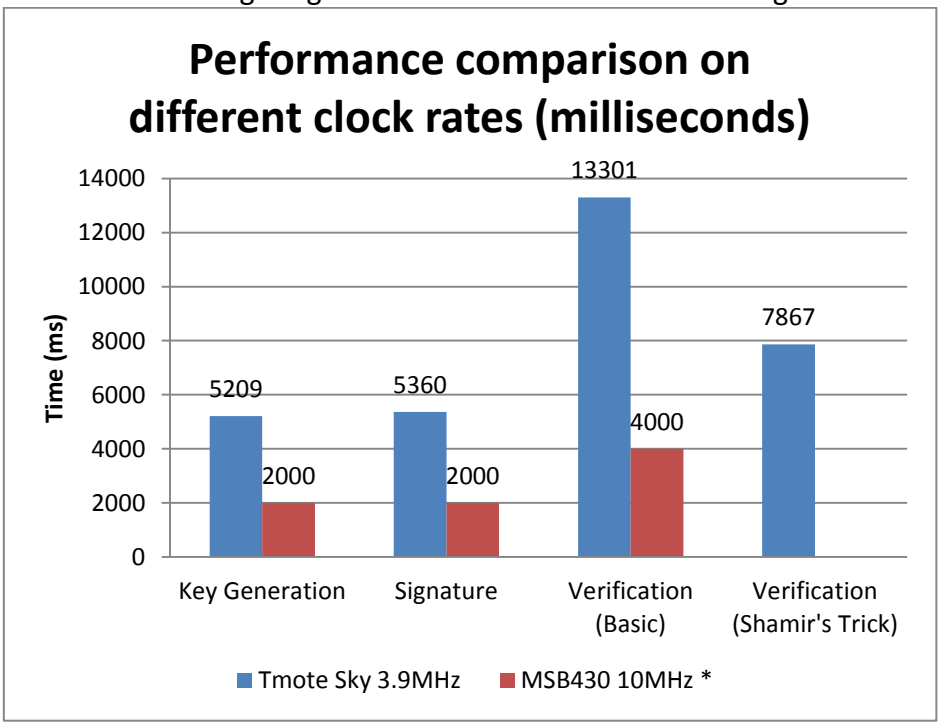


**Figure 7: Performance comparison**

* The results for MSB430 are estimated using led blink technique. Concrete results cannot be taken due to synchronization problem of Contiki at maximum clock rate.

The above graph shows that higher clock rates will yield much better results and same improvement can be visualized for reduced use of energy by the sensor node. For this test Relic projective co-ordinates and SECG-160 curve were used. Shamir's trick was not utilized for this test as MSB430 has less memory and cannot support this optimization.

There are many standard curves that can be used for ECDSA functions. Two standard curves specified by SECG group were chosen for this project. SECG-160 and NIST-192 are used for all tests. The given graph depicts the comparison of time taken by these two curves using both libraries. It can be seen that SECG-160 takes less time as fewer operations are required as compared to NIST-192. The key point that every administrator needs to focus on, is the level of security required. The following graph depicts that with more level of security more time consumption is required. This also applies to all other major requirement factors like energy, RAM and ROM.

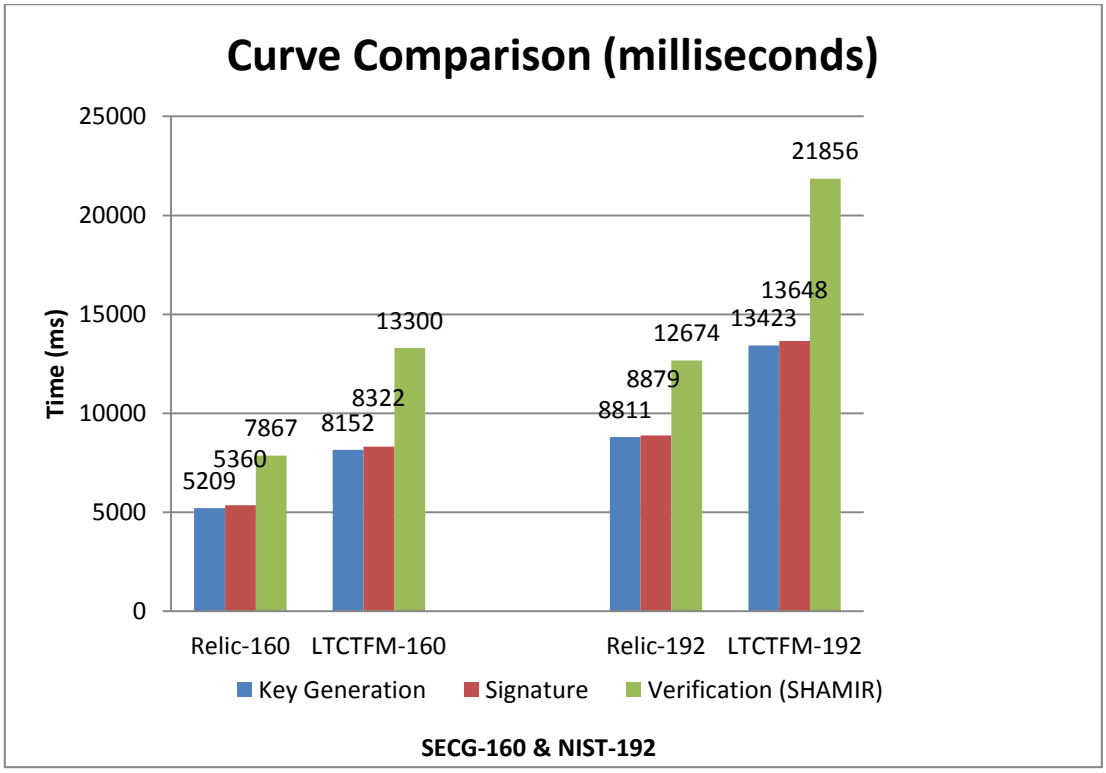


**Figure 8: Curves Comparison**

There are two types of co-ordinates system that can be used in any ECC based algorithm, Basic Co-ordinates and Projective co-ordinates. A comparison of these co-ordinates is shown below to visualize which co-ordinate system performs much faster and hence fulfils the requirements. LibTomCrypt used Basic co-ordinate system till version 1.0 and stopped using them after this and the latest version does not include support for basic co-ordinate system. Relic on the other hand has provided the code for basic co-ordinate system as well. So in this comparison the performance of these co-ordinates was reviewed using Relic library.

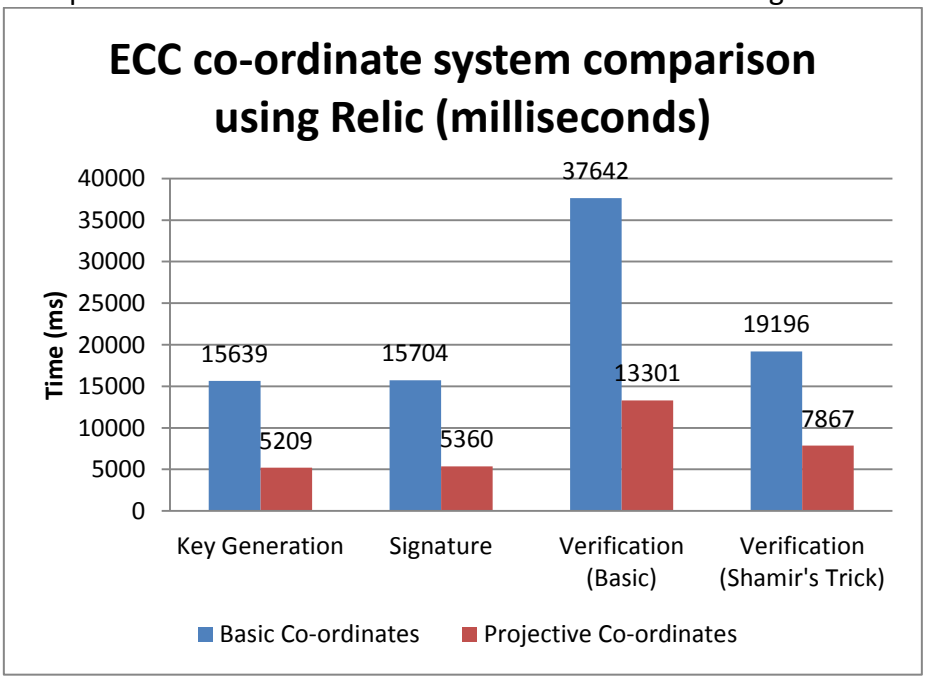


**Figure 9: Co-ordinates comparison**

It is clearly observed that Projective co-ordinate provide much better results in any scenario as compared to basic co-ordinates at the cost of more RAM and ROM resources. In case of

ROM there is not a big effect on addition of code size however Ram usage is much more but it is observed that sensor nodes that have even 5KB of Ram can easily support projective co-ordinates. Hence projective co-ordinates are recommended and used in all ECC algorithms for this project.

Modular Reduction Techniques play a vital role in providing efficient results. There are four techniques used in multi precision integers which reduce the large integer into a small number which can help a lot in performing large mathematical operations. These techniques are;

i) Basic Division based reduction
ii) Montgomery reduction
iii) Barret Reduction
iv) Fast reduction

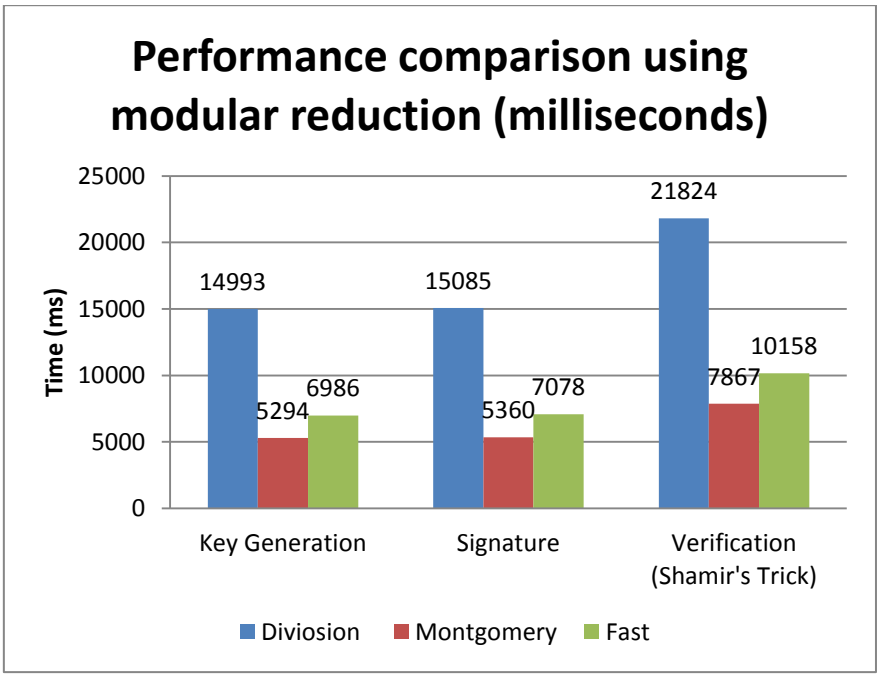


**Figure 10: Modular Reduction Technique Comparison**

Barret Reduction was not focused, primarily as TinyECC have provided information that Barret Reduction only helps in case of Pre-computation. Secondly both libraries have not used this techniques in their implementation for ECC operations. It is clearly seen from the graph that Montgomery reduction reduces the computation time to a very large extent hence recommend are made to use Montgomery reduction with all ECC operations.

## 6.5. Energy

Energy utilization is a major concern at this time. Most of the applications used in sensor networks consume energy in the micro joules, but public key cryptography consumes energy in millijoules. In order to show the detailed usage of energy in each phase of ECDSA separate graphs are presented for energy consumption of both libraries.

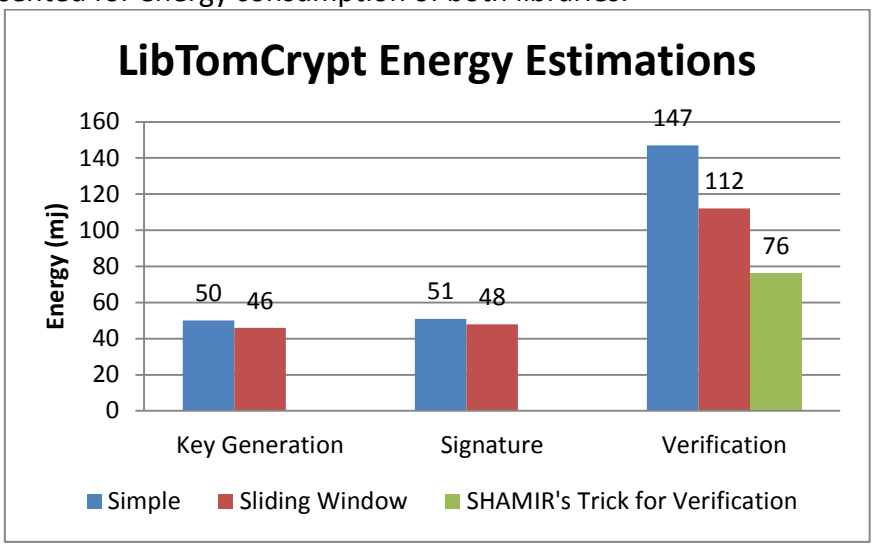


**Figure 11: LTCTFM Energy evaluation**

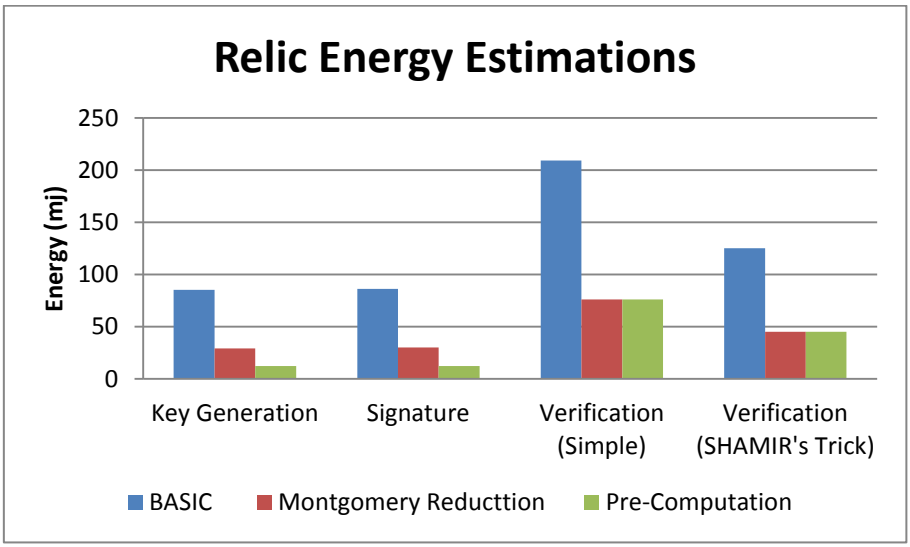


**Figure 12: Relic Energy Evaliation**

LibTomCrypt performs on an average scale while Relic consumes less energy when its optimizations are used. Overall Relic outperforms LibTomCrypt in saving the energy consumption of sensor node. In verification phase Shamir's trick takes the minimal energy as it takes less computational time. However it was not possible to take the results on higher clock rate due to serial rate synchronization problem.

## 6.6. ROM

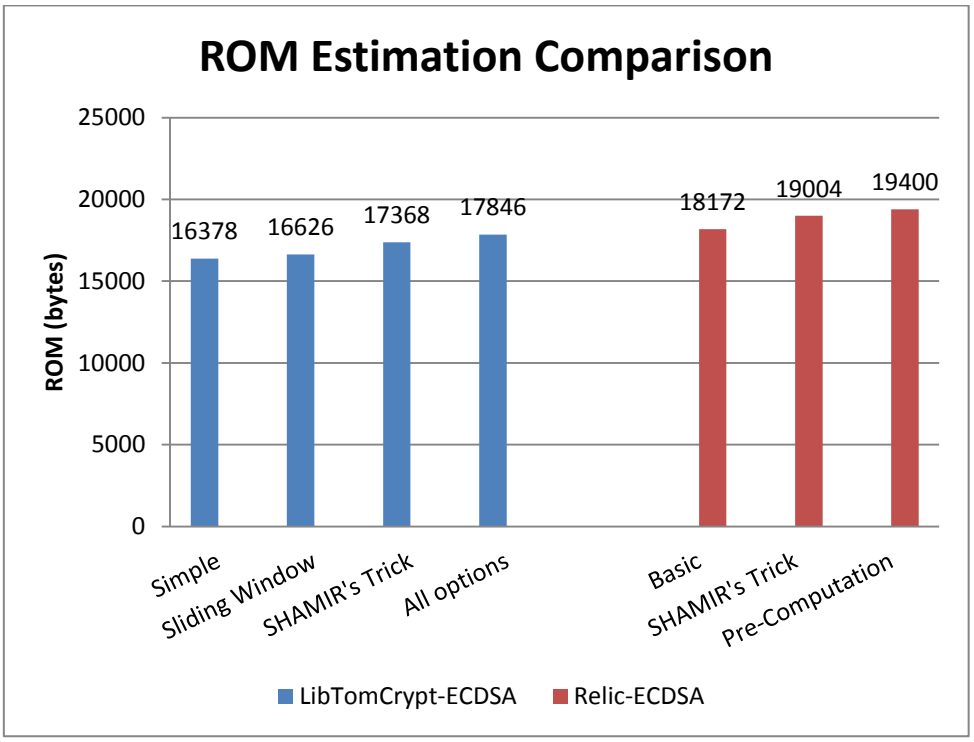


**Figure 13: Complete ROM estimations**

The above graph shows detailed code size requirements for adding elliptic curve cryptography into Contiki. The estimated code size for using elliptic curve cryptography is around 16-20 KB for both of libraries. Here it can be seen that the code size is not dependent on the level of security required. Similarly it is found that code size required for SECG-160 and NIST-192 curves doesn't change to a large extent that can affect the ROM size to be used.

During this project implementation work was performed in two phases. First phase was to strip all the code on a normal PC and test its validity. Once it is reduce to a minimal size and it correct behavior is verified then the process of porting such code to Contiki platform was started. This was a major problem and it took most of the time in this project as it was uncertain until mid of project about how to reduce different aspects of code for Contiki. Many factors needed to be focused, like code reusability (in case of rand function) and variable usage optimizations/reductions.

A simple example of such case is the use of SHA1 in the ECDH code. A normal SHA1 takes 10 KB on normal PC but once it is ported to Contiki then it overflows the stack space and hence cannot be compiled. After some research information about an 8bit SHA1 was found which was used later on with the code and then it uses minimal code size, and hence able to port program into Contiki.

## 6.7. RAM

In order to analyze the RAM requirement for complete ECDSA process a mix approach was used to calculate the estimated RAM size.
Cooja provides a stack viewer which can let users view the real-time stack size utilization of the compiled code. This estimation was enough for relic RAM estimation as relic was configred to use all variables from the stack memory space. Malloc based allocation was not used in case of Relic instead stack was preferred for the whole process. There is only one exceptional scenario of pre-computation which uses static variables. Hence in case of Pre-Computation static variable usage was added with stack to estimate the total Ram usage.

In LibTomCrypt a mix approach for memory allocation was finalized after many code readjustments. Due to different tests and observations it was decided to use a mix of dynamic memory allocation with stack. So in order to find the total RAM size usage for LibTomCrypt based implementations heap allocation was added to stack view as well.

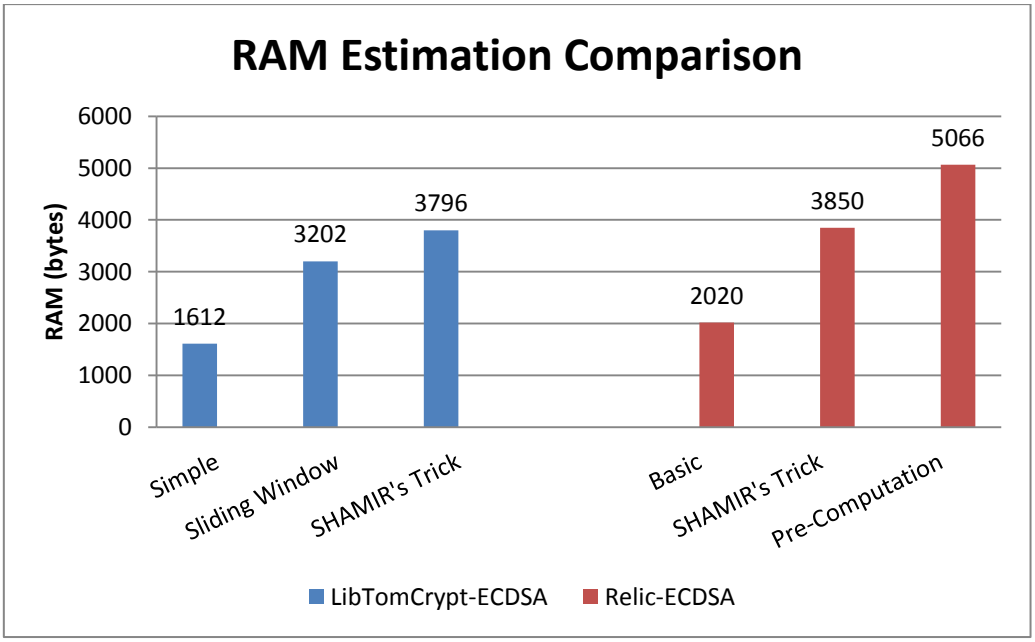


**Figure 14: RAM estomations**

The graph shows that LibTomCrypt with no optimizations included requires least RAM resources as compared to optimized LibTomCrypt and relic. Currently there is no sliding window implementation available for Relic, and LibTomCrypt doesn't include pre-computation methods, these merits cannot be compared for memory usage.

### ECDH & ECDSA Comparison Review

An additionaly analysis concerning the RAM estimations for ECDH implementation were reviewed for LibTomCrypt. ECDSA and ECDH in LibTomCrypt requires nearly same amount of

RAM in all scenarios. It is found that both libraries can run on sensor nodes with less than 5KB RAM if is configured to be resource efficient. However there was some monir differences in ROM size comaprison as ECDH includes more code for its operation.

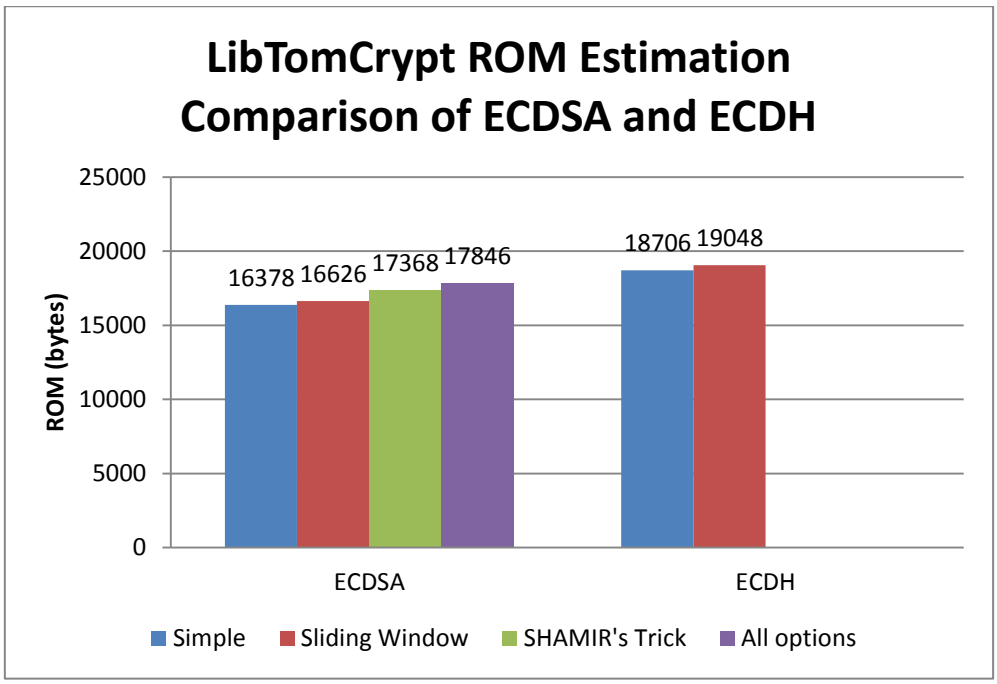


**Figure 15: Rom Estimations for ECDH and ECDSA**

There is a clear tradeoff between RAM usage and performance speed of algorithms. If more optimizations are used to reduce the execution time then it will require more RAM and vice versa. Hence it can be said that a review of these factors with the perspective of sensor nodes specifications are necessarily required before deployment.

## 6.8. Comparison with TinyECC

During all the project timeline, TinyECC was used as role model because it holds the same perspective as the work done in this thesis. Secondly TinyECC is the only standard present with context of operating systems usage in sensor networks that can be compared to see the performance merits. There are some optimizations present in TinyECC that are not available in Libtomcrypt and Relic (free version) hence it cannot be sais completely that the results are comparable, still it can provide a notion that the work done in this project is in the right path.

| | TinyECC | LTCTFM | Relic |
|---|---|---|---|
| Time | 3.2s (sign) 4.1s (verify) | 8.3s (sign) 25s (verify) | 2.3s (sign) 7.8s (verify) |
| Energy | 27.5 mJ + 17.1 mJ + 21.9mJ | 47mj + 48mj + 71mj | 25mj+ 12mj + 45mj |

| | | | |
|---|---|---|---|
| ROM | 13.5 KB | 19 KB | 19.5 KB |
| RAM | 1.5 KB | 2.5 KB | 1.2 KB |

**Table 5: All optimizations Enabled**

In above table TinyECC provide a much better result due to addition of optimizations which include Hybrid Multiplication and Inline assembly code. Relic can provide much better results once their developers will make the MSP430 assembly code available in their free version.

| | **TinyECC** | **LTCTFM** | **Relic** |
|---|---|---|---|
| Time | 4.2s (sign) 8.6s (verify) | 9 s (s) 13s (v) | 15s(s) 36s (v) |
| Energy | 230 mJ + 463mJ | 47mj + 48mj + 71mj | 91 mj + 92mj + 117mj |
| ROM | 8.2 KB | 15 KB | 17KB |
| RAM | 160 bytes | 312 bytes | 100 bytes |

**Table 6: All Optimizations disabled**

The table shown above is not used for comparison but it is more of a reference related data. Basic co-ordinates in LibTomCrypt and Relic are not used as they take more computational time, and preferred to use projective co-ordinates completely. In conclusion it can be said that Relic is much more efficient when same optimizations are used in all of compared libraires. However TinyECC shows much better results when no optimizations are used. In case of Contiki Relic seems to be a much better option as it has more optimizations included and it is specificly written for sensor networks.

Chapter 7

# Project Recommendations

## 7.1. Contiki

Contiki is an operating system for wireless sensor networks that is comparatively new and getting stable day by day as compared to TinyOS development which is more mature now. Due to these changes that arrive in Contiki many factors are still uncertain about the integration of public key cryptography in it. Some main factors that still need to be focused in this case are;

i) Running Contiki on maximum clock rate
ii) UART synchronization issues for debugging

The results presented in previous chapter are dependent on these considerations. Due to these best results were not obtained with use of Contiki. Contiki is not able to run fully functional in maximum clock rate of the sensor node. The core developers of Contiki were contacted for this matter and they told that still there are many fix value implementations so Contiki will be unstable if it is run at full clock rate. The only way for us to make it possible is to read and understand core of Contiki with reference to individual datasheets of the sensor nodes but this is not part of the scope of project. Still Contiki is made stable and used in experiments until the half of processor speed. Hence in this case MSB430 remained stable to work and show results through serial link up to 4.7MHz and on Tmote Sky using Cooja it can show proper results up till 3.9MHz.

During the last phase of project one developer contributed on Contiki by providing a hack to run Contiki on full speed. The only problem still unsolved at maximum clock rate is serial link synchronization. Hence Contiki ca now run on full speed but the results were not visible on serial link based display. The solution used to solve this problem was to utilize LEDS on MSB430. leds_on() function was place in code at locations where the code is checked and added tests at important checkpoints in the code. Hence it was assured that code executed completely and gives positive results at full clock speed. Same solution cannot be used for Tmote Sky inside Cooja as Cooja becomes unstable and doesnot even show the proper results for leds.

## 7.2. Cryptographic adaptations suitable for Contiki

In order to integrate Public Key cryptography with any sensor network operating system many important factors needed to be considered before deployment.

## Level of Security

First and foremost information required to deploy public key is to determine how much level of security is required. If sensor nodes are deployed in military or any other security environment then a higher level of security should be considered. In this case at least 192-bit elliptic curve cryptography should be the used. On the other hand if the sensor nodes are deployed in some peaceful zones and just required for gathering information then 160bit based ECC is enough to ensure message integrity. Consequently higher the level of security is implemented higher the RAM, ROM and energy requirements will be.

| | **High Level of Security** | **Low Level of Security** |
|---|---|---|
| **Key Size** | At least 192 bit | At least 160 bit |
| **Hash Function** | SHA256 | 8-Bit SHA1 |

## Sensor Node Processing Speed

There are many large mathematical computations required to complete public key cryptography. These computations seem to have very normal requirements in PC's as they can handle large instruction-sets, but sensor nodes have very small microcontroller with some balanced functionality. Currently most of the available microcontrollers used in sensor networks have a very low processing power, hence the public key computation takes more time but still it is not far to see good processing power in sensor nodes. An example of such case is computation time of SECG-160 taken from TinyECC. TelosB running at 4MHz took 3169 milliseconds (3.16 seconds) for ECDSA based signature while Imote2 running at 416MHz performed same work in only 11.8 milliseconds (0.011 seconds). Contiki is not fully tested to be used at maximum clock rate. Currently it can run stable under low power mode 1 (LPM1). Once Contiki is stable for full computational speed then much better results can be seen.

If processing power of any sensor node is not very high or not fully utilized then high key size based ECC should be used minimally as high key sizes will take more time which can affect many critical resources.

| | **Low Processing Power** | **High Processing Power** |
|---|---|---|
| **Key Size Recommended** | 160 bit | 192 bit |

## RAM consideration

Sensor nodes come with very low resources hence their RAM and ROM sizes are very limited, this puts an additional pressure on choosing the appropriate key size for ECC. It is experienced many times in this project. Some optimizations are very good for saving time but they require a lot of RAM which may not be available. The most important scenario for

such case is use of Shamir's trick. Shamir's trick can help reduce time for verification but it requires additionally. Most of the research shows that public key cryptography is even possible with very low resources while in our case it is different from other findings. All other implementations except TinyECC have run on sensor node without use of any operating system. Similarly most of them have used machine language code for optimizing the performance and resources. In this project no machine language code is used and all the code is running inside an operating systems "Contiki". Due to use of operating system many other considerations were changed like less RAM and ROM allocation, interrupt based handling and low power mode usage. Here recommendations regarding dfiferent ECC optimizations are summarized which can be used with respect to variable RAM requirements.

| | MSB430 (5KB RAM) | Tmote Sky (10KB RAM) |
|---|---|---|
| Projective Co-ordinates | Applicable | Applicable |
| Montgomery Reduction | Applicable | Applicable |
| Simple Multiplication | Applicable | Applicable |
| Sliding Window | Not Applicable | Applicable |
| Shamir's Trick | Not Applicable | Applicable |
| Pre-Computation | Not Applicable | Applicable |

**Table 7: ECC optimizations used on different sensor nodes**

There was more focus on RAM as compared to ROM as most of the current sensor nodes arrive with enough ROM to fill the required code. Low ROM based sensor node Tmote Sky was used which has only 48KB total ROM, and the maximum code size for this project was around 20KB. Hence the total code size including Contiki was around 45 KB. However an important point is case of Tmote Sky as it will not be able to add more applications into Contiki system once PKC is added to it. Below total code size estomations in case of two sensor nodes is highligted.

| | MSB430 (55 KB ROM) | Tmote Sky (48KB ROM) |
|---|---|---|
| Total ROM size | 55KB | 48KB |
| ECDSA-All optimizations used | 36406 bytes | 44298 bytes |
| ECDSA- No optimization used | 36406 bytes | 41276 bytes |

**Table 8: Total Rom Usage**

In case of MSB430 only 1500 bytes of RAM is available hence any advanced optimizations present cannot be used. This shows that RAM is the major factor as compared to ROM when it comes to use optimizations for ECC in sensor nodes.

Chapter 8

# Conclusion

In this thesis a complete feasibility study for deploying ECC based algorithms in wireless sensor networks under the Contiki operating system is presented. Public key cryptography is still expensive in the field of sensor networks as compared to most of the other applications. However it is showed that Contiki can also achieve high level of security using smaller key sizes like TinyOS has performed by use of TinyECC.

Moreover, the details description about parameters required to evaluate a cryptographic library are presented, and how it can be integrated into Contiki. Two most relevant cryptographic libraries LibTomCrypt and Relic are evaluated, with relevance to performance, energy, ROM and RAM usage inside Contiki. Relic has proven to be much better choice as it is specifically designed for wireless sensor networks.

ECC is expensive in wireless sensor networks if it is used without any additional optimizations available. Evaluations concerning the performance and resource efficiency of ECDSA and ECDH algorithms on the MSB430 platform and simulation based Tmote Sky are documented as well. Experiments have showed that it is possible to utilize ECC along with its optimizations even in low resource based sensor networks.

Chapter 9

# Future Work

The project focused on only one cryptography technique which is widely used and considered efficient for low resource based sensor networks. The main emphasis revolved around one ECC-based algorithm i.e. ECDSA. More algorithms like ECDH and ECIES can be added and compared to make the complete package of security based on ECC to be used with Contiki. There are certain ECC optimizations that are still not being used and tested. These include Hybrid Multiplication, and assembly code usage for multiplication and squaring operations. Once these are used it is expected that the results will be more efficient. ECC over Prime field was selected over Binary field as its being proven by TinyECC to be more resource efficient. In future Binary Curves can be added to the code which can build ground for more cryptographic techniques.

Pairing based cryptography is also becoming more feasible to be used with WSN. Many publications have shown that using pairing based algorithms with ECC can provide much more security to time ratio for the sensor networks.

In future ContikiSec framework can be modified to include ECC. This will make a concrete framework for the security of wireless sensor networks using Contiki.

# *Bibliography*


1. TinyOS Community Forum. [Online] http://www.tinyos.net.
2. The Contiki Operating System. [Online] http://www.sics.se/contiki/.
3. **SHAH BHATTI, JAMES CARLSON, HUI DAI.** s.l. : Springer Science, 2005. ISSN: 1383-469X.
4. *Full TCP/IP for 8-Bit Architectures.* **Dunkels, Adam.** San Francisco, California : Proceedings of the 1st international conference on Mobile systems, applications and services, 2003.
5. *Poster Abstract: Making Sensor Networks IPv6 Ready.* **Mathilde Durvy, Julien Abeillé, Patrick Wetterwald , Colin O'Flynn, Blake Leverett, Eric Gnoske, Michael Vidales, Geoff Mulligan, Nicolas Tsiftes, Niclas Finne, Adam Dunkels.** s.l. : Proceedings of the 6th ACM conference on Embedded network sensor systems, 2008. ISBN:978-1-59593-990-6.
6. *ContikiSec: A Secure Network Layer for Wireless Sensor Networks under the Contiki operating System.* **Lander Casado, Philippas Tsigas.** s.l. : Springer Berlin, 2009. ISBN: 978-3-642-04766-4_10.
7. MSP430 Ultra-Low-Power Microcontrollers. *Texas Instruments.* [Online] http://www.ti.com/corp/docs/landing/mcu/index.htm.
8. **ScatterWeb.** ScatterWeb - MSB430. [Online] http://www.scatterweb.com/content/products/research_line/msb430.en.html.
9. mspgcc- GCC toolchain for MSP430. [Online] http://mspgcc.sourceforge.net/.
10. *Mspsim - an extensible simulator for msp430-equipped sensor boards.* **Joakim Eriksson, Adam Dunkels, Niclas Finne, Fredrik Osterlind.** Netherlands : European Conference on Wireless Sensor Networks, 2007.
11. Moteiv Hardware Product Transition Notices. [Online] http://www.sentilla.com/moteiv-transition.html.
12. Fredrik Osterlind. [Online] http://www.sics.se/~fros/cooja.php.
13. **F. Amin, A.H Jahangir, and H. Rasi fard.** Analysis of Public-Key Cryptography for Wireless Sensor Networks Security. World Academy of Science, Engineering and Technology, 2008.
14. **Chris Townsend, Steven Arms.** *Wireless Sensor Networks: Principles and Applications.* s.l. : microstrain.com.
15. **Jing Deng, Richard Han, Shivakant Mishra.** *Enhancing Base Station Security in Wireless Sensor Netowrks.* s.l. : University of Colorado, Department of Computer Science. Technical Report CU-US-951-03.
16. *ContikiSec: A Secure Network Layer for Wireless Sensor Networks unider the Conitki Operating System.* **Lande Casado, Philippas Tsigas.** s.l. : Proceedings of the 14th Nordic Conference on Secure IT Systems NordSec 2009, 2009. ISBN: 978-3-642-04765-7.
17. **John Paul Walters, Zhengqiang Liang, Weisong Shi, and Vipic Chaudhary.** Wireless Sensor Network Security: A Survey. *Security in Distributed, Grid, and Pervasive Computing.* s.l. : CRC Press , 2006.
18. **Apostolos, Pyrgelis.** *Cryptography and Security in Wireless Sensor Networks.* [Presentation Slides] Greece : Department of ComputerEngineering and Informatics, 2009.
19. *Jamming Detection Mechanism for Wireless Sensor Networks.* **Murat Cakiroglu, Ahmet Turan Ozcerit.** s.l. : ICST, 2008. ISBN: 978-963-9799-28-8.
20. *Emergent Properties: Detection of the Node-capture Attack in Mobile Wireless Sensor Networks.* **Mauro Conti, Roberto Di Pierto, Luigi V. Mancini, and Alessandro Mei.** s.l. : Proceedings of the first ACM conference on Wireless Netowrk Security, 2008.

21. **Chris Karlof, David Wagner.** *Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures.* s.l. : University of California at Berkeley, 2003.
22. **Wassim Znaidi, Marine Minier, Jean-Phillippe Babau.** *An Ontology for Attakcs in Wireless Sensor Networks.* s.l. : INRIA, 2008. inria-00333591, version 1.
23. *SPINS: security protocols for sensor networks.* **Adrian Perrig, Robert Szewczyk, J. D. Tygar, Victor Wen, David E. Culler.** s.l. : Wireless Networks, Volume 8 , Issue 5 (September 2002), Pages: 521 - 534, 2002. ISSN:1022-0038.
24. *TinySec: A Link Layer Security Architecture for Wireless Sensor Networks.* **Chris Karlof, Naveen Sastry, David Wagner.** Baltimore : 2nd International Conference on Embedded Networked Sensor Systems SenSys 2004, 2004. ISBN:1-58113-879-2.
25. *TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks.* **An Liu, Peng Ning.** s.l. : IEEE Computer Society, 2008. ISBN:978-0-7695-3157-1.
26. Tmote Sky: Product Description. [Online] http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf.
27. *COOJA/MSPSim: interoperability testing for wireless sensor networks.* **Joakim Eriksson, Fredrik Osterlind, Niclas Finne, Nicolas Tsiftes, Adam Dunkels, Thiemo Viogt, Robert SAuter, Pedro Jose Marron.** Rome, Italy : International Conference On Simulation Tools And Techniques For Communications, Networks And Systems & Workshops, 2009. ISBN:978-963-9799-45-5.
28. *Unleashing public-key cryptography in wireless sensor networks.* **Lopez, Javier.** s.l. : IOS Press Amsterdam, The Netherlands, 2006. ISSN:0926-227X.
29. RSA Algorithm. [Online] http://pajhome.org.uk/crypt/rsa/contrib/RSA_Project.pdf.
30. RSA Project. [Online] http://pajhome.org.uk/crypt/rsa/contrib/RSA_Project.pdf.
31. **Hnerik Ahlsrom, Karl-Johan Skoglund.** *Encryption in Delocalized Access Systems.* Linkoping : Linkpoing Tekniska Hogskola, 2007. LITH-ISY-EX--07/4046--SE.
32. **Chou, Wendy.** *Elliptic Curve Cryptography and Its Applications to Mobile Devices.* s.l. : University of Maryland, College Park.
33. **Institute, SANS.** *Elliptic Curve Cryptography and Smart Cards.*
34. **Microsoft.** Overview of ECDH Algorithm. [Online] Microsoft, 2010. http://msdn.microsoft.com/en-us/library/cc488016.aspx.
35. LibTomCrypt. [Online] http://libtomcrypt-cug.googlecode.com.
36. TomFastMath. [Online] http://www.freshports.org/math/tomsfastmath/.
37. **Debian.** [Online] http://packages.debian.org/sid/libtommath-dev.
38. Relic Toolkit. [Online] http://code.google.com/p/relic-toolkit/.
39. BitInt - Multiprecision llibrary for Contiki. [Online] http://www.ohloh.net/p/bitint-multipre.
40. **Dunkels, Adam.** The contiki Operating Systems. *Contiki.* [Online] http://www.sics.se/contiki/developers/memory-block-management-in-contiki.html.
41. **Toledo, Sivan.** *Exercise in Embedded Computing: Contiki Basics.* [pdf] s.l. : Tel-Aviv University.
42. *Micro power meter for energy monitoring of wireless sensor networks at scale.* **Xiaofan Jiang, Prabal Dutta, David Culler, Ion Stocia.** Cambridge, Massachusetts, USA : Proceedings of the 6th international conference on Information processing in sensor networks, 2007. ISBN:978-1-59593-638-X.
43. *Software-based on-line energy estimation for sensor nodes.* **Adam Dunkels, Fedrik Osterlind, Nicolas Tsifes, Zhitao He.** Ireland : ACM, Proceedings of the 4th workshop on Embedded networked sensors, 2007. ISBN:978-1-59593-694-3.
44. **Moller, Bodo.** *Algorithms for multi-exponentiation.* Darmstadt : Technische Universit at Darmstadt, Fachbereich Informatik, 2001. Technical Report TI-8/01.