UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

SECURITY PRIMITIVES FOR ULTRA-LOW POWER SENSOR NODES IN WIRELESS SENSOR NETWORKS

by

An-Lun (Alan) Huang

Submitted in partial fulfillment of the requirements for the degree

Master of Engineering (Computer Engineering)

in the

Faculty of Engineering, the Built Environment and Information Technology

UNIVERSITY OF PRETORIA

October 2005

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

SECURITY PRIMITIVES FOR ULTRA-LOW POWER SENSOR NODES IN WIRELESS SENSOR NETWORKS

by

An-Lun (Alan) Huang

Study leader: Prof. W. T. Penzhorn
Department: **Computer Engineering**
Faculty of Engineering, Built Environment and Information Technology
Degree: **M. Eng (Computer)**

SUMMARY

The concept of wireless sensor network (WSN) is where tiny devices (sensor nodes), positioned fairly close to each other, are used for sensing and gathering data from its environment and exchange information through wireless connections between these nodes (e.g. sensor nodes distributed through out a bridge for monitoring the mechanical stress level of the bridge continuously). In order to easily deploy a relatively large quantity of sensor nodes, the sensor nodes are typically designed for low price and small size, thereby causing them to have very limited resources available (e.g. energy, processing power).

Over the years, different security (cryptographic) primitives have been proposed and refined aiming at utilizing modern processor's power e.g. 32-bit or 64-bit operation, architecture such as MMX (Multi Media Extension) and etc. In other words, security primitives have targeted at high-end systems (e.g. desktop or server) in software implementations. Some hardware-oriented security primitives have also been proposed. However, most of them have been designed aiming only at large message and high speed hashing, with no power consumption or other resources (such as memory space) taken into considerations. As a result, security mechanisms for ultra-low power (<500μW) devices such as the wireless sensor nodes must be carefully selected or designed with their limited resources in mind.

The objective of this project is to provide implementations of security primitives (i.e. encryption and authentication) suitable to the WSN environment, where resources are extremely limited. The goal of the project is to provide an efficient building block on which the design of WSN secure routing protocols can be based on, so it can relieve the protocol designers from having to design everything from scratch.

This project has provided three main contributions to the WSN field.

- Provides analysis of different tradeoffs between cryptographic security strength and performances, which then provide security primitives suitable for the needs in a WSN environment. Security primitives form the link layer security and act as building blocks for higher layer protocols i.e. secure routing protocol.

- Implements and optimizes several security primitives in a low-power microcontroller (TI MSP430F1232) with very limited resources (256 bytes RAM, 8KB flash program memory). The different security primitives are compared according to the number of CPU cycles required per byte processed, specific architectures required (e.g. multiplier, large bit shift) and resources (RAM, ROM/flash) required. These comparisons assist in the evaluation of its corresponding energy consumption, and thus the applicability to wireless sensor nodes.

- Apart from investigating security primitives, research on various security protocols designed for WSN have also been conducted in order to optimize the security primitives for the security protocols design trend. Further, a new link layer security protocol using optimized security primitives is also proposed. This new protocol shows an improvement over the existing link layer security protocols.

Security primitives with confidentiality and authenticity functions are implemented in the TinyMote sensor nodes from the Technical University of Vienna in a wireless sensor network. This is to demonstrate the practicality of the designs of this thesis in a real-world WSN environment.

This research has achieved ultra-low power security primitives in wireless sensor network with average power consumption less than 3.5 μW (at 2 second packet transmission

interval) and 700 nW (at 5 second packet transmission interval). The proposed link layer security protocol has also shown improvements over existing protocols in both security and power consumption.

Keywords: Wireless Sensor Network (WSN), ultra-low power, security, cryptographic primitives, Message Authentication Code (MAC), UMAC, OCB.

OPSOMMING

'n Draadlose sensornetwerk (DSN) behels klein toestelle (sensornodusse) wat naby mekaar geplaas word. Hierdie toestelle versamel data van die omgewing en ruil die data met ander nodusse deur gebruik te maak van draadlose verbindings. 'n Voorbeeld van 'n draadlose sensornetwerk is sensornodusse wat oor 'n brug versprei is om die meganiese stresvlakke wat daarop inwerk deurloops te monitor. Die sensornodusse word tipies ontwerp om goedkoop en klein te wees sodat 'n relatiewe groot aantal in werking gestel kan word. Dit beteken egter dat elke sensornodus min hulpbronne beskikbaar het (bv. energie, verwerkingskrag).

Verskillende sekuriteitselemente (kriptografiese elemente) was voorgestel en verbeter deur die jare. Die doel van die verbeterings was om moderne verwerkers se krag in te span (32- bis en 64-bis verwerking), argitekture soos MMX ("Multimedia Extension") te gebruik, ens. Die sagteware implementering van sekuriteitselemente was dus gemik op hoë-vlak stelsels ("desktop" of bediener rekenaars). Sommige hardeware-georienteerde sekuriteitselemente was ook voorgestel, maar die ontwerp van hierdie elemente konsentreer hoofsaaklik op hoë spoed hutsing, sonder enige ag vir kragverbryk of ander hulpbronne (soos geheuespasie). Sekuriteitselemente vir lae-drywing (<500µW) toestelle, soos die draadlose sensornodusse, moet daarom versigtig gekies word of ontwerp word met begrip van die beperkte hulpronne.

Die oogmerk van hierdie projek is om die implementasie van sekuriteitselemente (syfering en waarmerking) wat geskik is vir die DSN (waar hulpbronne uiters beperk is) te voorsien. Die doel van die projek is om 'n doeltreffende boublok te voorsien waarop die ontwerp van versekerde DSN roeteringsprotokolle uitgevoer kan word. Hierdie boublok het tot gevolg dat protokolontwerpers nie 'n hele projek van nuuts af hoef te ontwerp nie.

Hierdie projek bied drie bydraes tot die DSN vakgebied.

- Bied 'n analise van verskillende wisselwerkings tussen kriptografiese sterkte en werkverrigting. Hierdie analise bied sekuriteitselemente wat geskik is vir die behoeftes van die DSN omgewing. Sekuriteitselemente vorm die skakelvlak sekuriteit en dien as boublokke vir hoë-vlak protokolle (bv. die versekerde roeteringsprotokol).

- Implementeer en optimiseer verskeie sekuriteitselemente in 'n lae-drywing

mikrobeheerder (TI MSP430F1232) met beperkte hulpbronne (256 grepe RAM, 8 kilogrepe "flash" programgeheue). Die verskillende sekuriteitselemente word vergelyk in terme van die aantal verwerkersiklusse wat per verwerkte greep benodig word, asook die spesifieke argitekture wat benodig word (bv. vermenigvuldiger, groot bisskuif) en hulpbronne wat benodig word (RAM, ROM/"flash"). Hierdie vergelykings help in die evaluasie van die ooreenstemmende energieverbruik, en daarom die toepaslikheid tot draadlose sensornodusse.

- Navorsing was ook uitgerig op verskeie sekuriteitsprotokolle wat vir die DSN ontwerp is. Die doel van hierdie navorsing is om die sekuriteitselemente vir die sekuriteitsprotokol ontwerpsneiging te optimeer. 'n Nuwe skakelvlak sekuriteitsprotokol wat geoptimeerde sekuriteitselemente gebruik word ook voorgestel. Hierdie protokol toon 'n verbetering oor bestaande skakelvlak sekuriteitsprotokolle.

- Sekuriteiteselemente met vertroulikheids- en waarmerkingsfunksies word in 'n DSN geimplementeer met die "TinyMote" sensornodusse van die Technical University of Vienna. Hierdie nodusse word gebruik om die doelmatigheid van die ontwerpte stelsels van hierdie tesis in 'n realistiese DSN omgewing te demonstreer.

Hierdie navorsing het lae drywing sekuriteitselemente in 'n DSN bereik. Die gemiddelde kragverbruik was minder as 3.5 µW (met 2 sekonde pakkie versendingsintervalle) en 700 nW (met 5 sekonde pakkie versendingsintervalle). Die skakelvlak sekuriteitsprotokol wat voorgestel is toon verbeterings oor bestaande protokolle in beide sekuriteit en kragverbruik.

LIST OF ABBREVIATIONS

AES	Advanced Encryption Standard
CBC	Cipher Block Chaining
CSMA-MPS	Carrier Sense Multiple Access with Minimum Preamble Sampling
CTR	Counter mode
DoS	Denial of Service attack
ECB	Electronic Codebook
KDF	Key Derivation Function
MAC	Message Authentication Code
MANET	Mobile Ad-Hoc Network
MMX	Multi Media Extension
OCB	Offset Codebook
OTP	One time pad
PDF	Pad Derivation Function
PRF	Pseudo Random Function
PRG	Pseudo Random Generator
SAFER	Secure And Fast Encryption Routine
SIMD	Single Instruction Multiple Data
SNEP	Sensor Network Encryption Protocol
SPINS	Security Protocols for Sensor Networks
STEM	Sparse Topology and Energy Management
TDMA	Time Division Multiple Access
TEA	Tiny Encryption Algorithm
WEP	Wired Equivalent Privacy
WSN	Wireless Sensor Network
µTESLA	Micro Timed, Efficient, Streaming, Loss-tolerant Authentication

CONTENTS

PAGE

1 INTRODUCTION 12

1.1. Background 12

1.2. Problem Statement 13

1.2.1. The Need for Confidentiality and Authenticity in Wireless Sensor Networks 13

1.3. Overview of Current Literature 14

1.4. Contributions 15

2 WIRELESS SENSOR NETWORKS 17

2.1. Wireless Sens or Network versus Mobile Ad-Hoc Networks 17

2.2. Properties of Wireless Sensor Network 21

2.2.1. Limited Computational Power 22

2.2.2. Limited Memory 22

2.2.3. Limited Energy Resources 22

2.2.4. Small Data Packet Size 25

2.2.5. Wireless Communication 25

2.2.6. Susceptible to Physical Capture 26

2.3. Existing Wireless Sensor Networks 26

2.3.1. PicoRadio 26

2.3.2. WiseNET 27

2.3.3. EYES 27

2.3.4. TinyMote from the Technical University of Vienna 28

2.3.5. Crossbow Smart Dust WSN 28

2.4. Comparisons between TinyMote and MICA2/MICA2DOT 29

2.5. Conclusions 33

3 SECURITY IN WIRELESS SENSOR NETWORK 34

3.1. Trust Models 34

3.2. Threat Models 35

3.3. Security Requirements 36

3.3.1. Confidentiality and Authenticity 36

3.3.2. Availability 39

3.4. Conclusions 41

4 CRYPTOGRAPHIC CIPHERS 42

4.1. TEA 42

4.1.1. Cryptanalysis of TEA 44

4.2. SAFER K-64 45

4.2.1. Cryptanalysis of SAFER K-64 50

4.3. TREYFER 50

4.3.1. Cryptanalysis of TREYFER 51

4.4. Other Block Ciphers 51

4.4.1. AES 52

4.4.2. RC5 53

4.5. Block Cipher Modes of Operation 54

4.5.1. Cipher Block Chaining (CBC) 54

4.5.2. Counter (CTR) Mode 57

4.5.3. Offset Codebook (OCB) 60

4.6. RC4 65

4.6.1. Cryptanalysis of RC4 66

4.7. Conclusions 66

5 UMAC 68

5.1. Standard UMAC 68

5.1.1. NH 70

5.1.2. Pseudo Random Generator (PRG) 72
5.1.3. Pseudo Random Function (PRF) 72

5.2. Refined UMAC 73
5.2.1. Key Derivation Function (KDF) 74
5.2.2. Pad Derivation Function (PDF) 75
5.2.3. UHASH 75

6 LINK LAYER SECURITY PROTOCOLS 77

6.1. Sensor Network Encryption Protocol (SNEP) 77
6.1.1. Confidentiality 77
6.1.2. Authenticity 78

6.2. TinySec 80
6.2.1. Confidentiality 80
6.2.2. Authenticity 81

6.3. Other Link layer Securities 82

7 IMPLEMENTATIONS 84

7.1. Implementation Environment 84
7.1.1. TinyMote Network Behavior 87
7.1.2. TinyMote Packet Structure 90
7.1.3. TinyMote Power Consumption 91

7.2. Block Ciphers 93
7.2.1. XTEA 93
7.2.2. SAFER K-64 94
7.2.3. TREYFER 94
7.2.4. OCB Mode 95

7.3. RC4 Stream Cipher 96

7.4. UMAC 97
7.4.1. UMAC-Block Cipher 97
7.4.2. UMAC-RC4 100

7.5. Conclusions 101

8 RESULTS AND DISCUSSIONS 102

8.1. Power Consumption in the MSP430 Microcontroller 102

8.2. Cryptographic Ciphers 103

8.2.1. Observation and Analysis 106

8.3. UMAC 107

8.3.1. UMAC-XTEA 107

8.3.2. UMAC-RC4 108

8.4. OCB-XTEA 109

8.5. Security Primitives Implementations 110

8.5.1. Observation and Analysis 114

8.6. Secure Link Layer Protocol 115

8.6.1. Block Cipher Based 115

8.6.2. RC4 Stream Cipher Based 118

8.6.3. Observation and Analysis 118

8.7. Security Primitives Power Consumption 119

8.8. Improvements to Existing WSN Link Layer Securities 120

8.9. Conclusions 122

9 CONCLUSION AND FUTURE WORK 123

REFERENCES 126

ADDENDUM A 132

LIST OF FIGURES 132

LIST OF TABLES 134

1

INTRODUCTION

1.1. Background

A wireless sensor network (WSN) is one in which tiny devices (sensor nodes), positioned fairly close to each other, are used to sense and gather data from their environment and to exchange information through wireless connections between these nodes. Apart from the built-in sensors, these sensor nodes also have wireless transceivers and power sources built-in allowing them to work autonomously. Sensor networks have been undergoing extensive research and studies in recent years because of their various potential applications, such as monitoring the safety and security of buildings or homes (intelligent buildings and homes), measuring traffic flows, tracking environmental pollutants, monitoring factory instrumentations, monitoring temperature and lightings on a farm or in a greenhouse. Sensor nodes can even be distributed throughout a bridge allowing them to continuously sense and monitor the mechanical stress level of the bridge. In order to easily deploy a relatively large number of sensor nodes, the sensor nodes are typically designed for low price, small size and long operation life, which causes them to have very limited resources available (e.g. energy, processing power and memory size).

When speaking of the security of any system, it can be categorized into three main concerns: Confidentiality, Integrity and Authenticity (or sometimes Availability) (C.I.A.). Security primitives are used to achieve these basic concerns. For example, encryption

algorithms are used to achieve confidentiality, while cryptographic hash functions or message authentication codes (MAC) are used for achieving integrity and authenticity. Over the years, different security primitives have been proposed and refined aiming at utilizing modern processing power e.g. 32-bit or 64-bit systems, SIMD (Single Instruction Multiple Data) architecture such as MMX (Multi Media Extension) etc. In other words, security primitives have targeted the high-end systems (e.g. desktop or server) in software implementations. Several hardware-oriented security primitives have also been proposed. However, most of them have been designed aiming only at large messages and high-speed processing, with no power consumption or other resources (such as memory space) taken into consideration. As a result, security mechanisms for ultra-low power devices such as wireless sensor nodes must be carefully selected or designed with their limited resources in mind. Ultra-low power at the moment is typically referring to power consumption less than 500μW.

1.2. Problem Statement

Many available WSN systems lack even the link layer security features or, in those cases where the security features are implemented, they are too resource intensive. In other words, data communicated in a WSN lacks basic security mechanisms such as confidentiality (privacy) and authenticity, which is a major problem in certain applications.

The objective of this thesis is to provide implementations of security primitives (i.e. encryption and authentication) suitable to the WSN environment, where resources are extremely limited. The goal is to provide an efficient building block on which the design of WSN secure routing protocols can be based, so that it can relieve the protocol designers from having to design everything from scratch.

1.2.1. The Need for Confidentiality and Authenticity in Wireless Sensor Networks

As pointed out in [7], many WSN applications have had to compromise on the security of the sensor network due to the constrained sensor node resources. To many sensor networks,

information confidentiality may not be too much of an issue e.g. sensor readings monitoring the habitats of a lake or temperature at different parts of a factory. However, information authenticity and integrity are needed in most WSN [7] [9], and they may be crucial to prevent catastrophic losses due to malicious attacks. For example, it would be undesirable to allow any person to modify sensor readings on the status of valves in a factory, and thereby causing incorrect system responses due to this wrong sensor reading. On the other hand, as sensor networks become more widely used, many other applications will need both information confidentiality and authenticity. For example, although people may not be concerned with others obtaining sensor readings monitoring habitat of a lake, they are more likely to be concerned if their personal living environment or the readings in a health monitoring application become known by others. Furthermore, sensor networks deployed in a military environment would also require confidentiality apart from authenticity of the information.

1.3. Overview of Current Literature

Wireless sensor network routing protocols, because of the extremely limited resources available, have been designed without a strong sense of security. Perrig et. al. [1] have proposed a security protocol named SPINS (Security Protocols for Sensor Networks) to provide data confidentiality, data authentication (both two-party and broadcast authentication) and data freshness. In other words, SPINS provides security for the link layer protocols. SPINS adopted a block cipher as its cryptographic primitive on which all of its operations are based (i.e. encryption, authentication and random number generation). However, as pointed out by Menezes et. al. in [2], this is not a very good implementation; the randomness of the random number generator that is based on a block cipher may lead to unexpected security problems. Another link layer security system, TinySec [6], was proposed several years after SPINS, also by UC Berkeley but by a different group of researchers. TinySec uses a different block cipher to provide authenticity and confidentiality. Karlof and Wagner [3] have conducted a study of different attacks on a variety of WSN routing protocols. They have shown that it is also not enough to apply link layer security protocols (i.e. SPINS or TinySec) directly to an existing routing protocol as a

security patch. According to Karlof and Wagner [3], it is unlikely a sensor network routing protocol can be made secure by incorporating security mechanisms after the rest of the design has been completed. Therefore, it is important to design a routing protocol with security in mind right from the beginning, which is beyond the scope of this paper. This paper is aimed at providing the security primitives needed to allow such secure routing protocols to be built on, or in other words, providing security at the link layer.

Ganesan et. al. [4] have showed comparisons of a variety of security algorithms with respect to their performance costs such as number of clock cycles, processing time and required code size. These comparisons are performed on a wide range of platforms, thus providing a good indication when choosing security algorithms for low-end embedded devices. Ganesan et. al. in [4] have provided a good guidance to several well-known security algorithms on a wide-range of microprocessors and microcontroller architectures. However, all algorithms in [4] are studied from a software implementations perspective; no hardware implementations of security algorithms have been studied (i.e. a security algorithm on a dedicated piece of hardware). Law et. al. in [5] have proposed further improvements over SPINS in [1] by categorizing WSNs into different “profiles”. It also used a different block cipher than the one used in SPINS, and was implemented on a Texas Instrument MSP430 microcontroller. Similar to the studies in [4], Law et. al. in [5] have also only concentrated on implementations of security algorithms from a software perspective with a microcontroller. Apart from focusing mainly on software implementations for microcontrollers, most current studies have also been focusing only on block ciphers.

1.4. Contributions

This dissertation provides three main contributions to the WSN field.

- Provides an analysis of different tradeoffs between cryptographic security strength and performance, which can then be used to select security primitives suitable for the needs of a WSN environment. Security primitives form the link layer security

and act as building blocks for higher layer protocols i.e. secure routing protocol.

- Implements and optimizes several security primitives in a low-power microcontroller (TI MSP430F1232) with very limited resources (256 bytes RAM, 8KB flash program memory). The different security primitives are compared according to the number of CPU cycles required per byte processed, specific architectures required (e.g. multiplier, large bit shift) and resources (RAM, ROM/flash) required. These comparisons assist in the evaluation of its corresponding energy consumption, and thus the applicability to wireless sensor nodes.

- Apart from investigating security primitives, research on various security protocols designed for WSN have also been conducted in order to optimize the security primitives for the security protocols design trend. Further, a new link layer security protocol using optimized security primitives is also proposed. This new protocol shows an improvement over the existing link layer security protocols.

Security primitives with confidentiality and authenticity functions are implemented in the TinyMote [12] sensor nodes (equipped with MSP430F1232 MCU) in a WSN. This is to demonstrate the practicality of the designs of this thesis in a real-world WSN environment.

The remaining of this dissertation is organized as follows: chapter 2 and 3 discuss and provide a background understanding on the existing WSNs and security concerns of WSNs; chapter 4 discusses various cryptographic ciphers and their applicability to the WSN environment; chapter 5 discusses the UMAC authentication algorithm; chapter 6 discusses several types of existing link layer security protocols in WSNs; chapter 7 shows how different cryptographic ciphers are adapted and implemented in this dissertation; chapter 8 and chapter 9 provide the results and analysis of the implementations.

2

WIRELESS SENSOR NETWORKS

Wireless technologies are becoming more advanced and mature with many off-the-shelf wireless chips available in the market. On the other hand, microcontrollers are also becoming smaller and consuming less power. It is therefore inevitable for the combination of the two technologies, in addition with sensor technologies, to result in wireless sensor networks. In order to deploy a fairly large-scale network of sensor nodes, the cost of individual nodes must be minimized. As a result, sensor nodes are often very resource constrained: small code size/memory space, limited computation power and limited energy resources. This has made security concerns in WSN very different from traditional network security systems. This chapter discusses these properties found in wireless sensor networks. A brief overview of existing wireless sensor networks and a comparison between the commercialized sensor node MICA2 and the TinyMote is also provided. The TinyMote is the sensor node used in this paper.

2.1. Wireless Sensor Network versus Mobile Ad-Hoc Networks

Before sensor network security are discussed in detail, it is important to first identify what a wireless sensor network (WSN) is and what it is not! First of all, a WSN should not be considered as an equivalent to a mobile ad-hoc network. When referring to mobile ad-hoc networks (also known as MANET), the nodes involved are often heterogeneous devices in nature and are usually mobile with no fixed position [8]. This means that the network

topology is highly dynamic with nodes joining, leaving or roaming in the network. These nodes may also be connected to the ad-hoc network for only a short amount of time. For example, an ad-hoc network may be temporarily formed at a conference meeting where the nodes involved are devices such as PDAs, cell phones, laptops or tablets (tablet PC). This kind of ad-hoc network is also known as spontaneous networking [23]. The nodes involved in an ad-hoc network may be any devices ranging from customized ad-hoc devices to cell phones or PDAs.

On the other hand, a wireless sensor network often has a more closed architecture with specific design goals and purpose e.g. measuring temperature at different places within a building. As a result, the nodes involved in a wireless sensor network are mainly identical in hardware (homogeneous) and are designed aiming at an extremely low-cost for a large amount of deployment. As pointed out by Akyildiz et. al. in [10], the number of sensor nodes in a WSN can be several orders of magnitude higher than the number of nodes in an ad-hoc network. Due to the extremely low-cost nature of WSN nodes, these nodes are usually even more resource constrained than most ad-hoc network nodes, with less memory and computation power in order to achieve lower cost and longer battery life. WSN are also usually immobile and with different traffic patterns to an ad-hoc network [9]. One or more base stations often exist in WSN. Base stations are more powerful nodes with rich computational, memory, energy and radio resources. A base station may exist in the form of a PC or server and it is where the sensor data flows to and is stored. It is therefore also known as the sink node. Base stations may act as a gateway between a WSN and another network; therefore base stations may be connected to outside TCP/IP networks. These resourceful nodes are sometimes also known as rich uncles [5]. Table 2.1 provides a summary of comparisons between MANET and WSN.

Table 2.1. A comparison between mobile ad-hoc network and wireless sensor network.

Mobile Ad-Hoc Network (MANET)	Wireless Sensor Network (WSN)
Open architecture allowing heterogeneous devices (nodes) (e.g. PDA, cell phone and laptop).	Closed architecture with specific design goals in mind therefore homogeneous devices (nodes).
Requires no interaction with the environment.	Requires interaction with the environment.
Dynamic topology (mobile nodes).	Both fixed (static nodes) and dynamic topologies exist.
Point-to-point communications.	Broadcast communication paradigm.
May not be using any base station.	Base station exists.
Nodes often have more resources available (memory, computation power, energy).	Nodes have less resources compared to ad-hoc networks.
Operational lifetime varies, often less than that of a WSN.	Need to operate for a longer period of time (<6 months).

The concept of an ad-hoc network has been around for longer than the concept of a wireless sensor network and so there is more literature on ad-hoc networks. Although both ad-hoc network security and WSN security need to be designed with resource constraints in mind, in many cases the security designs for ad-hoc network are still too heavy for the resources available in WSN in terms of memory usage and per packet processing overhead [9]. An example would be the secure routing protocol designed for ad-hoc network in [11], which is still too resource intensive to be implemented in WSN. WSN in general have fixed topologies, however, some sensor networks (e.g. EYES [21] and PicoRadio [31]) share some of the properties of a MANET i.e. they are ad-hoc and mobile. The security primitives in this paper are implemented and demonstrated on a fixed topology WSN, where sensor nodes are immobile. Figure 2.1 is a representation of a typical WSN architecture.

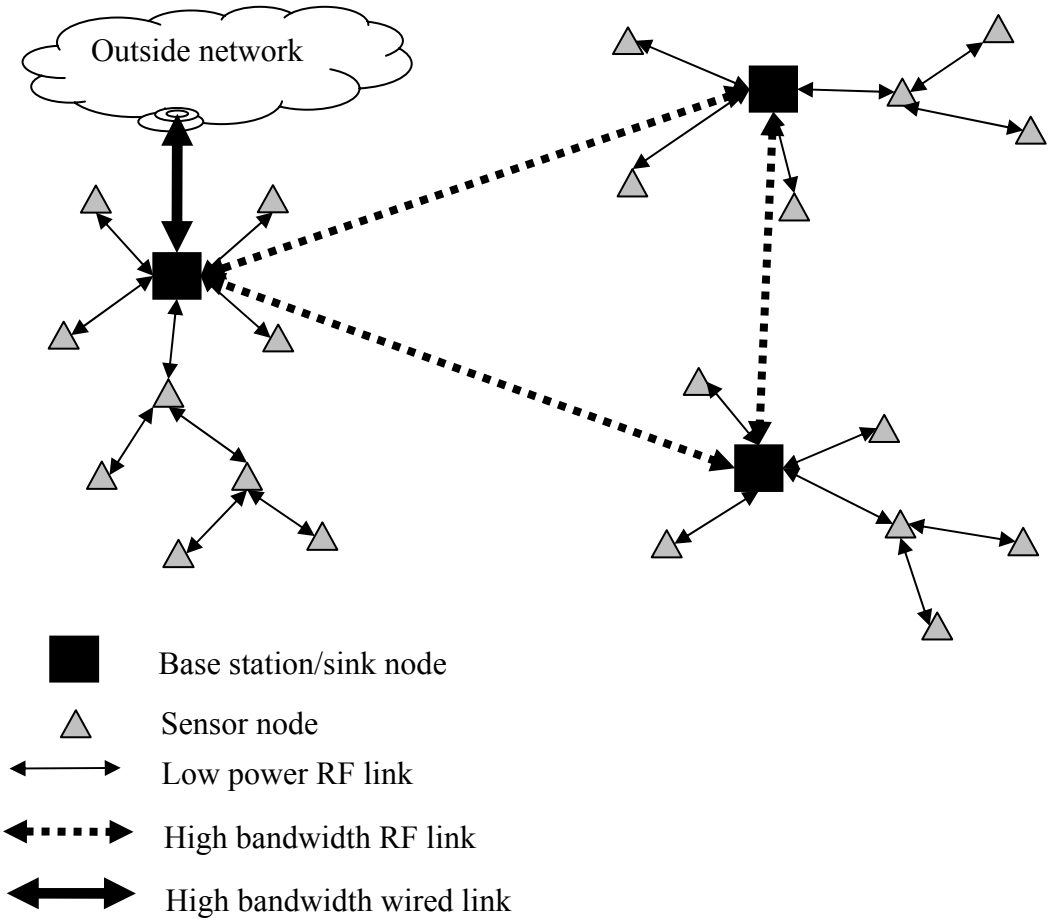

Figure 2.1. Representation of a wireless sensor network architecture.

Base stations (sink nodes) are often connected to each other via low latency, high bandwidth, and long distance RF links. A base station may also act as a gateway to be connected to a wired outside network (e.g. Ethernet). On the other hand, sensor node RF links have lower power consumptions which increase the sensor node's operating life due to the limited energy resources in sensor nodes.

2.2. Properties of Wireless Sensor Network

When designing security functions for WSN, there are many properties of WSN that need to be considered which are not present in conventional networks. Therefore, security primitives and protocols that are used in conventional networks may not be suitable for direct use in a WSN environment. As mentioned in the previous section, even security protocols designed for the already resource constrained ad-hoc network may not be suitable for WSN. Some of the properties of WSN and their consequences to security functions are noted in this section.

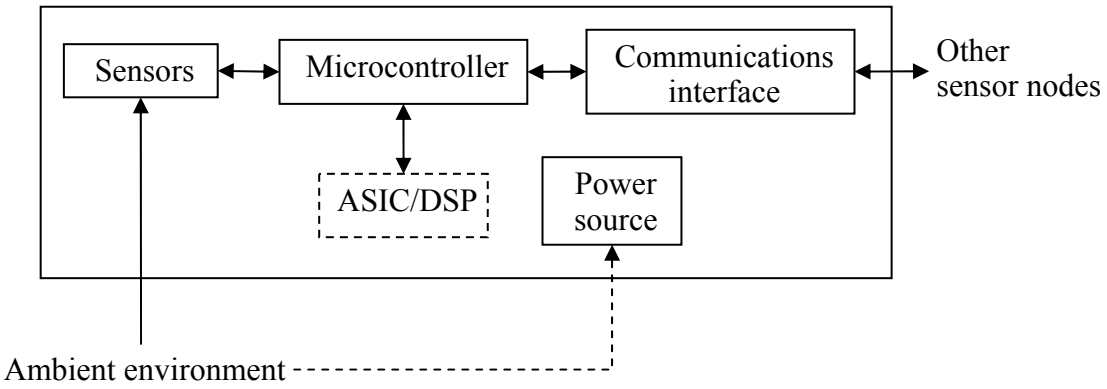

Figure 2.2. Functional diagram of a wireless sensor node.

Figure 2.2 shows the functional diagram of a typical wireless sensor node. In some sensor nodes the ambient environment can provide power to the sensor node, usually by converting light or acoustic noise into electricity. Therefore ambient environments not only provide sensor readings, but may also form an energy source. Some sensor nodes may also be equipped with application specific integrated circuits (ASICs) or digital signal processors (DSPs) to perform certain functions that may be too resource intensive for the general purpose microcontroller (e.g. cryptographic functions). Reconfigurable hardware (e.g. field programmable grid arrays (FPGAs)) may also exist to provide low level protocol functions that would otherwise be too resource intensive for the microcontroller.

2.2.1. Limited Computational Power

In order to achieve large-scale deployment of sensor nodes, a low-cost processing unit (e.g. a low-cost microcontroller) is often selected when designing sensor nodes. Furthermore, computational power is also often sacrificed in order to achieve lower power consumption. As a result, asymmetric encryption (public key cryptography) is not appropriate for WSN as it typically requires several times more computation than symmetric encryption, which violates the aim of conserving power [5] [9].

2.2.2. Limited Memory

Due to their low-cost and low-power nature, microcontrollers chosen for WSN purposes have very limited code memory and RAM. One example is TinyMote [12], which has only 256 bytes of RAM and 8Kbytes of flash. As a result, cryptographic functions requiring large code size or a lot of RAM are not suitable for WSN. For example, the block cipher AES needs large look-up tables which takes up code space; the stream cipher RC4 needs at least 256 bytes for a look-up table in RAM to provide the byte swapping mechanisms. Furthermore, the difference between microcontroller word sizes will also greatly influence the code size required [4]. For example, certain 8-bit microcontrollers may require up to 50% more code space when compared to a 16-bit microcontroller [4].

2.2.3. Limited Energy Resources

The most common power sources for WSN are the batteries. WSN need to be able to operate for a long time without human intervention, therefore every milliamp need to be used wisely. Power conservation can be said to be of highest importance in sensor networks. As sensor nodes may be deployed at places which are difficult to reach due to terrain obstacles, it is important for the sensor nodes to conserve energy so that it may last for a relatively long period of time (ranging from 6 months up to 10 years!).

When speaking of the ultra-low power consumption of sensor nodes, it is important to first understand what is meant by ultra-low power. The following table gives an overview on

the ranges between different devices' power consumptions.

Table 2.2. An overview of different devices' power consumptions.

Application	Power source	Power range
Desktop computer	Power grid	150W – 500W
Laptop	High capacity battery	10W – 120W
Mobile devices	Battery	5W – 10W
Handheld devices	Battery	100mW – 5W
Wireless sensors (including ad-hoc network nodes)	Tiny batteries	1mW – 100mW
Ultra-low power wireless sensors	Tiny batteries or energy scavenging	1μW – 500μW

As shown in Table 2.2, ultra-low power typically refers to power below 500μW. Energy scavenging is when energy is "scavenged" from the environment; for example converting energy from ambient acoustic noise, vibration, heat or light into electrical energy (Figure 2.2). Sensor nodes that are powered solely by power scavengers are called self-powered. Self-powered sensor nodes are autonomous and are suitable to be placed at locations that may be inaccessible after deployment [20].

Power consumption in WSN can be divided into three domains [10]: communication, data processing and sensing.

2.2.3.1. Power consumption in communication

Wireless transceiver is the major source of power consumption in WSN. Modern wireless transceivers are designed for lower and lower current consumptions and are now in the range of tens of milliamps. However, this power consumption is still too high for the ultra-lower power WSN environment, which as mentioned above, requires less than hundreds of micro-watts. As a result, in order to reduce power consumption due to communication,

sensor nodes have often been designed with low operational duty cycle and short data packet size.

A low duty cycle is one in which the sensor node (including the transceiver) is switched off or to a sleep mode when it is not transmitting any data, which allows the sensor node to remain in a power conserving mode most of the time. For example, the Intel mote [13] is aimed at maintaining an operational duty cycle of less than 1%. Ideally, in order to maximize efficiency and prevent wasted power, transmission or reception of data should take place right after the sensor node has woken up i.e. sensor nodes should be synchronized. This is the responsibility of a medium access control protocol[1] (a link layer protocol). Some example medium access control protocols are WiseMAC [15], STEM [16], BitMAC [17] and CSMA-MPS [18].

Transceivers have both wakeup and active power consumption. Most sensor nodes have been adopting low bit rate transceivers of up to 115Kbps, because the data volume is very small in WSN. However, recent research ([18] [19]) has shown that high bit rate (greater than 1Mbps) with short transmit and receive turn-around time can greatly reduce power consumption in communication. This is especially true for medium access control protocols that need to constantly switch between short intervals of transmit (TX) and receive (RX) mode (e.g. CSMA-MPS [18], STEM [16]); therefore reducing the transmit and receive turn-around time essentially reduces the start-up time and thereby also the wakeup power consumed.

2.2.3.2. Power consumption in data processing

In order to reduce power consumption, the time taken for local data processing at every node should be kept minimal. This also helps to achieve the aim of a low duty cycle. Different low-power microcontrollers are used in sensor nodes, for example MSP430 (16-

[1] The abbreviation for Medium Access Control is also MAC. Do not confuse this with the Message Authentication Code abbreviation MAC. Since this dissertation discusses security issues of WSN, MAC in this paper will only be used as message authentication code.

bit) in TinyMote [12], AVR ATmega 128 (8-bit) in Crossbow MICA motes [14] and ARM (16/32-bit) in Intel motes [13]. It is worth noting that different microcontroller word sizes will influence performance on different operations [4]. For example, a larger word size will benefit large bit size shift operations. As a result, encryption algorithms such as RC5 perform better in a 16-bit architecture than in an 8-bit architecture. On the other hand, RC4, which was designed for efficient byte operations, performs better in an 8-bit architecture [4]. The most common ways to conserve power consumed by microcontrollers is to switch between the different low-power modes provided by the microcontroller. However, as pointed out in [10], other low power techniques such as dynamic voltage scaling and CPU organization strategies still need to be explored. Similarly to wireless transceivers, microcontrollers also need wakeup time from the power saving sleep mode; therefore faster wakeup time will benefit power saving because sensor nodes need to frequently switch between sleep mode and active mode.

2.2.3.3. Power consumption in sensing

Sensor nodes may be used to collect various types of ambient data such as lighting, temperature, humidity, sound etc. Most sensor nodes have sensors designed as modular sensor boards allowing easy expansion [14][13]; therefore different sensor applications will have different power consumptions.

2.2.4. Small Data Packet Size

As discussed in the previous section, one of the most power consuming components in a sensor node is the RF transceiver. In order to reduce the usage of the transceiver, the size of packets sent wirelessly must be kept minimal [6] [7]. As packet size becomes smaller, the wakeup power for the transceiver becomes more important because the time needed to transmit a packet may be comparable to the transceiver wakeup time.

2.2.5. Wireless Communication

Wireless communication makes it possible for eavesdropping on a communication if it was

sent in plaintext. Physical jamming of the RF signal can also occur with wireless communication. However, it is possible to avoid such attacks by using frequency hopping and spread spectrum communication [9].

2.2.6. Susceptible to Physical Capture

Sensor nodes are often widely scattered across a large area, therefore it may be physically captured by an attacker. This characteristic, together with the characteristic that shared symmetric keys are often used in WSN, leads to the disadvantage that once a node is compromised, it means the secret key for the whole network is known. Law et. al. [5] consider tamper-resistant sensor nodes to be a must in WSN in order to protect the symmetric keys in all sensor nodes. However, Shi et. al. [9] and Karlof and Wagner [3] on the other hand consider that because sensor nodes are targeted for low cost, tamper-resistant hardware is unlikely to prevail.

2.3. Existing Wireless Sensor Networks

2.3.1. PicoRadio

PicoRadio [31] is a project at the Berkeley Wireless Research Centre (BWRC). It is designed to be a low-power self-configuring ad hoc network that covers a wide range of applications. Applications range from environmental monitoring, smart homes, warehouse inventories, health monitoring and drug administration in hospitals to interactive toys. In order to achieve low power, the PicoRadio project is aimed at addressing a complete system design throughout all protocol layers.

PicoRadio relies on dual radio transceivers: a main radio and a wakeup radio. The wakeup radio is an ultra-low-power transceiver that runs at 100% duty cycle. It monitors the wakeup channel for beacons to wakeup the main radio; therefore the main radio can remain switched off most of the time, thus saving energy. The power consumption goal for the ultra-low-power radio is 1 μW, however, this has not yet been realized. In the medium

access control protocol, because wakeup radio runs at 100% duty cycle, the sensor nodes require no synchronization as they can be woken up at any time. The main radio is a multi-channel transceiver which allows different channels to be used for communication to reduce collision, which also reduces power for retransmission.

2.3.2. WiseNET

Wireless Sensor Networks (WiseNET) [32] is a long-term research project at the Swiss Centre for Electronics and Micro-technology (CSEM). The main objective is to develop a low-power wireless ad hoc network with distributed sensor nodes combining sensing and signal processing capabilities.

Much of the design emphasis in WiseNET has been in the lower layers: in the medium access control protocol and the low voltage ultra-low power radio transceiver. The WiseMAC medium access control protocol is a novel design that combined concepts from time division multiple access (TDMA) and carrier sense multiple access (CSMA) protocols. A low voltage ultra-low power radio transceiver has also been designed for sensor network applications. The current consumption of the receiver is typically 2 mA at 0.9 V supply voltage. The average power consumption is 25 μW at a duty cycle of about 1.5%.

2.3.3. EYES

The EYES project [21] is a three year European research project to develop the architecture and the technology to enable the creation of sensors that can be networked together and support a large variety of mobile sensor applications. It is aimed at supporting various devices such as laptops, PDAs, mobile phones etc. As a result, the EYES network overlaps with both the fields of wireless sensor networks (WSN) and mobile ad hoc networks (MANET). In the EYES approach, two distinct abstraction layers are defined. One is the sensor and networking layer containing nodes and communication protocols up to the network layer. The other one is the distributed service layer containing lookup services and information services.

The EYES prototype wireless node has the following specifications: 16-bit 8 MHz microcontroller; 60 KB flash, 2 KB RAM; additional 1MB serial RAM; 868.35 MHz radio; and a bandwidth of 115.2 Kbps. EYES nodes use an operating system called PEEROS [35] (PreEmptive Eyes Real-Time Operating System), which requires about 10KB of code memory. The 16-bit microcontroller used is the Texas Instrument MSP430x149 [33], which is the same microcontroller series used in TinyMote [12]. The security primitives in this paper are implemented on TinyMote nodes.

2.3.4. TinyMote from the Technical University of Vienna

TinyMote [12] is a sensor node developed by the Technical University of Vienna (TUV). At the moment only medium access control protocols have been designed for TinyMote nodes, so the nodes have only been set up as a simple multi-node sensor network without a complex networking protocol. All security primitive experiments, comparisons and adaptations of this paper are based on TinyMote nodes.

The medium access control protocol, CSMA-MPS (carrier sense multiple access with minimum preamble sampling), combines design concepts from protocols WiseMAC and STEM [16]. CSMA-MPS has been shown to be more efficient than either WiseMAC or STEM, particularly when it is used with a high bit rate transceiver (greater than 1 Mbps) (refer to section 2.2.3.1 for more explanations). As a result, the average power consumption in TinyMote nodes due to radio communication is less than 50% of the total power consumption. A total power consumption of less than 200μW can be achieved (at 2 second sensor sampling interval); resulting in few years of battery life (two AA batteries). TinyMote nodes have the following specifications: 16-bit 1MHz microcontroller; 8 KB flash; 256 bytes of RAM; 2.4 GHz radio and 1 Mbps of bandwidth. The 16-bit microcontroller used is the Texas Instrument MSP430F1232 [33].

2.3.5. Crossbow Smart Dust WSN

Crossbow [14] is a company that manufactures the Mica series of commercialized sensor nodes. The Mica nodes inherit much of their design concepts from the UC Berkeley Smart

Dust nodes. MICA nodes also use the 8-bit Atmel AVR [34] microcontrollers and run the TinyOS operating system. MICA nodes use separate sensor board modules for different sensor requirements. The sensor board modules vary from light, temperature, humidity and acoustic to acceleration and GPS.

The medium access control protocol and the networking protocol are integrated into one protocol called the XMesh (previously known as Surge Time Sync or ReliableRoute). This is a network time synchronized protocol (within 1ms). The average battery life of MICA nodes ranges from a few months to less than 2 years (1.5 years at a 6-minute sampling interval). A typical MICA2 node has the following specifications: 8-bit 8MHz microcontroller (ATMega128L); 128KB flash (code); 512KB flash (data logging); 4KB EEPROM (configuration); 4KB RAM; ISM radio bands and a bandwidth of 38.4 Kbaud.

2.4. Comparisons between TinyMote and MICA2/MICA2DOT

In the previous section, several other existing sensor networks are briefly discussed. However, it is also important to have an understanding of how other available WSN differs from the TinyMote WSN platform used in this paper. The MICA2/MICA2DOT nodes are popular commercially available sensor nodes found in Crossbow Smart Dust WSN [14]. The tables below summarize comparisons between TinyMote and MICA2/MICA2DOT sensor networks. Both sensor networks have no security primitives implemented.

Table 2.3. Comparisons between TinyMote and MICA2/MICA2DOT sensor nodes.

	Power consumption			**RF current consumption**		
TinyMote	5s	10s		RX	TX (-5dBm)	TX (0 dBm)
	141µW	70µW		24mA	15mA	19mA
MICA2/ MICA2DOT	1 min	3 min	6 min	RX	TX (maximum power)	
	1.83mW	850.5µW	529.2µW	8mA (MICAz – 19.7mA)	25mA (MICAz – 17.4mA)	

(a)

	Memory				**word size**	**RF data rate**	**RF range**
TinyMote	8KB flash (code)	256 byte flash (config)	256 byte RAM		16-bit	1 Mbps	100m (LOS)
MICA2/ MICA2DOT	128KB flash (code)	512KB flash (code)	4KB EEPROM (config)	4KB RAM	8-bit	38.4 Kbaud (MICAz – 256 Kbps	304.8m (1000 ft) (LOS)

(b)

	Processor frequency	**OS**	**MAC[2] protocol**	**Networking protocol**	**Typical operation life**
TinyMote	1MHz @ 2.0V (1MIPS)	none	CSMA-MPS	under development	5.5 years @ 5s sampling interval
MICA2/ MICA2DOT	8MHz @ 2.7V (8MIPS)	TinyOS	XMesh	XMesh (Surge Time Sync)	<1.5 years @ 6min sampling interval

(c)

	Sensors	**Power source**
TinyMote	Light, temperature, humidity. (currently fixed sensor design)	Battery, solar cell with ultra capacitor
MICA2/ MICA2DOT	Light, temperature, humidity, acceleration, GPS, acoustic. (expandable sensor boards)	Battery

(d)

[2] MAC refers to Medium Access Control

It is observed that TinyMote WSN is aimed at lower level, less complex applications:

- Most sensors are fixed on board.
- No operating system, little memory space allowing essentially no user applications (low flexibility).
- Designed for difficult physical access of sensor nodes after deployment. Long operation life (up to 9 years).
- Lower costs per node (component-wise).

On the other hand, MICA2/MICA2DOT WSN can be observed as aiming at a higher level, more complex applications:

- Large varieties of sensors available.
- Large memory space together with the use of an operating system (TinyOS) allows multiple user applications (highly flexible).
- Allows remote sensor node data logging with data base.
- Physical access to sensor nodes should be relatively easy. Approximately 1 year battery life.
- Higher costs per node (component-wise).

2.5. Conclusions

In this chapter the key properties of a wireless sensor network (WSN) and how it differs from a mobile ad-hoc network (MANET) is discussed. Some of the existing WSNs are also briefly discussed. Therefore this chapter provides an understanding of the WSN and where its constraints such as memory and energy resources may lie.

3

SECURITY IN WIRELESS SENSOR NETWORK

Security in any network system does not simply involve only one or two layers, but rather needs to be viewed across all layers as a whole. The security issues for a conventional network differ greatly to the security issues in WSNs because of the extremely limited resources available in sensor nodes. This chapter provides an overview of security considerations in the context of the WSN.

3.1. Trust Models

One or more base stations often exist in WSN. Base stations are more powerful nodes with rich computational, memory, energy and radio resources. By radio resources it means that they have more powerful transceivers for a wider communication range and higher bandwidth links for communication amongst other base stations. A base station may exist in the form of a PC or server, where the sensor data flows to and is stored. Therefore they are also known as sink nodes. Base stations may act as a gateway between WSN and another network; therefore may be connected to an outside TCP/IP network. These resourceful nodes are sometimes also known as rich uncles [5]. Base stations are more expensive nodes, and are often assumed to be physically protected or have tamper-proof hardware.

As a result, in a WSN environment, a base station usually plays the role of a central trusted

authority (point of trust). A point of trust base station is what the other standard sensor nodes trust for its authenticity and accepts the keys managed by the base station. In a base station trust model, for two nodes to communicate directly with each other, they need to first rely on the base station to establish a shared secret key between them before communication can take place. However, scalability may become a problem for base stations. If every sensor node in the network has a unique secret key, then for two nodes to communicate with each other they need to first go through the trusted base station to establish a shared secret key. If every node needs to communicate with its neighboring nodes, then the base station becomes a scalability bottleneck. This paper also assumes the base station as the trusted authority in the trust model.

3.2. Threat Models

Attacks in the WSN can be categorized into insider and outsider attacks. Outsider attacks occur when the adversary is not an authorized participant of the sensor network. The adversary may be a passive attacker by eavesdropping wireless communications and tries to steal confidential data. Active attack may also take place in the form of spoofing or altering packets in order to infringe authenticity of communication; or injecting interfering wireless signals to jam the network. Disabling a sensor node is another form of outsider attack. The adversary can inject useless packets to drain the receiver node's battery power (a type of denial-of-service (DoS) attack) to disable a node. The adversary can also physically capture or destroy a sensor node to disable it. However, it is important to note that not all disabled nodes are a result of an attack. Catastrophic climatic change or battery depletion can also result in a failed node, which is indistinguishable from a disabled node.

Insider attacks are essentially referring to compromised nodes. Unlike outsider attacks which may result in disabled nodes, compromised nodes continuously disrupt or paralyze normal operations of the sensor network. A compromised node may exist in the form of a physically captured and reprogrammed sensor node; or it can be a different device (e.g. laptop) with more resources such as computational power, energy resources, memory resources and powerful radio signals. According to Shi and Perrig [9], a compromised

node has the following properties:

1. The node runs some malicious code that is different from the code on a legitimate node.

2. The node has compatible radio (in respect to spectrum, modulation scheme etc.) to other legitimate sensor nodes so that it can communicate with the sensor network.

3. The node is an authorized participant in the sensor network. If the communication is encrypted and authenticated using cryptographic primitives, then the compromised node must be in possession of the cryptographic keys of a legitimate node in order for communication to take place.

3.3. Security Requirements

When speaking of security requirements of any system, it is usually referring to: confidentiality, authenticity, availability and integrity. However, due to the extremely resource constrained nature of sensor nodes, security requirements may be very different compared to conventional networks. As a result, with sensor networks security, the aim is to achieve security requirements discussed in this section to make WSN robust against outsider attacks. However, in the case of insider attack, a graceful degradation of performance in proportion to the number of compromised nodes is aimed. For this is because the detection of compromised nodes is not always possible; therefore it is most likely that not all security requirements can be achieved.

3.3.1. Confidentiality and Authenticity

In WSN, data packets are communicated in a shared wireless medium, making them susceptible to eavesdropping and injection of malicious or spoofed packets. Data encryption protects confidentiality of data packets communicated. Authentication allows

verification of the packet sender to be a legitimate source (source authentication), and also verifies that the packet has not been modified (data authentication).

Encryption provides some confidentiality which protects the data packets from being revealed to passive attackers through eavesdropping. Encryption can be achieved using standard cryptographic primitives such as block ciphers (e.g. AES, TEA) or stream ciphers (e.g. RC4). However, encryption alone can not completely protect the confidentiality of data. A passive attacker can still perform traffic analysis on the encrypted data packets (ciphertext) which may reveal sensitive information about the data. On the other hand, an attacker can obtain the secret key used for encryption in a compromised node. If this secret key is shared globally amongst all sensor nodes, then the attacker will now be able to decrypt all communicated data packets in the sensor network.

Authentication ensures that the communicated data packets are from legitimate nodes and that they have not been changed by an attacker (authenticity check) or corrupted due to bad radio signals (integrity check). Authentication can be achieved by appending a message authentication code (MAC) to the data packet. A message authentication code is a piece of fixed size code that is computed from processing the data message with a key and a MAC function.

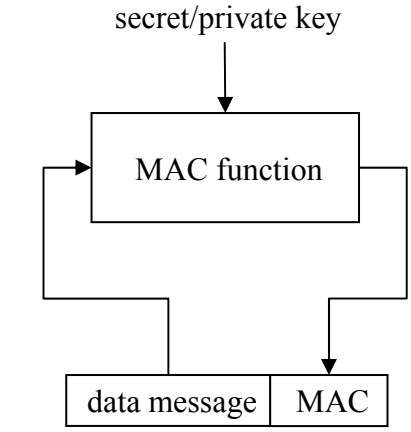

Figure 3.1. Representation of a data packet with MAC appended.

To verify that a received data packet is indeed from a legitimate node that has the secret key and the data packet has not been modified by an attacker; the receiver has to compute the received data for another MAC and compare this MAC with the one that was appended to the data packet. If the two MACs are different, then either the data packet has been modified by an attacker, or it has been corrupted due to bad radio signals.

Message authentication codes can be calculated in many ways. For example, using a CBC-mode block cipher, cryptographic hash functions with HMAC, or other custom designed MAC functions (e.g. UMAC or Message Authenticator Algorithm (MAA)). The figure below shows different categories of MACs.

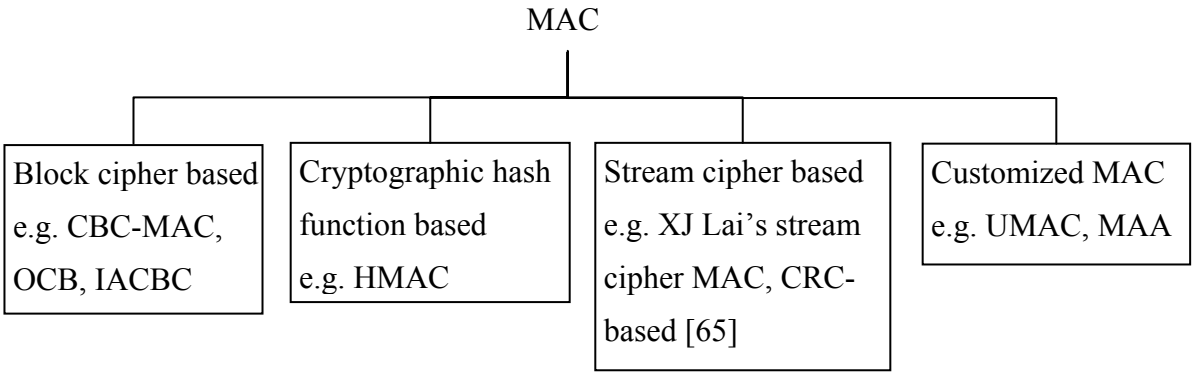

Figure 3.2. MAC categories.

In many scenarios, using a public key cryptography (asymmetric encryption) for authentication would be more desirable than using a shared secret key (symmetric encryption). In a shared secret key environment, if one sensor node is compromised and its secret key revealed, then the security is breached for all other nodes with the same secret key. When using public key cryptography, a sensor node will only be able to reveal its own private key if it is compromised. However, public key cryptography typically requires many more orders of computational cost and packet overhead, which makes it unsuitable for WSN even if it is used for key establishment when sensor nodes are initially installed [9]. Another problem for using public key cryptography is that it may lead to easy denial of service attack (DoS). Since the MAC verifying process is very computationally intensive, therefore if an attacker continuously sends out false packets requesting nodes to verify them, then it will greatly waste the receiving nodes' resources only to realize that the data packet it is a fake. As a result, public key cryptography is generally not considered for implementing WSN securities [1][5][9]. There are other researches on the key establishment for WSN, such as random key predistribution schemes [22], but are beyond the scope of this paper.

3.3.2. Availability

Availability refers to when the sensor network is functional throughout its designed

operation lifetime. The type of attack on network availability is often referred to as denial of service attack (DoS). DoS attack can occur in different layers of the network, such as physical, link and network layer. There are more different types of DoS attacks in the routing protocols at the network layer.

In the physical layer, DoS attack is achieved by jamming or interfering with the radio signals of the sensor network. This attack can also inject irrelevant data packets to a node in order to drain its energy resource (i.e. battery power) for radio reception. This type of physical layer DoS attack can be prevented by using frequency hopping and spread spectrum communication [9].

A link layer protocol, particularly the medium access control protocol can be exploited to achieve link layer DoS attack. For example, an attacker may cause malicious collisions or attempt to get an unfair share of the radio resource.

Karlof and Wagner in [3] have discussed several routing protocol attacks in the network layer. These attacks may all lead to DoS attacks.

- Spoofed, altered, or replayed routing information – This attack is to alter routing information exchanged amongst nodes to create routing loops, attract or repel network traffic, extend or shorten source routes, generate false error messages, partition the network, increase end-to-end latency etc.

- Selective forwarding – The malicious node may behave like a black hole by refusing to forward all or certain data packets it sees. However, this malicious node often needs to include itself in an actual path of the data flow.

- Sinkhole attacks – This attack tries to lure almost all the traffic of a particular area to go through a malicious node. This can be achieved by making the malicious node appear as a better choice of route to the surrounding nodes for sending data packets.

- Sybil attack – In a Sybil attack, a single malicious node fakes multiple identities to other nodes in the network. This can reduce the effectiveness of fault-tolerant schemes.

- Wormholes – This attack is where the adversary tunnels data packets received in one part of the sensor network and replay them in a different part. Wormholes can be used to create sinkholes.

- HELLO flood attack – Many routing protocols require nodes to broadcast HELLO packets to announce themselves to their neighboring nodes, so that their neighboring nodes will know who is within their radio range. An malicious node can use a high power transmitter to broadcast HELO packets and thus fooling every node in the network into thinking that the malicious node is its neighbor.

- Acknowledgement spoofing – Some routing protocols rely on implicit or explicit link layer acknowledgements. An adversary may spoof these acknowledgements to convince the sender that a weak link is strong or that a disabled link is alive, thereby causing packets sent to these links to be lost.

3.4. Conclusions

In this chapter the security concerns and security models of WSNs have been discussed. It is clear that the security solutions for the traditional networks are not entirely applicable to wireless sensor networks. This is because of the different characteristics and constraints found in WSNs.

4

CRYPTOGRAPHIC CIPHERS

Cryptographic ciphers often provide the most basic security requirements such as confidentiality, authenticity and integrity checking in any system. However, not all cryptographic ciphers that are suitable for conventional networks will also be suitable for WSNs. This chapter discusses security primitives through the use of cryptographic ciphers and their applicability to the ultra-low power WSN environment. The background of block ciphers as well as modes of operation are investigated and discussed here. The only stream cipher implemented in this paper, RC4, is also discussed here.

4.1. TEA

TEA (Tiny Encryption Algorithm) [36] and its related variants (XTEA, Block TEA, XXTEA) are symmetric key block ciphers designed for modern 32-bit word architecture. The emphasis of TEA is on small code size and easy implementation with typically few lines of codes. It uses a large number of iterations rather than a complicated algorithm. All TEA and its variants are based on the Feistel structure, every TEA cycle consists of two Feistel rounds (Figure 4.1).

TEA and XTEA operate on two 32-bit words as a 64-bit data blocks with a 128-bit key, therefore all operations are done in 32-bit words. Block TEA and XXTEA operate on variable-length blocks of arbitrary multiples of 32 bits size. The advantage of Block TEA

and XXTEA is that it eliminates the need for using a mode of operation (CBC, OFB, CFB, OCB etc.) on messages larger than one block. i.e. they can be applied directly to a complete message.

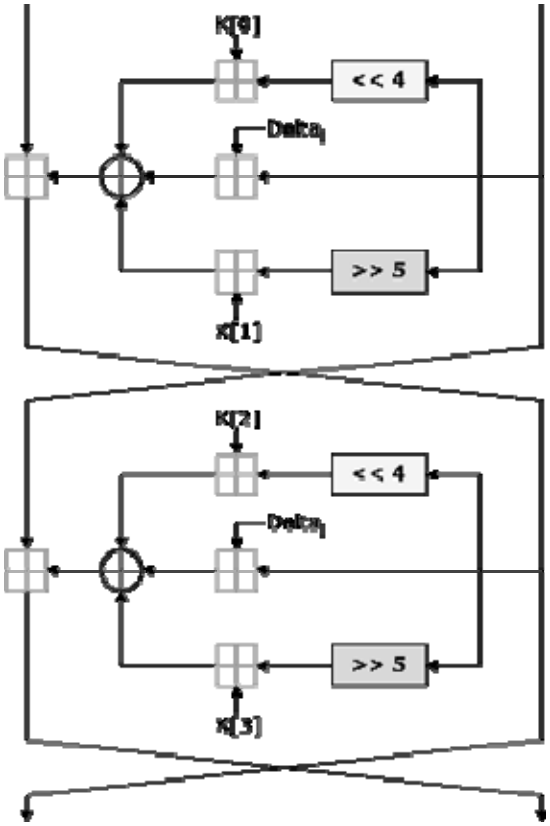

Figure 4.1. One TEA cycle (two Feistel rounds) [37].

XTEA (also known as TEAN) (Figure 4.2) was proposed in response to several weaknesses found with TEA. XTEA has the same basic operations as TEA, but the subkeys are mixed less regularly particularly to prevent key-schedule attacks on TEA. On the other hand, XXTEA is the updated version of Block TEA, as it prevents weaknesses found in Block TEA.

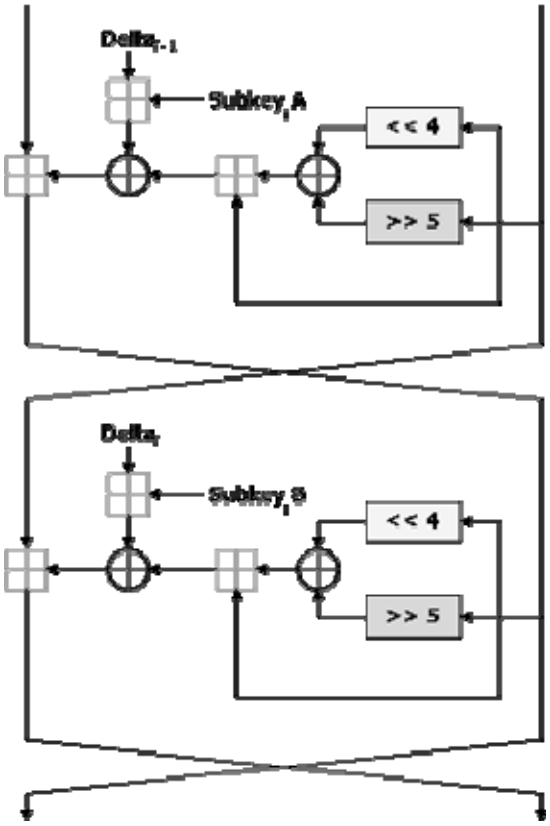

Figure 4.2. One XTEA cycle [37].

4.1.1. Cryptanalysis of TEA

TEA suffers from two types of cryptanalysis, the related-key [39] and equivalent-key [38] attacks. The equivalent-key attack is targeted at TEA's extremely simple key-schedule. This results in the problem that when flipping the most significant bits of the first two 32-bit words of the key, the encryption will not be affected. This attack has allowed hackers to successfully run Linux operating system on the Microsoft's Xbox gaming console. The best related-key attack on TEA requires 2^{23} chosen plaintexts and 2^{32} computation time to recover the key. XTEA is proposed by TEA designers to prevent weaknesses found in TEA. The best attack so far on XTEA is a related-key differential attack on 27 rounds [40]. This attack requires $2^{20.5}$ chosen plaintexts and has a time complexity of $2^{115.15}$ 27-round XTEA encryptions.

Block TEA key can be recovered with 2^{34} chosen ciphertext queries [41]. XXTEA was proposed to fix the weaknesses found in Block TEA. The best attack so far on XXTEA requires 2^{80} chosen plaintexts on 6 rounds, and a time complexity of 2^{127} to recover the key [41].

4.2. SAFER K-64

SAFER K-64 [44] (stands for Secure And Fast Encryption Routine with a Key of length 64 bits) is a non-proprietary secret (symmetric) key block cipher. The block length is 64 bits (8 bytes) and only byte operations are used for key scheduling, encryption and decryption.

The encryption structure of SAFER K-64 is shown in the following figure.

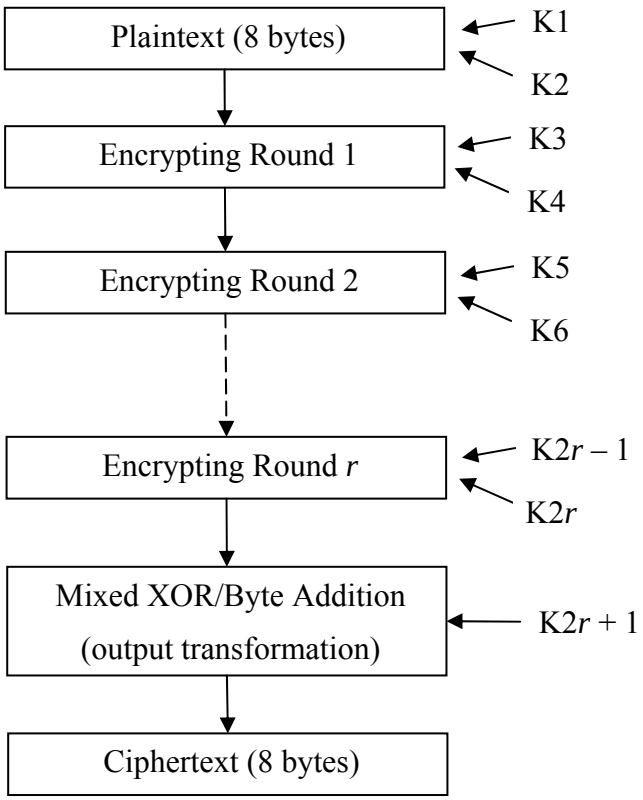

Figure 4.3. Encryption structure of SAFER K-64.

The encryption/decryption algorithm consists of r rounds, typically 6 rounds are recommended. Each round (shown in Figure 4.4) requires two 64-bit (8 bytes) subkeys and the output transformation needs one 64-bit subkey. In total $2r + 1$ subkeys are needed, which is derived from the user-selected secret key "K1". The output transformation involves byte XOR and byte addition (modulo 256) of the last subkey (K2r + 1) with output from the r-th round. The decryption structure is similar to the encryption structure except that the output transformation now becomes the input transformation and is executed first. The subkeys in the decryption structure are also used in a reversed order.

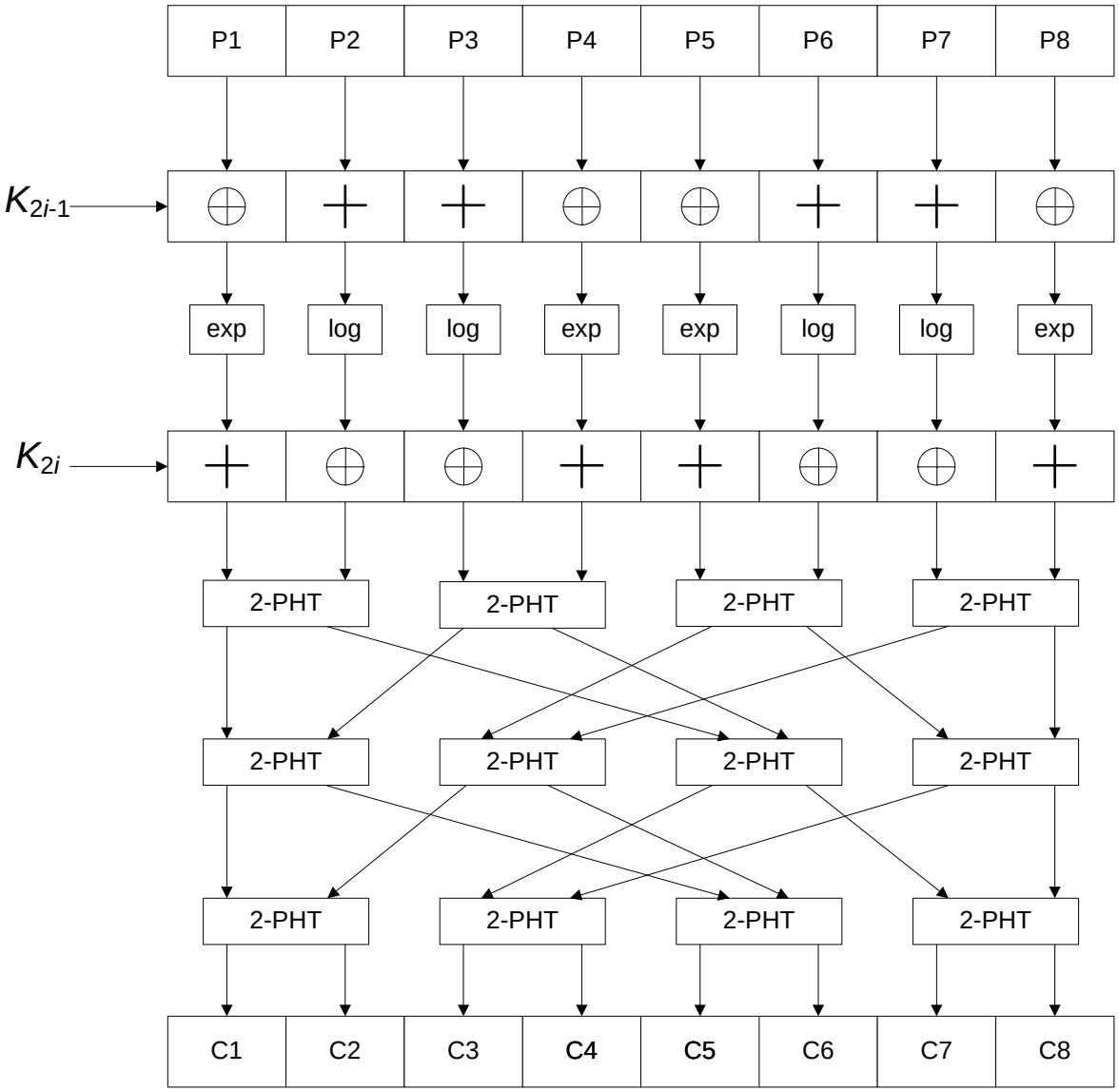

Figure 4.4. One encryption round structure of SAFER K-64.

As shown in the above figure of one encryption round, the 8 input bytes (64 bits) are represented by symbols "P1" to "P8", the output bytes are represented by symbols "C1" to "C8". The "⊕" and "+" symbol represents byte XOR and byte addition (modulo 256) respectively with the subkey bytes. If the input is represented by " j ", then the "exp" box output is: $45^{(j)}$ modulo 257, (except that this output is taken to be 0 if the modular result is 256); and the "log" box output is: $\log_{45}(j)$, (except that this output is taken to be 128 if the

input is: j = 0). These two transformations are nonlinear and invertible, which is favorable to a cipher algorithm. These two transformations can also be realized using two look-up tables of 256 bytes each, thus exchanging code size for computational time (which also saves energy). The "2-PHT" box stands for 2-point Pseudo Hadamard Transform, which is a linear transformation to achieve diffusion of even small changes in the plaintext. The 2-PHT consists of only three addition (modulo 256) operations.

The decryption round structure is similar to the encryption round structure. Instead of using 2-PHT, 2-IPHT (2-point inverse pseudo Hadamard Transform) is used. Both subtraction and addition (modulo 256) operations are required in 2-IPHT. The "2-IPHT" boxes are executed first in the decryption round, then followed by the non-linear transformations of exponential and logarithmic operations.

The key scheduling algorithm to derive the needed keys is shown in the following figure.

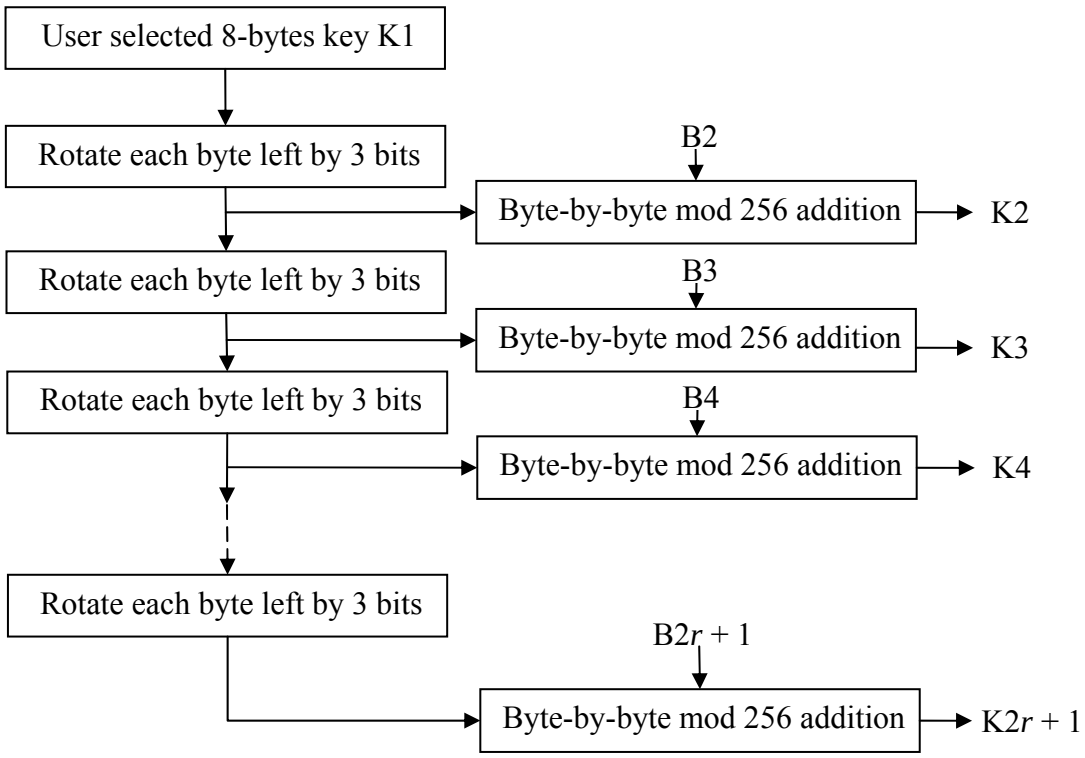

Figure 4.5. Key scheduling algorithm of SAFER K-64.

The key scheduling algorithm takes the user-selected key K1 to derive K2r + 1 number of subkeys. The key biases: B2, B3, … , B2r+1 are 8-byte fixed values introduced to ensure that the subkeys appear to be random, thereby eliminating the possibility of user-selected weak keys. The key biases can be calculated using the following formula:

$$b[i, j] = 45^{V} \bmod 257 \tag{4.1}$$

$$\text{where} \quad V = 45^{9i+j} \bmod 257 \tag{4.2}$$

Where b[i,j] denotes the j-th byte of bias Bi. If the exponential of 45 has been implemented as 256 bytes of look-up table as mentioned earlier, then the calculation of this formula can be simplified. The key biases can also be directly implemented as a look-up table, which further reduces computation time needed for setting up key biases. With 6 round SAFER

K-64 (r = 6), 12 key biases (8 bytes each) are needed, therefore a look-up table of 96 bytes is required.

4.2.1. Cryptanalysis of SAFER K-64

The best cryptanalysis on SAFER K-64 published so far, to the knowledge of the author, is by Wu et. al. in [45]. Wu et. al. has successfully applied a truncated differential attack on SAFER K-64 and SAFER SK-64 (with the modified key schedule) using 5 or 6 rounds. For an attack on 5-round SAFER K-64, 2^{38} chosen plaintexts and computation time of 2^{46} 5-round encryptions are required. For attack on a 6-round SAFER K-64, 2^{53} chosen plaintexts and a computation time of 2^{61} 6-round encryptions are required. This attack cannot be applied to SAFER K-64 with 7 rounds or more. Therefore it is suggested that an 8-round SAFER K-64 should be considered invulnerable to truncated differential attacks [45].

4.3. TREYFER

TREYFER is a 64-bit block cipher with 64-bit symmetric key and is proposed by Yuval [42]. It is aimed at applications with extremely limited resources, e.g. smart card and is designed to be very compact (less than 50 bytes of code on an 8051 microcontroller with assembler language). It can be executed on a very constrained architecture, for example an 8051 microcontroller with typically 1 KB flash EPROM, 64 bytes RAM, 128 bytes EPROM and a peak instruction rate of 1 MHz. TREYFER is designed to use only byte operations and requires fixed bit rotations and modulo 256 additions. The algorithm is as follows:

```
for (r = 0; r < NumRounds; r++){
        text[8] = text[0];
        for(i = 0; i < 8; i++)
                text[i+1] = (text[i+1] + Sbox[(key[i]+text[i])%256]) <<< 1;
                //rotate 1 left
        Text[0] = text[8];
}
```

In the above pseudo code, "text" represents the 8-byte plaintext, "Sbox" is the 256×8-bit (256 bytes) S-box chosen at random, and "NumRounds" is the number of rounds executed in TREYFER, which is typically 32. One of the motivations of the TREYFER design is the use of a large number of rounds (32) to thwart any possible practical attacks in spite of the simple round function design. The S-box was suggested by the author to be taken from another place in the memory running non-cryptographic codes. In this way there is no need to explicitly define a 256-byte S-box and thus code space is saved.

4.3.1. Cryptanalysis of TREYFER

An attack on TREYFER has been found by Biryukov and Wagner in [43]. It requires 2^{32} known plaintexts, 2^{44} TREYFER encryptions of computation times and 2^{32} memory. This proposed attack is also independent of the number of rounds and of the choice of the S-box. However, round counters can be introduced into the round function of TREYFER as a more complex key scheduling method in order to thwart such an attack. Besides this attack proposed by Biryukov and Wagner, no other attacks on TREYFER have been published to the knowledge of the author.

4.4. Other Block Ciphers

This section briefly discusses some other popular and secure block ciphers that are available, but are not implemented in this paper.

4.4.1. AES

AES (Advanced Encryption Standard) was published by NIST (National Institute of Standards and Technology) to replace DES (Data Encryption Standard). Out of the many candidates for AES, the Rijndael cipher was eventually selected to become the new AES [26]. AES is a symmetric key block cipher with a block size of 128 bits and three key size alternatives of 128, 192, or 256 bits.

Unlike many conventional symmetric key block ciphers, AES does not use the Feistel structure, where typically half of the data block is used to modify the other half of the data block before the two halves are swapped in the next round. AES processes the entire data block (128 bits) in parallel during each round. AES typically has 10 rounds; each round has four different stages, one of permutation and three of substitution. The encryption and decryption functions in AES differ. The encryption and decryption speed does not vary significantly, however, the key setup performance is slower for decryption and requires more memory than for encryption. All AES operations can be byte operations allowing it to be efficiently implemented on 8-bit processors. Its operations can also be defined in 32-bit words for efficient implementation on 32-bit processors [26].

Although AES has been well studied over the years and proven to be secure, it does not seem to be suitable for the platform which this paper is based on, or in many other WSN environments. One of the main reasons is that although AES has been designed for low-end 8-bit microcontroller, its baseline version still uses over 800 bytes of look-up tables. A speed optimized AES version, which runs about 100 times faster, uses over 10 KB of look-up tables. This memory requirement is not acceptable to many sensor node platforms. For example, the microcontroller MSP430F1232 used in the sensor nodes (TinyMote) of this paper has only 8KB of flash code memory in total. Apart from the large code size, AES also requires large RAM space to store expanded subkeys, typically larger than 156 bytes. Furthermore, because of the small packet size of WSN, a cipher with 128-bit (16 bytes) block size may not be very efficient. For example the last cipher call may only need to encrypt the last two bytes of the data packet, since the cipher uses 16-byte block, then the other 14 bytes of processing are wasted.

More on the performance comparisons of AES to other block ciphers is discussed in the results and discussions chapter (section 8.2).

4.4.2. RC5

RC5 is a symmetric encryption algorithm with a block size of 32, 64, or 128 bits [26]. The key length ranges from 0 to 2040 bits. RC5 encrypts two-word blocks, for example a 32-bit block has a word size of 16-bit. The maximum number of RC5 rounds is 255, but typically 12 rounds encryption/decryption algorithm is suggested.

RC5 has a simple structure similar to a Feistel structure. Instead of half of a block being updated as in the classic Feistel structure, both halves are updated in each RC5 round [26]. RC5 uses only three primitive operations: modulo 2^n addition/subtraction (n is the word size), XOR, and circular rotation. The encryption/decryption algorithm is very simple and can be implemented in few lines of codes. These characteristics make RC5 suitable for both hardware and software implementations. RC5 requires complex key expansion operations on user-selected secret keys. The number of subkeys that are needed is $2r + 2$, where r is the total number of rounds.

RC5 has also been around for some years and appears to be secure. Although it was designed to be of small size for efficient software and hardware implementation, its smallest word size is still 16-bit. The key setup operations have been shown to be very time consuming [5] and also require a relatively large amount of RAM space to store the expanded subkeys [54]. Furthermore, RC5 rotation operations are data-dependent, meaning that it has to rotate variable number of bits and often requires a large number of bit rotations. This large number of bit rotations is especially time consuming for processors with a word size smaller than that of the RC5 word size (e.g. 16-bit RC5 word on an 8-bit processor).

Law et. al. [54] have compared RC5, AES and several other block ciphers on the same family of microcontrollers (TI MSP430) as the one used in this paper. These comparisons

have shown that RC5 is not the most efficient cipher nor does it have the smallest code size. More on the performance comparisons of RC5 with other block ciphers is discussed in the results and discussions chapter (section 8.2).

4.5. Block Cipher Modes of Operation

Block ciphers provide encryption (confidentiality) to a fixed size block of data (e.g. 64 bits block). In order to provide confidentiality and authenticity security primitives using block ciphers, different block cipher modes of operation need to be used.

The easiest way of encrypting messages larger than a block is by using the electronic code book (ECB) mode. The message is divided into smaller chunks or blocks of a fixed size. ECB simply encrypts each plaintext block separately. However, with ECB mode, the same plaintext block will always result in the same ciphertext block when using the same key. In other words with a given key, every plaintext block corresponds to a unique ciphertext block. This will allow the attacker to draw up a table to map all plaintext-ciphertext pairs or perform statistical analysis on the ciphertext available; thereby finding out what is the plaintext for each corresponding ciphertext.

4.5.1. Cipher Block Chaining (CBC)

CBC mode can be used to provide encryption (confidentiality) of large messages as well as authenticity through CBC-MAC.

4.5.1.1. CBC mode for confidentiality

As shown in Figure 4.6, CBC mode divides the message to be encrypted into N number of blocks. The first message block m_1 is XORed ($\oplus$) with the initialization vector (IV) before encryption (E_k) with key k, thereafter every next message block will be XORed with the previous ciphertext block and then encrypted to obtain the next ciphertext block.

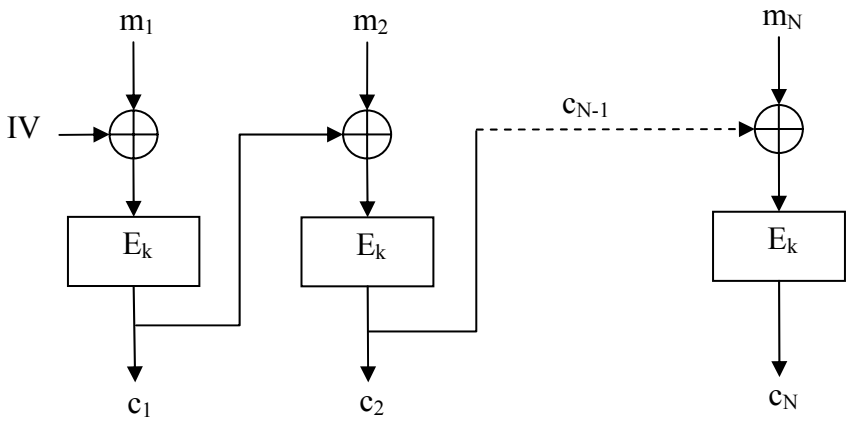

Figure 4.6. Cipher Block Chaining (CBC) mode encryption.

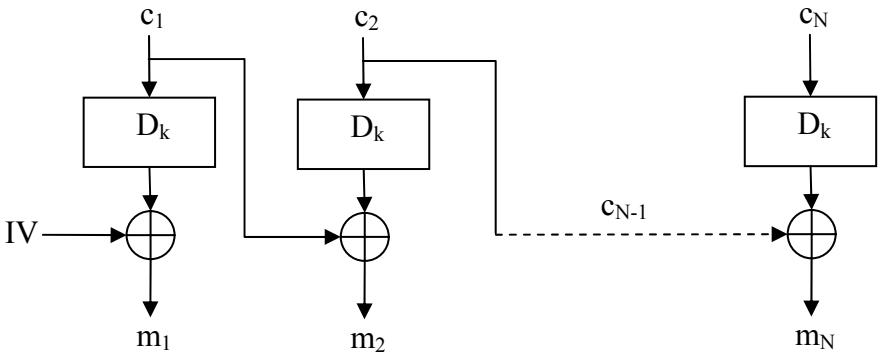

Figure 4.7. Cipher Block Chaining (CBC) mode decryption.

The IV needs to be known to both sender and receiver (decryption at receiver side is shown in Figure 4.7). IV provides the variation needed to ensure that every resulting ciphertexts will be different even if the same plaintext is being encrypted again. As a result, IV needs to be carefully selected with the following properties:

- IV must be random.
- Every new message must use a new, non-repeating IV.
- For maximum security, IV must not be a counter.
- IV can be a non-secret value.

CBC mode for encryption is provably secure when IVs do not repeat [46]. However, if IV repeats, only the length (in blocks) of the longest shared prefix of the messages is revealed. Therefore information leakage is not as bad as in the case of a repeated counter in the counter (CTR) mode, which we will discuss in section 4.5.2. More information leakage than a repeated IV will occur if IVs are simply taken from a counter. As a result, an IV is often derived from encrypting a nonce (use once only value), which can be an encrypted counter.

4.5.1.2. CBC mode for authenticity

A Message Authentication Code (MAC) is a fixed value calculated from data message with a key. It is often appended to the end of a data packet by the sender. If the data message has been altered, then the receiver calculated MAC will not be the same as the appended MAC. Figure 4.8 shows the standard CBC-MAC algorithm to generate a MAC.

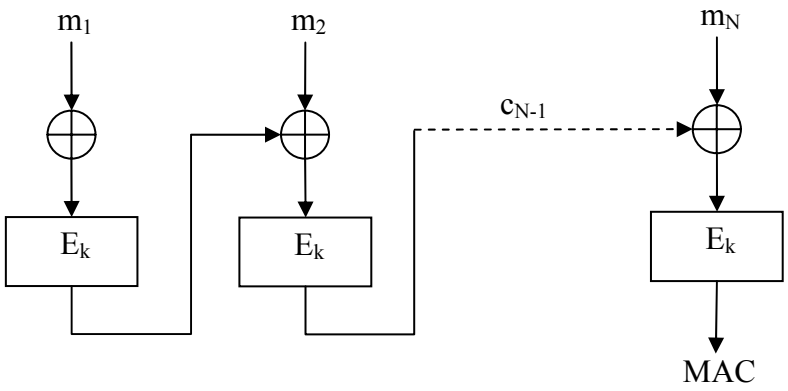

Figure 4.8. Standard CBC-MAC.

CBC-MAC operates exactly the same as the CBC mode encryption. Symbol *k* is the key for calculating MAC and *N* is the number of blocks that a message is divided into. The last output ciphertext is taken as the MAC. Therefore if a block cipher with a block size of 64-bit is used, then the resulting MAC can range from 1 to 64 bits.

Although the standard CBC-MAC is provably secure [47], it still has several flaws: it is not secure for variably sized messages; and if only one key is used to calculate all MACs, then for example a 64-bit MAC will have collisions occurring after 2^{32} messages because of the Birthday paradox [27]. As a result, key separation to generate other keys is recommended to overcome the Birthday attack and the use of other CBC-MAC variants (e.g. OMAC [62]) can overcome the insecure variably sized messages.

4.5.2. Counter (CTR) Mode

The counter (CTR) mode encryption and decryption operations are shown in the following two figures (Figure 4.9 and Figure 4.10). A counter is used similar to the use of an IV as it also provides the variation needed for every encryption to result in different ciphertexts even if the plaintexts are the same. However, counters are simpler than IVs used in CBC

mode. A counter can be of any value and then incremented by 1 for each subsequent block. For each encryption, the counter is encrypted (E_k) and then XORed with the plaintext block (P_n). Therefore the encrypted counter is used as a one-time encryption pad. For decryption, this same encryption pad must be XORed with the ciphertext block to recover the plaintext block (Figure 4.10). Stream ciphers also use one-time encryption pads (also known as one-time-pad, otp) to XOR with plaintexts to get ciphertexts, therefore CTR mode is also a stream cipher mode. However, CTR mode cannot be used for authentication like CBC-MAC.

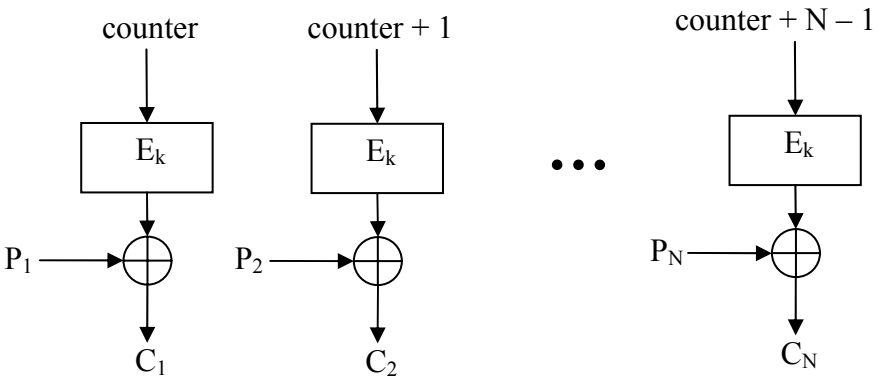

Figure 4.9. Counter (CTR) mode encryption.

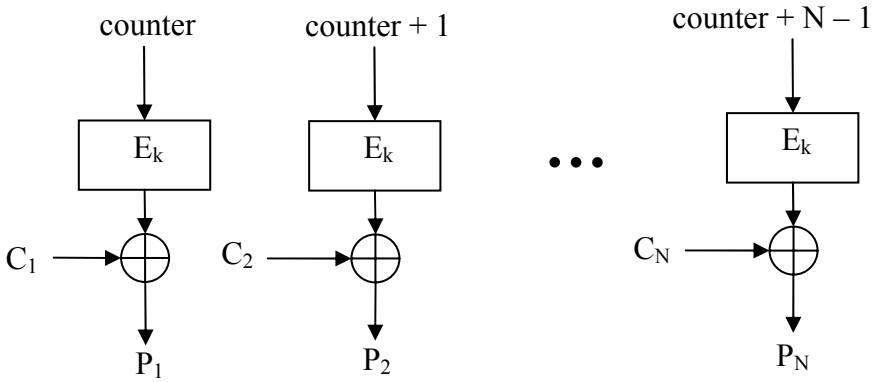

Figure 4.10. Counter (CTR) mode decryption.

The counter used in a CTR mode can be a non-secret value, but this counter value must never repeat. Unlike a repeated IV in the CBC mode which only leaks limited information, a repeated counter value leaks a significant amount of information. To illustrate consequences of reusing a counter in CTR mode, consider two different plaintext blocks P_1 and P_2 and two encryption pads Pad_1 and Pad_2. If the counters used for generating the encryption pads are the same, then $Pad_1 = Pad_2$. This results in the following: $C_1 \oplus C_2 = Pad_1 \oplus P_1 \oplus Pad_2 \oplus P_2 = P_1 \oplus P_2$. When plaintexts have sufficient redundancy, it is often possible to recover most or all P_1 and P_2 just from $P_1 \oplus P_2$ [48].

Besides the strict security requirement for a non-repeating counter, CTR mode has many advantages compared to the commonly used CBC mode [26]:

- Hardware efficiency: Unlike other chaining modes (e.g. CBC) where the encryption/decryption of the current block must be completed before the next block can begin, CTR mode allows parallel encryption/decryption on multiple blocks since it does not require previously computed blocks to perform encryption/decryption of the next block.

- Software efficiency: Similar to the hardware efficiency, if the processor supports parallel features (e.g. multiple pipelining, Single Instruction Multiple Data (SIMD) instructions), then multiple blocks can be processed simultaneously.

- Preprocessing: It is possible to pre-process the encryption pads with only the knowledge of the secret key and the counter value and store these encryption pads in memory. When plaintexts/ciphertexts arrive, all that needs to be done is to XOR the pre-processed encryption pads with plaintexts/ciphertexts to perform encryption/decryption.

- Random access: Any block of plaintext/ciphertext can be processed in a random-access fashion. This is again due to the fact that to encrypt/decrypt any block one

does not need any information from any previous blocks.

- Simplicity: Unlike for example the CBC mode, CTR mode only requires the implementation of the encryption algorithm and not the decryption algorithm.

4.5.3. Offset Codebook (OCB)

In the recent years, the National Institute of Standards and Technology (NIST) of the US has been reviewing new block cipher modes of operation particularly aiming at more efficient ways to provide both encryption and authentication security primitives [49]. The most commonly used method to provide both authentication and encryption using block ciphers has been using CBC mode for encryption and CBC-MAC for authentication. Using CBC requires two different keys for CBC encryption and CBC-MAC. The message also needs to be processed twice (two-pass), one for CBC encryption and the second time for CBC-MAC authentication. In other words, $2N$ block cipher calls are required, where N is the number of blocks that the message has been divided into.

NIST has reviewed several new modes of operation that provide encryption and authentication by processing the message only once (one-pass) plus some additional processing overheads. Some of the more promising new modes of operation which provide both authentication and encryption are: Integrity Aware CBC (IACBC)/Integrity Aware Parallelizable Mode (IAPM) [51], eXtended CBC (XCBC) [50] and Offset Codebook (OCB) [52]. OCB is a follow-on work to Jutla's IAPM [51].

When a message has been divided into N number of blocks, then to provide both authentication and encryption OCB requires $N + 2$ number of block cipher calls, and some additional overheads. Figure 4.11 and Figure 4.12 show how OCB provides authentication to the message and message header, and encryption only to the message. When sending a data packet over the network, it mainly consists of header bytes and message data bytes. It is often desirable to provide authentication to both headers and message data, but only encryption to the message data. OCB provides authentication to both headers and message

data using one MAC, but only encryption to the message data. This is desirable in many situations where the headers need to be authenticated but not encrypted to facilitate efficient relaying of data packets in the network. The MAC has the size of a block length of the block cipher used.

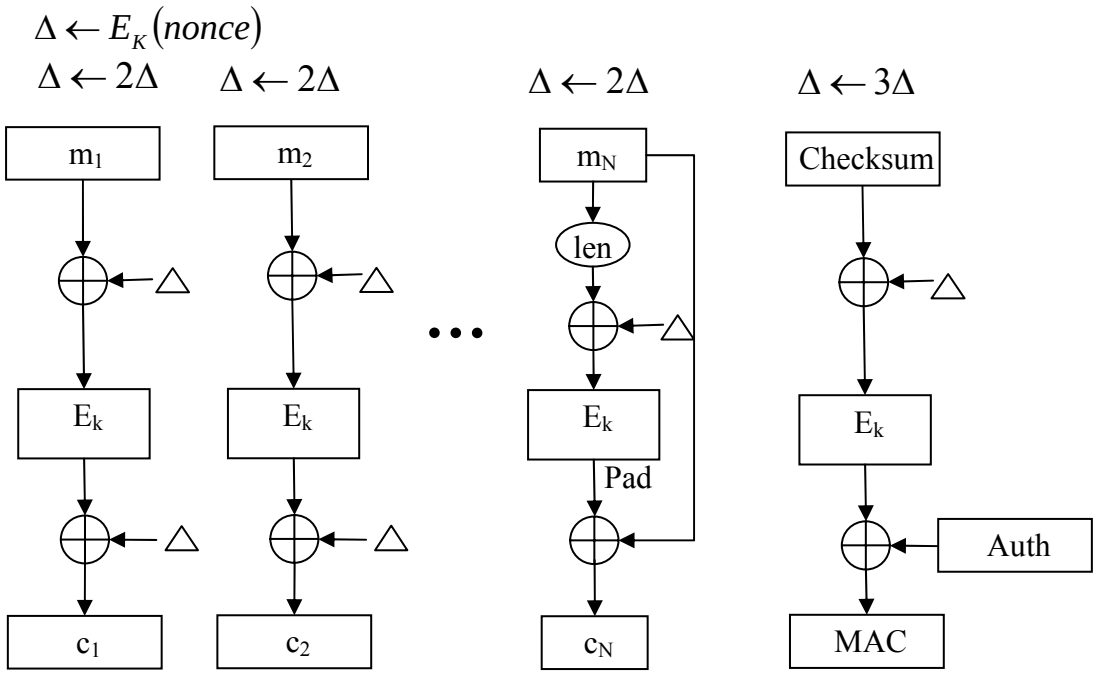

Figure 4.11. OCB encryption and authentication on a message.

In Figure 4.11, m_N and c_N are message and ciphertext block respectively. "$E_K(nonce)$" is the encryption of a nonce with secret key "K". The symbol "Δ" refers to the offset needed for OCB, and $\Delta \leftarrow 2\Delta$ means that the new offset value is replaced by the 2Δ operation. Having offsets allow a single secret key to provide both authentication and encryption. Offset calculations in the previous version of OCB (OCB 1.0) depended on Gray codes and the calculation time was not constant per offset [53]. The offset calculation in OCB 2.0 is as follows:

When offset Δ is a 64-bit block, then:

If MSB (most significant bit) = 0

Then $2\Delta = \Delta << 1$

Else $2\Delta = (\Delta << 1) \oplus 0x1B$

$3\Delta = 2\Delta \oplus \Delta$

$5\Delta = 2(2\Delta) \oplus \Delta$ or $5\Delta = 3(3\Delta)$

The offset operations: 2Δ, 3Δ and 5Δ are the polynomial multiplication of Δ by x, $x+1$, and x^2+x+1 within the field with 2^{128} elements [53]. The meaning of $\Delta << 1$ is to perform left bit-shift by 1, and the byte *0x1B* is represented in 64-bit format when XORed ($\oplus$).

The operation "*len*" in Figure 4.11 is to represent any input value that is smaller than 64 bits in a 64-bit format. OCB is able to encrypt messages of arbitrary length into a ciphertext of equivalent length. Even if the last block of plaintext message is not a full block size, the resulting ciphertext is not padded and will have the equivalent length as the plaintext block. Note that at the last plaintext block (m_N) of OCB encryption is XORed with the first $|m_N|$ bits of the "*Pad*". The "*Checksum*" is the value: $m_1 \oplus m_2 \oplus m_3 \ldots m_{N-1} \oplus (c_N 0^* \oplus Pad)$, where $c_N 0^*$ means appending c_N with enough 0-bits to get a 64-bit value. The value "*Auth*" is the authenticator for the header bytes of the message, which is included into the MAC. Figure 4.12 and Figure 4.13 show how header bytes are authenticated and not encrypted. Similar to previous figure, "Θ" is the offset for OCB operations on headers. Figure 4.12 shows the case when the header bytes are multiples of the block size, whereas Figure 4.13 shows the case when the header bytes are not multiples of the block size i.e. the last header block is smaller than one block size. In Figure 4.13, the last header block h_N is appended with a 1-bit followed by enough 0-bits to get a full block size (i.e. 64-bit in this example).

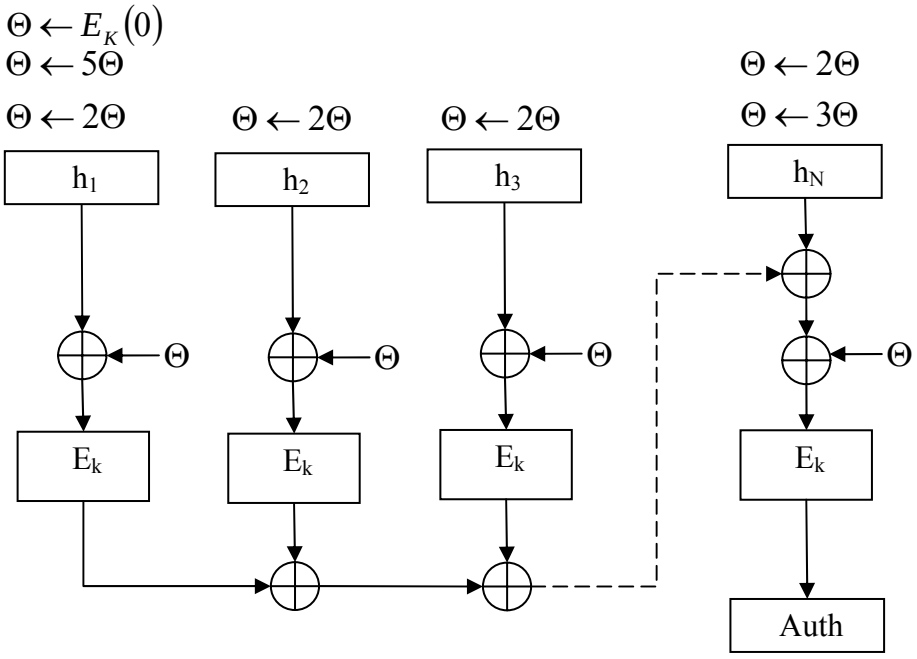

Figure 4.12. OCB authentication on message header of multiple block size (PMAC).

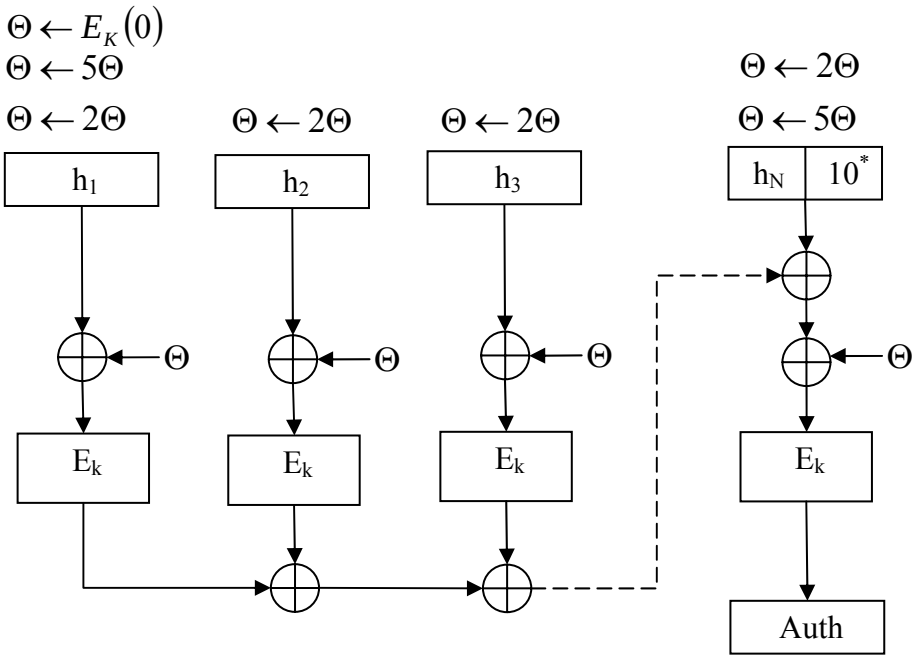

Figure 4.13. OCB authentication on message header not multiple of block size (PMAC).

The nonce used in OCB is unlike the initialization vector (IV) used in a CBC mode encryption, it is less strict compared to an IV. An OCB nonce is a value with the size of the block size; it also need not be secret. Furthermore, it also need not be random but it must not be repeated. A simple counter can be used as a nonce in OCB.

Some additional properties of OCB are:

- OCB requires a single block cipher key to provide both authenticity and confidentiality security primitives.

- Assuming the underlying block cipher is secure, OCB is provably secure.

- OCB does not involve modular 2^n addition (n is the cipher block size, e.g. 64-

bit/128-bit), which is not parallelizable and can be expensive especially for dedicated hardware [52] [53]. OCB also does not use the modular *p* arithmetic (*p* is a fixed prime number), which is a weaker algebraic structure [53] and is also a more expensive operation. OCB uses the less expensive GF(2^n)-based approach, which mainly uses XOR operations.

- OCB ciphertext will have the exact same length as the plaintext. Therefore it is more appropriate for short messages because the ciphertext does not expand in size.

4.6. RC4

RC4 is a stream cipher designed in 1987 by Ron Rivest for RSA Security. It has byte-oriented operations and generates streams of key bytes as encryption pad (one-time-pad) to be XORed with plaintext bytes (or XOR with ciphertext to obtain plaintext). It allows variable key length ranging from 1 to 256 bytes. The user-selected key is used to initialize a 256-byte S-box (also known as the state vector). This S-box contains all 8-bit numbers from 0 through 255. As a key byte is generated, the S-box is again permuted.

The following pseudo codes show how RC4 stream bytes are generated [26]. The variable *S* represents the 256-byte S-box.

```
i, j = 0;
while (true) {
        i = (i + 1) % 256;
        j = (j + S[i]) % 256;
        SWAP (S[i], S[j]); // swaps the two bytes at two different positions in S
        t = (S[i] + S[j]) % 256;
        k = S[t];
}
```

As it can be seen, the stream cipher RC4 is very simple requiring only few lines of codes to generate a key byte. However, the initialization of the S-box (not shown above) with user-selected secret key requires fairly large amount of CPU cycles, which is shown in section 8.2.

4.6.1. Cryptanalysis of RC4

RC4 has been proposed and in practical use for many years and no major weaknesses have yet been found. The best attack so far on RC4 has been targeted at its first few hundred key bytes generated. Therefore by dumping the initial 512 key bytes generated can thwart such attack on RC4 [66].Apart from a cryptographic cipher's weaknesses, it is also important to use the cryptographic cipher in a properly designed security protocol. Although RC4 itself has shown to be secure, however, the use of RC4 in the WEP (Wired Equivalent Privacy) protocol has been proven to be insecure [61]. WEP is a security protocol which is part of the IEEE 802.11 (WiFi) standard.

4.7. Conclusions

Several cryptographic ciphers that are implemented in this paper have been discussed in this chapter. Well-known conventional block ciphers such as the AES and RC5 are also briefly discussed but not implemented in this paper.

Although TEA has been designed for 32-bit word architecture, it was chosen because of its small code size, and its little memory requirement as it does not need to expand and store additional subkeys. Furthermore, as it will be shown in the results and discussions chapter, TEA in fact performs fairly well on the 16-bit microcontroller used in this paper.

The SAFER K-64 block cipher is designed for byte (8-bit) operation which makes it suitable for many low-end microcontrollers in sensor networks. It also requires fairly small code space, although not as small as TEA or TREYFER. The SAFER K-64 also requires additional RAM to store the expanded subkeys. However, as it will be shown in later

chapters, SAFER K-64 has the best performance in all block ciphers discussed in this paper, making it suitable for sensor networks requiring performance (thereby even lower power consumption) and has more memory resources.

The TREYFER block cipher is also designed for byte operation and aimed at extremely small code size, which is the main reason for choosing TREYFER. It requires the least code space in all block ciphers discussed in this paper. Similar to TEA, TREYFER also does not require any key initialization process to store expanded subkeys. However, as it will be shown in later chapter, TREYFER has the poorest performance in all block ciphers discussed in this paper. It is therefore not strongly recommended for WSNs.

RC4 is the only stream cipher implemented in this paper. The RC4 stream cipher also operates on byte-level and requires very small code space. As it will be shown in later chapters, RC4 has the best performance compare all other ciphers discussed in this paper. However, a stream cipher cannot be used as in the case of block ciphers with different modes of operation to provide authentication. Therefore it is more difficult to provide authentication using a stream cipher. Apart from this, RC4 has shown to be a good choice for the resource-constrained WSN field.

Other popular block ciphers have also been analyzed in this chapter. They have been shown to be inappropriate for WSNs in many aspects. One example is the large code space required by AES. In the results and discussions chapter, other block ciphers have also shown to be able to provide better performance with smaller code size.

5

UMAC

This chapter provides a brief overview of UMAC from a practical implementation point of view. The concept of a UMAC is discussed in the standard UMAC section and the practical UMAC implementations in systems today is discussed in the refined UMAC section. The detailed security definitions and the security proofs of universal hash functions and UMAC are beyond the scope of this paper and are therefore not discussed.

5.1. Standard UMAC

UMAC is an authentication algorithm using the universal hash function family, NH. NH is a new universal hash function family developed specifically for UMAC [55]. In simple terms, universal hash functions are collections of hash functions that map messages into short output strings such that the collision (pairs of different inputs with identical outputs) probability of any given pair of messages is small. UMAC has been particularly designed to utilize the SIMD (Single Instruction Multiple Data) parallelism of modern processors to achieve high speed. A 64-bit hash code UMAC optimized with MMX (Multimedia extensions) instructions can achieve a speed of more than 1 byte/cycle with messages larger than 256 Kbytes (on a Pentium II machine with MMX) [55]. UMAC allows user to select the underlying cryptographic primitives (e.g. cryptographic hash functions or block ciphers). No new heuristic primitives are developed in UMAC; therefore it is secure as long as the underlying cryptographic primitives are secure.

Figure 5.1 shows how the standard UMAC is implemented using a pseudo random function (PRF) to authenticate messages.

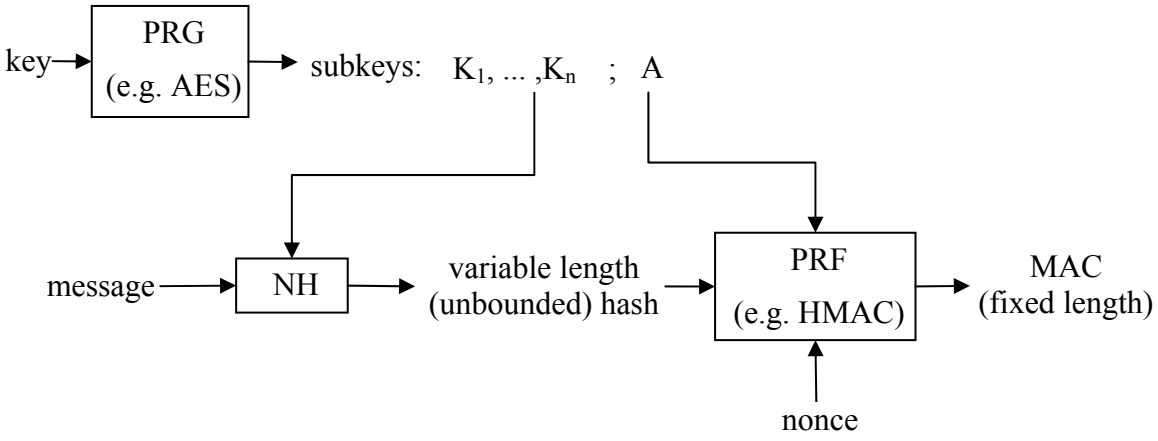

Figure 5.1. UMAC.

The notations used for UMAC are defined as follows:

- *key* is the user selected secret key.
- *w* is the adopted word size.
- K_1, ... ,K_n are the subkeys generated to be used in NH. All subkeys are the size of a word, *w*. *n* is the total number of subkeys for use in NH.
- *A* is the subkey generated to be used in the PRF.
- *PRG* is the pseudo random generator used to expand the secret key into subkeys needed for the different stages of UMAC.

- *NH* is the universal hash function used in UMAC.

- *PRF* is the pseudo random function applied to the variable length hash code to produce a fixed size MAC.

- *nonce* is a non-repeating, non-secret value that is to be sent by the message sender to the receiver.

5.1.1. NH

The message to be authenticated needs to be represented in words of size *w*. For illustration purpose, let *w* be 32 bits. The message is then divided into blocks of *n* number of words, let $n = 4$ as shown in Figure 5.2. This *n* number of words is the amount of words that the NH hash function will process when it is called. The actual *n* values range from 32 to 2^{28} bytes, typically $n = 256$ (equivalent to 1024 bytes when $w = 32$ bits) [57].

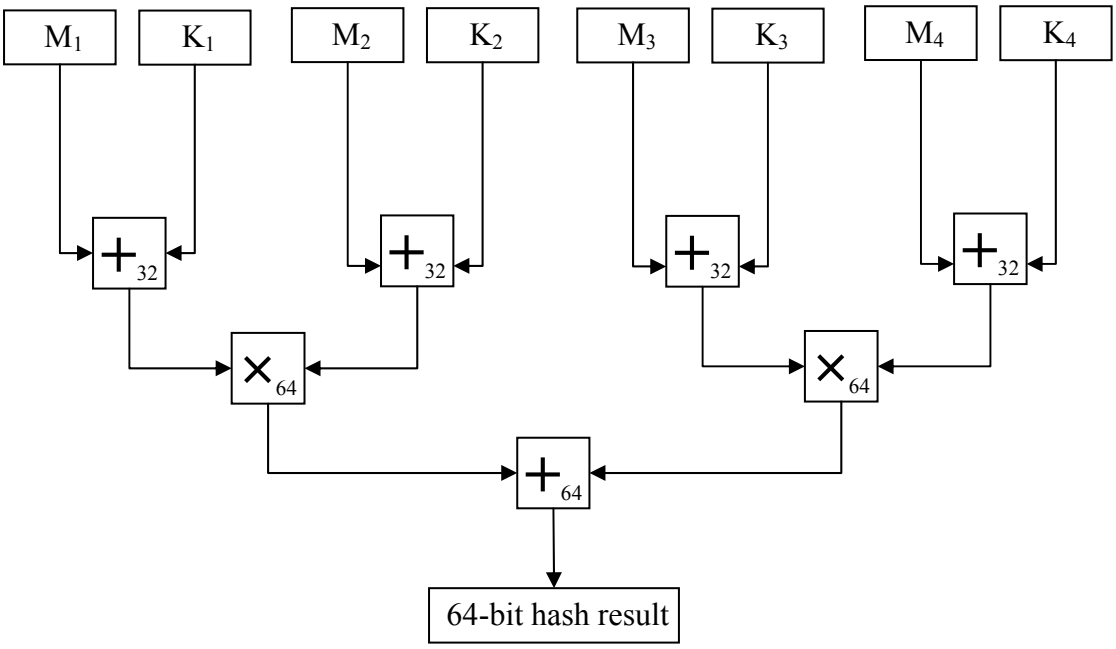

Figure 5.2. NH hash function with word size *w* = 32, number of words processed *n* = 4.

The message words (*M*) are processed by the NH hash function as shown in the above figure. The *n* words subkeys (*K*) are generated by a pseudo random generator (PRG). With 32-bit words, each message word is added (modulo 32) with the subkey word and then multiplied (modulo 64) with the next word that is also the result of subkey and message word addition. All multiplied (modulo 64) results of an NH call are then added (modulo 64) to get a 64-bit hash code. Repeated NH hash function calls are performed on all message words at *n* words per NH function call. The *n* subkeys remain the same for all NH calls; therefore with the same secret key, subkeys only need to be generated once. The multiple 64-bit hash codes resulting from multiple NH calls are concatenated together as an unbounded (variable length) hash code. Although this hash code is smaller than the original data message, it is still proportional to its size.

Increasing *n* increases the number of words to be processed in one NH call and results in smaller sized unbounded hash codes after NH calls. This tends to speed up MAC

generation on large messages, but requires more memory (for subkeys K_1, ... ,K_n) for processing and could potentially slow the processor by overflowing the processor's cache memory.

NH operations can be optimized using SIMD instructions such as MMX (Multi Media Extensions) instructions. NH calls are heavily used in UMAC, thus by optimizing NH calls one can greatly optimize UMAC.

5.1.2. Pseudo Random Generator (PRG)

A PRG is used to expand the user selected secret key into subkeys needed for internal stages of UMAC. Subkeys K_1, .. ,K_n are needed for the NH hash function, whereas subkey A is needed for pseudo random function (PRF) to produce a fixed size MAC. UMAC allows any PRG, but typically a block cipher (e.g. AES) is used. A block cipher takes the user selected secret key to encrypt a block of pre-defined value known as the index value. The resulting ciphertexts from encrypting different index values are used as the subkeys. The block cipher may have to be called several times to obtain enough ciphertexts as subkeys.

5.1.3. Pseudo Random Function (PRF)

The unbounded (variable length) hash codes obtained from NH calls are first appended with a nonce. A pseudo random function (PRF) is then performed on it to obtain a fixed size MAC output. HMAC-SHA1 can be used as a PRF, but any PRF is allowed. With HMAC-SHA1, the subkey A generated by the PRG is needed.

A non-repeating nonce is also needed for PRF to ensure that every MAC generated is different even if the data messages are the same. The nonce in UMAC has similar properties as the nonce used in OCB. It can be a simple non-secret incrementing counter that is sent with the data message and the appended MAC. For secure operations, the nonce should never be repeated within the life of a single UMAC secret key. To provide protection against replay attacks, the receiver needs to check that no nonce value is used

twice. This can be easily achieved when the nonce is a counter.

Although HMAC-SHA1 itself alone can be used to generate a MAC, UMAC using HMAC-SHA1 as PRF is more efficient since the input to HMAC-SHA1 is already a lot smaller than the original data message size.

5.2. Refined UMAC

After the release of the standard UMAC, UMAC authors have further refined UMAC and achieved three main goals [56] [57]:

1. Improved UMAC performance on short messages.

2. Minimize the use of underlying cryptographic primitives.

3. Selective-assurance verifiability is achieved. For example if a 64-bit MAC is computed, the receiver can choose to verify only the first 32 bits at nearly twice the speed of verifying the full 64-bit MAC.

Figure 5.3 shows the functional diagram of the refined UMAC [57].

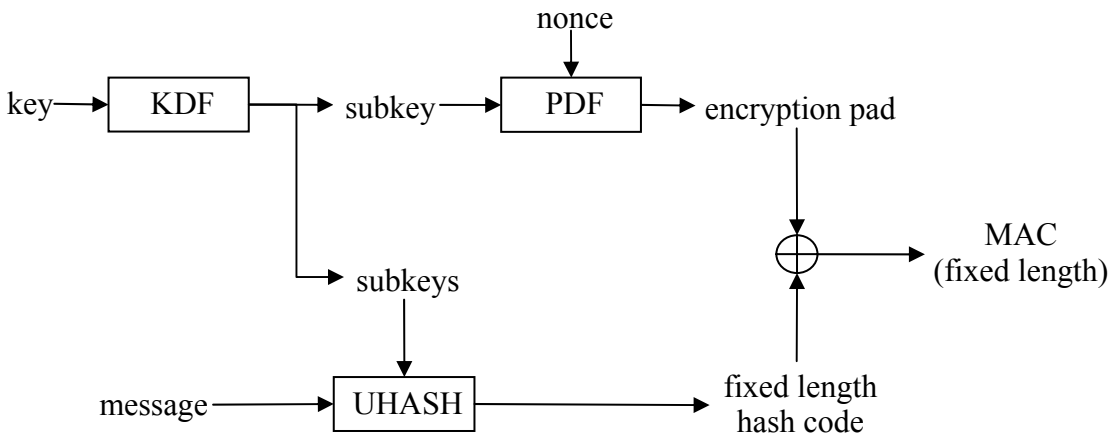

Figure 5.3. Refined UMAC.

At first glance, the standard UMAC (Figure 5.2) appears to be very different than the refined UMAC (Figure 5.3). However, as will be discussed now, the differences are in fact fairly subtle. The key derivation function (KDF) contains the pseudo random generator (PRG) in standard UMAC. Whereas the pseudo random function (PRF) (e.g. HMAC-SHA1) is replaced by the keyed hash function UHASH and the pad derivation function (PDF).

5.2.1. Key Derivation Function (KDF)

The user selected secret key is expanded into more subkeys using the KDF. The subkeys are used internally by UMAC in UHASH and the pad derivation function (PDF). KDF is equivalent to the PRG process in the standard UMAC. Block ciphers (e.g. AES) are used in output feed back (OFB) mode to produce the required subkey bits. The OFB mode used in KDF first encrypts a pre-defined index value, and then takes the resulting ciphertext output as the next block to be encrypted. This chain of ciphertext outputs is used as the required subkeys.

5.2.2. Pad Derivation Function (PDF)

The PDF is needed to generate the one-time encryption pad to be XORed with the fixed size hash code to produce the MAC. This one-time-pad (also known as otp) is obtained by applying the PDF to a nonce with a subkey generated by the KDF. A block cipher is typically used in the PDF to encrypt the nonce. The resulting ciphertext bytes are used as the one-time-pad. The nonce is defined the same as in the standard UMAC.

5.2.3. UHASH

UHASH is a keyed hash function, which takes an arbitrary length input data message, and produces as output a fixed length hash code. Figure 5.4 shows the function diagram of the UHASH.

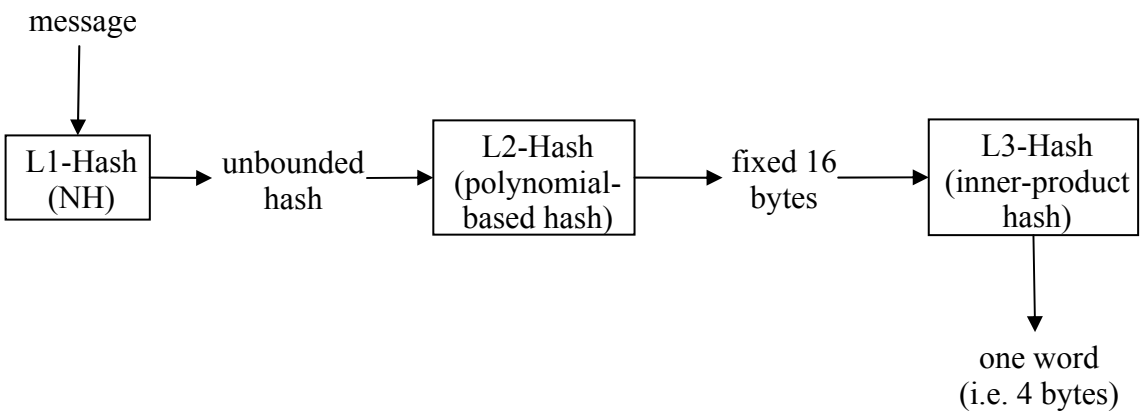

Figure 5.4. UHASH with word size $w = 32$.

UHASH consists of three layers. The first layer is the NH hash function, which is used to compress input messages into strings many times smaller than the input message. The NH in layer one is the same as the NH defined previously.

The second layer is a polynomial-based hash function that takes the unbounded (variable

length) hash results from NH, and produces a fixed-length 16-byte output (when $w = 32$). The polynomial hash function includes prime modulus operations. The security guarantee assured by polynomial hashing degrades linearly with the increasing length of the message being hashed and the prime number value. The prime modulus can be dynamically increased to ensure that the collision probability never grows beyond a certain pre-set bound when hashing a long message.

The third layer is an inner-product hash function that hashes the fixed 16-byte input to a fixed length word (i.e. 4 bytes when $w = 32$). A 36-bit prime modulus operation is used to improve security. Detailed discussions of layer two and layer three implementations are beyond the scope of this paper.

These three layers (UHASH) are repeated (with some different subkeys) until enough output MAC bytes are produced. For example, with 32-bit word size, UHASH needs to be called twice to obtain a 64-bit MAC. This shows that each MAC word can be computed and verified independently; therefore allowing the receiver to repeat UHASH lesser times to compute only some prefix of a UMAC MAC, thereby achieving faster verification speed (selective-assurance verification).

6

LINK LAYER SECURITY PROTOCOLS

The link layer security provides the first-line security just above the physical layer (where medium access control resides). All other higher level securities rely on the secureness of link layer securities. There are however, only very few link layer security protocols that have been proposed for WSNs. This chapter provides a review of the available link layer security protocols for WSN.

6.1. Sensor Network Encryption Protocol (SNEP)

One of the early studies on the security concerns for WSNs at the link layer is by Perrig, Szewczyk et. al. [1]. They proposed a security protocol called SPINS (Security Protocols for Sensor Networks), which contains two sub-protocols called: SNEP (Sensor Network Encryption Protocol) and µTESLA. SPINS was designed for the UC Berkeley SmartDust program [24]. The prototype SmartDust sensor node on which SPINS was implemented has the following specifications: 8-bit 4MHz microcontroller, 8 KB instruction flash, 512 bytes RAM, 916 MHz radio, 10Kbps bandwidth. The operating system running on the nodes is called TinyOS [25], which is also developed by UC Berkeley.

6.1.1. Confidentiality

SNEP (Sensor Network Encryption Protocol) is a link layer security protocol as it provides

confidentiality, authenticity and integrity. Additionally, SNEP also provides data freshness for protection against replay attacks. All these security primitives are constructed from a single block cipher, RC5. To achieve confidentiality, a counter (CTR) mode encryption is used (section 4.5.2). A shared counter is known between the sender and the receiver, therefore there is no need for transmitting the counter value with the data packet, which saves communication overhead. The overhead for SNEP is 8 bytes per message. The use of a counter value prevents electronic code book (ECB) attacks [26]. The aim for SNEP to use CTR mode as opposed to cipher block chaining (CBC) mode is that, with CBC mode, a random or secret initial vector (IV) is needed to protect against ECB attacks [1][26]. However, having to send different IVs for every transmission adds overhead to the data packet, which increases the power consumption in the sensor node. Since the counter value in CTR mode does not need to be random, it can be known and shared between sender and receiver in advance. Another reason for using CTR mode is that the size of the ciphertext is exactly the same size as the plaintext and not a multiple of the block size.

6.1.2. Authenticity

To achieve authenticity and integrity, a CBC message authentication code (CBC-MAC) using RC5 is used (section 4.5.1). Initialization vectors (IV) are not required in the CBC mode when it is used for generating a MAC; therefore it does not have the problem of having to transmit large IVs wirelessly. The counter value shared between the sender and the receiver is also used in generating the MAC. Using a counter here provides protection against replaying of old messages.

When using SNEP, the message that will be sent from *A* to *B* looks as follows:

$$A \rightarrow B : \left\{ E_{(K\_enc,C)}(M), MAC_{K\_mac}\left(C \middle| E_{(K\_enc,C)}(M)\right)\right\} \tag{6.1}$$

- $E_{(K\_enc,C)}(M)$, means RC5 encryption *E* with encryption key *K_enc*, using counter value *C*, on message data *M*.

- $C|E_{(K\_enc,C)}(M)$, is the concatenation of counter value *C* with $E_{(K\_enc,C)}(M)$.

- $MAC_{K\_mac}(C|E_{(K\_enc,C)}(M))$, is the message authentication code function (CBC-MAC) with MAC key *K_mac* on $C|E_{(K\_enc,C)}(M)$.

In the prototype sensor node for which SPINS was implemented, every node has a master key. Both encryption key *K_enc* and MAC key *K_mac* are derived from this master key by encrypting two different constant values with the master key to obtain two ciphertexts. The resulting ciphertexts are used as the encryption key and the MAC key.

Another protocol designed for SPINS is the µTESLA ("micro" version of the Timed, Efficient, Streaming, Loss-tolerant Authentication) protocol. µTESLA allows the sender to broadcast authenticated data to the entire sensor network with potentially untrusted receivers in it. Using symmetric MAC is insecure: If a node has been compromised with its symmetric MAC key known to the attacker, then the compromised node can impersonate the sender and forge messages to other receivers, therefore an asymmetric mechanism is required. Typically asymmetric cryptography is used for this purpose, but for the case of WSN asymmetric cryptography is too resource intensive. µTESLA allows authenticated broadcast from symmetric cryptography only (using RC5), and introduces asymmetry with delayed key disclosure and one-way function key chains. Authenticated broadcast is beyond the scope of link layer security and is therefore not further discussed in this paper.

Note that the µTESLA security protocol assumes the following data flow patterns only: node to base station, base station to node, and base station to all nodes. These communication patterns do not include sensor node to sensor node communication, which is found more often in mobile ad-hoc networks (MANET).

6.2. TinySec

TinySec [6] is another link layer security protocol also proposed by UC Berkeley but several years later and by different developers. It is designed to provide only link layer security: message confidentiality, integrity and authenticity. TinySec is designed to be implemented into the operating system TinyOS [25], which is the OS running on many sensor nodes (e.g. Mica mote series [14]). As a result of integrating with TinyOS, the TinySec stack can be used with simple function calls from the TinyOS application programmers' point of view. A representative sensor node that implements TinySec through TinyOS is the Mica2 [14]. It has the following specifications: 8MHz 8-bit Atmel ATMega128L microcontroller with 128KB flash for code, 4KB RAM for data, 512KB flash for data logging, ISM (industrial, scientific and medical) radio band and 19.2Kbps radio bandwidth.

6.2.1. Confidentiality

To provide confidentiality, TinySec adopts CBC mode encryption using the block cipher SkipJack. Although the optimized RC5 assembler code performs better than SkipJack [6], SkipJack is still chosen by the TinySec developers because of its lower key setup costs and because it is patent free. Recall that in the previous section it has been discussed that CBC mode encryption requires secret or random initial vectors (IV). TinySec uses an 8 byte IV (Figure 6.1), but 4 of the 8 bytes are from existing header fields, therefore only 4 additional bytes of overhead are added. In the 4 bytes overhead, 2 bytes are used as a counter for generating non repeating IV. Although transmitting a plaintext counter as IV (non-secret IV) does not achieve the strongest security in CBC mode encryption [26], this is a security tradeoff which TinySec accepts due to the resource-constrained sensor nodes. The remaining 2 bytes overhead is used to represent the source node address. This is to prevent every node from having the same IV when the counter value starts from zero. The 2 bytes counter together with the unique source address ensures IVs between nodes are different. This is different than the SNEP protocol, where all nodes share the same counter value used for CTR mode. With TinySec the counter is used to derive IV. However, a 2-byte counter allows only 2^{16} different IV values per node. Therefore IV reuse occurs after

sending only 65536 data packets. TinySec relies on the fact that data rates for data packets are very low (typically one packet per minute per node), therefore IV reuse will only occur after a longer period of time, at which point a new key should have been used already and thus IV counter can start anew again. At a data rate of one packet per minute, IV reuse will not occur for over 45 days. Even if IV reuse has occurred, only limited information may be revealed. This is another security tradeoff which TinySec considers to be tolerable.

Figure 6.1 shows the packet formats for TinySec with authentication and encryption (TinySec-AE), TinySec with authentication only (TinySec-Auth) and TinyOS packet format without TinySec security protocol. The common fields of TinyOS packets are: destination address (*Dest*), active message (*AM*) type, and the packet length (*Len*). The explanations for these fields are beyond the scope of this paper. Additional fields as overheads of TinySec are: source address (*Src*), counter (*Ctr*) and MAC (*MAC*). Note that a cyclic redundancy check (CRC) code is not used when a MAC is used. This is because MAC also provides the integrity check originally provided by CRC.

6.2.2. Authenticity

To provide authentication and integrity, CBC-MAC is used. The additional MAC overhead size is 4 bytes. Unlike SNEP, protection against replay attacks is not provided in TinySec. This is because TinySec nodes do not share the same counter value; therefore in order to detect replay attacks every node needs to maintain a table of past counter values from all the nodes that it receives. Whereas with the shared same counter value, every node knows exactly what counter values have already been used, thus preventing replay attacks. TinySec developers believe that replay attack protection should be provided in layers higher than the link layer.

Figure 6.1 shows the TinyOS packet format without security and with security (TinySec). The grayed fields are protected by MAC, field with grid is authenticated and encrypted and the byte size of each field is indicated in brackets.

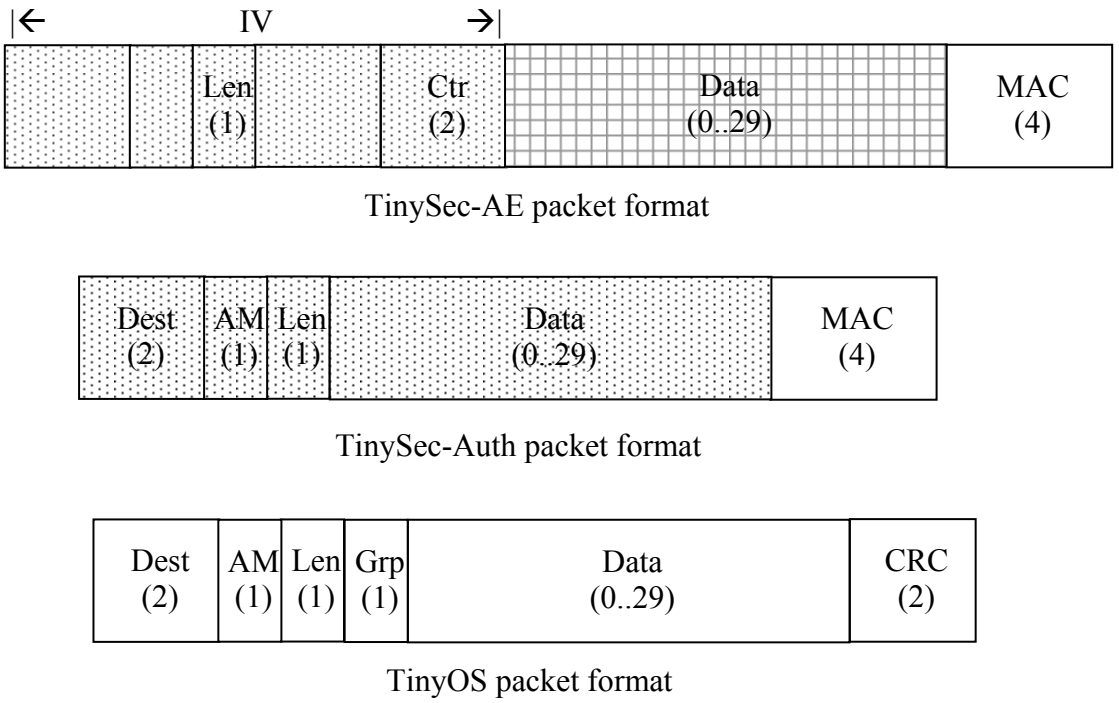

Figure 6.1. The TinySec and TinyOS packet format.

TinySec has been implemented with the programming language used for TinyOS called nesC. The implementation requires 728 bytes of RAM and 7146 bytes of code space. Or an alternative implementation that requires 256 bytes of RAM and 8152 bytes of code space, and also a 6% slower block cipher operation.

6.3. Other Link layer Securities

Other link layer security ideas have also been proposed or even adopted in prototype sensor networks. But many of them have not been documented as a proper security protocol. One such example is the link securities in the EYES project (refer to section 2.3.3). The security designers for the EYES network have proposed profiling application patterns [5]. The different “profiles” specify parameters that will be most suitable for the

different application environments, instead of designing a one-size-fits-all security solution. Example profile parameters such as the need for: data confidentiality, tamper resistance hardware, public key cryptography and etc. The securities for EYES network assumes public key cryptography capability because of the large 1MB serial RAM, and also adopts symmetric block cipher. Proposed block ciphers to be used are: MISTY1, TEA and AES (depending on the memory resources available).

7

IMPLEMENTATIONS

In this chapter, the implementation environment and security primitives implemented in this paper are discussed. The implementation decisions and approaches as well as the reasons for choosing particular security primitives are also explained.

7.1. Implementation Environment

All algorithms from this paper are implemented on the MSP430 low power microcontroller family from Texas Instruments [33]. This microcontroller family is used by sensor nodes from the EYES project [21] and the TinyMote nodes in the Technical University of Vienna's WSN project [12]. TinyMote nodes are used as the WSN platform on which the implementations of this paper are based. Figure 7.1 shows the TinyMote sensor nodes with the antenna built into PCB (printed circuit board), and a sensor node fitted with two AA batteries.

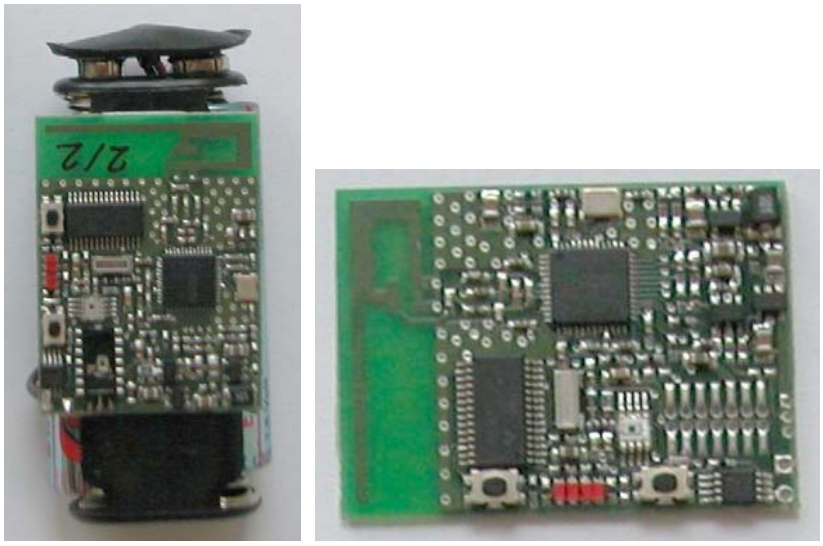

Figure 7.1. TinyMote with 2 AA batteries (left).

TinyMote has the following characteristics:

- Dimension: 38mm × 28mm
- Microcontroller:
  - TI MSP430F1232
  - 16-bit RISC CPU
  - 200 μA at 1 MHz (1 MIPS), 2.2V
  - 8KB flash memory (code memory), 256 bytes RAM
- Wireless interface:
  - Chipcon CC2400 [60] 2.4 GHz band

- Range: 100 m (line-of-sight)
- Data rate: 10 Kbps, 250 Kbps, 1 Mbps
- RF wakeup time: 1.23 ms
- Onboard antenna

- Sensors:
  - Onboard: temperature, brightness, relative humidity
  - Optional mountable sensor board (analog input 10-bit resolution)

The TinyMote is programmed by connecting it to a custom made TinyMote USB dongle, which is in turn connected to a TI JTAG interface to a PC. The USB dongle can be either powered either by connecting it to a USB port, or by connecting it to two AA batteries. The programming software used is the IAR Embedded Workbench IDE (integrated development environment) for MSP430 with C/C++ compiler version 3.21A [59]. The IAR Embedded Workbench contains both the programmer for TinyMote and the C/C++ compiler with which the algorithms from this paper are implemented. The compiler is set for maximum speed optimization for all code implemented in this paper.

The software running on TinyMote nodes is developed by the Technical University of Vienna [12], and is at version 2.4.6. The security primitives that are implemented in TinyMote nodes are first integrated into this version of the node software (written in C), then the software is recompiled and uploaded to the nodes.

With the current design of TinyMote WSN, the sink node (base station) is a standard TinyMote node with source address “0”. It is connected via a USB dongle to a PC, to which the received data packets are sent. The PC is also the power source for the sink node

through the USB port. The human interface and the display of sensor information take place at the sink node connected PC.

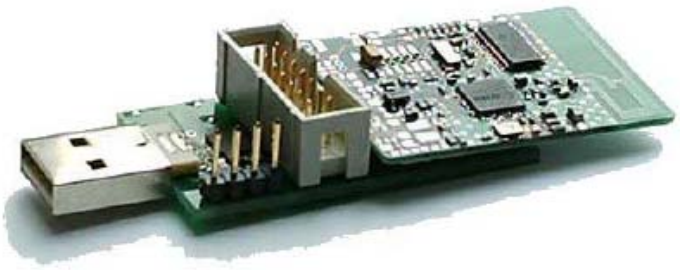

Figure 7.2. TinyMote connected to USB dongle.

The security model in this research is based on the base station as the point of trust (central trusted authority). The sink node (base station) and sensor nodes are connected similar to Figure 2.1. However, since the current design of the TinyMote WSN uses a standard sensor node as the base station, the base station in the TinyMote WSN lacks many characteristics often found in other WSN base stations. These include the high bandwidth base station-to-base station communications (as shown in Figure 2.1) and rich computational and memory resources. However, since the TinyMote base station is connected to the PC through the USB dongle, it has no limits on its energy usage.

7.1.1. TinyMote Network Behavior

The TinyMote network behavior is depicted as a flow chart in Figure 7.3. The current version of the node software implements the medium access control protocol CSMA-MPS [18] and only simple routing algorithms.

The traffic pattern of the TinyMote WSN in this paper is many-to-one. Multiple sensor nodes send sensor values to the sink node and there are no sensor node-to-sensor node communications. The network topology is also static (the node positions are fixed).

Each sensor node periodically listens to the channel at two second intervals. It listens for

wakeup signals from its neighboring nodes and receives data packets. A node is synchronized with its neighboring node through this periodic channel listening. The TinyMote node also performs sensor measurements every two seconds. If sensor values have changed significantly then a data packet with new sensor values will be sent back to the sink node, possibly with several hops of other nodes in between. If a data packet has failed to be delivered (or the transmitting node has not received acknowledgment from the receiving node) after two trials, the node will then search for an alternate neighboring node to deliver the data packet. If this third delivery trial also fails, the data packet will be dropped and the node will enter sleep mode for 60 seconds to save power, because it is probably not connected to the network.

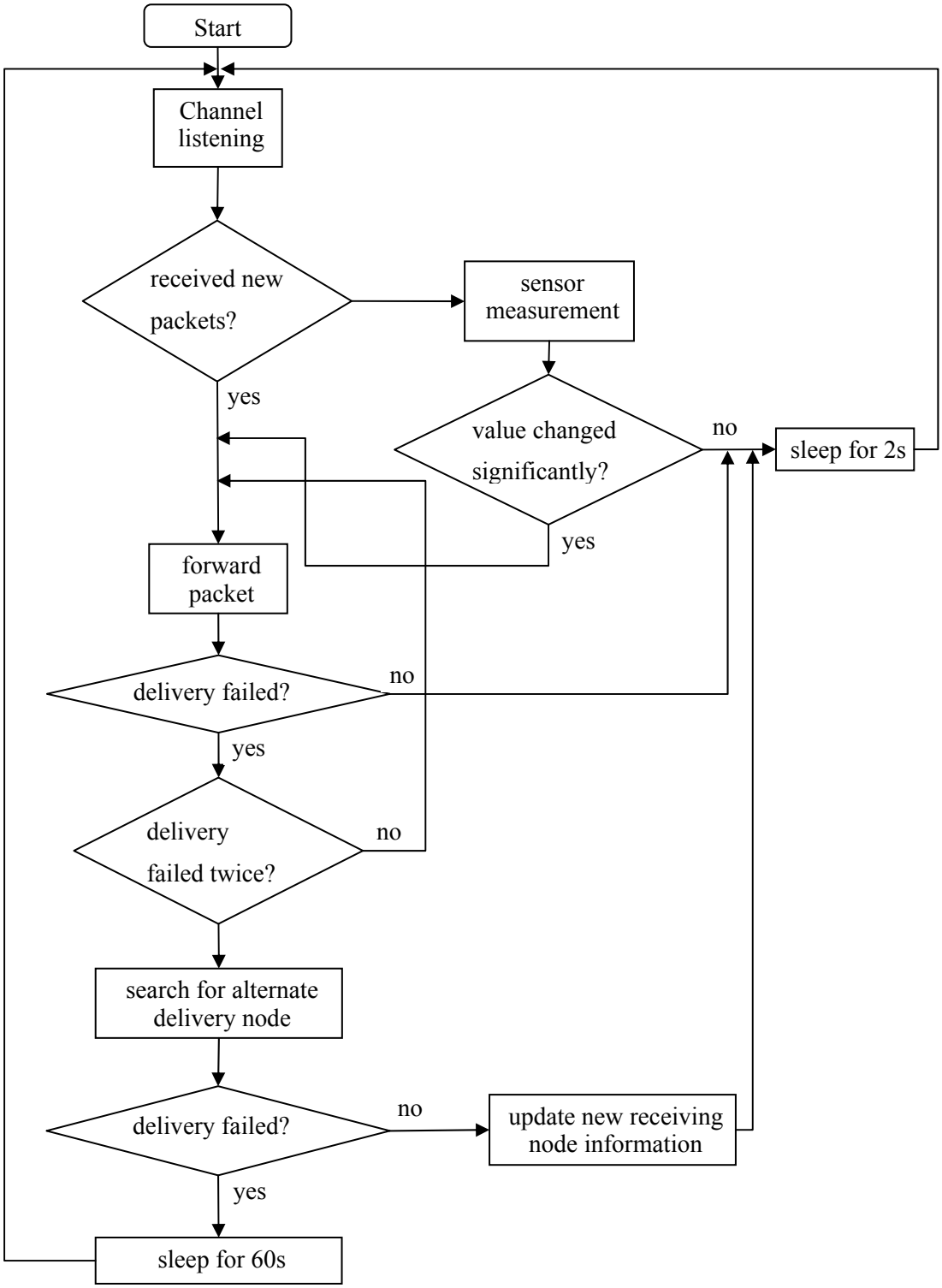

Figure 7.3. TinyMote network behavior.

7.1.2. TinyMote Packet Structure

The TinyMote packet structure is shown in Figure 7.4. The transceiver fields in Figure 7.4 (a) are added by the RF transceiver hardware. Figure 7.4 (b) shows the detailed view of the data field from Figure 7.4 (a). The user specific fields are handled by TinyMote software in the microcontroller. The first three fields (packet length, source address, last hop count) in Figure 7.4 (a) are required for normal operations of the medium access control protocol CSMA-MPS. Hop counts (HC) are values indicating how many transmissions are needed before a data packet reaches the sink node. For example, a node with source HC = 2 indicates that the data packet it sends needs to go through another node before reaching the sink node. The mandatory fields in Figure 7.4 (b) are required by the current version of the TinyMote software, but are not required by the CSMA-MPS protocol; therefore it is possible to reprogram these fields for alternative purposes.

Transceiver		**User specific**				**transceiver**
Preamble (32 bits)	Sync word (16 bits)	Packet length (8 bits)	Source address (16 bits)	Last HC (8 bits)	Data (max 11 bytes)	CRC (16 bits)

(a)

Data							
Mandatory				**optional**			
source HC (8 bits)	sensor type (8 bits)	TX slot counter (8 bits)	V_{CC} (8 bits)	V_{Sol} (8 bits)	temperature (16 bits)	brightness (16 bits)	humidity (16 bits)

(b)

Figure 7.4. TinyMote packet structure.

The meaning of different data fields in Figure 7.4 (b) are as follows:

- Source HC – hop count value of the source node (where the data packet originates).

- Sensor type – bit encoded byte indicating which types of sensor values are included in the optional fields.

- TX slot counter – a debugging value for CSMA-MPS protocol.

- V_{CC} – the voltage level of the source node.

- V_{Sol} – the solar cell voltage level of the source node (refer to section 7.1.3).

- Temperature – the temperature of the source node with accuracy up to one tenth of a degree Celsius.

- Brightness – the brightness of the source node measured in LUX.

- Humidity – the relative humidity of the source node measure as a percentage (%).

Note that the CRC (Cyclic Redundancy Check) field for integrity checking will not be needed if the authentication security primitive is implemented (e.g. MAC). This is because an authentication code provides both authentication and an integrity check.

7.1.3. TinyMote Power Consumption

The TinyMote power consumption levels (with no security primitives) at different measurement intervals and data transmission intervals are shown in Figure 7.5 and Figure 7.6. A data packet is only sent when sensor readings differs significantly, therefore the packet transmission interval is usually longer than the measurement and channel listening interval.

It is worth noting that the TinyMote based WSN is designed to be energy self-sufficient, meaning the sensor network lifetime is not limited by energy resources (e.g. batteries). To achieve this, TinyMote has been designed to be connected to a solar cell which has a solar panel and two 10F ultra-capacitors. The ultra-capacitors are used as an energy storage medium. They are superior to rechargeable batteries because ultra-capacitors achieve a lot more charge-discharge cycles and thus a longer lifetime than any rechargeable batteries.

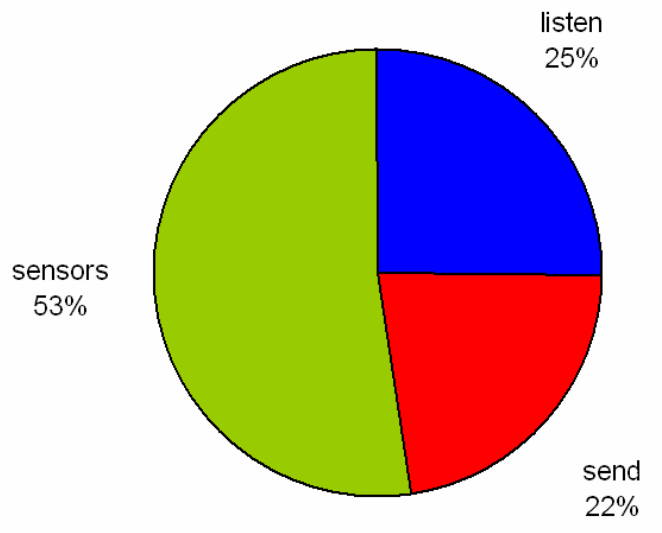

Figure 7.5. Power consumption 187 µW (channel listening: 47 µW, packet sending: 41.14µW, sensors: 98.38 µW) (2s measurement interval, 10s transmission interval).

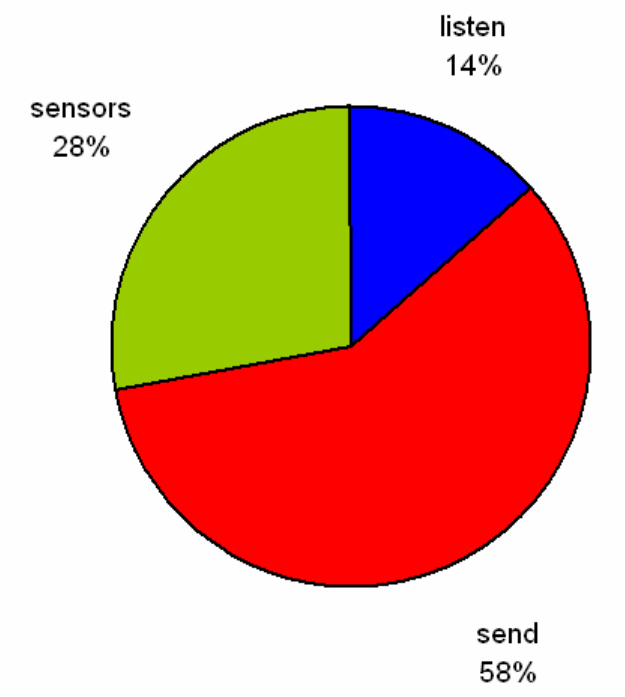

Figure 7.6. Power consumption 70 µW (channel listening: 9.50 µW, packet sending: 41.14µW, sensors: 19.68 µW) (10s measurement interval, 10s transmission interval).

7.2. Block Ciphers

This section discusses which block ciphers are implemented in this paper, how they are implemented and why these block ciphers and these particular configurations are chosen.

7.2.1. XTEA

XTEA (Extended Tiny Encryption Algorithm) as implemented in this paper is adapted from the XTEA reference C code [36]. The reference XTEA C code is ported to the IAR C/C++ compiler for MSP430. The number of XTEA cycles is fixed at 15 (30 Feistel rounds); therefore certain XTEA internal values can be pre-computed. This results in a slightly more optimized version of XTEA with a smaller code size and a faster execution speed. A faster execution speed means lesser processor cycles are required and it therefore requires lower power consumption.

One of the main reasons for choosing XTEA is its small code size (typically less than 30 lines of C codes). XTEA also requires very little RAM space because it does not require an initialization process to generate and store subkeys like many other block ciphers. Subkeys are mixed and generated within the XTEA rounds. XTEA is chosen instead of the even simpler original TEA or other TEA variants because the original TEA has proven to be insecure; Block TEA and XXTEA are designed to be more efficient on longer messages, whereas in a WSN environment most messages are fairly short. A 15-cycle XTEA is chosen because the best proposed attack on XTEA is on 27 rounds, which is less than 14 cycles. Therefore for a balance between security and performance, a 15-cycle (30 rounds) XTEA is chosen.

Although XTEA operates directly on 32-bit words and requires 32-bit shifts, it requires only short fixed number bit shifts. Furthermore, the MSP430 is a 16-bit microcontroller; therefore operating on 32-bit words does not impose as huge a penalty as would be found on an 8-bit microcontroller.

7.2.2. SAFER K-64

The implemented SAFER K-64 (Secure And Fast Encryption Routine with a Key of length 64 bits) in this paper is adapted from the Turbo Pascal code provided by the SAFER K-64 designer [44]. For execution speed optimization, three look-up tables are provided which are not included in the reference Turbo Pascal code. They are namely the log, exponential and the key bias tables. These look-up tables allow SAFER K-64 to avoid logarithmic, exponential and modulus operations during key scheduling and round functions. A 7-round SAFER K-64 is used in this paper.

SAFER K-64 has many advantages for implementation in a resource-constrained environment. When using look-up tables to replace logarithmic and exponential computations the SAFER K-64 requires only byte level XOR and addition operations, making it suitable for any word size microcontrollers. SAFER K-64 is also a relatively small code size block cipher. Another reason for choosing SAFER K-64 is that, although it seems to be suitable in an embedded environment, there are not many studies on SAFER K-64's performance in a WSN environment; therefore it is chosen to see how it compares against some other well known block ciphers. A 7-round SAFER K-64 is chosen because so far the best feasible attack has been found on 5-round SAFER K-64. The authors of the proposed attack indicate a 7 or 8-round SAFER K-64 is secure against such attack. With a 6-round SAFER K-64, although such attack is still applicable, the required encryption computation time approaches 2^{64} (attack computation time: 2^{61}); therefore an attack on a 6-round is also not very feasible.

7.2.3. TREYFER

A 32-round (nominal) TREYFER is implemented in this paper. The TREYFER codes are adapted from the C code provided by the designer [42]. In the original code, explicit modulo 256 operations are required. However, this operation consumes a lot of processor cycles as it requires to performing division operations. The adapted TREYFER code avoids explicit use of modulo 256 operations and thus achieves optimization for both code size and execution speed (33% smaller code size and 83% faster execution speed than the original TREYFER C code). The designer of TREYFER suggested "stealing" the required 256 bytes of S-box from any other place in code memory running with TREYFER.

However, in this paper the 256 bytes S-box is explicitly defined together with the TREYFER code.

The main reason for choosing TREYFER is its extremely small code size (even smaller than the TEA block cipher). TREYFER also requires only byte level operations. Similar to the TEA block cipher, TREYFER also does not require the generation and storage of subkeys; therefore it also requires less RAM. Another reason for investigating TREYFER is that, similarly to SAFER K-64, although it is also suitable for a resource-constrained environment, it has not yet been properly studied in a WSN environment. The 32-round TREYFER is chosen because it is the nominal round value used by the designer. Although the designer suggests any round value larger than 8 should provide enough security, however there are no attempts to cryptanalysis on TREYFER rounds smaller than 32; therefore for security reason a 32-round TREYFER is chosen.

A feasible attack on TREYFER has been shown to be possible and is independent of the number of rounds. However, a counter-measure against such an attack is also possible (refer to section 4.3.1.)

7.2.4. OCB Mode

Offset Codebook (OCB) version 2.0 block cipher mode as implemented in this paper is developed from scratch, but using snippets of C code from the OCB designer's web site [53] as a guideline. The only difference is that the underlying block cipher has changed from AES (128-bit block size) to XTEA (64-bit block size).

One of the main reasons for choosing OCB is its ability to provide authentication and encryption in only one-pass of the data message (data message processed once only), plus two additional block cipher calls and some processing overhead for creating the sequence offsets. Other generic modes usually require one-pass for encryption (e.g. CBC, CTR mode) and another pass for processing of the data message for authentication (e.g. CBC-MAC). Compared to other one-pass modes, OCB is a follow up work on XCBC [50] and is proposed after many earlier proposed one-pass modes. As a result, OCB has been designed with several improvements and is more efficient and less complex than many other one-

pass modes. For example, OCB uses XOR instead of large bit addition or modulo prime number operations. OCB version 2.0 has particularly reduced the complexity of generating sequence offsets compare to other one-pass modes, where offset generation has contributed to most of the processing time after the block cipher operations in any one-pass modes. Another advantage of OCB, also described in section 4.5.3, is that it is designed to be able to authenticate a packet's associated data (e.g. header bytes in a data packet) without encrypting it. This allows both header bytes and data bytes in a packet to be authenticated, while only data bytes are encrypted. This is an important feature in many networks including WSN, because a sensor node may need to see the headers in plaintext in order to quickly relay the data packet to the next node and return to sleep to conserve power. If the headers are also encrypted, then in order for a node to correctly relay the packet to the next node, it needs to first decrypt the headers, which imposes more processing time and thus higher power consumption.

7.3. RC4 Stream Cipher

The RC4 stream cipher in this paper is not implemented on the MSP430F1232 microcontroller that is used in the TinyMote sensor nodes. It is implemented and simulated on a MSP430F149 microcontroller. The reason for this is that RC4 requires at least 256 bytes of RAM to store its constantly changing S-box content, but MSP430F1232 has only 256 bytes of RAM. Therefore it is implemented on a MSP430F149 which has a larger RAM space (2 KB RAM) (another WSN, the EYES [5] project, uses MSP430149 as its node microcontroller). RC4 in this paper is adapted from the pseudo code in [26]. However, the initialization function of the adapted RC4 has been improved by avoiding the use of modulo key length operations, which greatly reduces the initialization processor time (76% faster) and also with slightly smaller code size (3% smaller).

RC4 is chosen because it is an extremely fast and small size stream cipher, even though it can not be implemented on the TinyMote platform. Therefore it is implemented in this paper to investigate its alternative use. RC4 also only requires byte level operations, making it suitable for any word size microcontrollers. RC4 on its own provides only confidentiality (encryption) and it cannot be used like block cipher modes to provide

authenticity. Therefore in this paper, RC4 is used to generate pseudo random numbers needed for UMAC operations, as an alternative to using block ciphers to generate the needed pseudo random numbers. Lastly, although not as popular as the block cipher modes providing authentication, there are still other proposals suggesting how stream ciphers can be used to provide authentication, but these are not investigated in this paper.

7.4. UMAC

The UMAC implementations in this paper have been greatly modified to be more suitable to a WSN environment. They are implemented from scratch using UMAC internet drafts [57] as guidelines.

7.4.1. UMAC-Block Cipher

In the UMAC shown in Figure 5.3 and Figure 5.4 from section 5.2, the L2-HASH and L3-HASH are used to further reduce the size of the unbounded (variable length) hash code generated by L1-HASH (NH hash function). After the L2-HASH and L3-HASH the output hash code is a fixed size. However, the output hash code size of the NH hash function is proportional to the input message size. Therefore in a closed WSN environment where data message size is small, fixed and known, it is possible to customize UMAC to eliminate the need for the L2-HASH and L3-HASH. The UMAC implemented in this paper is similar to Figure 5.3, except that only the NH hash function is needed in place of UHASH as shown in the figure below.

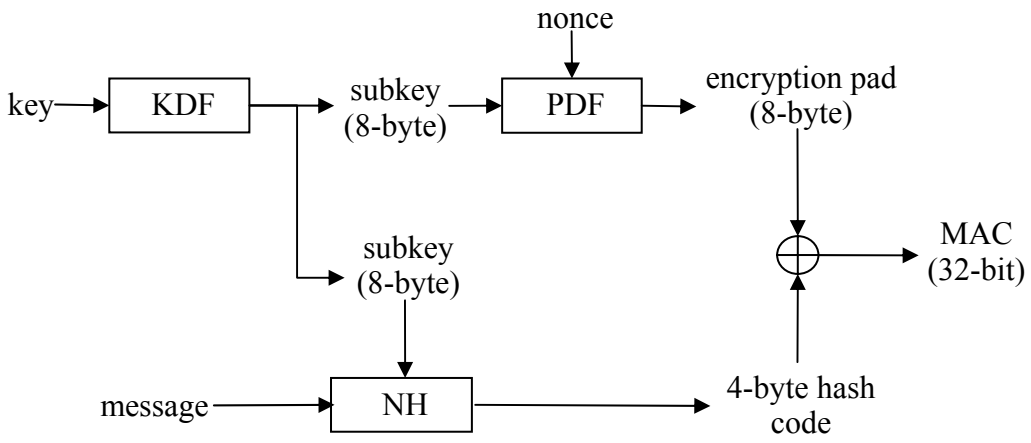

Figure 7.7. customized UMAC.

The customized UMAC in this paper (Figure 7.7) uses an underlying block cipher with a 64-bit block size. Therefore the KDF (key derivation function) only needs to call the block cipher twice to generate two 8-byte subkeys to be used in the PDF (pad derivation function) and NH hash function. Note that the subkeys only need to be generated once, and remain the same for the lifetime of the same secret key. One block cipher call in PDF produces 8 bytes of encryption pads to be XORed with a 4-byte (32-bit) hash code. Therefore every PDF call produces enough encryption pads for generating two MACs. The nonce used with PDF is realized with a simple incrementing counter.

The NH hash function is shown in the following figure (Figure 7.8).

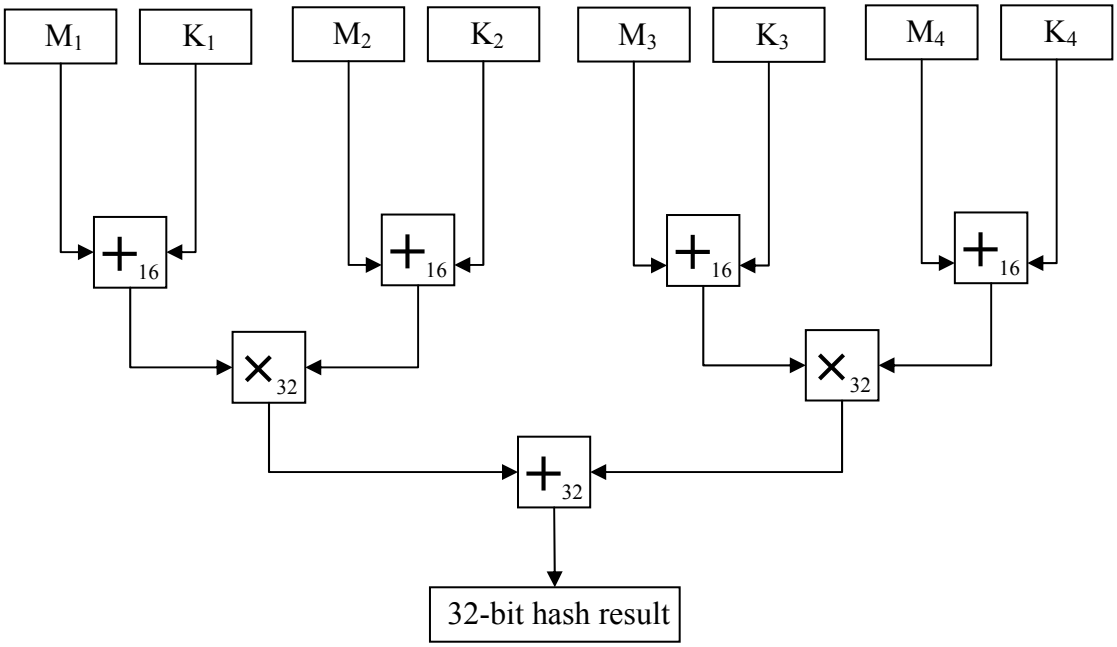

Figure 7.8. NH hash function with word size *w* = 16 and number of words processed in a NH block *n* = 4.

The word size processed in the NH hash function is 16-bit, and the number of words processed in a NH block is equal to 4 words. Therefore the number of bytes processed in one NH hash function call is 8 bytes. Word size is chosen to be 16-bit to achieve a 32-bit hash code after two 16-bit multiplications, and also because the word size of the microcontroller used is also 16-bit.

The data message is divided into 8-byte chunks as input for the NH hash function. The hash result of a NH function call is added to the hash result of the previous NH function call. Therefore, the final hash code will always be of a fixed 4-byte size. For example, to process a 24 byte data message, three NH function calls are required, and all three hash results are added. This method of implementing the NH hash function has also been proposed by Yüksel [58], however, in a hardware implementation only.

The advantage of using such a customized UMAC is clear. After the initial subkeys have been generated, to produce two MACs for any messages afterwards only requires one KDF

call (i.e. one block cipher call) and several NH hash calls (depending on the size of the message). Using only one block cipher call to generate every two MACs can save a significant amount of processing time, particularly when the block cipher calls are more expensive than NH function calls. The underlying block cipher of UMAC can also be used to provide encryption that is not provided by UMAC.

7.4.2. UMAC-RC4

The KDF and PDF in UMAC are needed to produce subkeys and encryption pads respectively. However, UMAC is not limited on using only block ciphers to generate pseudo random numbers needed in KDF and PDF. In this paper, the RC4 stream cipher is used to generate pseudo random numbers as the subkeys needed for the NH hash function and the encryption pads needed to XOR the NH hash code. The following figure (Figure 7.9) shows how RC4 is used with an NH hash function to generate a MAC. The NH hash function used here is the same as the one described previous in Figure 7.8. A nonce (counter) is not needed in this implementation because a nonce is used with the block cipher to ensure that every encryption pad produced will be different. Also, the RC4 stream cipher is designed to continuously generate pseudo random key stream bytes as encryption pads; therefore a nonce is not required.

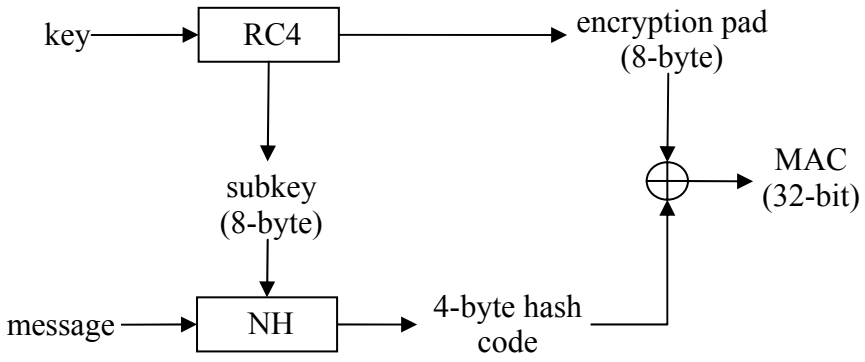

Figure 7.9. UMAC with RC4.

The main advantage of incorporating UMAC into RC4 is its efficiency and simplicity. RC4

replaces the need of KDF and PDF for generating pseudo random numbers, resulting in a smaller code size. Apart from generating pseudo random numbers needed for UMAC, RC4 can also generate additional encryption pads to encrypt the data message; thus providing encryption apart from the authentication by UMAC. However, since there is no nonce being used in RC4, the message receiver of the UMAC-RC4 authenticated and RC4 encrypted messages must keep track of the sender's RC4 S-box status in order to know which key stream bytes are being used at the moment.

7.5. Conclusions

Adaptations and implementations of several cryptographic algorithms are discussed in this chapter. Blocks ciphers of XTEA, SAFER K-64, and TREYFER are implemented. The OCB mode block cipher encryption and authentication is also implemented. The only stream cipher implemented and evaluated is the RC4 stream cipher. Furthermore, UMAC authentication algorithm is also adapted for WSN and implemented with XTEA as its underlying block cipher.

8

RESULTS AND DISCUSSIONS

This chapter provides the results of the various implementations discussed in the previous chapter. These results include the performance, power consumption and security level of the different security primitives; and how link layer security protocols are improved compare to existing protocols.

8.1. Power Consumption in the MSP430 Microcontroller

Different instructions may require different numbers of clock cycles, resulting in different amounts of energy consumption per cycle. Even different instructions with the same number of clock cycles may consume different amount of energy per cycle because of the nature of the instruction itself. For example an instruction that accesses the main memory (RAM) or registers will consume less energy than an instruction that accesses the flash memory.

However, Law et. al. [54] have shown that the energy per cycle is fairly consistent for the MSP430 microcontroller family with a mean deviation of 6%. Groβschadl et. al. [64] have further shown that variable energy consumption per cycle has more influence on high-end microcontrollers and DSPs. For example, Intel's StrongARM SA-1100 has a more complex power management strategy such as the use of conditional clocking trees, which ensures only the presently required units in the microcontroller are clocked and other units remain static; thereby resulting in the difference in energy consumption per cycle.

Therefore it is safe to say that the MSP430F1232 microcontroller used in this paper running at 1 MIPS, 2.2V, and at an average current of 200 µA requires an average power consumption of: $2.2V \times 200\mu A = 440\mu W$.

8.2. Cryptographic Ciphers

Table 8.1 shows the code size, look-up table size, user key size and expanded subkey size required for each cryptographic cipher. Both code and look-up table are stored in the flash memory and the key and expanded key are stored in RAM. The size of the standard and optimized versions of XTEA, TREYFER and RC4 are also shown.

Table 8.1. Cryptographic ciphers memory requirements.

	Code (Flash)	Look-up table (Flash)	Key (RAM)	Expanded key (RAM)
XTEA (std.)	712 bytes	N/A	16 bytes	N/A
XTEA (opt.)	620 bytes	N/A	16 bytes	N/A
SAFER K-64	850 bytes	624 bytes	8 bytes	112 bytes
TREYFER (std.)	294 bytes	256	8 bytes	N/A
TREYFER (opt.)	196 bytes	256 bytes	8 bytes	N/A
RC4 (std.)	512 bytes	N/A	16 bytes	256 bytes
RC4 (opt.)	492 bytes	N/A	16 bytes	256 bytes

The code size consists not only of the algorithm for the cipher, but also the necessary simple code for setting up testing vectors (e.g. plaintexts and ciphertexts) to execute the cipher.

All three block ciphers implemented in this paper are implemented and measured in ECB (electronic codebook) mode processing an 8-byte block. This is to allow measurements to be focused on the actual block cipher algorithm's speed performance. However, Law et. al. [54] have shown that the performance in a speed optimized block cipher differs very little

between generic modes of operation (e.g. CBC, CFB, CTR etc.).

Table 8.2 below shows the performance (in CPU cycles to process one byte) of each cipher. Key setup is the number of cycles required to initialize expanded subkeys and is only executed once for the lifetime of the same user selected secret key.

Table 8.2. Cryptographic ciphers CPU usage.

	Encrypt	Key setup
XTEA (std.)	303	N/A
XTEA (opt.)	287	N/A
SAFER K-64	126	3578
TREYFER (std.)	6527	N/A
TREYFER (opt.)	1110	N/A
RC4 (std.)	103	47515
RC4 (opt.)	103	11154

From the tables above it can be seen that the optimized XTEA achieves smaller size (12% smaller) and slightly faster execution speed (5.3% faster) than the standard XTEA code size. XTEA does not require setting up expanded subkeys; therefore there is no additional RAM or CPU usage needed for storing and setting up subkeys.

Both SAFER K-64 and TREYFER are the only cryptographic ciphers that require look-up tables. It can be seen that SAFER K-64 requires the most total memory usage. However, SAFER K-64 is also the fastest block cipher compared to the other two block ciphers. If a 6-round SAFER K-64 is used, then the execution speed is comparable to the stream cipher RC4 (6-round SAFER K-64: 110 CPU cycles per byte).

The optimized TREYFER is approximately 33% smaller in size and significantly faster (83% faster) than the standard TREYFER. When used in a real-life application, the 256 bytes TREYFER look-up table values can be derived from other parts of the same application in memory running with TREYFER. It can be seen from the above tables that,

although TREYFER has the smallest code size, it also requires the highest CPU usage.

The optimized RC4 is only slightly smaller (3% smaller code) than the standard RC4 code size. However, with optimized RC4, a significant CPU usage saving is obtained in key initialization (76% faster). RC4 does not require a look-up table in flash, but it requires 256 bytes of S-box in RAM for generating key streams. Stream cipher RC4 is the fastest and requires the least flash memory space.

Table 8.3 shows additional block cipher memory requirements and CPU usage conducted by Law et. al. in [54] (compiler also set for speed optimization). The code size contains both code and look-up tables of the block cipher. CPU usage for encryption is also measured in CPU cycles per byte (on an 8-byte block), and key setup is measured in number of CPU cycles needed. MISTY1 is another royalty-free 128-bit key, 64-bit block cipher; it is not further discussed in this paper. Note that the version of AES used by Law et. al. in [54] has been optimized for speed, sacrificing the storage space with larger code size.

Table 8.3. Additional cryptographic ciphers memory requirements and CPU usage.

	Code (Flash)	**Expanded Key (RAM)**	**Encrypt (CBC)**	**Key setup**
RC5	6312 bytes	152 bytes	620	40556
AES	15842 bytes	240 bytes	400	1313 (encrypt) 5034 (decrypt)
MISTY1	8492 bytes	64 bytes	490	584

There are three differences between the implementation environment in this dissertation and the implementation environment in [54].

1. The block ciphers in [54] are implemented on the same microcontroller family (MSP430) as the one used in this paper. The only difference is that the microcontroller used in [54] has larger memory resources and a built-in hardware

multiplier. However, none of the block ciphers in Table 8.3 requires intensive multiplication operations.

2. The development software used in [54] is the IAR C/C++ compiler version 2.20A, which is older than the version used in this paper. However, the performance differences between the two compiler versions are very slight, especially when the compiler is set to optimize for code speed.

3. The block ciphers in Table 8.3 are implemented in CBC mode, which has higher CPU usage than the ECB mode results in this paper (Table 8.2). However, it is also shown in [54] that the performance differences between different modes are minimal, particularly when the compiler is to compile codes for speed optimization. Note that with the speed optimized codes in this compiler, CBC mode produces code approximately 3 KB bigger than ECB mode.

Therefore despite some minor differences, the results from Law et. al. (Table 8.3) still provide good comparisons against the results obtained in this paper (Table 8.1 and Table 8.2).

8.2.1. Observation and Analysis

From the above performance results and memory requirements, it can be seen that the stream cipher RC4 requires the least flash memory and CPU usage to encrypt one byte; however, it requires the most RAM and fairly high CPU usage for key initialization. Also, being a stream cipher, RC4 cannot be easily adapted with block cipher modes of operation to provide authentication besides encryption. On the other hand, if an efficient stream cipher authentication algorithm and a microcontroller with sufficient RAM is used, RC4 will be a very good choice as a low-power security solution to provide both encryption and authentication. Another advantage of RC4 is that because it produces one keystream byte at a time, it can be customized to the exact packet size of different WSN applications.

The SAFER K-64 block cipher is the fastest block cipher compared to all other block

ciphers. However, it requires more flash and RAM memory than XTEA and TREYFER. TREYFER requires the least flash memory, but it performs poorly and requires the most CPU cycles to encrypt one byte. XTEA appears to be a good compromise between speed performance and code size. Therefore XTEA is suitable for an extremely memory constrained environment, while still providing fairly low power consumption. SAFER K-64 is well suited for environments with slightly more memory where it can be used to achieve even lower power consumption (less CPU usage). However, note that the security of XTEA is higher than the SAFER K-64 because of its 128-bit key length compared to the SAFER K-64's 64-bit key length.

It is observed that some of the popular block ciphers (e.g. RC5 and AES) that are found in many traditional network security packages do not perform that well in the embedded environment. Although AES is faster compared to RC5 and MISTY1, it however requires very large flash memory space (more than 10 KB). Even so, it is still slower than XTEA and SAFER K-64. Both RC5 and MISTY1 also require higher CPU usage than XTEA and SAFER K-64 to encrypt one byte. Furthermore, RC5 also requires fairly high CPU usage for its subkey initialization. The higher CPU usage needed for subkey initialization, the less energy-efficient it is to change its secret key. Therefore both RC5 and AES are not suitable for a WSN environment. Law et. al. [54] recommended using MISTY1 for a memory constrained environment. However, the results of this paper have shown that XTEA requires both lower CPU usage and less memory than MISTY1. SAFER K-64 also requires less flash memory (but more RAM) and performs better compared to MISTY1.

8.3. UMAC

UMAC provides only authentication by calculating the MAC (message authentication code). In this paper, UMAC is customized for small size data using XTEA (block cipher) and RC4 (stream cipher) as its underlying pseudo random number generator.

8.3.1. UMAC-XTEA

The customized UMAC-XTEA can be subdivided into three components: KDF, PDF and

NH. The key derivation function (KDF) generates two subkeys (total of 16 bytes) needed for both the PDF and the NH hash function. The pad derivation function (PDF) generates a one-time encryption pad to be XORed with the hash code. The NH hash function processes eight-byte blocks and produces four-byte (32-bit) hash codes. Every PDF call invokes an XTEA cipher call, which produces an eight-bytes encryption pad; therefore one PDF call provides encryption pads for generating two MACs.

Table 8.4 shows the number of CPU cycles needed for the different function calls within UMAC-XTEA. The NH hash function involves 16-bit multiplication operations, which makes the CPU cycle usage depending on the input value to the NH function. The NH function CPU usage below is the average value across several different input values. Furthermore, some MSP430 microcontrollers have built-in hardware multiplier (HW multiplier) (e.g. MSP430F140). When such microcontrollers are used, the performance of the NH hash function is improved.

Table 8.4. UMAC-XTEA CPU usage.

	KDF	**PDF**	**NH (without HW multiplier)**	**NH (with HW multiplier)**
CPU usage	4679	2300	570	190

The flash memory required for UMAC-XTEA code in the above table is 1333 bytes. This includes code for setting up a simulated 24 byte data packet. If a data packet is of size 24 bytes (3 blocks), then to authenticate such data packet using UMAC-XTEA requires one PDF call and three NH hash function calls (excluding the key setup KDF call).

8.3.2. UMAC-RC4

The customized UMAC-RC4 is similar to the block cipher UMAC-XTEA. It uses the same NH hash function to process eight-byte data blocks and to produce four-byte (32-bit) hash codes. The key derivation function (KDF) in UMAC-RC4 only needs to generate one

eight-byte subkey for the NH. The pad derivation function (PDF) generates four-byte encryption pads to be XORed with the hash code.

Table 8.5 shows the CPU usage (number of CPU cycles) of function calls within UMAC-RC4. The initialization function initializes the RC4 S-box in RAM. Both initialization and KDF only need to be executed once for the lifetime of the same secret key. The NH functions CPU usage is the same as the previous result with UMAC-XTEA.

Table 8.5. UMAC-RC4 CPU usage.

	Initialization	**KDF**	**PDF**	**NH (without HW multiplier)**	**NH (with HW multiplier)**
CPU usage	11154	529	277	570	190

The flash memory required for UMAC-RC4 code in the above table is 1429 bytes. This includes code for setting up a simulated data packet of 24 bytes. To authenticate a data packet of size 24 bytes (3 blocks), with UMAC-RC4 requires one PDF call and three NH hash function calls (excluding RC4 initialization and key setup call of KDF).

8.4. OCB-XTEA

OCB (Offset Codebook) is a block cipher mode that provides both encryption and authentication. OCB is capable of authenticating associated data (e.g. header bytes) without encrypting it. The customized OCB-XTEA in this paper is divided into two components: PMAC and OCB_ENC. PMAC authenticates header bytes (associated data) into 8-byte (64-bit) authentication tags. OCB_ENC encrypts and authenticates the data bytes, and combines the data authentication tag with the header authentication tag from PMAC to obtain a final 64-bit MAC.

In Table 8.6, results (number of CPU cycles) are obtained from simulated data packets of

size 24 bytes. The first 8 bytes are the header bytes and the last 16 bytes are message data bytes. Therefore, the OCB-XTEA-encrypted and -authenticated data packet will have 8-byte headers in plaintext, 16 bytes of encrypted message data, and a 64-bit MAC authenticating both header and message data bytes.

Table 8.6. OCB-XTEA CPU usage on 24 bytes data (8-byte header, 16-byte data).

	PMAC (8-byte header)	**OCB_ENC (16-byte data)**
CPU usage	4898	9790

The flash memory required for OCB-XTEA codes in the above table is 1749 bytes, which includes code for setting up the simulated data packet of 24 bytes.

For a data packet consisting of an 8-byte header and a 16-byte data block (for a total of 3 blocks); PMAC requires one block cipher call, OCB_ENC requires two block cipher calls for encryption, and another block cipher call for producing the final MAC. Another additional block cipher call is required to encrypt the nonce to be used in OCB offsets. Therefore with N blocks, $N+2$ block cipher calls are required. Apart from the block cipher calls, OCB also requires several offset operations (refer to section 4.5.3), which requires a lot less CPU cycles than the block cipher calls.

8.5. Security Primitives Implementations

To provide all security primitives: confidentiality, authenticity and integrity are needed. Encryption provides confidentiality, while authentication using MAC provides both message authentication and integrity checking. This section suggests several combinations of using the abovementioned cryptographic functions to provide encryption, authentication and integrity checking in WSN.

- OCB-XTEA – OCB block cipher mode provides both encryption and

authentication with *N+2* of block cipher calls (*N* being the number of blocks) and some offset operations. Only one secret key is needed in OCB for both encryption and authentication.

- UMAC-XTEA + XTEA – UMAC-XTEA provides only authentication. However, because block cipher XTEA is used as UMAC's underlying pseudo number generator, the same XTEA code can be reused to perform the encryption function. Two different secret keys are needed for UMAC (authentication) and XTEA encryption.

- UMAC-RC4 + RC4 – UMAC-RC4 provides only authentication. The underlying stream cipher RC4 is used as pseudo number generator for UMAC. However, the pseudo random bytes (keystream bytes) generated by RC4 can also be used as encryption pad to be XORed with the message data for encryption. Only one secret key is used for RC4 in the implementation of this paper. However, using two different keys for two RC4 instantiation is possible.

- Generic Block Cipher modes – Generic block cipher modes have been widely used to provide encryption and authentication (e.g. CBC-MAC). Therefore it is included in the performance measurements for a comparison to other non-conventional methods of providing encryption and authentication. Two different secret keys are needed for the encryption mode and the authentication mode.

Table 8.7 shows the CPU usage (number of CPU cycles) of the suggested combinations of cryptographic functions. A simulated data packet of 24 bytes is used, with first 8 bytes being the header and the last 16 bytes being the data bytes. All block cipher performance measurements are done using XTEA. The aim is to observe which cryptographic combinations are most suitable; therefore other block ciphers can be used instead of XTEA.

Table 8.7. Encryption and Authentication CPU usage (24 bytes packet size).

	Encryption & Authentication	**Key setup**
OCB-XTEA	14688	N/A
UMAC-XTEA + XTEA	7446	4679
UMAC-RC4 + RC4	3635	11683
Generic block cipher modes (XTEA) (CBC-MAC + CTR mode)	11465	N/A

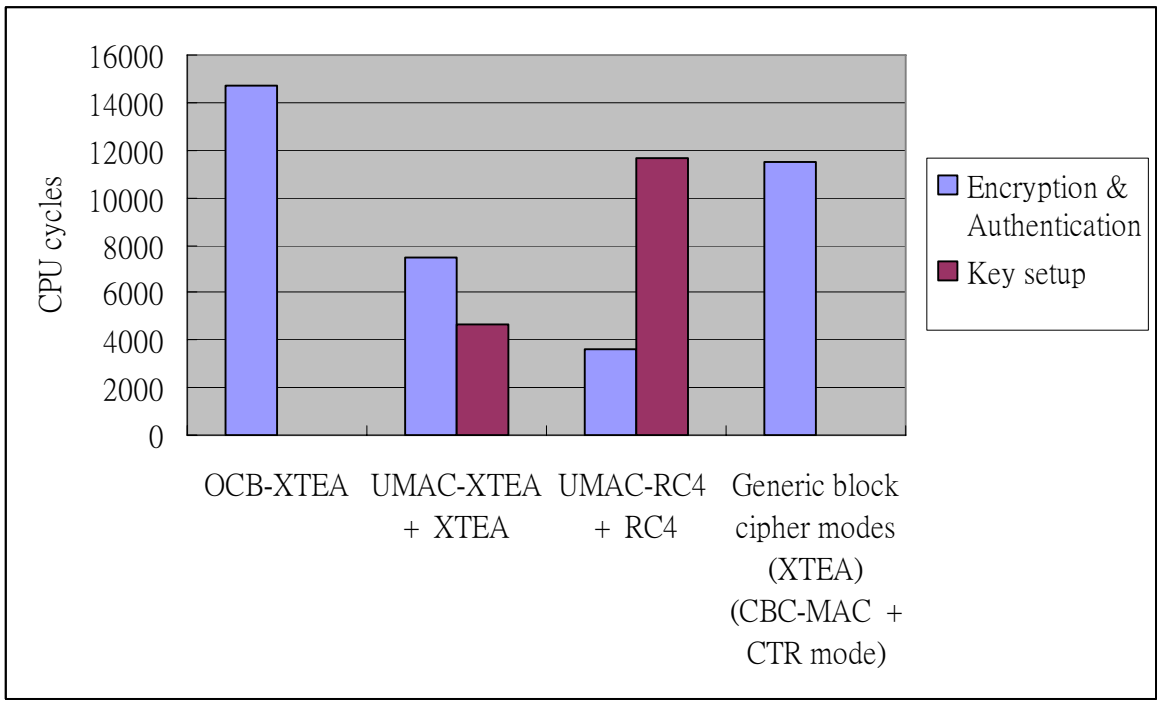

Figure 8.1. Encryption and Authentication CPU usage (24 bytes packet size).

Table 8.8 and Figure 8.4 shows the performance of block cipher cryptographic functions on a packet size of 3 blocks (24 bytes). However, different cryptographic function combinations will perform differently with different packet sizes:

- OCB-XTEA – Requires $N+2$ block cipher calls (N is the number of blocks) and

additional offset operations to provide encryption and authentication. The additional offset operations typically require a lot less CPU cycles than block cipher calls.

- UMAC-XTEA + XTEA – Unlike OCB, the number of block calls in this combination depends also on the header size. This is because header bytes only need to be authenticated and not encrypted; therefore as the header increases in size compared to the message data portion, the number of block cipher calls required will decrease. The block cipher calls required are: $(N - H) + 0.5$ (where N is the total number of blocks and H is the number header blocks). UMAC-XTEA requires half a block cipher call and N NH function calls for authentication. $N - H$ block cipher calls are needed for encryption. For example, a 24 bytes (3 blocks) packet with 8 bytes header (1 block) requires half a block cipher call plus four NH hash function calls for UMAC-XTEA authentication; along with two XTEA calls for encryption.

- Generic block cipher modes – When using generic block cipher modes to encrypt and authenticate N blocks of data, $2N$ block cipher calls are required. However, when authenticating N blocks of data and encrypting $N - H$ blocks of message data, the required number of block cipher calls is: $2(N - H) + H$ (N is the number of blocks and H is the number header blocks). For example, a 24 bytes (3 blocks) packet with an 8 bytes header (1 block) requires three block cipher calls for authentication and two block cipher calls for encryption (five block cipher calls in total).

Table 8.8 shows the number of block calls needed for various size data packets. WSN packet sizes are typically less than 30 bytes. TinyMote [12] has a maximum packet size of 15 bytes (including header bytes). Therefore a comparison of packet sizes between 16-byte to 32-byte systems is reasonable.

Table 8.8. Number of block cipher calls needed for various size data packets.

	16-byte (8-byte header, 8-byte data)	**24-byte (8-byte header, 16-byte data)**	**32-byte (8-byte header, 24-byte data)**
OCB-XTEA	4	5	6
UMAC-XTEA + XTEA	1.5	2.5	3.5
Generic block cipher modes (XTEA) (CBC-MAC + CTR mode)	3	5	7

8.5.1. Observation and Analysis

As shown in Table 8.7, the combination of the UMAC-RC4 + RC4 and the RC4 security primitives requires the least CPU usage to provide both encryption and authentication. Using RC4 also requires fairly little flash memory. However, RC4 needs to maintain a 256-byte S-box in RAM, which may be too much for certain extremely memory constrained sensor node (e.g. TinyMote with only 256 bytes of RAM). Furthermore, as mentioned in section 7.4.2, unlike block cipher modes, the stream cipher RC4 does not use a nonce; therefore the receiver has to keep a copy of the sender's S-box in order to produce the same keystream bytes being used on the received data packet. If two secret keys are used for encryption and authentication, then two RC4 instantiations are required, thus two 256-byte S-boxes will be needed at both the sender and the receiver.

When using a block cipher to provide encryption and authentication, Table 8.7 shows the UMAC (authentication) + block cipher (encryption) has the least CPU usage when the packet size is 24 bytes with 8-byte header. Table 8.8 also shows that UMAC-XTEA + XTEA requires the least block cipher calls and N NH hash function calls (where N is the number of blocks). It has also been noted that NH function calls typically need a lot fewer CPU cycles than block cipher calls. As a result, optimized UMAC + block cipher encryption seem to be a viable security primitives solution in the resource constrained WSN environment.

In Table 8.8, it can be seen that OCB performs better with higher numbers of blocks when compared to the generic block cipher modes of providing encryption and authentication. Even when OCB and generic block cipher modes require the same number of block cipher calls as in the case with 24-byte packet (with 8-byte header), OCB still has a higher CPU usage (Table 8.7). This is because OCB requires additional offset operations than generic block cipher modes. Therefore, with a smaller packet size, it is more efficient to use generic block cipher modes than OCB to provide encryption and authentication; whereas OCB performs better with larger packet size.

For generic block cipher mode authentication, other variants of CBC-MAC can also be used to overcome some shortcomings of CBC-MAC (as discussed in section 4.5.1.2).

8.6. Secure Link Layer Protocol

Secure cryptographic functions on their own cannot assure the security of the network. If the security primitives are not properly implemented in the communication protocol, then attacks on the security flaws of the protocol may be possible even if the cryptographic functions themselves are secure. A well known example of such an attack is in the WEP (Wired Equivalent Privacy) protocol, which is part of the IEEE 802.11 (WiFi) standard. The WEP protocol uses RC4 as its underlying cryptographic function, which is secure. However, the WEP protocol itself has been found to have security flaws and thus has been broken, allowing the secret key to be easily found [61].

This section suggests one way of implementing security primitives as a secure link layer protocol. A detailed study of various methods for implementing security primitives in the link layer is beyond the scope of this paper.

8.6.1. Block Cipher Based

For the suggested secure link layer protocol, the CTR (counter) mode encryption with 8-byte counter is recommended for all the block cipher cryptographic combinations mentioned previously. A 32-bit MAC is recommended for authentication and integrity

checking.

8.6.1.1. CTR mode

The main reason for choosing CTR mode encryption is its simplicity to implement. Unlike the initialization vector (IV) of the CBC mode encryption, a simple incrementing counter can be used as the counter in CTR mode encryption. The security of CBC mode encryption is affected if a counter is used as an IV in the CBC mode (refer to section 4.5.1.1). In the case when using UMAC-block cipher + block cipher encryption (e.g. UMAC-XTEA + XTEA), a simple incrementing counter can be used as the nonces required by UMAC. Therefore with CTR mode encryption, the same counter can be used for both encryption and UMAC authentication.

The counter value in CTR mode must never be repeated within the lifetime of the same secret key. As discussed in section 4.5.2, if the counter value is repeated, severe information leakage occurs. In order to prevent repeated counter value, 8-byte counter is recommended.

The counter (also being the nonce for UMAC) is a non-secret value and is transmitted together with the data packet. Therefore if the full 8-byte counter is being transmitted with the data packet, then it will add too much overhead and greatly increase power consumption. As a result, only the lowest byte of the 8-byte counter should be transmitted with the data packet. The sender and receiver (e.g. base station) in WSN is often synchronized, therefore the receiver is expected to have the remaining higher 7 bytes of the counter value. With the transmitted lowest counter byte, the receiver can regain synchronization with sender if packet loss has occurred. Therefore it can be seen as a "window" which buffers for some packet losses due to an unreliable wireless channel.

If multiple sensor nodes all start with a counter value of zero, then this counter value is in essence repeated in all sensor nodes. This will result in the same encryption pads being produced and used for multiple data packets, which leads to the security problem discussed in section 4.5.2. To prevent this from happening, each node's unique node address can be XORed with the counter value before it is encrypted to derive the encryption pad (as

shown in Figure 8.2, where *P* and *N* are the plaintext and ciphertext block respectively). In this way, no two sensor nodes will have the same encryption pad for encryption even if they are using the same counter value.

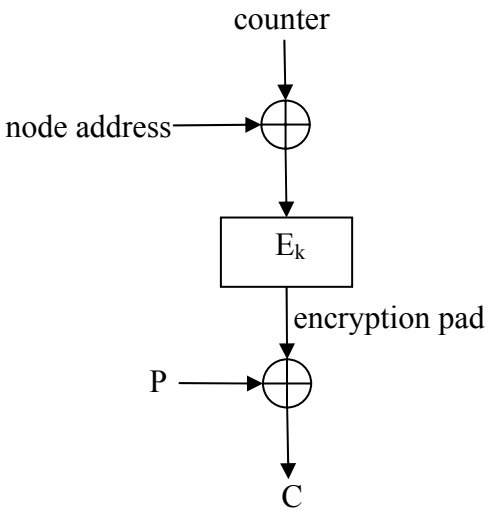

Figure 8.2. Counter XORed with unique node address in the CTR mode.

8.6.1.2. 32-bit MAC

A MAC of 32 bits may seem to be insufficient from a traditional security application's point of view. It is however, sufficient from a practical WSN application's point of view.

In order to forge a MAC, the attacker needs to be "on-line" and to continuously interact with the entity verifying the MAC (e.g. a WSN base station). However, in order to preserve energy in the WSN, communications usually only take place at intervals of a few seconds or a few minutes. Consider the case where sensor nodes take sensor measurements and communicate these readings every two seconds (as in a TinyMote sensor node, refer to section 7.1.1). Then with a 32-bit MAC, there will be $2^{32} = 4.29 \times 10^9$ possible MAC combinations. Then in order to forge a MAC, the attacker needs to communicate all $4.29 \times 10^9 \div 2$ at 2-second intervals. This means a total time of $4.29 \times 10^9 \div 2 \times 2$ seconds is required by the attacker, which translates to approximately 136 years!

8.6.2. RC4 Stream Cipher Based

With the UMAC-RC4 + RC4 implementation, a 32-bit MAC is also recommended. However, as discussed in section 7.4.2, RC4 does not use a nonce (counter) to produce different encryption pads as in the case of CTR mode. Therefore, in an unreliable communication channel, packet loss may cause the sender and the receiver to lose synchronization (receiver loses track on which RC4 keystream bytes are being used). To solve this problem, every packet transmitted can be numbered with a sequence number. If packet losses have occurred, the receiver can detect this by realizing that several sequence numbers have been skipped, so the receiver can also skip some keystream bytes accordingly to decrypt the received packet and regain synchronization.

8.6.3. Observation and Analysis

As a result of the 32-bit MAC and the transmitted lowest counter byte, the new TinyMote packet with encryption and authentication is shown in Figure 8.3, as opposed to the normal packet structure shown in Figure 7.4.

Transceiver		**User specific**				
Preamble (32 bits)	Sync word (16 bits)	Packet length (8 bits)	Source address (16 bits)	Last HC (8 bits)	Data (max 10 bytes)	MAC (32 bits)

(a)

Data						
Mandatory			**optional**			
source HC (8 bits)	sensor type (8 bits)	counter (lowest 8 bits)	V_{CC} (8 bits)	temperature (16 bits)	brightness (16 bits)	humidity (16 bits)

(b)

Figure 8.3. TinyMote packet structure with encryption and authentication.

The grayed bars are authenticated fields, and the authenticated and encrypted fields are covered with grayed dots. Note that the counter must also be authenticated.

Comparing the above figure with the original TinyMote packet at Figure 7.4, it can be seen that the field's CRC, TX slot counter, and V_{Sol} have been discarded. CRCs (Cyclic Redundancy Checks) are used for integrity checking, therefore it is no longer required when using MAC. The TX slot counter has been used only for debugging purpose and can be discarded. The V_{Sol} field is used specifically for measuring the solar cell voltage level. It may also be discarded because the TinyMote sensor nodes only need one power source at a time, thus the same V_{CC} field can be used for indicating either battery voltage or solar cell voltage. As a result, even with the security primitives of both encryption and authentication implemented in TinyMote WSN, only one byte of packet overhead is imposed!

8.7. Security Primitives Power Consumption

As discussed in section 8.1, it is safe to say that energy per cycle is more or less constant for the MSP430 microcontrollers. Therefore the energy consumption for security primitives can be calculated from the number of CPU cycles it requires.

Consider the MSP430F1232 operating at 1 MIPS and an average power of 440 µW. The time to complete one instruction is: $\frac{1}{1\times10^{-6}}$ = 1µs. With OCB-XTEA, it requires 14688 CPU cycles to encrypt and authenticate 24 bytes of data (as shown in Table 8.7). Therefore the time needed for such operation is: 14688 cycles × 1 µs = 14.688 ms. Thus the average power consumption for such OCB-XTEA operation is: $440\mu W \times 14.688ms = 6.46\mu W$. However, if the sensor node only need to perform the security function at a 2-second interval (i.e. packet transmitted at 2-second interval), then the average power would be: $\frac{6.46\mu W}{2} = 3.23\mu W$.

The following table and figure shows the average power consumption per second of the different cryptographic function combinations at 2 second and 10 second packet transmission intervals.

Table 8.9. Encryption and authentication power consumptions.

	2 second	10 second
OCB-XTEA	3.23 µW	646.27 nW
UMAC-XTEA + XTEA	1.64 µW	327.62 nW
UMAC-RC4 + RC4	799.7 nW	159.94 nW
Generic block cipher modes (XTEA) (CBC-MAC + CTR mode)	2.52 µW	504.46 nW

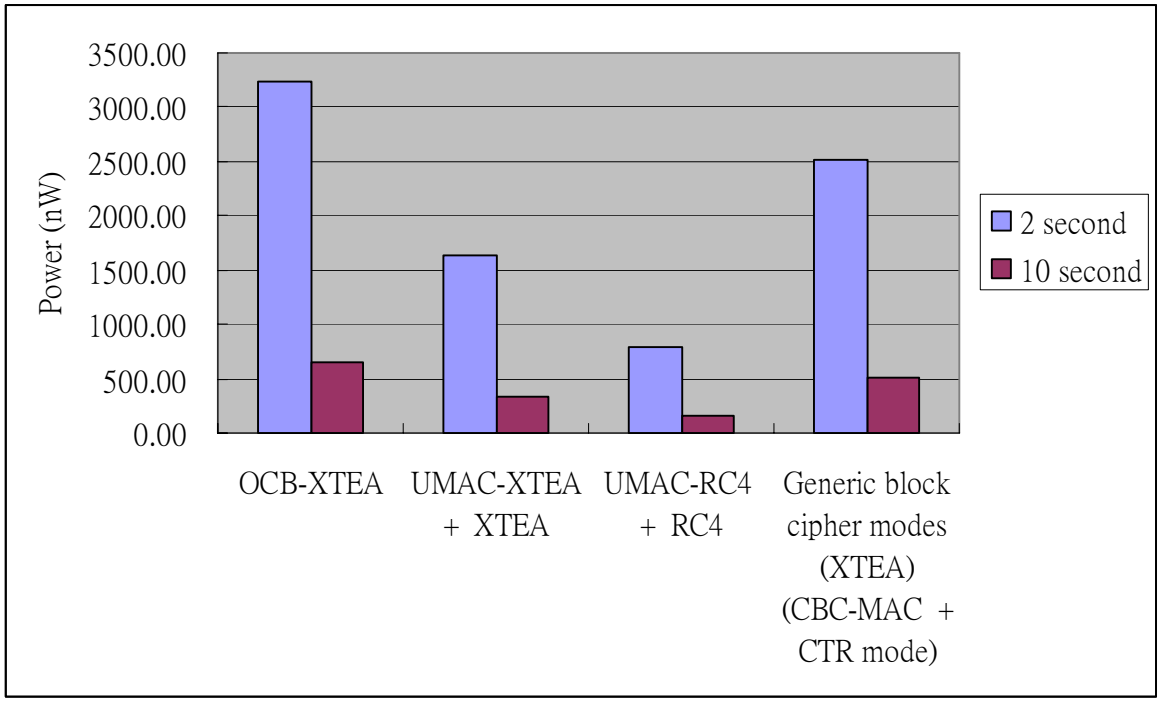

Figure 8.4. Encryption and authentication power consumptions.

8.8. Improvements to Existing WSN Link Layer Securities

The proposed security primitives and the secure link layer protocol implemented in TinyMote sensor networks have shown several improvements when compared to existing secure link layer protocols.

- Packet overhead – The TinySec [6] security protocol requires 5 bytes overhead per transmitted packet to provide encryption and MAC authentication. The SNEP [1] security protocol requires 8 bytes overhead. The proposed security protocol requires only 1 byte overhead for each packet of the TinyMote sensor network. Smaller overhead also means lower power consumption while transmitting data packets.

- Power consumption – Results of the customized UMAC for a small data size have been shown it to be more efficient than the generic use of block ciphers to provide authentication. Therefore the resulting authentication and encryption has shown significant reduction in power consumption compared to other security protocols using generic block cipher modes to provide encryption and authentication.

- Synchronization – SNEP uses the CTR mode for encryption, but it does not transmit the counter value together with the packet. It relies on the receiver to share the same counter value with the sender. However, in an unreliable communication channel, sender and receiver may loose synchronization due to loss of data packets. The proposed security protocol transmits the lowest counter byte with the data packet so that even when a few packet losses have occurred, with the received counter value it is still possible to let the receiver regain synchronization with the sender.

- Security – The proposed security link layer protocol for TinyMote have better security than the existing security protocols in many aspects:

    - The TinySec security protocol uses a counter as its IV (initialization vector) for CBC mode encryption, which results in information leakage issues. The proposed security protocol uses CTR mode and needs only simple incrementing counter.

    - With a transmitting rate of one packet per minute, TinySec IV reuse will occur after every 45 days, which results in leaking some information on

messages with repeated IV. The proposed security protocol uses an 8-byte counter (with only the lowest byte being transmitted in the packet), which will not repeat in a few hundred years even if packets are sent at one second intervals.

- TinySec does not provide protection against a replay attack. The proposed security protocol provides protection against a replay attack by authenticating the counter (as shown in Figure 8.3). If a packet is verified successfully, the receiver knows the packet must be sent after the previously verified packet because it will have a larger counter value.

- SNEP also uses CTR mode for encryption. It however, does not cater for the situation when multiple sensor nodes all start with the same counter value. This repeated counter value give rise to serious security issues as discussed in section 4.5.2. The proposed security protocol prevents such problem by XORing the counter value with the sensor node's unique node address (as shown in Figure 8.2).

8.9. Conclusions

In this chapter the results of the various implementations are recorded. It can be seen that although the TREYFER block cipher has been designed for small size, but has performed poorly. OCB has been designed to improve encryption and authentication performance, but its performance gain is observed only when processing larger data packets. On the other hand, UMAC and XTEA have not been designed for an embedded 16-bit environment, but have been adapted and shown to perform fairly well (as to be discussed in the next chapter).

9

Conclusion and Future Work

In this dissertation, several cryptographic ciphers, block cipher modes and authentication algorithms have been investigated for their power consumption and code size for their applicability in an ultra-low power wireless sensor network environment.

The well known AES and the WSN-popular RC5 block ciphers have been shown to be not very suitable for WSN. The block cipher SAFER K-64 has been investigated for the first time for its applicability in WSN. Compared to other block ciphers investigated for WSN environment, SAFER K-64 achieves the best performance in CPU usage known to the author. It, however, requires slightly more RAM. XTEA requires a fairly small amount of flash/ROM memory and no RAM is needed for the subkeys setup. Even though XTEA is designed for a 32-bit architecture, it performed well on the 16-bit MSP430 platform and outperformed both AES and RC5 on the same MSP430 platform. Although TREYFER requires the least flash memory and also does not need RAM for the subkey setup, it requires a considerable number of CPU cycles. RC4 is the only stream cipher implemented in this paper, but it has been shown to require the least amount of CPU usage. Improvements have been made on XTEA, TREYFER and the initialization of RC4 to further optimize their performance and code size.

The three types of security primitives required for use in wireless sensor networks are encryption, authentication and integrity checking. A fairly new block cipher mode, OCB has been implemented to study its performance in WSN. The UMAC authentication algorithm has also been studied and implemented.

OCB performs better with higher numbers of data blocks to process. When it comes to lower numbers of data blocks, using the generic block cipher modes to provide authentication and encryption performs better than the OCB mode.

Although UMAC is originally designed for a modern 32/64-bit architecture and for authenticating longer messages, in this paper it has been adapted and optimized for the short message WSN environment. Security primitives using UMAC + XTEA for authentication and XTEA (CTR mode) for encryption have been implemented and shown to perform very well. Its power consumption is: 1.64 μW and 327.62 nW at 2 second and 10 second packet transmission intervals respectively, with a packet consisting of an 8-byte header and an 16-byte data (24 bytes in total). It has been shown to be better than other block cipher based security primitives such as OCB mode and the generic block cipher modes for any number of data blocks. UMAC-RC4 has even better speed performance than UMAC-XTEA. It however, also requires the most RAM resources.

In the proposed security link layer protocol, the use of block cipher in CTR mode with an 8-byte counter and a 32-bit MAC is proposed. The proposed security link layer protocol has shown many improvements over the existing security link layer protocols such as the TinySec and the SNEP security protocol.

During the code implementation and optimization, several tradeoffs between performance, code size and RAM usage have been observed. Apart from the conventional tradeoffs between the code size and the RAM space requirements, tradeoffs also exist between code flexibility and code performance. Furthermore, it is also observed that repeated function calls can be very expensive and may require a significant amount of CPU cycles. This is accounted for by the fact that every function calls needs to setup the parameters being passed and to push register data on to stack memory and pop the data back from stack memory when returning from the function call. Therefore these function call performance overheads add up quite significantly when a function needs to be called repeatedly.

This paper has achieved in providing analysis and solutions for security primitives in wireless sensor networks. However, possible future work may include:

- Similar to the TinySec security protocol, the proposed security primitives in this paper can also be packaged into a set of security primitive APIs (application program interfaces) to allow ease of use for higher layer WSN protocol designers.

- The research on security primitives in this paper are based on a microcontroller platform. However, cryptographic ciphers designed for hardware implementations also exist, particularly in stream ciphers. One example is the linear-feedback-shift-register (LFSR) based stream ciphers, which have been used in smart card microcontrollers for encryption [63]. Therefore it is necessary to conduct further researches into the possibility of using either ultra-low power cryptographic hardware or existing smart card microcontrollers.

REFERENCES

[1] A. Perrig, R. Szewczyk, V. Wen, D. Culler and J.D.Tygar, "SPINS: Security Protocols for Sensor Networks", *Proceedings of 7th Annual International Conference on Mobile Computing and Networks (Mobicom)*, pp. 189-199, Rome, Italy, 2001.

[2] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, *Handbook of Applied Cryptography*, 5th ed., CRC Press, 1996.

[3] C. Karlof and D. Wagner, "Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures", *Proceedings of the 1st IEEE International Workshop on Sensor Network Protocols and Applications (SPNA)*, pp. 113-127, Anchorage, USA, May 2003.

[4] P. Ganesan, R. Venugopalan, P. Peddabachagari et. al., "Analyzing and Modeling Encryption Overhead for Sensor Network Nodes", *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, San Diego, USA, 2003.

[5] Y. Law, S. Dulman, S. Etalle et. al., "Assessing Security-Critical Energy-Efficient Sensor Networks", Department of Computer Science, University of Twente, Tech. Rep. TR-CTIT-02-18, 2002.

[6] C. Karlof, N. Sastry and D. Wagner, "TinySec: A Link Layer Security Architecture for Wireless Sensor Networks", *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor System (SenSys)*, vol. 47 issue 6, Baltimore, USA, November 2004.

[7] S. Mahlknecht, "Energy-Self-Sufficient Wireless Sensor Networks for the Home and Building Environment", Doctor's thesis, Technical University of Vienna, 2004.

[8] H. Y. Yang, H. Luo, F. Ye et. al., "Security in Mobile Ad Hoc Networks: Challenges and Solutions", *IEEE Wireless Communications Magazine*, pp. 38-47, February 2004.

[9] E. Shi and A. Perrig, "Designing Secure Sensor Networks", *IEEE Wireless Communications Magazine*, pp. 38-43, December 2004.

[10] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam and E. Cayirci, "A Survey on Sensor Networks", *IEEE Communications Magazine*, pp. 102-114, August 2002.

[11] Y. Chun Hu, A. Perrig, and D. Johnson, "Ariadne: A Secure On-demand Routing Protocol for Ad Hoc Networks", *Proceedings of 8th Annual International Conference on Mobile Computing and Networks (Mobicom)*, Atlanta, USA, September 2002.

[12] S. Mahlknecht and M. Rötzer, "Energy-Self-Sufficient Wireless Sensor Networks",

Technical University of Vienna, Tech. Rep., 2005.

[13] Intel Mote, http://www.intel.com/research/exploratory/motes.htm. Last accessed on June 2005.

[14] Mica Mote, http://www.xbow.com. Last accessed on June 2005.

[15] A. El-Hoiydi, "WiseMAC, An Ultra Low Power MAC Protocol for the WiseNET Wireless Sensor Network", *Proceedings of the 1st ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, Los Angeles, USA, November 2003.

[16] C. Schurgers, V. Tsiatsis, S. Ganeriwal and M. Sirvastava, "Optimizing Sensor Networks in the Energy-Latency-Density Design Space", *IEEE Transactions on Mobile Computing*, vol. 1 no. 1, pp. 70-80, January 2002.

[17] M. Ringwald and K. Römer, "BitMAC: A Deterministic, Collision-Free, and Robust MAC Protocol for Sensor Networks", *Proceedings of the 2nd European Workshop on Wireless Sensor Networks*, pp. 57-69, Istanbul, Turkey, January 2005.

[18] S. Mahlknecht and M. Böck. "CSMA-MPS: A Minimum Preamble Sampling MAC Protocol for Low Power Wireless Sensor Networks", *Proceedings of IEEE International Workshop on Factory Communication Systems*, pp. 73-80, Vienna, Austria, September 2004.

[19] S. Mahlknecht and M. Böck. "On the use of High Bit Rate Transceivers for Low Duty Cycle Wireless Sensor Networks", *IEEE 7th Africon Conference in Africa*, pp. 1235-1238, Gaborone, Botswana, September 2004.

[20] Cryptography for Ultra-Low Power Devices, http://www.crypto.wpi.edu. Last accessed on October 2005.

[21] EYES: Energy Efficient Sensor Networks, http://eyes.eu.org. Last accessed on June 2005.

[22] H. Chan, A. Perrig, and D. Song, "Random Key Predistribution Schemes for Sensor Networks", *Proceedings of IEEE Symposium on Security and Privacy*, pp. 197-213, Berkeley, USA, May 2003.

[23] L. Feeney, B. Ahlgren, and A. Westerlund, "Spontaneous networking: An Application-oriented Approach to Ad Hoc Networking", *IEEE Communications Magazine*, pp. 176–181, June 2001.

[24] SmartDust, http://robotics.eecs.berkeley.edu/~pister/SmartDust/. Last accessed on July 2005.

[25] TinyOS, http://www.tinyos.net. Last accessed on October 2005.

[26] W. Stallings, *Cryptography and Network Security Principles and Practices*, 3rd ed.,

Prentice Hall, 2003.

[27] B. Schneier, *Applied Cryptography : protocols, algorithms, and source code in C*, 2nd ed., John Wiley & Sons, 1995.

[28] J.D. Golić, V. Bagini, and G. Morgari, "Linear Cryptanalysis of Bluetooth Stream Cipher", *Advances in Cryptology – EUROCRYPT'02*, LNCS 2332, Springer-verlag, 2002.

[29] E. Barkan, E. Biham, and N. Keller. "Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication", *Advances in Cryptology – CRYPTO'03*, LNCS 2729, Springer-verlag, 2003.

[30] Wireless Medium Access Control and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs). IEEE Standard, 802.15.4-2003, ISBN 0-7381-3677-5, May 2003.

[31] PicoRadio Project, http://bwrc.eecs.berkeley.edu/Research/Pico_Radio. Last accessed on October 2005.

[32] C. Enz, A. El-Hoiydi, J. Decotignie and V. Peiris, "WiseNET: An Ultra Low-Power Wireless Sensor Network Solution", *IEEE Computer Magazine*, pp. 62-70, August 2004.

[33] Texas Instrument MSP430 Microcontroller, http://www.ti.com. Last accessed on October 2005.

[34] Atmel AVR Microcontroller, http://www.atmel.com/products/avr/. Last accessed on September 2005.

[35] J. Mulder, S. Dulman, D. van Hoesel and P. Havinga, "PEEROS - System Software for Wireless Sensor Networks", August 2003, http://tobasco.ctit.utwente.nl/~dulman/docs/systemsoft.pdf. Last accessed on October 2005.

[36] M. Needham and D. Wheeler, "TEA Extensions", Computer Laboratory, University of Cambridge, Tech. Rep., October 1997.

[37] M. Russell. "Tinyness: An Overview of TEA and Related Ciphers", Feburary 2004, http://www-users.cs.york.ac.uk/~matthew/TEA/. Last accessed on June 2005.

[38] J. Kelsey, B. Schneier, and D. Wagner, "Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES", *Advances in Cryptology – CRYPTO'96*, LNCS 1109, pp. 237-251, Springer-verlag, 1996.

[39] J. Kelsey, B. Schneier, and D. Wagner, "Related-key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X NewDES, RC2, and TEA", *Proceedings of the 1st*

International Conference on Information and Communication Security, LNCS 1334, pp. 233-246, Springer-verlag, Beijing, China, 1997.

[40] Y. Ko, S. Hong, W. Lee et. al., "Related Key Differential Attacks on 26 Rounds of XTEA and Full Rounds of GOST", *Proceedings of Fast Software Encryption – FSE'04*, LNCS 3017, Springer-verlag, New Delhi, India, 2004.

[41] M. Saarinen, "Cryptanalysis of Block TEA", http://www.cc.jyu.fi/mjos/block_tea.ps. Last accessed on October 1998.

[42] G. Yuval, "Reinventing the Travois: Encryption/MAC in 30 ROM Bytes", *Fast Software Encryption – FSE'97*, LNCS 1267, pp. 205-209, Springer-verlag, 1997.

[43] A. Biryukov, D. Wagner, "Slide Attacks", *Fast Software Encryption - FSE'99*, LNCS 1636, pp.245, Springer-verlag. 1999.

[44] J. Massey, "SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm", *Fast Software Encryption – FSE'94*, LNCS 809, pp. 1-17, Springer-verlag, 1994.

[45] H. Wu, F. Bao, R. H. Deng et. al., "Improved Truncated Differential Attacks on SAFER", *ASIACRYPT'98*, LNCS 1514, pp. 133-147, Springer-verlag, 1998.

[46] M. Bellare, A. Desai, E. Jokipii, et. al., "A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operation", *Proceedings of 38th Annual Symposium on Foundations of Computer Science (FOCS 97)*, Miami, USA, 1997.

[47] M. Bellare, J. Kilian, and P. Rogaway. The Security of the Cipher Block Chaining Message Authentication Code. *Journal of Computer and System Sciences*, 61(3):362 – 399, December 2000.

[48] E. Dawson and L. Nielsen, "Automated Cryptanalysis of XOR Plaintext Strings", *Cryptologia*, pp. 165-181, April 1996.

[49] NIST web page on different modes of operation, http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/. Last accessed on September 2005.

[50] V. Gligor and P. Donescu, "Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes", *Fast Software Encryption – FSE'01*, LNCS 2355, pp. 92-108, Springer-verlag, 2001.

[51] C. Jutla, "Encryption Modes with Almost Free Message Integrity", *Advances in Cryptology – EUROCRYPT'01*, LNCS 2045, Springer-verlag, 2001.

[52] P. Rogaway, M. Bellare, J. Black et. al., "OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption", August 2001, http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/. Last accessed on July 2005.

[53] P. Rogaway, "OCB: Background FAQ", March 2005, http://www.cs.ucdavis.edu/~rogaway/ocb/. Last accessed on June 2005.

[54] Y. Law, J. Doumen, and P. Hartel, "Benchmarking Block Ciphers for Wireless Sensor Networks (Extended Abstract)", *Proceedings of 1st IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, Ft. Lauderdale, USA, 2004.

[55] J. Black, S. Halevi, H. Krawczyk et. al., "UMAC: Fast and Secure Message Authentication", *Advances in Cryptology – CRYPTO'99*, LNCS 1666, pp. 216. Springer-verlag, 1999.

[56] J. Black, S. Halevi, H. Krawczyk et. al., "Update on UMAC Fast Message Authentication", May 2000, http://www.cs.ucdavis.edu/~rogaway/umac/update.pdf. Last accessed on April 2005.

[57] T. Krovetz, "UMAC: Message Authentication Code using Universal Hashing" 2000, internet draft, http://www.ietf.org/internet-drafts/draft-krovetz-umac-07.txt. Last accessed on April 2005.

[58] K. Yüksel, "Universal Hashing for Ultra-Low-Power Cryptographic Hardware Applications", Master's thesis, Worcester Polytechnic Institute, May 2004.

[59] IAR Embedded Workbench IDE for MSP430, http://www.iar.com. Last accessed on August 2005.

[60] Chipcon AS Inc., datasheet CC2400 (Rev. 1.3), Oct. 2004, http://www.chipcon.com. Last accessed on July 2005.

[61] S. Fluhrer, I. Mantin and A. Shamir, "Weaknesses in the Key Scheduling Algorithm of RC4", *Proceedings of 8th workshop on Selected Areas in Cryptography (SAC)*, LNCS 2259, Springer-verlag, Toronto, Canada, 2001.

[62] T. Iwata and K. Kurosawa, "OMAC: One-key CBC MAC", *Fast Software Encryption – FSE'03*, LNCS 2887, pp. 129-153, Springer-verlag, 2003.

[63] Shrinking generator in Smart Card, http://www.maxking.com/basiccard.htm. Last accessed on September 2005.

[64] J. Großschadl, R. Avanzi, E. Savas et. al., "Energy-Efficient Software Implementation of Long Integer Modular Arithmetic", *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, LNCS 3659, pp. 70-90, Springer-verlag, Edinburgh, UK, 2005.

[65] X. Lai, R. Rueppel, and J. Woollven, "A Fast Cryptographic Check-sum Algorithm based on Stream Ciphers", *Advances in Cryptology – AUSCRYPT'92*, LNCS 718, pp. 339-348, Springer-verlag, 1992.

[66] I. Mironov, "(Not So) Random Shuffles of RC4", *Advances in Cryptology – CRYPTO'02*, LNCS 2442, Springer-verlag, 2002.

ADDENDUM A

LIST OF FIGURES

Figure 2.1. Representation of a wireless sensor network architecture. 20
Figure 2.2. Functional diagram of a wireless sensor node. 21
Figure 3.1. Representation of a data packet with MAC appended. 38
Figure 3.2. MAC categories. 39
Figure 4.1. One TEA cycle (two Feistel rounds) [37]. 43
Figure 4.2. One XTEA cycle [37]. 44
Figure 4.3. Encryption structure of SAFER K-64. 45
Figure 4.4. One encryption round structure of SAFER K-64. 47
Figure 4.5. Key scheduling algorithm of SAFER K-64. 49
Figure 4.6. Cipher Block Chaining (CBC) mode encryption. 55
Figure 4.7. Cipher Block Chaining (CBC) mode decryption. 55
Figure 4.8. Standard CBC-MAC. 57
Figure 4.9. Counter (CTR) mode encryption. 58
Figure 4.10. Counter (CTR) mode decryption. 58
Figure 4.11. OCB encryption and authentication on a message. 61
Figure 4.12. OCB authentication on message header of multiple block size (PMAC). 63
Figure 4.13. OCB authentication on message header not multiple of block size (PMAC). 64
Figure 5.1. UMAC. 69
Figure 5.2. NH hash function with word size $w = 32$, number of words processed $n = 4$. . 71
Figure 5.3. Refined UMAC. 74
Figure 5.4. UHASH with word size $w = 32$. 75
Figure 6.1. The TinySec and TinyOS packet format. 82
Figure 7.1. TinyMote with 2 AA batteries (left). 85
Figure 7.2. TinyMote connected to USB dongle. 87
Figure 7.3. TinyMote network behavior. 89
Figure 7.4. TinyMote packet structure. 90
Figure 7.5. Power consumption 187 µW (channel listening: 47 µW, packet sending: 41.14µW, sensors: 98.38 µW) (2s measurement interval, 10s transmission

interval)..... 92
Figure 7.6. Power consumption 70 µW (channel listening: 9.50 µW, packet sending: 41.14µW, sensors: 19.68 µW) (10s measurement interval, 10s transmission interval)..... 92
Figure 7.7. customized UMAC. 98
Figure 7.8. NH hash function with word size $w = 16$ and number of words processed in a NH block $n = 4$. 99
Figure 7.9. UMAC with RC4. 100
Figure 8.1. Encryption and Authentication CPU usage (24 bytes packet size)..... 112
Figure 8.2. Counter XORed with unique node address in the CTR mode. 117
Figure 8.3. TinyMote packet structure with encryption and authentication..... 118
Figure 8.4. Encryption and authentication power consumptions. 120

LIST OF TABLES

Table 2.1. A comparison between mobile ad-hoc network and wireless sensor network... 19
Table 2.2. An overview of different devices' power consumptions..... 23
Table 2.3. Comparisons between TinyMote and MICA2/MICA2DOT sensor nodes. 30
Table 8.1. Cryptographic ciphers memory requirements. 103
Table 8.2. Cryptographic ciphers CPU usage. 104
Table 8.3. Additional cryptographic ciphers memory requirements and CPU usage. 105
Table 8.4. UMAC-XTEA CPU usage. 108
Table 8.5. UMAC-RC4 CPU usage. 109
Table 8.6. OCB-XTEA CPU usage on 24 bytes data (8-byte header, 16-byte data). 110
Table 8.7. Encryption and Authentication CPU usage (24 bytes packet size). 112
Table 8.8. Number of block cipher calls needed for various size data packets..... 114
Table 8.9. Encryption and authentication power consumptions..... 120