Cryptographic Key Distribution in Wireless Sensor Networks Using Bilinear Pairings

By
Piotr Szczechowiak
BEng, MSc

Thesis directed by:
Dr. Martin Collier and Prof. Michael Scott

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

September 2010

Faculty of Engineering & Computing
Dublin City University

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy *is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.*

Signed:

ID number: 56124023

Date: 22/09/2010

Abstract

It is envisaged that the use of cheap and tiny wireless sensors will soon bring a third wave of evolution in computing systems. Billions of wireless senor nodes will provide a bridge between information systems and the physical world. Wireless nodes deployed around the globe will monitor the surrounding environment as well as gather information about the people therein. It is clear that this revolution will put security solutions to a great test.

Wireless Sensor Networks (WSNs) are a challenging environment for applying security services. They differ in many aspects from traditional fixed networks, and standard cryptographic solutions cannot be used in this application space. Despite many research efforts, key distribution in WSNs still remains an open problem. Many of the proposed schemes suffer from high communication overhead and storage costs, low scalability and poor resilience against different types of attacks. The exclusive usage of simple and energy efficient symmetric cryptography primitives does not solve the security problem. On the other hand a full public key infrastructure which uses asymmetric techniques, digital signatures and certificate authorities seems to be far too complex for a constrained WSN environment. This thesis investigates a new approach to WSN security which addresses many of the shortcomings of existing mechanisms. It presents a detailed description on how to provide practical Public Key Cryptography solutions for wireless sensor networks. The contributions to the state-of-the-art are added on all levels of development beginning with the basic arithmetic operations and finishing with complete security protocols.

This work includes a survey of different key distribution protocols that have been developed for WSNs, with an evaluation of their limitations. It also proposes Identity-Based Cryptography (IBC) as an ideal technique for key distribution in sensor networks. It presents the first in-depth study of the application and implementation of Pairing-Based Cryptography (PBC) to WSNs. This is followed by a presentation of the state of the art on the software implementation of Elliptic Curve Cryptography (ECC) on typical WSN platforms. New optimized algorithms for performing multiprecision multiplication on a broad range of low-end CPUs are introduced as well. Three novel protocols for key distribution are proposed in this thesis. Two of these are intended for non-interactive key exchange in flat and clustered networks respectively. A third key distribution protocol uses Identity-Based Encryption (IBE) to secure communication within a heterogeneous sensor network. This thesis includes also a comprehensive security evaluation that shows that proposed schemes are resistant to various attacks that are specific to WSNs. This work shows that by using the newest achievements in cryptography like pairings and IBC it is possible to deliver affordable public-key cryptographic solutions and to apply a sufficient level of security for the most demanding WSN applications.

Acknowledgments

First of all I would like to thank the Faculty of Engineering and Computing for funding my research under the Connect scholarship scheme. I am grateful to many people for help, both direct and indirect, in completing this thesis. I would never achieved the goal without the help of my colleagues, friends and family.

I would like to express my deepest gratitude to my supervisors Prof. Mike Scott and Dr. Martin Collier for their continuous support and encouragement. The work that I have done would not be possible without the guidance and dedication of Prof. Scott. I wish to thank Dr. Liam Marnane, Dr. Noel Murphy and Dr. Stephen Blott for serving on my doctoral committee and for their helpful comments and suggestions.

I cannot forget to acknowledge all my friends from Dublin City University. I would like to thank my fellow lab members, past and present Szymon, Damien, Amitabha, Mei and Alan. I am particularly grateful to Szymon for his friendship and great advices. Those long hours in the lab would not be the same without frequent discussions in our native language. Big thanks goes to all the football lads Aubrey, Brendan, Sean and others. It was always a pleasure to kick the ball with you to get rid of the everyday stress.

Chciałbym rownież podziękować mojej rodzinie a w szczególności moim kochanym rodzicom za miłość, oddanie i wielkie wsparcie, które okazywali mi przez te wszystkie lata trwania doktoratu. Wiem, że wszystko to co osiągnąłem nie byłoby możliwe bez waszej pomocy. Jestem wam za to niezmiernie wdzięczny. Tą pracę dedykuję wam drodzy rodzice.

Na koniec chciałbym bardzo podziękować osobie, która zajmuje szczególne miejsce w moim sercu. Dziękuję kochana Karolinko za twoją miłość, cierpliwość i wsparcie. Dziekuję za codzienne rozmowy i częste wizyty w Irlandii. Nadałaś większy sens mojemu życiu. Chcę żebyś wiedziała, że wszystko co robię, robię dla nas.

Poznań, Dublin, September 2010 *Piotr Szczechowiak*

List of Publications

- P. SZCZECHOWIAK AND M. SCOTT, *Enabling Practical Public Key Cryptography in Wireless Sensor Networks*, Research Colloquium on "Wireless as an enabling technology: Innovation for a critical infrastructure", April 2010, Royal Irish Academy, Committee for Communications and Radio Science

- P. SZCZECHOWIAK, M. SCOTT AND M. COLLIER, *Securing Wireless Sensor Networks: An Identity-Based Cryptography Approach*, International Journal of Sensor Networks (IJSNet), Volume 8, Number 4, 2010, Inderscience

- P. SZCZECHOWIAK AND M. COLLIER, *TinyIBE: Identity-Based Encryption for Heterogeneous Sensor Networks*, in ISSNIP 2009: Proceedings of the 5th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Melbourne, Australia, December 2009.

- P. SZCZECHOWIAK AND M. COLLIER, *Practical Identity-Based Key Agreement for Secure Communication in Sensor Networks*, in ICCCN '09: Proceedings of the 18th International Conference on Computer Communications and Networks, San Francisco, USA, August 2009, pages 1-6, IEEE.

- P. SZCZECHOWIAK, A. KARGL, M. SCOTT AND M. COLLIER, *On the Application of Pairing-based Cryptography to Wireless Sensor Networks*, in WiSec '09: Proceedings of the Second ACM Conference on Wireless Network Security, Zürich, Switzerland, March 2009, pages 1-12, ACM.

- P. SZCZECHOWIAK, L.B. OLIVIERA, M. SCOTT, M. COLLIER AND R. DAHAB, *NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks*, in Wireless Sensor Networks – EWSN 2008, Lecture Notes in Computer Science vol. 4913, pages 305-320, Springer-Verlag, February 2008.

- M. SCOTT AND P. SZCZECHOWIAK, *Optimizing Multiprecision Multiplication for Public Key Cryptography*, Cryptology ePrint Archive, Report 2007/299, http://eprint.iacr.org/2007/299, August 2007.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF ACRONYMS

AES	Advanced Encryption Standard
ALU	Arithmetic Logic Unit
ANSI	American National Standards Institute
ARM	Advanced RISC Machine
ASIC	Application Specific Integrated Circuit
BDHP	Bilinear Diffie-Hellman Problem
CCTV	Closed Circuit Television
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DES	Data Encryption Standard
EEPROM	Electrically Erasable Programmable Read-Only Memory
FPGA	Field Programmable Gate Array
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
LEACH	Low-Energy Adaptive Clustering Hierarchy
LED	Light-Emitting Diode
LSB	Least Significant Bit
MIRACL	Multiprecision Integer and Rational Arithmetic C/C++ Library
MSB	Most Significant Bit
NIST	National Institute of Standards and Technology

PCB Printed Circuit Board

PDA Personal Digital Assistant

RAM Random Access Memory

RF Radio Frequency

RISC Reduced Instruction Set Computer

ROM Read Only Memory

RSA Rivest-Shamir-Adleman

SSL Secure Sockets Layer

TDMA Time Division Multiple Access

WEP Wired Equivalent Privacy

WSN Wireless Sensor Network

CHAPTER 1

Introduction

Today's computing systems have evolved much since the 70's when the first personal computers were released. Advances in electronics and circuit miniaturization has brought us to a point where a fully operational computing system can fit on a tip of a finger. Taking into consideration the size of computers and their numbers, we can easily distinguish three waves in evolution of computing (Fig 1.1). In the beginning computers were very rare and usually operated by few people. The advent of mobile phones and notebook computers has resulted in millions of units sold at the turn of the century. Nowadays, the third wave of evolution is about to happen. Billions of tiny and cheap computing devices deployed around the world will soon gather, analyze and exchange information about the people and the environment they live in. Those pervasive computing devices are not personal computers as we tend to think of them, but very small devices, that will be eventually embedded in almost any type of object imaginable, including cloths, cars, tools, appliances and various consumer goods.

Those tiny devices embed sensors and wireless transceivers to communicate with each other and form distributed networks connected to the Internet. Their main task is to build a bridge between the physical world and the digital world that is displayed on our computer screens. In this way it is possible not only to monitor the physical world but also actuate remotely depending on the measurement data. With advances in this field, new intelligent control systems will be available that are completely autonomous and can

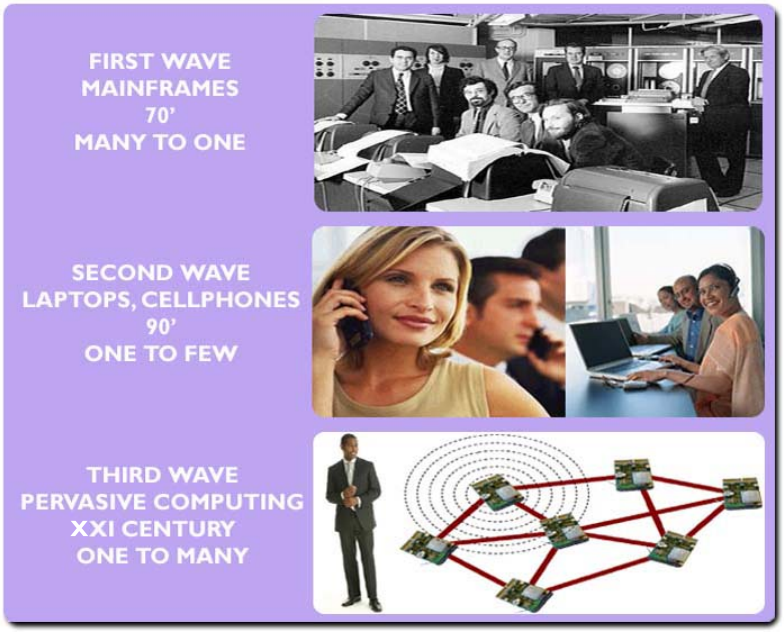

Figure 1.1: *Three waves of computing systems evolution*

act adequately without human supervision.

At the dawn of a new century we are facing new problems that will need definite solutions in the near future. Our planet's climate is changing rapidly and we need global and accurate measurement data to better understand what is happening around us. We also need to reduce communication systems costs, and conserve energy and natural resources. Wireless sensors technology brings us a step closer to resolving those (and many more) issues. Sensors can even alert us about incoming natural disasters and help fight against environmental degradation. The emergence of Wireless Sensor Networks (WSNs) is brought about by a convergence of advanced electronic and wireless technologies. There is no doubt that WSNs will be deployed on a large scale in many different parts of the world. This next generation of computing brings many new challenges not only because we are dealing with very constrained devices, but also because scale really matters here.

The whole WSN paradigm is often described with terms such as "smart dust" [119], the "internet of things" or "pervasive computing" that will revolutionize the way people live. This revolution, however, can only happen if we find the solution to crucial problems of security and privacy in these systems. This is especially important for widespread adoption of WSN technology in many different domains. The range of WSN applications spans from environmental monitoring and home automation to more complex ones like

traffic control and health care. All these applications require various security levels and services. Simple applications like habitat monitoring (e.g [84]) need only basic security assurances that can be fulfilled by using symmetric key systems.

There are also many practical applications, where sensor devices can control the operation of critical equipment, monitor assembly lines and perform condition based monitoring of critical structures. For example, sensor devices are deployed on the Golden Gate Bridge in San Francisco to monitor structure vibrations [72]. The importance of security in such an application justifies the use of a high security level in the network.

We can identify a whole range of commercial applications that are deployed in public areas, where the threat of a physical attack is larger than usual. These applications include water quality monitoring, tunnel lighting control, street traffic and parking monitoring. In many cases, commercial buildings waste vast amounts of energy by inefficient Heating, Ventilation and Air Conditioning (HVAC) usage. Wireless sensor networks can provide intelligent control based on precise real-time measurements to reduce the energy consumption of those systems. Sensor networks can also provide energy monitoring and automatic meter readings in our homes. All such systems require reliable security solutions that will allow wider deployments in the real world.

There are also other important applications in the military and health-care spheres, where security has the highest priority. Such systems should depend on public-key technology in order to fulfil all the security requirements. There is a need for a scalable public key cryptography solution that would set the stage for an array of new and innovative applications in wireless sensor network arena.

Sensor networks are easier to attack and harder to protect than other types of networks. In many cases, sensor networks are deployed in open, unattended areas that anyone can access. The wireless nature of communication between the sensor nodes makes it easy to eavesdrop, intercept and inject bogus information into an unprotected network. The security problem becomes even more crucial when we allow wireless senors not only to gather data about people and the environment, but also to actuate on their own, based on their sensor readings. It is obvious that such a revolution in computing puts security solutions to a great test.

Wireless sensor networks are clearly a very challenging environment for applying security services. They differ in many aspects from traditional fixed, networks and stan-

dard cryptographic solutions cannot be used in this application space. Looking at existing security mechanisms for WSNs, we can see that they are still in their early stages and the level of security provided is not satisfactory for many applications. Despite many research efforts, the problems of key distribution and authentication are still open and require new cryptographic solutions. These new security mechanisms have to take advantage of specific sensor network features and meet the strict limitations of WSN hardware platforms.

1.1 Wireless Sensor Network overview

Wireless sensor networks can be described as an emerging new technology with a very promising future. However, someone may actually ask the question; why do we really need wireless communication in this area when wired sensor systems are working fine for years now. Wired sensors can be found in many places. Buildings are full of CCTV cameras, intrusion sensors, smoke detectors, light sensors and many others. It may seem that moving to the wireless domain is not necessary, as the RF spectrum is already crowded by other communication systems. The main advantage of WSNs is that they have much lower installation and connection cost. This is especially important in the case of large distributed systems where the price of wiring can be really significant. Additionally, wireless communication allows deployment in new inaccessible areas even by people without any prior engineering experience.

We can define a WSN as a wireless network of spatially-distributed autonomous devices that use sensors to cooperatively monitor physical or environmental conditions, such as temperature, humidity, sound, vibration, pressure, motion or pollution, at different locations [102]. A typical WSN consist of a large number of tiny computing devices that relay information through each other until it reaches the destination node (sink node). The sink node is a gateway device that has a connection with the fixed network and in some cases is a higher class device with greater capabilities. An example of a typical WSN scenario is presented in Fig 1.2.

Usually each node in a sensor network can collect data, act as a router and be a source of information at the same time. Such a diversity in functionality is not common in traditional fixed networks where a dedicated device is typically used for each function. This approach requires that each sensor node can support a multi-hop routing algorithm for

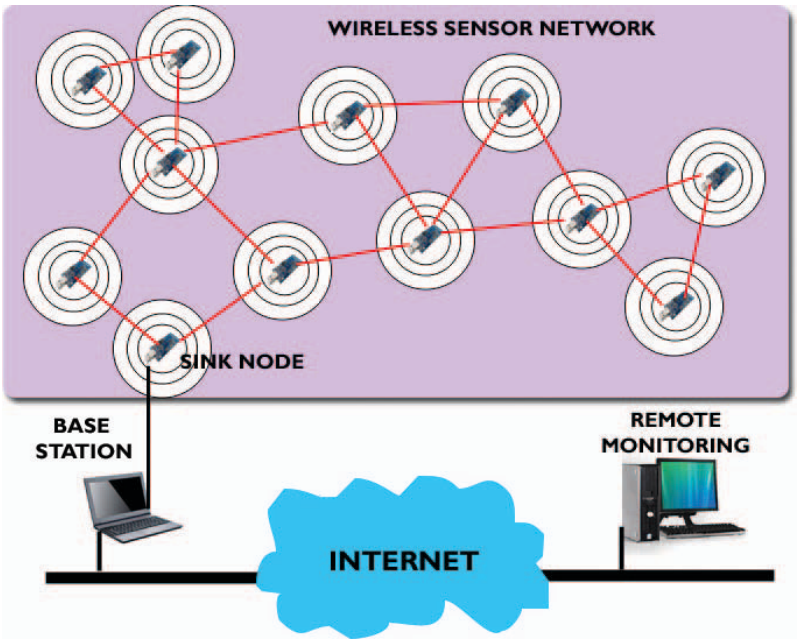

Figure 1.2: *Typical wireless sensor network scenario*

finding the routes in the network.

A single WSN device (apart from some sensors) is typically equipped with a radio transceiver, memory, battery and a Central Processing Unit (CPU) that controls the operation of the whole device. Sensor nodes are often called *motes* [1] and come in many different shapes and sizes. The smallest System-on-a-Chip (SoC) devices designed for research purposes (Smart Dust project) were just $2.5mm \times 2.5mm$ [62]. The crucial factor here is the size of the battery and the antenna. Basically the smallest fully operational devices are the size of the smallest batteries manufactured so far. Typical commercial motes are usually no smaller than two AA batteries or coin cells.

Wireless sensors are designed to support unattended operation for long periods of time. For many applications their expected lifespan on a single pack of batteries is around 2 years. This of course varies a lot depending on the algorithm's computational complexity, communication overhead and duty cycling. Energy conservation is one of the most important issues in sensor networks that needs to be considered when designing a network protocol. Some platforms try to solve this problem by using alternative, renewable energy sources. The most promising concepts are the ones that utilizes solar cells, thermal power or vibration energy. Intensive research is ongoing in the area of energy harvesting for WSNs [104].

The size of a sensor network differs a lot depending on a given application and the

coverage area. Typical transceivers have the range of approximately $100m$ in open area, but in real-life applications motes are deployed much closer to each other. The number of nodes varies from a few to several hundreds, so the scalability issue is important in case of every network mechanism. The largest WSN deployment so far had 800 motes that formed a single large self organized network [121].

The cost of a single sensor node depends on many factors and ranges from hundreds of euro to a few cents. This variety depends mainly on the complexity of the particular devices used to build the network. Size and cost are two major constraints for sensor devices and they limit significantly the node's resources such as bandwidth, memory, energy and computing power.

1.1.1 WSN hardware platforms

From a security point of view proper understanding of hardware limitations and capabilities is really crucial. It is clear that the level of security achieved is directly proportional to the processing power required. This resource, together with memory is usually very limited on sensor devices. The processing time needed for cryptographic calculations should be short due to the short battery lifetime. There would be no use for a system that offers a high level of security but at the same time depletes all of mote's energy in the matter of few days. That is why WSN nodes require a lightweight cryptosystem.

WSN devices have to meet requirements that are usually specific to a given application, but we can still distinguish several common features. Nodes should be small, cheap, energy-efficient, and have to be equipped with the right set of sensors and have enough memory and computational power to perform given tasks. Wireless communication within the network should be highly reliable even in the face of interference and link failures. The WSN design space is very large and hardware platforms are suppose to support more than one type of application. Due to the large variety in mote design and parameters (memory size, microcontroller type and speed), it is hard to develop a universal security solution that would fit all hardware architectures. For cryptography implementations available memory and specific CPU features are the most important parameters. The security solutions that target WSNs should be optimized to work with the most constrained devices (typically low-end 8-bit platforms).

From many available architectures we can choose several devices that are the most

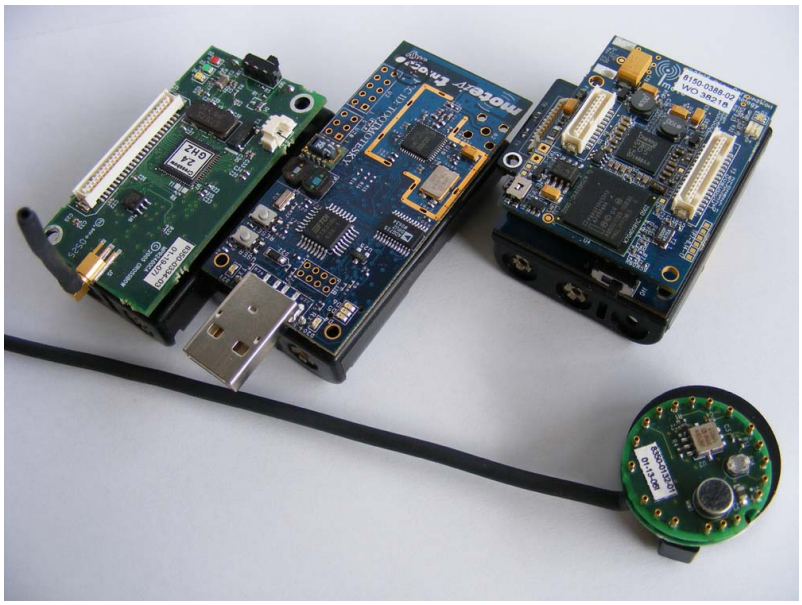

Figure 1.3: *Wireless sensor network hardware platforms, from the left: MICAz, Tmote Sky, Imote2, bottom: MICA2DOT*

popular platforms at the moment (Fig. 1.3). Each of these motes might be regarded as a representative of a certain class of sensor devices. The MICA2DOT represents nodes with the most limited resources available. It is one of the smallest commercially available motes and is powered by a 3V coin cell. MICAz is a widely recognized 8-bit sensor platform that was used in many different WSN research projects. Tmote Sky is a slightly more powerful device with 16-bit processing and a broad range of sensors embedded on the main Printed Circuit Board (PCB). The last device - Imote2 represents the most powerful group of 32-bit sensor platforms that have a lot more capabilities than previously mentioned nodes.

Cryptographic primitives are based on arithmetic operations and their efficiency depends mainly on the hardware support of a given microprocessor. In cryptography we have to deal with large numbers that can have more than one thousand bits. Arithmetic on such large integers is a time consuming task, especially on 8-bit CPUs. In order to optimize those operations we need to take advantage of all the specific features of a given device.

Table 1.1 brings together many important parameters of popular WSN platforms. Typical devices have 8 or 16-bit architectures and clock speeds around 4-8MHz. The data memory available can be as low as 4kB and program storage might be limited to only 48kB. These constraints make it really difficult to implement some of the more complex security primitives (as required in Public Key Cryptography - PKC). Another limitation is

Table 1.1: *Comparison of typical wireless sensor platforms*

	MICA2DOT	**MICA2**	**Tmote Sky**	**Imote2**
Processing	8-bit	8-bit	16-bit	32-bit
CPU	Atmega128L	Atmega128L	MSP430	PXA271
Clock speed	4MHZ	7.38MHz	8.19MHz	13MHz
Program memory	128kB	128kB	48kB	32MB
RAM	4kB	4kB	10kB	256kB
FLASH	512kB	512kB	1MB	32MB
Arithmetic support	Limited	Limited	Limited	High
Active power	8mW	24mW	10mW	26mW
Sleep power	75μW	75μW	6μW	330μW
Wake up time	180μs	180μs	6μs	-
Radio	CC1000	CC1000	CC2420	CC2420
Frequency	433MHz	433MHz	2.4GHz	2.4GHz
Data rate	38.4kb/s	38.4kb/s	250kb/s	250kb/s
Receive power	29mW	29mW	38mW	38mW
Transmit power	42mW	42mW	35mW	35mW
Battery	3V coin cell	2x AA	2x AA	3x AAA

the fact that many CPUs do not have good arithmetic support. Only high-end platforms like the Imote2 have hardware capable of fast arithmetic operations with clock speeds adjustable up to 416MHz.

Looking at the power consumption figures in Table 1.1 it is evident that radio communication is very expensive and usually takes more energy than CPU operation. This observation is extremely important and makes a big impact on every network protocol design. Since wireless transmission is a major cause of power drainage, the number of messages in the network should be reduced to a minimum. Low duty cycling should be incorporated as well, so all the nodes in the network can stay asleep most of the time and wake up only when it is really necessary. An ideal security solution for WSNs would have a small bandwidth overhead and low computational complexity to allow long network operation.

1.2 Security issues in WSNs

Security is one of the most important and challenging problems in wireless sensor networks [117]. Sensor networks like any other communication systems need fundamental security services. A secure WSN should posses the following security features [97]:

- key distribution and management;
- information confidentiality and privacy;
- secure routing;
- intrusion detection;
- data integrity;
- entity authentication;
- secure data aggregation.

All of the above features need to be provided if we want to have a fully secure WSN system. Information confidentiality and data integrity can be fulfilled by incorporating simple link-layer security mechanisms that encrypt packets and use message authentication codes. Intrusion detection algorithms should control the access to the network and allow only legitimate nodes to join the WSN. Authentication is an important security property in sensor networks as it ensures the receiver, that the message did in fact originate from the claimed sender. In many cases the confidentiality of simple sensor readings is not as important as the origin of the data.

The establishment and distribution of shared keys between the sensor nodes is one of the most important security services needed to ensure reliable networking. The above security requirements can be fully addressed only by building upon a solid key distribution framework. Key management is an essential cryptographic mechanism upon which other security primitives are built. Secure routing, messages confidentiality and entity authentication are not possible without a proper key distribution mechanism that bootstraps the security in the network.

The task of applying an appropriate level of security within a sensor network is difficult. In order to develop a security solution for WSNs we need to overcome several challenges [112]:

Limited hardware. As mentioned earlier, sensor nodes are very constrained in terms of resources. Only a small percentage of the total memory can be used for cryptography and the overall security mechanism should be fast and energy efficient.

Wireless communication. This type of communication is prone to jamming, eavesdropping and injection of malicious messages into the network. An additional problem is the limited bandwidth of the system. Extra overhead required in providing the security should not substantially degrade the overall efficiency of the system.

Scalability. The security protocol must scale well with the size of the network. Traditional security solutions are designed for two-party settings and add a significant overhead in large scale sensor networks.

Versatility. Sensor networks have many different hardware platforms and configuration modes that make them suitable for a wide range of applications. A security solution designed for a flat network topology will be unlikely to be the right choice in a clustered network scenario. Optimal solutions should be tailored to a specific network type.

Physical attacks. WSN nodes do not have secure storage for cryptographic keys, which means that an active attacker can capture one or more nodes, intercept messages and decode them using the derived secret key. Low cost sensor devices exclude the possibility of using tamper-proof hardware.

All the severe constraints mentioned earlier, and the demanding deployment environments, make WSN security more challenging than for traditional fixed networks. However, several properties of sensor networks may actually help to design security services in such environment. The fact that the networks will usually comprise of large numbers of nodes may help to increase the security as redundant motes may be used to detect and defy possible threats. Another advantage from a security point of view is that in the context of a WSN, there is a clearly identified single "trusted authority", and that is the original deployer of the network. The clearly identified existence of such an entity, which is not always the case in cryptography, makes it easier to develop a solution. The unique features of sensor networks allow novel solutions and security protocols that are not possible in conventional networks.

WSNs are still at the design and research stage and security solutions may ultimately be incorporated into the very foundations of system's architecture. It is important to avoid the same mistakes as in case of Wireless Local Area Networks (WLANs) where security was not obligatory from the beginning. To achieve a secure and reliable system

it is necessary to integrate security into every single component, since components without security can become a potential point of attack.

1.3 Thesis contribution

The area of security for wireless sensor networks is very broad with many topics and open problems. With the widespread use of WSN deployments the need for reliable security mechanisms will grow significantly. Existing solutions are not sufficient for many applications and new cryptographic schemes need to be developed. Despite the intense research efforts there are still many security issues that have not been fully addressed yet. Cryptographic key distribution and node authentication inside the sensor network are the main problems that lack practical solutions. The solution to these problems is essential for proper security bootstrapping in WSNs and is the main topic of this thesis.

1.3.1 Motivation

One approach to the problem of key distribution on low-powered sensor devices is to restrict the set of cryptographic building blocks to the simplest cryptographic primitives (block ciphers and hash functions), and then try to build a solution from these. This approach guarantees the lowest computational complexity at the cost of the level of security provided. A more ambitious and challenging strategy is to imagine a close-to-ideal public key cryptography solution, and then somehow try to make it practical. Following this strategy is a lot more difficult, but offers security levels and services comparable with the ones used in traditional computer networks.

Most of the WSN security solutions that have been proposed so far are very restricted and rely only on simple symmetric primitives (e.g. [69], [40], [43]). Many of them reduce the computational cost, but are not scalable and tend to dramatically increase the communication overhead. They also do not provide a good trade-off between resilience and storage of cryptographic keys. Symmetric key solutions are prone to physical attacks and they do not work when a small number of nodes are captured by attackers. The security of such systems is very limited when compared to Public Key Cryptography solutions.

Symmetric key techniques proposed in the literature usually defend against a particular security threat making the sensor network vulnerable to a range of dangerous attacks.

It is difficult to combine many such mechanisms into a unique security solution due to different or even conflicting underlying assumptions. In contrast, Public Key Cryptography provides a unified framework and a solid base for many security services that can be build on top of it. Public Key Cryptography offers a more flexible and simple interface, without the need for key pre-distribution, pair-wise key sharing or complicated one-way key chain schemes. The functionality of Public Key Cryptography schemes is highly desirable in WSNs to enhance the security level of demanding applications.

Despite many advantages, Public Key Cryptography has its own drawbacks. It is much more computationally expensive and uses larger key sizes than most symmetric key algorithms. Traditional Public Key Cryptography systems also need authentication of public keys which is usually performed by certificates and digital signatures. The management of certificates is usually very expensive and requires trusted infrastructure which can be unavailable in many sensor network scenarios.

Typical security systems used in fixed networks rely on a hybrid approach. They use more expensive public key techniques only for session key establishment, and use fast symmetric key algorithms for message encryption and decryption. Similar solutions should be applied in the case of sensor networks. It is necessary that the nodes run cryptographic operations based on primitives such as symmetric key encryption, hash functions and Public Key Cryptography algorithms. Hash functions and block ciphers are the standard building blocks that offer basic protection of the information that is exchanged between nodes. They provide important security services such as confidentiality of the communication channel and integrity of the messages. Additionally, Public Key Cryptography allows secure key exchange, node authentication, and protects against malicious insiders that try to participate in information exchange. Therefore, it is essential to use all three types of primitives to secure wireless sensor networks. However, many researchers have ruled out public key cryptography claiming that it is too heavyweight for constrained sensor devices [69], [97], [112], [98], [127].

Wireless sensor networks can have different network organization and communication patterns. A division into clusters is the most typical network topology in sensor networks. In such scenario each cluster has a designated node, which aggregates the data in the cluster (cluster head). The communication pattern in this type of a network is many to one, where all cluster members send data directly to the cluster head. How-

ever, some wireless sensor network applications require a different network organization. Sensor networks used for industrial automation use a mesh topology, where each node maintains communication with every other device in the network. This type of topology requires different communication protocols and a security solution designed for a clustered network scenario would not work here in an optimal way. Hence, the key distribution mechanism should be tailored to a specific network organization for optimal performance.

1.3.2 Research objectives

As WSN applications mature, the need for reliable security systems grows significantly and justifies the use of public key cryptography techniques to secure communication in sensor networks. However, published results have shown that traditional public key solutions (e.g. RSA [120]) are not suitable for constrained devices. Therefore Public Key Cryptography was for a long time considered as an option that is beyond the capabilities of wireless sensor nodes. The latest research results revisited this common opinion and showed that Public Key Cryptography primitives based on Elliptic Curve Cryptography (ECC) are feasible on sensor devices [57]. This fact was a turning point in the area of WSN security and has motivated work on efficient Elliptic Curve Cryptography implementations on sensor nodes.

There are many Elliptic Curve Cryptography implementations on WSN nodes but not much attention is focused on complete Public Key Cryptography solutions for sensor networks. So far only a few public key schemes have been proposed and none of them seems to be a practical solution for the key distribution problem. Despite the fact that Elliptic Curve Cryptography primitives are computationally feasible on sensor nodes, protocols based on them are not. They require exchange and storage of large keys, which is expensive, especially for the most constrained sensor devices. Practical Elliptic Curve Cryptography schemes also need authentication of public keys, which is usually performed by certificates and digital signatures. Sensor networks cannot afford a complicated Public Key Infrastructure (PKI) and need new security schemes that can provide authentication without using expensive certificates.

Elliptic Curve Cryptography brings us a step closer to applying Public Key Cryptography in WSNs, but we need simple public key infrastructure solutions that would

enable fully functional security protocols in sensor networks. Bilinear pairings and Identity Based Cryptography (IBC) are new, promising security techniques that can fulfil this goal. We need to show that not only Public Key Cryptography primitives are feasible but also that complete Public Key Cryptography protocols are practical in WSNs. The use of the latest achievements in cryptography can bring this vision to reality. The main objectives of this research are fourfold:

- to develop new PKC-based mechanisms that will solve many of the shortcomings of existing key distribution protocols in sensor networks;
- to provide efficient software implementations of different Public Key Cryptography primitives for sensor nodes;
- to investigate the application of Pairing Based Cryptography (PBC) in wireless sensor networks;
- to design efficient key distribution mechanisms for large scale sensor networks with various topologies.

The process of applying security in communication systems is a complex task that requires important design decisions. We need to decide if the cryptographic primitives will be implemented in hardware or in software. The first approach is the best solution performance-wise, but requires a dedicated hardware accelerator that would lift the total cost of a node and complicate its design. This of course goes against the low-cost/small size philosophy which is a priority for WSN devices. Software implementations are a few orders of magnitude slower and not as efficient in terms of resources utilization, but they are more flexible and cost-effective solutions with rapid development time. Software solutions also allow quick implementations of new cryptographic schemes on off-the-shelf devices and allow tests on already deployed networks. That is why this thesis investigates mainly software implementations of cryptographic schemes. This approach leads also to conclusions as to whether or not reasonable security and performance levels can be achieved with software-only cryptographic implementations, or is hardware support really needed?

1.3.3 Summary of contributions

This thesis presents a detailed description on how to provide practical Public Key Cryptography solutions for wireless sensor networks. The contributions to the state-of-the-art are added on all levels of development, beginning with the basic arithmetic operations and finishing with complete security protocols. The reader is presented with the mathematical foundations of all the cryptographic constructs used in this work. Based on this, several new key distribution protocols are proposed and are evaluated on various WSN platforms and network topologies. The thesis consists of 5 main contributions:

- Improved methods for finite field arithmetic on low-end CPUs are described - in particular, new optimised algorithms for large integer multiplication and binary polynomial multiplication.

- A software package called NanoECC is presented. This cryptographic library provides ECC-based public key cryptography operations that were specifically optimised for different sensor network platforms. NanoECC can be flexibly configured and integrated into sensor network applications to provide protocols such as Elliptic Curve Digital Signature Algorithm (ECDSA) and Elliptic Curve Diffie-Hellman key exchange protocol (ECDH). NanoECC shows that Public Key Cryptography based on Elliptic Curve Cryptography is not only viable, but in fact efficient for resource-constrained sensor nodes.

- The first in-depth study on the application of pairing-based cryptography to wireless sensor networks is presented. Micro-pairings present the state-of-the-art on the implementation of bilinear pairings on a range of sensor network devices. Efficient implementations of pairings enables such cryptographic schemes like Identity Based Encryption (IBE), and thus opens new ways for achieving security in sensor networks.

- On a system level, the application of identity-based cryptography is investigated. The thesis proposes two new identity-based key-exchange protocols. The first scheme was designed for flat network topologies to solve the key distribution problem among homogeneous sensor nodes. The second one targets hierarchical sensor networks where groups of nodes form distinct clusters.

- An efficient security bootstrapping mechanism called TinyIBE is presented. This scheme uses identity based encryption to distribute session keys in a heterogeneous sensor network environment. TinyIBE exploits the enhanced capabilities of high-end cluster heads to secure communication between different classes of sensor devices.

1.4 Thesis outline

The reminder of this thesis is organized as follows:

Chapter 2 presents an overview on cryptographic key distribution techniques. It describes the classical approach to key distribution together with recent advances in public key cryptography in the form of Elliptic Curve Cryptography, bilinear pairings and identity based systems. Chapter 2 presents also a survey of key-distribution methods in wireless sensor networks. It introduces an identity-based approach to key distribution and shows its advantages over traditional systems. Finally it defines all the building blocks that are necessary to implement Identity-Based Cryptography protocols in sensor networks.

Chapter 3 describes in detail finite field arithmetic on low-end processors. It gives a brief introduction to finite fields and presents optimized arithmetic operations over the binary and the prime field. In particular it presents new algorithms for multiprecision multiplication and binary polynomial multiplication. It describes also extension fields and shows how to accelerate arithmetic operations through hardware improvements on the CPU side.

Chapter 4 presents the details of elliptic curve cryptography implementations on WSN motes. It gives the mathematical background on elliptic curves, curve arithmetic and point representation. Chapter 4 presents also a brief survey on existing Elliptic Curve Cryptography implementations on sensor nodes. The last part of the chapter contains a description of the NanoECC package. It gives all the implementation details and compares the performance results with the state-of-the-art.

Chapter 5 reports on cryptographic pairing implementations on sensor nodes. The first part of the chapter presents a mathematical introduction to bilinear pairings and defines

different pairing types. The second part investigates the application of pairing based cryptography in wireless sensor networks. It discusses the security parameters and different pairing algorithms. The last part presents Micro-pairings - an efficient Pairing-Based Cryptography implementation for sensor nodes. The description includes all the pairings parameters together with performance evaluation on a broad range of sensor platforms.

Chapter 6 describes identity-based security protocols for sensor networks. The first part of the chapter defines various WSN models that have many distinctions and different requirements. Based on this, three new key-exchange protocols are proposed. The first two investigate the application of non-interactive key-exchange schemes in flat and hierarchical network scenarios. The third key distribution protocol uses identity-based encryption to secure communication within a heterogeneous sensor network. Chapter 6 includes also a comprehensive security analysis of all the proposed solutions together with possible security threats and defensive measures.

Chapter 7 concludes the thesis with a summary of contributions, suggestions for future research, and some final remarks.

CHAPTER 2

Key Distribution Techniques in Wireless Sensor Networks

After analyzing the constraints and limitations of sensor networks, it is clear that these kinds of environment need lightweight cryptography to achieving high level of security. On small sensor devices, a practical security solution must be a trade-off between cost, performance and security level. It is often very difficult to optimize all three design goals at the same time. In most situations developers sacrifice the security level by using cost-effective symmetric key solutions without proper mechanisms for key distribution.

For symmetric encryption to work, two nodes must obtain the same secret key which has to be protected from access by others. How the secret keys are deployed in the network, however, is a non-trivial problem that is difficult to solve using only symmetric key techniques. The problem is especially challenging in case of wide area distributed systems such as sensor networks. An unprotected environment requires frequent key changes which limit the amount of data compromised if an attacker learns the key. Therefore, the strength of the cryptographic system rests with the key distribution technique, a term that refers to the means of delivering a key to two parties who wish to exchange data in a secure manner.

One can imagine a small sensor network that has pair-wise secret keys pre-loaded into

the nodes memory before the deployment phase. Each sensor node has a list of $n-1$ keys, one for each of the other $n-1$ nodes in an n-node network. After the deployment phase the nodes can use the secret keys to encrypt the data that is exchanged in the network. The network deployer can also manually update the keys whenever it is required. However, the addition of new nodes is problematic and requires new keys for every single node in the network. Such a system might work well in case of a really small network which is deployed in a controlled environment, but is unacceptable in the case of large distributed networks.

Another solution to the problem might be to incorporate a Key Distribution Center (KDC). The KDC would be responsible for distributing keys to pairs of nodes whenever they would need to communicate with each other. Every node would have to share a unique symmetric key with the key distribution center for the purpose of key distribution and authentication. This approach significantly increases the number of packets related to security and may cause communication bottlenecks around the nodes directly connected to the KDC. The security of such a scheme relies on the security of the key distribution center which needs to be well protected. This method also requires the existence of a fixed infrastructure with trusted servers, which is usually not available in many sensor network deployments.

The above two examples show how limited symmetric key techniques are when it comes to key distribution in wireless sensor networks. Key distribution under symmetric encryption requires that two nodes already share a key (through pre-distribution) or receive keys from a trusted key distribution center. Sensor networks need more flexible methods of key distribution in the network, similar to the techniques used in traditional fixed networks.

2.1 Classical approach to key distribution

Symmetric encryption, also referred to as conventional cryptography, was the only type of encryption in use prior to the development of public-key methods in the 1970s. The main drawback of symmetric key techniques is the lack of a proper key exchange mechanism. Public key systems are an elegant solution to this problem. They simplify the key management and offer additional functionality that was not available in symmetric systems. In traditional fixed networks, Public Key Cryptography is a fundamental and

widely used technology that secures the communication both in private networks and across the Internet.

The distinguishing technique used in Public Key Cryptography is the use of pairs of keys, where the key used to encrypt a message is not the same as the key used to decrypt it. Each user holds a mathematically related key pair: a secret private key and a published public key. This type of security technique is called asymmetric cryptography.

2.1.1 Asymmetric cryptography

Public Key Cryptography was introduced first by W. Diffie and M. Hellman. Fig. 2.1 presents the main idea of a public key cryptosystem.

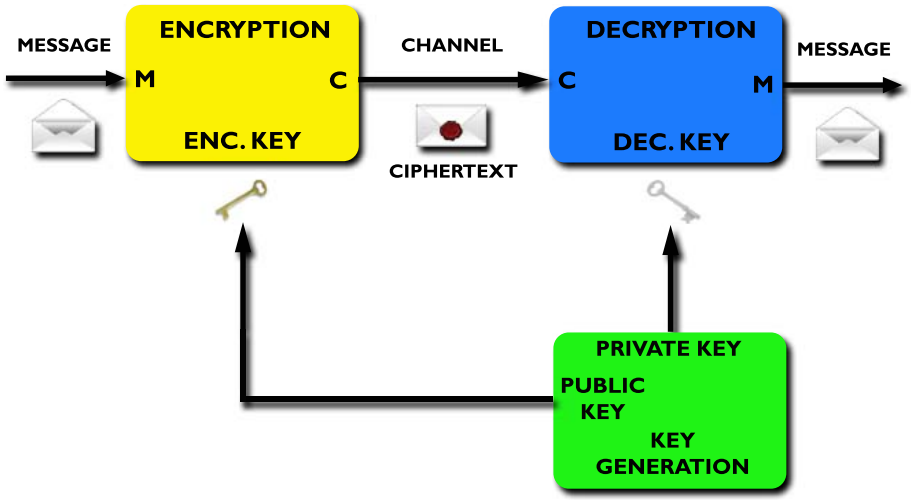

Figure 2.1: *Public-key cryptosystem*

Each user has a pair of keys: a public and a private key. The private key is kept secret, while the public key may be widely distributed, so other users can send encrypted messages to the designated recipient. The keys are related mathematically, but the private key cannot be practically derived from the public key. A message M encrypted with the freely available key can only be decrypted from the ciphertext C with the corresponding private key. Sometimes, messages are encrypted with the private key and decrypted with the public key. This operation is used in creating digital signatures.

In the wireless sensor network scenario, the whole scheme might by simplified. The original network deployer is a trusted entity that can deploy nodes with an embedded private/public key pair. In this way the trusted key generation center is no longer necessary. A single node can broadcast its public key to all its neighbours, who can then use it

to encrypt messages. However, in order to authenticate the public key there is a need for a trusted Certificate Authority which issues appropriate certificates.

Among all public key algorithms, there are three established families of practical relevance. The security of these systems is based on hard mathematic problems:

RSA Is named after its inventors Rivest, Shamir and Adleman [101]. The security of this scheme is based on the difficulty of factoring large numbers. It is the most famous asymmetric algorithm that is widely used until today for many important applications such as electronic commerce. Nonetheless, its operational requirements are very expensive with key sizes of a minimum 1024 bits.

Discrete logarithm based systems are another group of asymmetric algorithms that are commonly used in cryptography (i.e. ElGamal encryption [50]). The security of these schemes is based on the difficulty of calculating discrete logarithms in a finite field. The key sizes in these systems are comparable to those in RSA schemes.

Elliptic Curve Cryptography is based on the algebraic structure of elliptic curves and its strength relies on the difficulty of the discrete logarithm problem in this setting (ECDLP). Elliptic Curve Cryptography has lower requirements than the other two systems. Therefore it is considered the most attractive family for embedded devices.

As mentioned earlier, public key schemes are a few magnitudes more demanding in terms of resource utilization than symmetric key systems. Many researchers claim that Public Key Cryptography software implementations for sensor nodes are not possible due to the very limited amounts of memory and computing power [43], [69]. Hence hardware support may be needed for public key operations. Unfortunately most hardware extensions for sensor nodes that aim to improve the execution of asymmetric cryptography exist mainly as a proof-of-concept. None of the sensor node platforms currently available on the market has hardware support for Public Key Cryptography. An additional hardware accelerator would complicate the node design and rise the overall price of the device. Nonetheless, future WSN platforms would benefit from hardware that supports Public Key Cryptography operations. This support could be in a form of a specialized microcontroller or as an external chip that accelerates cryptographic operations.

There is an evident demand for cost effective Public Key Cryptography hardware solutions for sensor nodes. At the same time, software support for public key opera-

tions needs to be improved. In many situations Public Key Cryptography must be implemented purely in software because changes to the hardware are not possible. Software solutions may be also used for quick development of higher level protocols and to validate new security concepts. For many WSN applications, asymmetric cryptosystem would seem to be the best approach for distributing symmetric keys. It provides a high level of security in the network and offers better scalability and resistance against node capture than any symmetric key solution. Despite all these advantages, the application of public key ctyptography protocols in WSNs remains challenging. All three families of public key cryptography algorithms need authentication of public keys for proper usage. Practical deployments of public key solutions require the existence of a public key infrastructure in which certificate authorities authenticate public keys. The application of a traditional public key infrastructure in WSNs is very difficult due to the ad hoc nature of the network and the limited resources available.

2.1.2 Public Key Infrastructure

Public-key encryption schemes are secure only if the authenticity of the public key is assured. This service is provided with the use of certificate schemes. In cryptography, a public key infrastructure can be defined as an arrangement that binds public keys with respective user identities by means of a Certificate Authority. The main task of a PKI is to create, manage, store, distribute and revoke digital certificates. A typical public key infrastructure consists of:

- Certificate Authority (CA) - issues and verifies digital certificate;
- Registration Authority (RA) - performs initial authentication and acts as the verifier for the Certificate Authority before a digital certificate is issued to the user;
- Certificate repository - there can be one or more directories where certificates (with their public keys) are stored together with Certificate Revocation Lists (CLRs);
- Certificate management system.

Figure 2.2 presents a basic public key infrastructure and shows the information flow in the system. In order to participate in a public key infrastructure, user A must first enroll or register with the Registration Authority. The registration authority validates the user's

identity and forwards his public key to the Certificate Authority. The primary objective of the Certificate Authority is to bind the identifying information and credentials supplied by the Registration Authority with the public key of the user. The result of this process is the generation of a public key certificate. The binding is declared when a trusted Certificate Authority digitally signs the public key certificate with its own private key. The certificate authority issues each user with a digital certificate. When user B wants to communicate with user A he asks for his digital certificate. User B contacts the Certificate Authority in order to validate the received certificate. The certificate authority checks in the repository to see if the certificate is still valid (has not expired or been revoked). User B verifies the digital certificate by using the Certificate Authority's public key which is securely distributed among the users. After a positive certificate verification user B trusts user A and can send him encrypted data using A's public key. This relation is established thanks to the trusted third party that performs entity authentication on the users behalf.

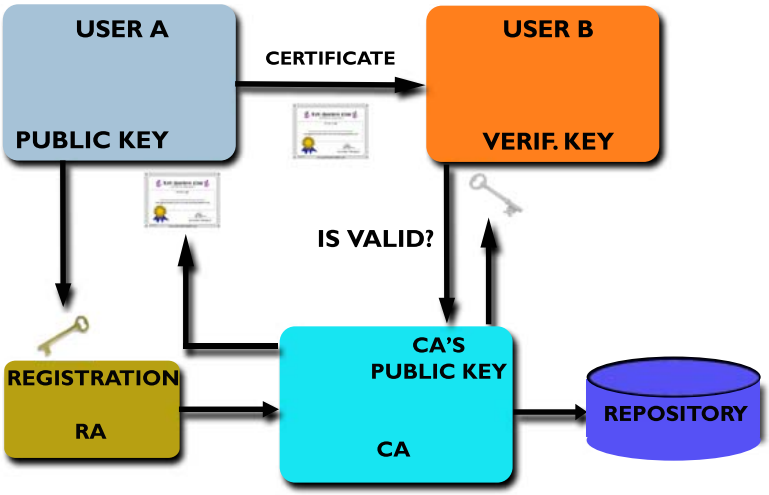

Figure 2.2: *Basic Public Key Infrastructure (PKI).*

In the context of wireless sensor networks, it is often difficult to provide a practical PKI. The system presented in Fig. 2.2 is quite complicated and has a significant communication overhead. Sensor nodes have strict energy limitations and cannot spend too much time on radio transmissions. Lack of a trusted infrastructure and sensor deployments in remote areas make it difficult to apply the concept of a Certificate Authority. It is also not possible for one of the nodes to play the role of a Certificate Authority as this entity must be unconditionally trusted and well-protected from attacks. However, the concept of a public key infrastructure in a sensor network scenario can be simplified thanks to several

design features of WSNs. Sensor nodes can be carefully configured in a secure environment which exists before the network deployment phase. The original network deployer is a trusted entity who can use a computer (base station) to upload all the global parameters and secret keys into nodes memory. It is clear that the base station can be considered as the Certificate Authority in the system. It can be responsible for creating digital certificates that associate the identity of a node with its public/private key pair. Moreover, the base station can take the role of a Registration Authority, since it is in charge of assigning identities to all the nodes in the network. The base station can also create the public/private key pair of a node, as it is not efficient for a node to create its own keys. In theory the base station may also act as a certificate repository, but this solution is not practical in sensor networks. Most sensor nodes would have to use multi-hop communication in order to reach the base station and retrieve the certificates. The energy cost of this communication would have a significant influence on the overall network lifetime. A better solution would require that every node is pre-loaded with its own certificate and the Certificate Authority's public key. After the network deployment, nodes provide the certificate to any neighbour that requests it and use the Certificate Authority's public key to verify the authenticity of other certificates.

Although the basic functionality of a public key infrastructure can be applied in sensor networks, the whole scheme is still not practical. The system is too complicated and not efficient in terms of resources utilization. In sensor networks of more than 100 devices, the nodes would have to store many certificates and public keys that would fill the limited storage space. Certificate management requires also the use of digital signatures and introduces large communication overhead in the network. Nodes are also required to contact the base station in order to validate certificates and check whether they have been revoked. Thereby the base station has to be always connected to the sensor network, which makes it vulnerable to attacks. This requirement also limits the autonomous operation of the network and the number of possible applications.

All the above features make traditional public key infrastructure schemes unpractical in WSNs. Sensor networks cannot afford a complicated public key infrastructure and need new security schemes that can provide authentication without using expensive certificates. These new public key systems should be simple, flexible and more efficient than methods currently used in traditional fixed networks.

2.2 Recent advances in Public Key Cryptography

Cryptography like many other areas of computer science is constantly evolving and new advances are made in the state-of-the-art. However, the adoption rate of new techniques in cryptography is slow due to many technological, economical and even political reasons. Emerging security techniques are often treated with great uncertainty and it takes a long time before the technology matures and can be adopted by government bodies and commercial organizations. This fact holds true in the case of the whole family of public key cryptosystems that are based on the algebraic structure of elliptic curves. The concept of elliptic curve cryptography has been known for over twenty years, but commercial products have appeared only recently on the market.

Cryptosystems based on Elliptic Curve Cryptography are especially interesting for sensor networks since they are more efficient in resource utilization than any other public key techniques [58]. The technology had been proven secure and there is no reason why it should not be used in the context of WSNs. Additionally recent advances in the field of Elliptic Curve Cryptography have shown that bilinear pairings on elliptic curves can be used to create useful cryptosystems. Cryptography using pairings is an emerging field related to Elliptic Curve Cryptography which has been attracting the interest of the international cryptography community, since it enables the design of original cryptographic schemes and makes well-known cryptographic protocols more efficient. Pairing-based cryptography has also allowed many long-standing open problems to be solved in an elegant way. Perhaps the most impressive among those applications is Identity Based Encryption (IBE), which in turn has allowed complete Identity Based Cryptography (IBC) schemes. WSNs can make use of these new achievements in cryptography to provide advanced security services at low cost. The following sections present a general overview of these new Public Key Cryptography systems.

2.2.1 Elliptic Curve Cryptography (ECC)

The use of elliptic curves in cryptography was suggested for the first time in 1985 independently by Neal Koblitz and Victor S. Miller. Since then the field of Elliptic Curve Cryptography has grown significantly and has been studied extensively. Many research papers has been published on the security and efficient implementation of Elliptic Curve Cryptography. ECC has been accepted commercially and has also been adopted by stan-

dardizing bodies such as: ANSI (American National Standards Institute), ISO (International Organization for Standardization), SECG (Standards for Efficient Cryptography Group) and NIST (National Institute of Standards and Technology). Elliptic Curve Cryptography is a proven technology that is used in many different commercial products such as mobile phones, smart cards, email systems and many others.

In every cryptographic scheme the fundamental security lies in the hardness of the underlying mathematical problem. The harder the number-theoretic problem is, the more secure will be the system , that can be build on top of that. The difficulty of these problems directly impacts the performance, since it dictates the size of the domain and key parameters. These values are very important in a security system as the performance of arithmetic operations relays heavily on them.

As mentioned earlier, the hardness of Elliptic Curve Cryptography is based on the Elliptic Curve Discrete Logarithm Problem (ECDLP). One of the main operations in elliptic curve cryptography is the calculation of a scalar point multiplication $Q = sP$, where Q and P are two points on a certain elliptic curve. The Elliptic Curve Discrete Logarithm Problem occurs when the coordinates of P and Q are known and the scalar s needs to be calculated. This computational problem becomes harder with the size of domain parameters. When the size of the underlying finite field is larger than 160 bits the calculation of ECDLP is considered to be computationally infeasible (with the current state-of-the-art in cryptanalysis).

So far only algorithms with exponential running time have been proposed to solve the discrete logarithm problem in the elliptic curve setting. The best method so far is the Pollard's ρ algorithm [100] which has computational complexity of $O(\sqrt{n})$. For other public key systems, that are based on integer factorization and discrete logarithm problems, fast algorithms are known that have a subexponential expected running time (e.g. the number field sieve lenstra93). This feature allows Elliptic Curve Cryptography based systems to provide the same level of security as traditional schemes but with smaller parameters. Table 2.1 [58] compares the key sizes in symmetric, RSA and Elliptic Curve Cryptography systems for achieving different security levels.

The numbers in Table 2.1 shows that much smaller key sizes can be used in the elliptic curve system than with RSA at a given security level. The five specific security levels correspond to the five different block ciphers and the key sizes that they use. The dif-

Table 2.1: *Key size comparison between symmetric, RSA and elliptic curve systems for achieving an equivalent security level. The security level of* k *bits means that the best algorithm known for breaking the system takes approximately* 2^k *steps.*

	Security level (bits)				
	80	112	128	192	256
Block cipher	SKIPJACK	3-DES	AES-small	AES-medium	AES-large
EC parameter p	160	224	256	384	512
RSA modulus n	1024	2048	3072	8192	15360

ference in key sizes between Elliptic Curve Cryptography and RSA is especially visible for higher security levels. At 256 bits of security Elliptic Curve Cryptography uses only a 512-bit key which is 97% smaller than the corresponding RSA key. The advantages that can be gained from smaller keys include not only faster computations and smaller memory requirements, but also energy savings for sensor devices, as fewer bits are required to be transmitted by the radio. All this advantages makes Elliptic Curve Cryptography the most attractive family of public key algorithms for wireless sensor networks.

One of the problems with Elliptic Curve Cryptography is that some of the protocols and implementations are covered by patents. The availability of patent free elliptic curve schemes and other legal issues has limited the widespread of this cryptographic technique. Another drawback of elliptic curve cryptography is that the whole system is a lot more complicated than the traditional RSA scheme. The mathematics behind Elliptic Curve Cryptography are rather complex and a broad range of different parameters makes the implementation more difficult. Nevertheless, small key sizes, relatively low computational requirements and high flexibility justifies the choice of Elliptic Curve Cryptography as a Public Key Cryptography technique.

2.2.2 Pairing-Based Cryptography (PBC)

Pairing-Based Cryptography is a relatively young area of cryptography that revolves around a particular function with interesting properties. Pairings, such as the Weil pairing, were first used in the context of cryptanalysis [88] to reduce the ECDLP into a discrete logarithm problem in the finite field. The first use of pairings in cryptography is the work of Sakai *et al.*[106] and Joux [67]. Both papers proposed pairings as the base for building complete cryptosystems. Since then many protocols have been proposed that use pairings as the underlying crypto primitives.

A pairing function is a mapping between two groups of elliptic curve points. It is often denoted as $\hat{e}(P, Q)$ where P and Q are two points on a special elliptic curve. The whole concept of using pairings in cryptography is based on the property of bilinearity:

$$\hat{e}(aP, bQ) = \hat{e}(P, bQ)^a = \hat{e}(aP, Q)^b = \hat{e}(P, Q)^{ab} \tag{2.1}$$

In equation 2.1 aP and bQ are elliptic curve point multiplication operations where a and b are scalars. The above property of the pairing allows the creation of novel cryptographic schemes (e.g. identity based encryption, short signature schemes) that cannot be constructed using other techniques. Pairings can be also used to implement existing security protocols in a more efficient way and with additional functionality (key agreement schemes, threshold cryptography). In the context of wireless sensor networks, pairings enable different identity-based cryptography schemes that can simplify key distribution in the network.

2.2.3 Identity-Based Cryptography (IBC)

The term *identity-based cryptography* refers to cryptosystems that use user's identities as public keys. The first concept of Identity-Based Cryptography was formulated by Adi Shamir in 1985. He described in his paper [111] the theoretical basis of the system, but did not propose a practical design solution.

The first identity-based cryptosystem based on pairings was proposed in the year 2000 by Sakai, Ohgishi and Kasahara [106]. They proposed a key-sharing scheme that used identities as public keys. The idea of the key agreement scheme is quite simple and relies on the bilinearity property of the pairing. Let H_1 be a hashing[1] and mapping function that maps unique identity information into an elliptic curve point. Private Key Generator (PKG) has a master key s and issues private keys to users of the form sP_A, where $P_A = H_1(ID_A)$ and ID_A is the identity of user A. After this step users A and B can calculate a shared secret that only they (and the PKG) can compute, namely:

$$\hat{e}(sP_A, P_B) = \hat{e}(P_A, P_B)^s = \hat{e}(P_A, sP_B) \tag{2.2}$$

Thanks to bilinearity of the pairing, both users can calculate the same value inde-

[1] Hashing defines a conversion of a large amount of data into a small value, using a mathematical function.

pendently of each other and use this shared secret to encrypt their communication. The above key sharing scheme does not require any communication between the parties. User A needs only his private key and the public key of user B (which can be derived based on public identity ID_B) to calculate the pairing.

Soon after the discovery by Sakai, Ohgishi and Kasahara, another application of bilinear pairings was presented. In 2001 D. Boneh and M. Franklin described an identity-based encryption scheme which was based on the Weil pairing. This scheme fully realized Shamir's vision from 1985 to encrypt messages using only the identity of the recipient. Boneh and Franklin's Identity-Based Encryption scheme remains today as one of the most famous protocols in Identity-Based Cryptography.

2.2.3.1 Identity-Based Encryption (IBE)

Identity-Based Encryption is a public-key cryptosystem that was developed in order to solve some of the problems of traditional public key algorithms. The whole scheme works as follows. Instead of generating a random pair of keys and publishing the public key, the user chooses his identity *ID* as his public key. The sender uses this information to encrypt data and transmits the ciphertext *C* through the channel. The user authenticates himself to the key generation center and receives his private key. The recipient's private key is calculated from his *ID* and must be send to him in a secure way. This operation enables the receiver to decode the message *M*. The whole IBE scheme is illustrated in Fig. 2.3.

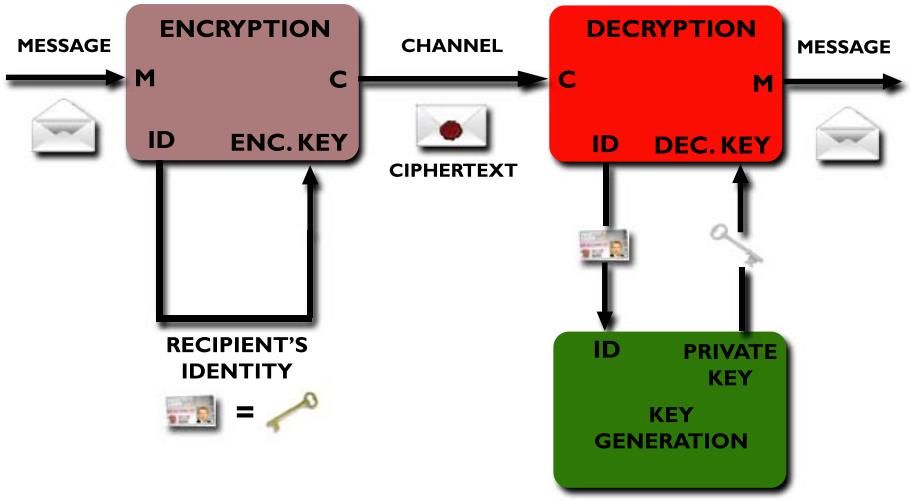

Figure 2.3: *Identity-based cryptosystem*

Identity-based encryption uses a unique identity as a public key rather than relaying

on the certificates and revocation lists that are used in traditional asymmetric cryptosystems. In identity-based systems, certificates are no longer needed to bind public keys to users identities. Thanks to this feature, the whole cryptographic system is simplified and easier to manage. The whole identity-based encryption scheme can be divided into four basic stages:

(1) Setup During this step all the system parameters are set and the master-key is generated. The parameters include the description of the message space *M* and the ciphertext space *C*. All the system parameters will be publicly known but the master-key has to be kept secret and is known only to the PKG.

(2) Extract This operation takes as input the system parameters, the master-key and the user identity *ID*. The master-key is used to generate the private key corresponding to an arbitrary string $ID \in \{0, 1\}$.

(3) Encrypt In this step plaintext $m \in M$ is encoded using a public key *ID*. The output is the corresponding ciphertext.

(4) Decrypt Takes as input a private key and ciphertext $c \in C$ and decodes the message giving the corresponding plaintext.

The Setup and Extract steps are performed by the private key generation center and they cannot be carried out by the users. Each request for private key has to be checked, so a proof of identity is needed before releasing the key. Both Encrypt and Decrypt algorithms are run by the users to encode and decode messages that are exchanged between them.

The security of the Identity-Based Encryption system depends mainly on the secrecy of the information stored at the key generation center, so a great effort has to be made to make it as secure as possible. Another important aspect of this scheme is the accuracy of identity checks that are performed before issuing private keys. Only valid users should receive secret keys that are necessary in the decryption step. The main advantage of identity-based encryption over traditional public key systems is that there is no need to store so many public keys. Public keys can be simply generated when necessary based on users identities. This process minimizes the communication overhead involved in public key authentication.

Identity-Based Encryption is a promising security technique that can be used in many different applications. So far it has been successfully adopted to secure files and secure e-mail systems. Recent developments have focused on implementing Identity-based Encryption on smart cards. Despite its many advantages Identity-Based Encryption have not been widely deployed in security systems. Besides the usual time it takes for new technologies to be adopted in security systems, there are certain drawbacks that hold back the widespread of this technology.

The main problem with Identity-Based Encryption and other identity-based systems is the need for a trusted key generation center. This entity is in charge of generating and escrowing user's private keys. It has the power to impersonate anybody else in the system. For that reason the private key generation center must be an entity that is unconditionally trusted by all network users. In most systems such an entity simply does not exist. However in the case of sensor networks the original network deployer is obviously a trusted entity that can play the role of a public key generator. Hence, security systems that are based on Identity-Based Cryptography present a promising solution to the key distribution problem in wireless sensor networks.

2.3 Key distribution in Wireless Sensor Networks

Many sensor network applications require protection against such problems as eavesdropping or injection and modification of packets. Cryptography is the standard defense against such attacks. The security requirements in WSNs depend mainly on the application type, but basic requirements that are common in all networks can be described as follows:

1) Data privacy, integrity and freshness;

2) Access control;

3) Non-repudiation and entity authentication;

4) Availability.

The first set of goals can be fulfilled by incorporating a link layer security mechanism that uses encryption and message authentication codes. Access control should allow only legitimate nodes to join the network. Guaranteeing availability involves min-

imizing the impact of Denial-of-Service (DoS) attacks. Non-repudiation is an important security property in sensor networks as it ensures the receiver that the message did in fact originate from the claimed sender. In many cases the confidentiality of simple sensor readings is not as important as the origin of the data. Symmetric key algorithms are used for message integrity checks, entity authentication, and encryption, while asymmetric cryptography additionally provide non-repudiation and access control. The above security requirements can be fully addressed only by building upon a solid key distribution framework. Key management is the essential cryptographic mechanism upon which other security primitives are built.

Despite much research effort, key distribution and management in wireless sensor networks still remains an open problem. After network deployment sensor nodes have to somehow agree upon a shared secret that will be used to encrypt the data. This issue can be addressed by using different security techniques. In general there are four basic classes of key distribution schemes that were considered for sensor networks:

- symmetric key solutions,
- random key pre-distribution,
- public-key algorithms,
- trusted server mechanisms.

Symmetric key solutions usually assume that a single key is used to encrypt and decrypt the communication in the network. This key is preloaded into the memory of every sensor node. The second type of key agreement protocols uses central servers to distribute keys within the network. This method is limited to only a few applications where a trusted infrastructure is available. Public key techniques use asymmetric cryptography which is more heavyweight than symmetric cryptography and requires authentication of public keys. The last approach to key establishment is via pre-distribution, where nodes are loaded with random keying material before their deployment.

2.3.1 Symmetric key solutions

Most of the security solutions for WSNs use basic symmetric key algorithms due to their simplicity and efficiency in resource utilisation. In typical real-life WSN deployments,

a network-wide key is used to encrypt all communication (e.g. in the residential mode in ZigBee WSNs [128]). Even though the system is simple and easy to implement its security is very limited and can be easily broken when the single key is revealed. The security can be enhanced by issuing one particular key for every pair of nodes, but this approach significantly extends the number of keys in the network to $\frac{1}{2}n(n-1)$, where n is the number of nodes. In large sensor network deployments each node would have to store many secret keys, which would quickly fill its limited memory space. Such a solution is not practical in WSNs and provides an insufficient level of security for many applications, especially those that deal with critical data. Symmetric key solution are very limited when it comes to key distribution, but they offer basic cryptographic primitives that can be used by higher level security protocols.

2.3.1.1 TinySec

TinySec [69] is a link layer security architecture for wireless sensor networks. It is a lightweight security package that developers can easily integrate into existing sensor network applications. The authors envisage that this architecture can cover the basic security needs for most applications. The two main goals of this mechanism are ease of use and minimal impact on network performance. TinySec provides access control, message integrity, data confidentiality and replay protection. The whole architecture does not address such problems like energy depletion attacks, physical tampering or node capture attacks. One of the important design goals of TinySec is that it is transparent to applications running on TinyOS[2]. Additionally it facilitates the customization of the provided security level.

TinySec supports two different security options. The first one, *authenticated encryption* (*TinySec-AE*) is a full security mode that offers encryption of the data payload and packet authentication with a Message Authentication Code (MAC). The second one, called *TinySec-Auth*, offers only authentication of the messages. The message authentication code is computed over the encrypted data and the packet header. Fig. 2.4 presents the packet format of *TinySec-AE*.

The packet format is based on the standard TinyOS packet [77]. It differs from the original structure by the addition of 6 bytes to the packet. The standard 2-byte Cyclic Re-

[2]TinyOS is a popular operating system used in wireless sensor networks.

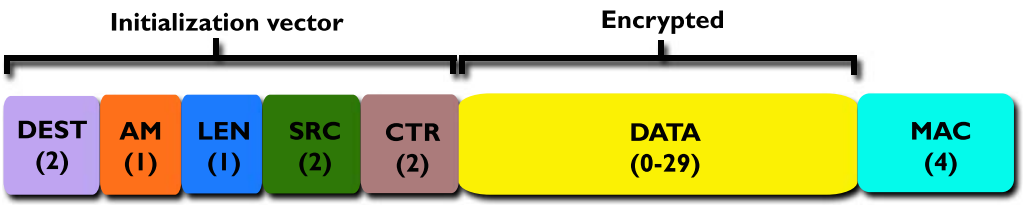

Figure 2.4: *TinySec-AE packet format (bytes) [69]*

dundancy Check (CRC) field is replaced by a 4-byte message authentication code. Cyclic redundancy check provides no security against malicious modification or forgery of the packet. Message authentication code also detects transmission errors, so the Bit Error Rate (BER) is not affected. TinySec uses a special 8-byte Initialization Vector (IV) with destination and source addresses, packet length, Active Message (AM) type and counter. Active message types in TinyOs are similar to port numbers in TCP/IP. Initialization vectors are used to achieve semantic security in a way that encrypting the same plaintext two times should give different ciphertexts.

TinySec is based only on symmetric key algorithms and uses the cipher block chaining mode for encryption and message authentication code generation. The default block cipher is Skipjack, but the whole architecture is cipher independent and other algorithms like RC5 or AES can be used instead. The timing for Skipjack reported in [69] on 8-bit MICA2 platform is $0.38ms$ (encryption of one 64-bit block). The whole TinySec implementation takes 3000 lines of nesC [51][3] code and requires 728 bytes of RAM and 7146 bytes of ROM on a MICA2 mote.

Experimental results on a 36 node network [69] showed that TinySec is an efficient link layer security protocol. Introduction of security services added less than 10% to the overhead in terms of packet latency and energy consumption. The network encountered only a 6% decrease in throughput. Much of the overhead can be fully explained by the increased packet length of TinySec. The results achieved in [69] demonstrate that a simple link layer cryptography mechanism can be efficiently implemented without any hardware support. Software implementation offers acceptable energy costs without major performance degradation.

The key distribution problem is not addressed in the TinySec architecture. Any particular key establishment protocol can be used in conjunction with the link layer mecha-

[3]nesC is a programming language similar to C that is used in TinyOs.

nism. It is expected that higher level security protocols will use TinySec as a basic security primitive. TinySec may reduce the effort of implementing such protocols by providing the right set of interfaces.

2.3.2 Random key pre-distribution

Key pre-distribution schemes are very popular in sensor networks due to their simplicity and low computational complexity. In a key pre-distribution scheme, a random set of keys from a certain pool is loaded into each sensor node before the deployment phase. During network operation, sensors perform a discovery process to identify shared keys between each other. Because of the random choice of keys, a shared key may not exist between some pairs of nodes. Those keys can be established with the use of neighbouring nodes that already share the keys on the path between the pair of nodes. In random key pre-distibution schemes any two nodes in the network share a common key with a certain probability.

The first probabilistic key sharing scheme for sensor networks was proposed by Eschenauer and Gligor in [43]. The Eschenauer-Gligor (E-G) scheme uses random graph theory to model connectivity in a sensor network. The whole protocol can be divided into three stages:

1) Key pre-distribution stage During this step, a large key pool of P keys and their identifiers is generated by a trusted key distribution server. From this key pool, m keys are randomly drawn and distributed offline into each node, together with key identifiers. The set of m keys is called the node's *key ring*.

2) Shared key discovery stage Once the nodes are initialized with keys, they are deployed in the field. After deployment, each node tries to discover the neighbours with which it shares common keys. There are many ways to determine whether two nodes share common keys or not. The simplest way is to make the nodes broadcast, in clear text, the list of identifiers of the keys on their key ring. If a node finds out that it shares a common key with a particular node, it can use it for secure communication. This approach does not reveal any important information for the adversary, but leaves room for a traffic analysis attack.

3) Path key establishment stage A link exists between two nodes only if they share a

key, but the path key establishment stage facilitates provision of the link between two nodes when they do not share a common key. If two nodes A and B do not share a key they can use their neighbours to establish secure communication. Node A sends an encrypted message to its neighbour C using a shared secret key. This message is a request for secure connection with node B. If node C has a common key with node B, it can generate a pairwise key for nodes A and B. After the shared key discovery stage is finished there will be a number of keys left in each sensor's key ring that are unused and that can be utilized by each sensor node for path key establishment.

The probability that two nodes share at least one key in their key rings of size m chosen from a given pool of P keys can be defined as:

$$p' = 1 - P_r[\text{two nodes do not share any key}] \tag{2.3}$$

The above equation takes into account that the size of the key pool (P) is not a sensor-design constraint. The key pool is generated offline and hence its size is usually much larger than m, which is limited by the node's memory size. In the case where each key of a key ring is drawn out of a pool of P keys without replacement, the number of possible key rings is equal to:

$$P_1 = \frac{P!}{m!(P-m)!} \tag{2.4}$$

The total number of possible key rings that do not share a key with a particular key ring can be calculated as the number of key rings drawn out of the remaining $P - m$ unused keys in the pool, namely:

$$P_2 = \frac{(P-m)!}{m!(P-2m)!} \tag{2.5}$$

Therefore, the probability that no key is shared between the two rings is the ratio of the number of rings without a match to the total number of rings:

$$\frac{P_2}{P_1} = \frac{m!(P-m)!(P-m)!}{P!m!(P-2m)!} \tag{2.6}$$

Hence the probability that two nodes share at least one key can be calculated as:

$$p' = 1 - \frac{P_2}{P_1} = 1 - \frac{((P-m)!)^2}{P!(P-2m)!} \tag{2.7}$$

The value of P is usually very large compared to m and hence the Sterling's approximation for $n!$ can be used to simplify the equation 2.7 into:

$$p' = 1 - \frac{(1-\frac{m}{P})^{2(P-m+\frac{1}{2})}}{(1-\frac{2m}{P})^{(P-2m+\frac{1}{2})}} \tag{2.8}$$

Figure 2.5 shows the probability of sharing at least one key between two nodes in the Eschenauer and Gligor scheme, based on formula 2.8. The plot is drawn for different values of the key pool size P. For example for a pool size of 1000 keys, only 26 keys need to be distributed to any two nodes to have the probability $p' = 0.5$ that they will share a key after deployment. If the pool is increased 5 times ($P = 5000$), the key ring has to be increased to 59 in order to achieve the same key sharing probability. For lower values of p' the Eschenauer-Gligor scheme scales well but higher values of p' require large key rings for each node. This is especially visible in the case of large key pool sizes of 10000 and more, which are necessary to provide higher levels of security in the system.

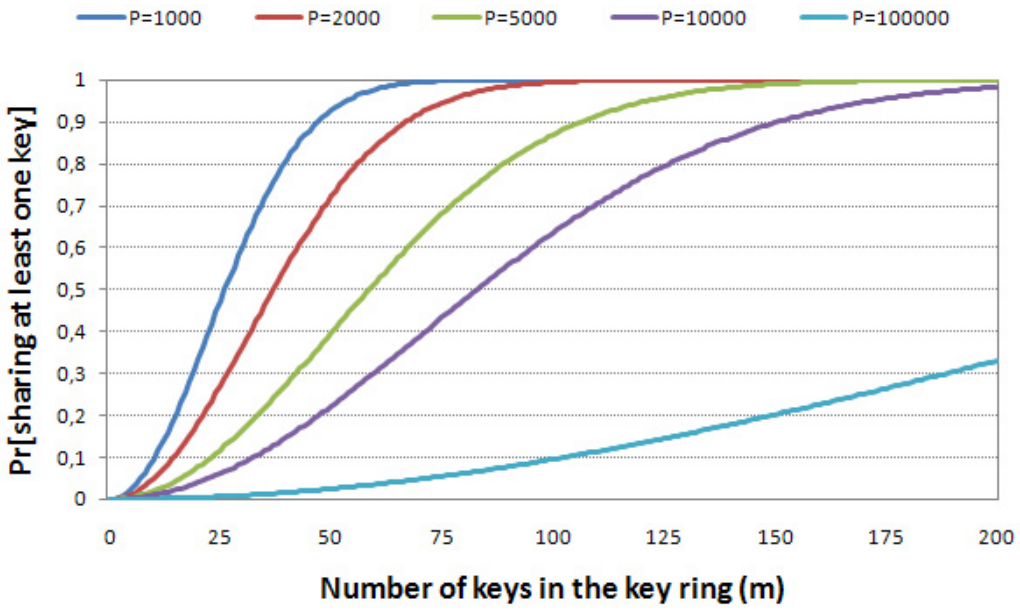

Figure 2.5: *Probability of sharing at least one key in the EG scheme.*

Since the development of the Eschenauer-Gligor scheme, random key pre-distribution for sensor networks has been extensively investigated in the literature. Pietro *et al* [31] questioned the validity of the random graph model in WSNs, and proposed another geo-

metric random model for sensor networks. Chan, Perrig, and Song in [25] introduced two variations of the Eschenauer-Gligor scheme: q-composite random key pre-distribution and multipath key reinforcement. The first scheme requires that two nodes have at least q common keys to set up a secure link. The nodes use all common keys instead of only one to establish the encryption key. The multipath reinforcement scheme is similar to the Eschenauer-Gligor protocol, but uses multiple indirect paths to establish a secure link. This approach increases the level of security in the system at the cost of increased communication overhead.

All of the above schemes are based on random key distribution where neighbouring nodes share a common key with certain probability. This feature, however, is also one of the main flaws of the system. Probabilistic key sharing does not give a guarantee that a perfect connection between communicating parties can be established. The application of such protocols in large scale sensor networks is problematic because of large key rings and complex re-keying procedures. The communication overhead of random key pre-distribution schemes has not yet been fully analyzed. Most papers describe how the key discovery and path key establishment stages work, but do not consider the cost of communication between the nodes. Especially, finding a secure path in a random graph is a NP-complete problem which is usually ignored by the authors. Despite its drawbacks, probabilistic key sharing is still considered as a plausible key distribution method in WSNs. However, security schemes that use random key pre-distribution offer lower security than systems based on public key cryptography. This fact makes it less likely that such methods will be widely used in practical applications.

2.3.3 Public Key Cryptography feasibility

For many WSN applications, asymmetric cryptosystem would seem to be the best approach for distributing symmetric keys in the network. However, Public Key Cryptography is computationally far more demanding in both hardware and software than symmetric key algorithms. The performance gap is especially large on small devices like 8-bit sensor nodes. For example, an optimized public key solution based on RSA performs 100 to 1000 times slower than a standard symmetric cipher such as the Advanced Encryption Standard (AES). This of course has a huge impact on energy consumption, which is two to three orders of magnitude higher for asymmetric techniques [42]. Many researchers

claim that due to these drawbacks Public Key Cryptography is beyond the capabilities of today's sensor nodes [69], [97], [112], [98], [127].

Despite the disadvantages, Public Key Cryptography schemes offer security services that are highly desirable in WSNs. This fact motivates further research in this direction. The main goal is to develop a secure and efficient key distribution mechanism that is practical in WSNs and allows simple key establishment in large scale sensor networks. The main efforts to apply public-key cryptography in WSNs is focused on implementing RSA and Elliptic Curve Cryptography. The following sections describe some of the attempts to implement these Public Key Cryptography primitives in sensor networks.

2.3.3.1 TinyPK

In [120] the authors describe the design of a public key protocol that provides authentication and key agreement between a sensor network and a third party. The same functionality is provided also for every pair of nodes within the sensor network. The proposed TinyPk scheme is based on the widely recognized RSA cryptosystem and the Diffie-Hellman key agreement technique. This protocol exploits the fact that RSA public operations (encryption and signature verification) are very fast compared to private key operations (decryption and signing).

To make the whole protocol practical for low end devices, the nodes are only required to perform the public key operations. RSA key size is set to 1024 bits as lower values for the keys are no longer considered secure with the current state-of-the-art in large number factorization. Each sensor node has to cube a 1024-bit integer and take its residue modulo a large prime which is a relatively fast operation. TinyPK uses $e = 3$ as the public exponent but this low exponent variant of RSA does not lower the security of the whole scheme. All the computationally expensive operations are performed by external servers. In this way the heavy burden of RSA private key operations is moved to devices that possess much higher computational capabilities than sensor nodes.

The main goal of the protocol is to allow secure communication between an external party and a sensor network. In many cases it is impractical and against the security policy to directly provide the session key to the external entity. Introduction of new session keys to all parties is a redundant and expensive operation. An authentication service is needed before a secure information exchange might be possible. TinyPK assumes the ex-

istence of a Certificate Authority in the system, which has a trusted pair of private/public keys. Any third party that wishes to interact with the motes also requires its own key pair and must have its public key signed with the Certificate Authority's private key. This establishes the external party's identity. Each mote in the network needs to have a copy of Certificate Authority's public key, before it is deployed in the field. The following describes successive steps of the TinyPK protocol:

1) The external party sends to the node the first part of the message containing its own public key signed by the CA's private key.

2) The external party sends the second part of the message that is signed with his private key. The message consists of a time stamp and a checksum. This information is not encrypted.

3) The sensor node receives the first part of the message and uses his preloaded CA's public key to verify the signature and extract the third party's public key.

4) The sensor node uses the external party's public key to verify the second part of the message. It validates the time stamp and the checksum. After passing the validation the third party has successfully authenticated to the sensor network.

5) The mote encrypts the session key together with the received time stamp using the third party's public key and sends back the message.

6) The external party decrypts the message with its private key, checks if the time stamp is the same as the one it sent. If validation is positive the external party stores the session key for future transmissions with the sensor network.

TinyPK also provides a secure key exchange service between two sensor nodes. This goal is achieved through the use of the Diffie-Hellman key exchange protocol [32]. This algorithm provides a shared secret-between two parties that can be then used to create a cryptographic key for symmetric encryption. TinyPK uses this mechanism to generate a secret-suitable for creating and replacing TinySec keys (see Section 2.3.1.1). This technique also helps to establish secure communication between two separate sensor networks that do not share a common key.

The whole key exchange scheme is presented in Fig. 2.6. Before the deployment phase, nodes are preloaded with a large prime n and a generator g. Both parameters are publicly

available. Each mote generates a private random number r and calculates $g^r \bmod n$. The calculated values are exchanged. After receiving the message both parties share a secret because of the property in Fig. 2.6.

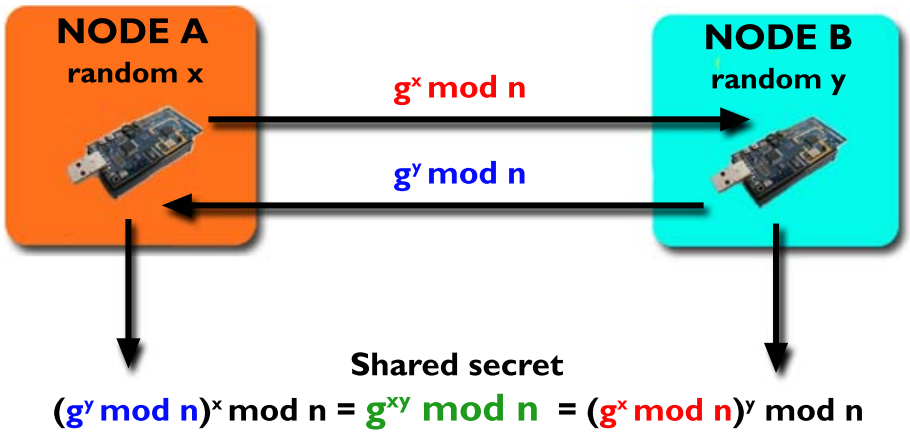

Figure 2.6: *Diffie-Hellman key exchange protocol in sensor networks*

The evaluation of TinyPK demonstrates that standard public key technology is not well-suited to constrained devices. The RSA 1024-bit public key operation takes 14.5s on the constrained MICA mote, which may be a reasonable value under the condition that such operations are performed very infrequently. However, the authors in [120] report that the implementation of RSA private operations is too slow to be used on sensor devices. The Diffie-Hellman key exchange performance is also not satisfactory. Modular exponentiation is very expensive on 8-bit processors and takes around 60s for a 1024-bit prime with a 176-bit exponent. The code for modular exponentiation required 12.4KB in ROM and 1.2KB RAM on the MICA2 platform.

One of the main drawbacks of TinyPK is the need to exchange many messages that are only related to the key establishment process. Communication in WSNs is very expensive in terms of energy consumption and the key distribution mechanism should minimize the number of exchanged messages. In TinyPK each sensor node has to store many public keys in addition to its private key. Each of those keys is 1024 bits in size and has to be authenticated before it can be used. The need for a trusted Certificate Authority in the system limits the range of possible applications. Additionally standard Diffie-Hellman key exchange scheme is not authenticated and allows dangerous man-in-the middle attacks. TinyPK shows that portions of the RSA cryptosystem can be successfully applied in WSNs, however, a complete system requires trusted infrastructure and hardware ac-

celeration for cryptographic operations.

2.3.3.2 Public Key Cryptography on 8-bit processors

Software implementations of Public Key Cryptography primitives on sensor devices were usually considered as not possible due to high memory usage and computational complexity. The authors in [57] revisited this common opinion by implementing elliptic curve and RSA primitives on two 8-bit microcontrollers: the CC1010 and the Atmega128. The latter is one of the most popular processors used in sensor networks.

RSA and Elliptic Curve Cryptography operations can be significantly accelerated with dedicated cryptographic coprocessors such as those used in smart cards. Unfortunately coprocessors require additional hardware which adds to the size and complexity of the devices. In many cases it is also easier to provide software implementations of basic Public Key Cryptography primitives.

Results presented in [57] show that modular multiplication and squaring of large integers are the performance-critical operations for RSA and Elliptic Curve Cryptography. The point multiplication routine on an elliptic curve is a fundamental operation of Elliptic Curve Cryptography systems. Multiprecision multiplication and squaring of large integers takes 77% of the execution time of the whole point multiplication routine on the Atmega128 processor. Therefore high performance implementations need to focus specifically on optimizing these operations in assembly language. Large integers multiplication not only involves arithmetic operations, but also a significant amount of data transport to and from memory. To optimize these operations the authors proposed a hybrid multiplication method that combines the advantages of row-wise and column-wise multiplication (see Section 3.3.2.1). This multiplication algorithm reduces the number of memory accesses and uses all available registers on the Atmega128 platform to achieve a 25% performance increase for elliptic curve point multiplication.

The authors in [57] describe the implementation of standardized RSA and Elliptic Curve Cryptography operations over NIST/SECG $\mathbb{F}_p$ elliptic curves. Several techniques like the Chinese Reminder Theorem (CRT), Montgomery multiplication and Optimized Squaring are used to accelerate RSA operations. Some optimizations are also performed for point multiplication on elliptic curves over $\mathbb{F}_p$. The authors use non-adjacent forms, projective coordinates and other curve-specific optimizations to improve the performance

of Elliptic Curve Cryptography. Implemented algorithms are evaluated with respect to performance, code size, and memory usage. All the results for RSA and Elliptic Curve Cryptography operations on Atmega128 processor are summarized in Table 2.2 [57].

Table 2.2: *Timings for different RSA and ECC operations on Atmega128.*

Algorithm	**Atmega128 @ 8MHz**		
	Time	ROM	RAM
ECC secp160r1	0.81s	3682B	282B
ECC secp192r1	1.24s	3979B	336B
ECC secp224r1	2.19s	4812B	422B
Modular exponentiation 512	5.37s	1071B	328B
RSA-1024 public-key $e = 2^{16} + 1$	0.43s	1073B	542B
RSA-1024 private-key w. CRT	10.99s	6292B	930B
RSA-2048 public-key $e = 2^{16} + 1$	1.94s	2854B	1332B
RSA-2048 private-key w. CRT	83.26s	7736B	1853B

Timings show that point multiplication on the standard secp160r1 curve [24] is faster than the RSA-1024 private-key operation and has comparable performance to the RSA-1024 public-key operation. The biggest difference in performance however is visible for higher security levels. Point multiplication on a secp224r1 curve takes only 2.19s compared to 83.26s for the RSA-2048 private key operation, which offers the same level of security. Memory requirements for a full RSA implementation with the Chinese Reminder Theorem are also significantly higher. Results in Table 2.2 prove for the first time that public key primitives can be implemented in an efficient way even on very constrained 8-bit architectures. This means that Public Key Cryptography is viable on embedded platforms without the use of hardware acceleration. Compared to RSA, Elliptic Curve Cryptography offers faster computation, memory, energy and bandwidth savings. Therefore it is a better suited solution for constrained sensor devices.

2.3.4 Identity based cryptography for sensor networks

Elliptic Curve Cryptography uses smaller parameters than other Public Key Cryptography techniques (e.g. RSA) making the cryptosystem more suitable for sensor networks. However, the problem of public key authentication is not solved by using Elliptic Curve Cryptography and the system still requires a practical public key infrastructure. One of possible solutions to this problem is to apply security schemes that are used in identity based cryptography.

As mentioned earlier, Identity-Based Cryptography is a public key technique that is based on identities of users. With Identity-Based Cryptography the wireless sensor node's identity can be used as the public key and hence there is no need for a certificate to bind a wireless sensor node's identity to its public key. Such a system provides practical public key encryption without the use of a complex public key infrastructure. In many ways an identity based scheme is a perfect solution for sensor networks. There is no need to maintain a public key directory, as the public keys can be derived from node's identities that are widely known in the network. Identity-Based Cryptography provides scalable security mechanism in which the number of keys is kept to a minimum. Nodes generate a public key for a given node only when they want to communicate with it for the first time. After agreeing upon a shared session key, nodes can use cheap symmetric key mechanisms (like TinySec) to encrypt the messages and to communicate in a secure manner.

Identity-Based Cryptography allows every node to send secure messages to all other nodes from the beginning of the network operation. No prior interaction between the nodes is needed. Exchange of the information does not require any service or assistance from a third party. However, identity-based systems assume the existence of a trusted key generation center, which issues private keys corresponding to user's identities. This authority can use its master key to decrypt user's messages. It can also impersonate anyone in the network. This feature introduces the key escrow problem, where the security of the whole system depends on the public key generator security. In many cases, a single unconditionally trusted entity in the network simply does not exist.

Fortunately, in sensor networks the original network deployer can be considered as a trusted entity that can act as the public key generator. It can generate a unique secret key based on each node's identity and pre-load this information to node's memory before the deployment phase. At this stage a secure channel clearly exists which allows careful configuration of the network. The application of identity based cryptography to wireless sensor networks is presented in Figure 2.7.

Security schemes that are based on Identity-Based Cryptography allow easy addition of new nodes to the network. There is no need to replace or add new keys to existing devices. New sensor nodes only have to be programmed with the domain parameters and a private key by the public key generator before deployment. The communication over-

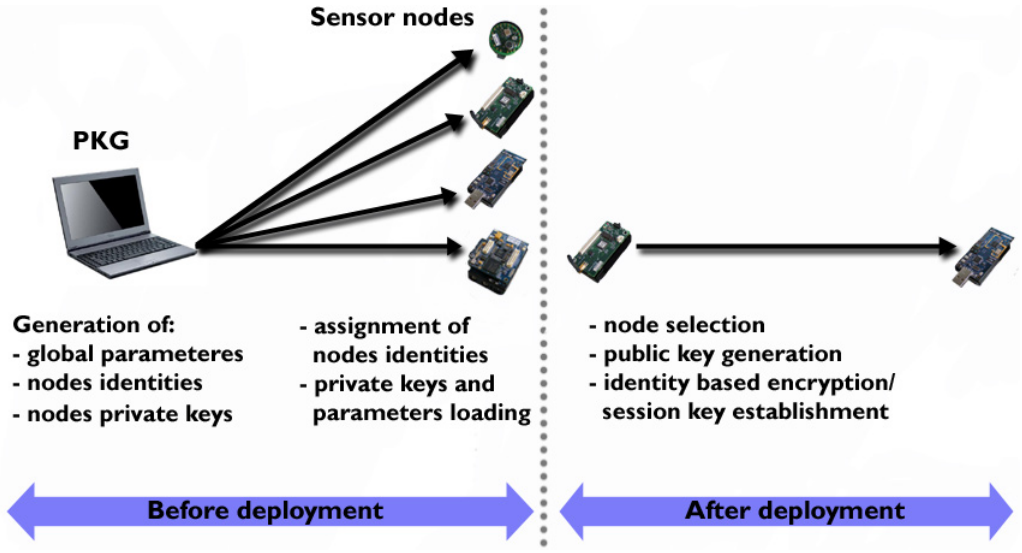

Figure 2.7: *Application of IBC in WSNs*

head in establishing session keys is minimal in Identity-Based Cryptography schemes. This makes the Identity-Based Cryptography approach much more suitable for low energy WSNs than traditional public key infrastructure schemes. The elimination of digital certificates lowers the energy consumption and makes the system more practical, especially in WSNs that are deployed in remote areas. In the case of sensor networks, Identity-Based Cryptography offers better security than other methods that are not based on Public Key Cryptography. The whole system also increases the resistance against the node capture attack. Subverting one of the nodes does not reveal anything about the communication between other pairs of nodes. It allows only to decrypt the messages received from other nodes. Network access control is also provided as only the public key generator issues identities and pre-loads sensors with valid private keys. An active attacker can encrypt messages to given identities but cannot decrypt any message without a proper private key.

Identity-Based Cryptography has clear advantages over traditional public key systems, but also has some inherent problems. One of them is key revocation. In traditional Public Key Cryptography systems, compromised keys are replaced with a new private/public key pair. In Identity-Based Cryptography systems, key revocation requires that users have to change their identity information that corresponds to given private keys. This might be especially problematic in cases where identities are chosen as nodes unique physical addresses (e.g. transceivers serial numbers). One solution to the problem

might be to use network addresses (e.g. IPv6 addresses) to identify nodes in the network. An alternative solution would be to combine date with the identity information to generate new private keys when necessary. The problem of key revocation highlights the importance of proper management of nodes identities in Identity-Based Cryptography systems.

Despite some minor problems Identity-Based Cryptography has many advantages when compared with other security schemes. Table 2.3 summarizes the main benefits that arise from using Identity-Based Cryptography to secure wireless sensor networks.

Table 2.3: *Identity based cryptography vs other security schemes for WSNs.*

	Symmetric key cryptography	**Random key pre-distribution**	**PKC**	**IBC**
Computational complexity	Low	Low	High	High
Communication overhead	Low	High	High	Low
Key distribution	Problematic	Simple	Complex	Simple
Number of keys	$O(n^2)$	O(n)	O(n)	n
Key directory	At each node	At each node	At each node or key center	No
Digital certificates	No	No	Yes	No
Forward encryption	No	No	No	Yes
Non-repudiation	No	No	Yes	Yes

Identity based cryptography seems to fit perfectly as a solution for the key distribution problem in WSNs. It provides advanced and flexible primitives that can be used to create novel key agreement schemes which can establish symmetric keys without any communication between the nodes. There is no need for random key pre-distribution, pair-wise key sharing or complicated one-way key chain schemes. The application of identity based cryptography in sensor networks is a promising new approach to WSN security that can provide practical Public Key Cryptography solutions in this challenging environment.

On the other hand the computational complexity of Identity-Based Cryptography is significant and it is unclear if such systems are even viable on constrained sensor devices. It is also uncertain if Identity-Based Cryptography can provide a sufficient level of security. There is a clear correlation between the level of security achieved and the

processing power required. It is important to demonstrate that efficient Identity-Based Cryptography implementations are indeed possible with security parameters set beyond the current cryptanalysis records. However, before implementation details can be discussed, there is a need to identify all the building blocks that are necessary to implement a complete Identity-Based Cryptography scheme in sensor networks.

2.3.5 IBC protocol building blocks

The implementation of Identity-Based Cryptography protocols is more complicated than in case of traditional public key schemes (eg. RSA). It requires many modules that perform various operations. Figure 2.8 illustrates the framework that is required to implement complete Identity-Based Cryptography protocols.

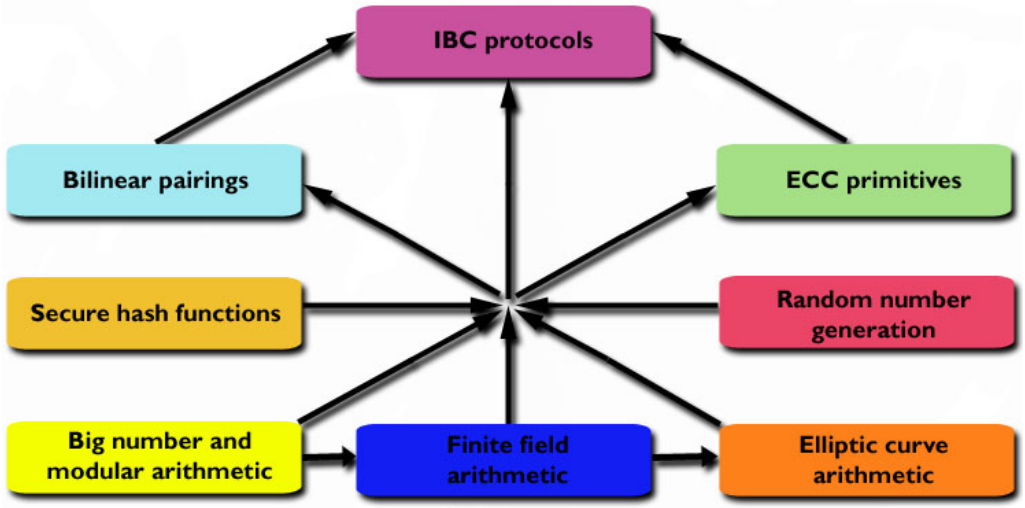

Figure 2.8: *IBC protocol building blocks*

Efficient implementation of pairing based cryptosystems in sensor networks is challenging and requires optimization of the following modules:

1) Finite field arithmetic. Elliptic curves used in cryptography are defined over finite fields. Therefore efficient arithmetic operations in the underlaying field are crucial to the overall performance of the system. Much effort must be devoted to speeding up basic arithmetic routines, as they are frequent in higher level operations. For best performance, platform-dependent hand-coded assembly routines should be used.

2) Big number and modular arithmetic. In cryptography, arithmetic operations are performed on numbers which are hundreds of bits long. Such numbers are not sup-

ported natively in many computing languages and must be implemented externally. Appropriate handling of big numbers and modular arithmetic is the key to efficient cryptographic operations.

3) Elliptic curve arithmetic. Cryptographic mechanisms based on elliptic curves depend on arithmetic involving the points of the curve. Elliptic curve operations rely on the arithmetic in the finite field, but the performance of point addition and doubling is also important. The efficiency of curve arithmetic depends on the type of the curve, the size of the underlying field, the point representation system and the implementation of the algorithms.

4) ECC primitives. One of the main cryptographic primitives in Elliptic Curve Cryptography is the scalar point multiplication. This operation is the base for many Elliptic Curve Cryptography schemes such as Elliptic Curve Digital Signature Algorithm, Elliptic Curve Diffie-Hellman and Elliptic Curve Integrated Encryption Scheme. The performance of point multiplication is crucial in Elliptic Curve Cryptography as it dominates the execution time of the above schemes. Point multiplication is an important primitive not only in standard Elliptic Curve Cryptography but also in Identity-Based Cryptography. Efficient implementation of point multiplication is difficult due to the variety of existing algorithms and different types of elliptic curves.

5) Cryptographic pairings. The calculation of cryptographic pairings is the most computationally intensive operation in Identity-Based Cryptography schemes. Pairing-Based Cryptography is a new area of research in cryptography which have not been extensively studied yet. Software implementations of pairings were considered as too heavyweight for embedded devices and the application of Pairing-Based Cryptography to sensor networks has not been investigated so far.

Each of the above building blocks cannot be viewed in isolation and has to be optimized with other modules in mind. For example finite field operations have to be optimized for various field sizes in order to support different curves and security levels. An optimization that may not usually help, could be beneficial when the behaviour of the whole system is taken into account. In some cases one optimization could adversely affect another.

As can be seen in Figure 2.8, identity-based systems require also other building blocks such as hash functions and random number generators. Secure hash functions are important in every cryptographic scheme, but they are not as computationally expensive as other operations in Identity-Based Cryptography and hence will not be discussed in detail in this work. The security and efficiency of random number generators is important in sensor networks but this topic is also beyond the scope of this thesis. The following chapters present a detailed description on how to implement complete identity-based security protocols in sensor networks. The implementations are discussed on all levels of development beginning with basic operations in the finite field, and finishing on new pairing based cryptography protocols for WSNs.

2.4 Summary

Cryptographic key distribution is a fundamental security service in every computer system. Other security services like authentication, data confidentiality and integrity can be built only upon a solid key management framework. The problem of symmetric key distribution is challenging especially in large distributed systems like wireless sensor networks, where the available resources are very limited.

Traditional fixed networks usually solve the key distribution problem with the use of Public Key Cryptography. This chapter presents the classical approach to key distribution. It presents the idea behind Public Key Cryptography and shows how a basic public key infrastructure works. It also describes the problems of using a traditional Public Key Infrastructure in a sensor network environment. Section 2.2 presented also a brief overview of recent advances in Public Key Cryptography such as Elliptic Curve Cryptography and cryptographic pairings. From these techniques pairings are especially interesting as they enable identity based cryptography which can be used in sensor networks to provide a practical Public Key Infrastructure and a simple way for symmetric key distribution.

The development of feasible key distribution mechanisms for WSNs is an active area of research. Despite much effort, key distribution and management still remains as an open problem and a main topic in WSN security. This chapter presents a survey of different key distribution techniques that were proposed in the literature. Most of these security solutions use basic symmetric key algorithms due to their simplicity and efficiency in

resources utilization. Symmetric key mechanisms (like TinySec) provide link layer security but they do not solve the key distribution problem and assume that the nodes already share network-wide or pair-wise keys. Random key pre-deployment schemes have low computational complexity but high communication overhead and they do not scale well with the size of the network. Probabilistic key sharing like other schemes based on symmetric key cryptography do not provide a good trade-off between resilience and storage of cryptographic keys. Other methods that are based on key distribution centers are also ill-suited to WSNs as they are inefficient in resources utilization and require a trusted infrastructure.

Symmetric key mechanisms offer a security level that is not acceptable in many WSN applications especially more mature ones that deal with critical data. Some researchers [57], including this author, believe that a proper level of security for those applications can only be achieved with the additional use of Public Key Cryptography methods. First efforts to apply Public Key Cryptography in sensor networks showed that RSA is not feasible on sensor devices as it requires large keys and its private key operations are very expensive in terms of computation. Further work in this area showed that Elliptic Curve Cryptography is a more suitable Public Key Cryptography method in a resource constrained environment because of smaller key sizes and faster execution time.

Practical use of Public Key Cryptography methods in WSNs requires not only computational feasibility but also efficient ways for public key authentication. The use of a traditional PKI with certificates and digital signatures is not possible in sensor networks, therefore new cryptographic mechanisms have to be developed. This chapter proposes the use of identity-based cryptography methods for distributing keys in the network. This approach has many advantages over existing solutions. It can provide simple key distribution without the use of expensive certificates. It can also introduce significant savings on radio transmissions and storage of cryptographic keys. Despite the advantages, the application of Identity-Based Cryptography in WSNs have not been fully investigate so far. It is uncertain if Elliptic Curve Cryptography and pairings can by efficiently implemented on sensor nodes without the use of hardware accelerators. There are also no pairing-based protocols that were designed specifically for sensor networks. These problems will be addressed in the following chapters of this thesis.

CHAPTER 3

Finite Field Arithmetic for Low-end Processors

Finite field arithmetic is one of the main building blocks of every Identity-Based Cryptography protocol. Finite fields are very important constructs, because they have special properties that can be exploited for the purpose of cryptography. They serve as elementary blocks in ECC because elliptic curves used in cryptography are always defined over finite fields. There are three main types of finite fields that are suitable for implementation of elliptic curve systems: prime fields, binary fields and optimal extension fields.

The efficient implementation of finite field arithmetic is an important prerequisite in elliptic curve systems because curve operations are performed using arithmetic operations in the underlying field. Operations on field elements such as addition, subtraction, multiplication, squaring and inversion operate on numbers which are usually hundreds of bits long. These arithmetic operations are repeated many times in higher level algorithms. Hence their efficiency has a significant influence on the overall performance of the system. The efficiency of the finite field operations depends on several factors:

- the hardware capabilities of a given platform;
- the type and size of the finite field;

- the way of representing the field elements;
- the algorithms used for field arithmetic.

Of all arithmetic operations multiprecision multiplication of field elements is the most time-critical routine in typical finite fields. There are operations that are more complex in terms of computation (e.g. inversion) but they are used infrequently in practice and hence are not time-critical. The efficiency of field multiplication is especially crucial on low-end processors such as the 8-bit Atmega128 CPU, which is used in current sensor node devices. The best implementation results on such a constrained hardware can only be achieved with the use of hand crafted assembly language code. These routines should be specifically optimized for a given hardware platform for maximum performance.

3.1 Introduction to finite fields

Fields are abstractions of familiar number systems and their basic properties. The set of real numbers $\mathbb{R}$ form a field, but the set of integers is not a field since every element does not have a multiplicative inverse. A field consist of a set $\mathbb{F}$ together with two binary operations, addition (+) and multiplication ($\cdot$), that satisfy the usual arithmetic properties [58]:

1) $(\mathbb{F}, +)$ is an abelian (commutative) group with (additive) identity denoted by 0;
2) $(\mathbb{F} \setminus \{0\}, \cdot)$ is an abelian group with (multiplicative) identity denoted by 1;
3) Distributive law: $(a + b) \cdot c = a \cdot c + b \cdot c$ for all $a, b, c \in \mathbb{F}$.

3.1.1 Prime fields

A field $\mathbb{F}$ is regarded as a finite field when its order is finite. The order of a finite field is the number of elements in the field. There exists a finite or Galois Field (GF) of order q if and only if q is a prime power. Such a field is denoted as $GF(q)$ or $\mathbb{F}_q$ with $q = p^m$ where p is a prime number called the characteristic of $\mathbb{F}$, and m is a positive integer. If $m = 1$, then $\mathbb{F}$ is called a prime field. In case where $m \geq 2$, then $\mathbb{F}$ is called an extension field. The prime field is denoted as $\mathbb{F}_p$ and consists of integers modulo p ($\{0, 1, 2, \ldots, p-1\}$) with addition and multiplication performed modulo p. The following presents an example of arithmetic operations in $\mathbb{F}_{17}$:

1) Addition: $9 + 12 = 4 \quad (21 \bmod 17 = 4)$;

2) Subtraction: $9 - 12 = 14 \quad (-3 \bmod 17 = 14)$;

3) Multiplication: $9 \cdot 12 = 6 \quad (108 \bmod 17 = 6)$;

4) Inversion: $9^{-1} = 2 \quad (9 \cdot 2 \bmod 17 = 1)$.

3.1.2 Binary fields

In the literature [58] finite fields of order 2^m are called binary fields or characteristic-two finite fields and are denoted as $\mathbb{F}_{2^m}$. Such fields can be constructed using polynomial or normal basis representation. All the software implementations described in this thesis use the polynomial basis as the multiplication operation is much more expensive when a normal basis[1] is used. When using a polynomial basis the elements of $\mathbb{F}_{2^m}$ are the binary polynomials (polynomials whose coefficients are in the field $\mathbb{F}_2 = \{0, 1\}$) of degree at most $m - 1$:

$$\mathbb{F}_{2^m} = a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \cdots + a_2z^2 + a_1z + a_0 : a_i \in \{0, 1\} \tag{3.1}$$

Addition of the elements in $\mathbb{F}_{2^m}$ is performed like a normal addition of polynomials, with coefficient arithmetic performed modulo 2 (XOR function). Multiplication of field elements is done in a special way as a reduction step is needed. For this purpose an irreducible[2] binary polynomial $f(z)$ of degree m is chosen. Such a polynomial exists for any m and can be found in an efficient way. Irreducibility of this polynomial means that $f(z)$ cannot be factored as a product of binary polynomials each of degree less than m. Multiplication of $\mathbb{F}_{2^m}$ elements is performed modulo the reduction polynomial $f(z)$. The following presents an example of arithmetic operations in $\mathbb{F}_{2^3}$. The irreducible polynomial for $\mathbb{F}_{2^3}$ is given by $f(z) = z^3 + z + 1$ and the field consists of 8 elements $\{0, 1, z, z + 1, z^2, z^2 + 1, z^2 + z, z^2 + z + 1\}$.

1) Addition: $(z^2 + z + 1) + (z^2 + 1) = z$;

2) Subtraction gives the same result as addition since $a = -a$ for all $a \in \mathbb{F}_{2^m}$;

3) Multiplication: $(z^2+z+1)\cdot(z^2+1) = z^2+z \quad (z^4+z^3+z+1) \bmod (z^3+z+1) = z^2+z$;

[1] A normal basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ is a basis of a form $N = \{\alpha, \alpha^q, \ldots, \alpha^{q^{n-1}}\}$

[2] Irreducible means that an object cannot be expressed as the product of two or more non-trivial factors

4) Inversion: $(z^2+z+1)^{-1} = z^2 \quad (z^2+z+1) \cdot z^2 \bmod (z^3+z+1) = 1$.

3.1.3 Extension fields

In addition to prime and binary fields elliptic curve cryptosystems can be also evaluated over extension fields. An extension field can be derived from the polynomial basis representation of binary fields. If p is prime, $k \geq 2$ and $\mathbb{F}_p[z]$ denotes the set of all the polynomials in the variable z with coefficients from $\mathbb{F}_p$, then the elements of the extension field $\mathbb{F}_{p^k}$ can be represented by polynomials in $\mathbb{F}_p[z]$ each of degree at most $k-1$:

$$\mathbb{F}_{p^k} = a_{k-1}z^{k-1} + a_{k-2}z^{k-2} + \cdots + a_2z^2 + a_1z + a_0 : a_i \in \mathbb{F}_p \tag{3.2}$$

The arithmetic in the extension field is derived from the usual arithmetic in polynomial rings. The addition of two polynomials is done by the addition of their coefficients modulo p. Extension field multiplication is computed by a complete multiplication of two polynomials and the subsequent reduction modulo the irreducible polynomial $f(z)$ of degree k. For efficiency reasons, some effort can be made to choose $f(z)$ to have a minimal number of terms and small coefficients. For example, for the field F_{p^2} , where p is a prime and $p = 3 \bmod 4$, $f(z)$ can be chosen as z^2+1, and elements can be represented as $a_1z + a_0$, with $a_1, a_0 \in \mathbb{F}_p$.

Extension fields that are used to build Elliptic Curve Cryptography systems are often in the form of Optimal Extension Fields (OEFs) [6]. The general idea with optimal extension fields $\mathbb{F}_{p^k}$ is to select p, k, and the reduction polynomial $f(z)$ to more closely match the underlying hardware characteristics. OEFs are usually defined as extension fields where the value of p is chosen as a small pseudo-mersenne prime[3] which fits in a single computer word. Such an approach simplifies the handling of carries in arithmetic operations. Despite this advantage, OEFs are not as widely standardized as prime and binary fields (e.g. IEEE P1363 standard [64]) and hence will not be discussed in this thesis. However the arithmetic operations in the extension field will be presented in this chapter as they are important in Pairing-Based Cryptography. The preference for small word-sized modulus p in OEFs makes these extension fields rather specialized. A much wider range of possible extension fields need to be considered in the context of cryptographic pairings.

[3]This prime has a form of $p \approx 2^d$, for example, $p = 2^{160} - 2^{31} - 1$

3.2 Embedded processors

Devices that are used in sensor networks embed different processors depending on the requirements for computing power and memory. The architectures range from low-cost 8-bit to more advanced 32-bit platforms (see Table 1.1 in Chapter 1.1.1). Most of the sensor nodes, however, have CPUs with very limited capabilities and low energy usage. Therefore the security protocols designed for WSNs should be first optimized to work with the low-end 8-bit platforms.

All the cryptographic operations presented in this thesis were tested on three different processors commonly used in sensor networks. The most popular CPU is the 8-bit ATmega128 [5] provided by the Atmel corporation. This chip is embedded in such platforms as MICA2DOT, MICA2 and MICAz. The second processor is the Texas Instruments MSP430F1611 CPU [115] which is used in Tmote Sky nodes. The last hardware platform is the PXA271 XScale microprocessor [66] which is based on an ARM core. This CPU is being used in recent Imote2 nodes, which have more computing power and much improved capabilities than the previously mentioned platforms. The above three processors can be treated as the typical representatives of the 8, 16 and 32-bit classes of sensor network devices.

3.2.1 ATmega128 8-bit processor

The ATmega128 belongs to the Atmel AVR family of processors which is based on a modern highly structured RISC design. It offers 133 instructions, most of which are executed in a single CPU cycle. It can achieve a maximum throughput of 16 MIPS at 16 MHz. However, the models used in sensor networks are usually clocked at 7.3828 MHz. The ATmega128 has a Harvard architecture with separate memory spaces for code and data. One of the most interesting features of this hardware platform is the large set of 32 general purpose 8-bit registers. This feature gives the developer a lot of flexibility when writing source code. The physical memory is split up in three parts: program memory with 128 KB in system-programmable flash memory, 4 KB of RAM and 4KB of configuration EEPROM. There are many development tools available for the Atmel AVR family. Atmel offers the free AVR Studio development environment. It contains a cycle-accurate simulator of the ATmega128 processor which is a great help when developing the assembly language code.

3.2.2 The 16-bit MSP430 microcontroller

The 16-bit MSP430F1611 produced by Texas Instruments differs in many ways from the Atmel chip. The MSP430 has a more traditional architecture that uses mainly memory-to-memory operations rather than the more classic RISC load-store approach of the ATmega128. Its instruction set is limited to 27 instructions, but the variety of 7 different addressing modes offers a lot of flexibility in data manipulation. This orthogonal architecture allows every instruction to be used with every addressing mode. An interesting feature of this CPU is the existence of an external hardware multiplier which performs fast multiplication of 8 and 16-bit integers. The MSP430 provides twelve general purpose registers. The remaining four registers `r0-r3` are used for Program Counter, Stack Pointer, Status Register and Constant Generator respectively. Registers `r4-r15` are general purpose and available for use at all times. This CPU offers also 10KB of RAM, but only 48KB of ROM which might be a problem in the case of large programs. A very nice feature of the MSP430 is its ultra low power consumption which is especially important on tiny devices in distributed environments where the available energy is very limited.

3.2.3 The 32-bit ARM processor

The 32-bit Marvell PXA271 processor is built around the ARM core and differs completely from the two previously described platforms. It is a much more advanced and powerful CPU with superior capabilities. Also the memory resources on this processor are a few orders of magnitude higher than on the other two devices (32 MB of ROM and 256 KB of RAM). This range of CPUs is commonly used in smartphones (e.g. the iPhone) and PDA class devices and opens new possibilities when used in the WSN environment. The ARM processor has a standard RISC load-store architecture, with several innovative features, including effectively free shifting of operands at no extra cost (using a built-in barrel shifter) and conditional execution of instructions. It has 16 registers, although three of these are reserved for special purposes, `r13` as a stack pointer, `r14` holds a function return address, and `r15` is the program counter. If needed, the content of `r14` can be pushed onto the stack and an extra register is then available until the end of the called function. The PXA271 offers full 32-bit processing and its clock rate can be changed dynamically by adjusting the input voltage. This interesting feature permits a tradeoff between energy consumption and processing power during the CPU operation.

3.3 Efficient prime field arithmetic on constrained CPUs

The prime field arithmetic must provide multiplication, squaring, addition, subtraction and reduction operations. Field inversion is also required for some operations but in many cases it can be replaced with a larger number of cheaper operations. For example using Montgomery's method [91] the modular reduction can be carried out without using division, and hence it is the modular multiplication and squaring which are significant.

Multiprecision multiplication is the time-critical requirement in the great majority of number-theoretic based methods for public key cryptography. It is required for the RSA algorithm, for methods based on elliptic curves, and also for new mechanisms that are based on cryptographic pairings. Elliptic curve systems are usually defined over finite fields $\mathbb{F}_p$ of large characteristic where p has at least 160 bits (for security reasons). Since the modulus is of a fixed size in bits, the code for addition, multiplication, squaring and modular reduction can be written in assembly language. The loops can be completely unrolled to achieve maximum performance at the cost of some additional storage.

The implementation of prime field arithmetic requires that each field element is represented using multiple machine words. In the case of $\mathbb{F}_p$ where p is a large prime, the field element a can be implemented as a series of W-bit unsigned integers $0 \leq a_i < 2^W$, where W is the wordsize on the target machine (e.g. W = 8, 16, 32) and $a = \sum_{i=0}^{t-1} a_i 2^{W_i}$. The number of required words t to hold a field element can be calculated from $t = \lceil \frac{\log_2 p}{W} \rceil$. The binary representation of a field element a can be stored in an array $A = (A[t-1], \ldots, A[2], A[1], A[0])$ where the rightmost bit of $A[0]$ is the Least Significant Bit (LSB).

3.3.1 Modular addition and subtraction

Multiprecision integer addition and subtraction is a straightforward operation on standard general purpose processors. It requires t additions (or subtractions) of wordsize integers. One thing that needs to be remembered is the propagation of carry bits throughout the calculations. On processors that have the add-with-carry (and subtract-with-carry) instruction, there is no need for an explicit check for carry. Listing 3.1 presents the assembly language code that performs addition of two wordsize integers on three different processors used in sensor networks.

In the case of the ATmega128, registers `Y` and `Z` store the addresses of two field elements a and b, whereas `X` holds the address of the result c. The MSP430 uses the registers

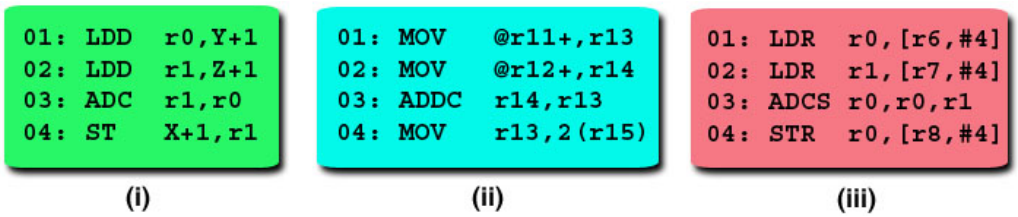

Figure 3.1: *Multiprecision addition of two wordsize integers on (i) ATmega128, (ii) MSP430 and (iii) ARM processors*

`r11`, `r12` and `r15` respectively with address increment of 2 bytes. The three registers in case of the ARM processor are `r6`, `r7`, `r8` with 32-bit addressing. Multi-word subtraction is similar to addition, the difference is in line `03` where a subtract with carry instruction needs to be inserted. In the context of subtraction the carry bit is often called a borrow.

Arithmetic in the field $\mathbb{F}_p$ requires modular additions $c = (a + b) \bmod p$ and subtractions $c = (a - b) \bmod p$. These operations require an additional step which performs reduction modulo p. In case of modular subtraction, after calculating t wordsize subtractions the borrow bit is tested. If the borrow is equal to 1 then the modulus p is added to the result c. Similarly in modular addition the carry bit decides if a reduction is needed. When the carry bit is equal to 1 then the modulus p needs to be subtracted from c. The reduction is also needed if carry is equal to 0 and $c \geq p$.

Table 3.1: *Timings in instruction cycles for modular addition and subtraction of 160-bit integers on standard WSN processors.*

Hardware platform	**Atmega128**	**MSP430**	**PXA271**
Modular addition (assembly)	404	386	155
Modular addition (C)	596	533	200
Modular subtraction (assembly)	392	377	149
Modular subtraction (C)	584	526	194

Table 3.1 presents the timings for modular addition and subtraction operations on 160-bit integers in $\mathbb{F}_p$. All the timings are given in clock cycles of a given hardware platform. They represent an average value over 100 operations because the reduction step is not always executed. The numbers in Table 3.1 prove that assembly implementation of modular addition and subtraction gives on average 25% better performance than the standard C implementation.

3.3.2 Multiprecision integer multiplication

There are two main methods to implement multiprecision integer multiplication in software. The first group uses the Karatsuba-Ofman techniques that divide the operands into smaller integers and thus reduce the number of multiplications. Unfortunately this approach has a large overhead in additional operations and is not efficient in finite fields of practical interest. The second group of techniques is more practical and uses different variations of basic schoolbook multiplication.

When doing a multiplication using the schoolbook algorithm the multiplication is divided into several partial products that are accumulated to get the final result. The partial products can be calculated in any order before they are added together. They can be arranged in rows from right-to-left or in columns by bit length. Figure 3.2 shows a 4×4 multiplication using both methods.

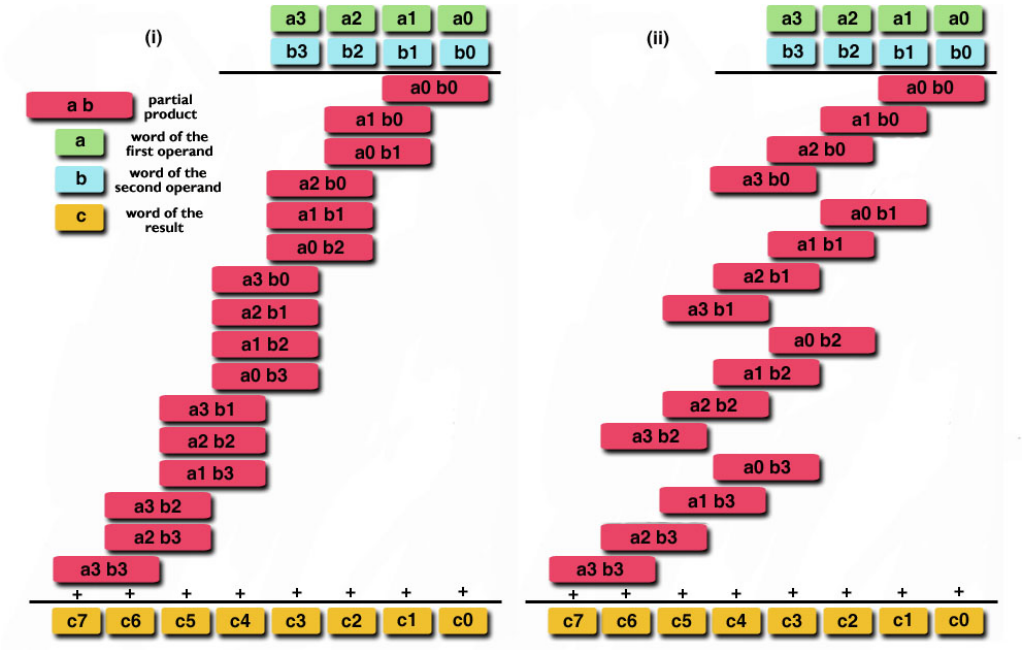

Figure 3.2: *(i) Column-wise and (ii) Row-wise multiplication*

The column-wise technique (also known as the product scanning method) is not so straightforward to program as the columns are of different lengths, although this is not a concern if the code is unrolled. The alternative row-wise method (also known as the operand scanning method) is much easier to implement as a short looped program. The word-by-word multiplication can be carried out using a simple pair of nested "for" loops. The row-wise algorithm requires more memory writes, but less memory reads of the

operands. Therefore the column-wise algorithm should be preferred as memory writes only occur at the foot of each column. A disadvantage of both methods is that they reload the same operands many times to calculate different partial products. This is obviously not an optimal approach especially if a processor have enough registers available to store some of these values for later use.

Recently in [57] it has been suggested that a hybrid method which combines features of both techniques might be preferable, particularly in a setting where the processor has many available general purpose registers. This method was first described in the context of the ATmega-128L 8-bit processor, which has 32 registers. In the same setting, Uhsadel *et al.* [116] have recently obtained improved results. The following sections presents a description of an optimized method of hybrid multiplication which simplifies the whole approach. The improved technique can be applied to a wide range of embedded processors and is superior to the earlier proposals.

3.3.2.1 Improved hybrid multiplication

The most efficient implementations of multiprecision multiplication use the method of Comba [28], which is based on the column-wise technique (see Fig. 3.2 (i)). Each W-bit pair of digits is multiplied together to create one $2W$ bit partial product. This partial product is accumulated in a triple register. The third register (called a "carry-catcher") is required to catch the carries that can arise as the full column is added up. After the column addition, the least significant register of the triple register is written to memory as a part of the result, and the other two registers shifted down, representing the carry to the next column. The maximum number that can arise in the third and most significant register is bounded by the number of digits in the multi-precision multiplication. This is likely to be much smaller than $2^W - 1$, which is the maximum number that can be stored in a register, and so sometimes a smaller register can be used here.

The hybrid method [57] keeps the advantages of the column-wise method, while exploiting any extra registers to avoid unnecessary reloads of data. The idea is straightforward – perform the multiplication as if the word length of the computer were actually $d \cdot W$, and perform the $d \times d$ multiplication that arises in the calculation of the larger partial product using the row-wise algorithm. Figure 3.3 (i) illustrates the hybrid method for the case $d = 2$. Four wordsize integers are loaded into registers. As each large 2×2 par-

tial product is calculated (represented by the large outer boxes), it must be accumulated into registers. The diagram illustrates the accumulation of a particular partial product, and curved arrows indicate the carries. A naive implementation would require five registers in this case to store the sum. However, in the worst case this might require up to five add-with-carry instructions, as in integer addition, a carry-out is always a possibility that must be catered for.

The original hybrid method [57], and the improved variant described in [116], use a rather complex method to deal with this carry propagation problem, based on the fact that the product of two words, plus two words, cannot overflow a double-word-sized register since:

$$(2^W - 1)(2^W - 1) + (2^W - 1) + (2^W - 1) = 2^{2W} - 1 < 2^{2W} \tag{3.3}$$

A new idea to improve the hybrid method is to simply employ even more registers to suppress the carry propagation. As well as using $2d$ column registers to hold the sum of each now wider column, an extra $2d - 1$ registers are deployed as "carry catchers". These registers are initialized to zero at the top of the column and catch any carries that may arise without the possibility of further carry propagation. When the column is finished, these carry catchers are simply added with carry to the main set of column registers. The case for $d = 2$ is shown in Figure 3.3 (ii). The most significant carry-catcher register performs exactly the same role as the fifth register in the naive implementation.

The $d \times d$ row-wise multiplication requires $d + 1$ registers, d registers to hold the entire row, each element of which is then multiplied by another register (which stores the current multiplier) to create d partial products. There are $2d$ column registers and $2d - 1$ carry-catchers, for a total of $5d$ registers. For $d = 1$ this is the original Comba method, which requires five registers, just about possible if the number of general purpose registers is $r = 8$. The choice $d = 2$ requiring ten registers is a good fit for a processor with a total of $r = 16$ registers, and $d = 4$ will take 20 registers for the case $r = 32$. The improved hybrid method can exploit even more registers if they are available on the target platform.

The above figures are approximate and may require modification for a particular architecture. Not all of the r registers on a given type of CPU might be available to the developer. At least three registers will be required to store the memory addresses of the

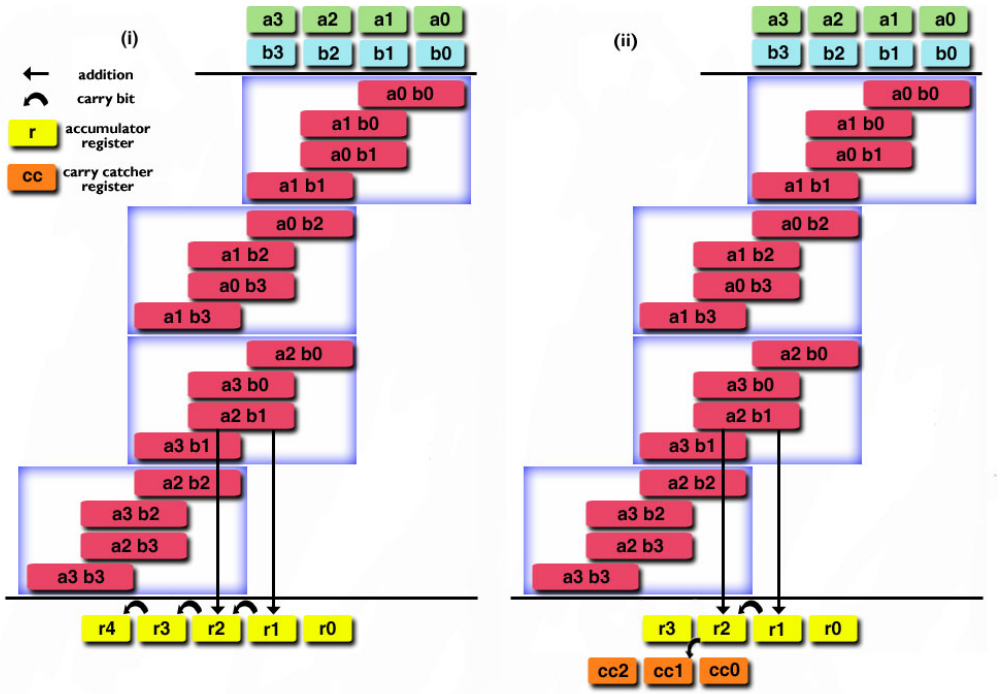

Figure 3.3: *The processing of a two word partial product using (i) A naive implementation of the hybrid method and (ii) The improved hybrid implementation*

operands and of the result. It is also sometimes helpful if a register fixed with the constant value of zero is available. The processor must support simple indexed memory load and store instructions, an integer multiply instruction which outputs the result in a pair of registers, and a simple add-with-carry instruction.

The implementation of the improved hybrid method is always processor specific, as CPUs differ in small but significant ways that can not be described in a one-size-fits-all way. Indeed the previous work is described only in the specific context of a particular processor [28], [57], [116]. However, in order to prove the generality of the improved hybrid multiplication, this approach was applied to three different embedded processors. In this context the new method achieved better performance (around 8-20%) than previous techniques.

3.3.2.2 The ATmega128 microcontroller

The ATmega128 processor has 32 registers, denoted `r0` to `r31`. There are three memory addressing registers, denoted `X`, `Y` and `Z`, which actually refer to the 16-bit register pairs `r26:r27`, `r28:r29` and `r30:r31`. The calculation of a typical single partial product on this processor using the standard Comba method is illustrated in listing 3.4 (i). The `Y` and

Z registers are pointing at 8-bit digits of the multiplicand and the multiplier respectively. The X register is pointing at the address which will eventually contain the sum of this column. Using standard indexed addressing and the LDD instruction, the multiplicand and multiplier are loaded into the r0 and r1 registers. The MUL instruction always places the 16-bit result in the register pair r0:r1. The triple register in this case is r6:r7:r8 and the register r5 is kept as a constant zero. Then an ADD and two ADC instructions are used to add the new product into the triple register, taking care of the carries.

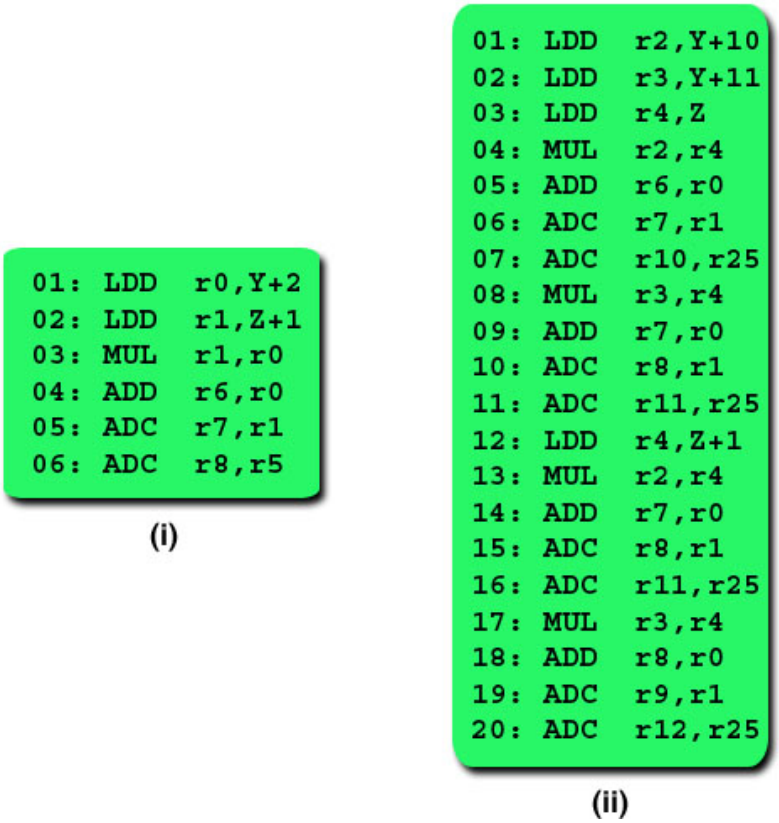

Figure 3.4: *The processing of (i) A single partial product (Comba method) and (ii) four partial products (improved hybrid method) on ATmega128*

Listing 3.4 (ii) shows the improved hybrid method with $d = 2$. In this case 4 partial products are calculated together in each step, requiring more registers. In this case the column registers are r6:r7:r8:r9, and r10:r11:r12 are used as the carry-catchers. This time r25 is used to hold the constant zero.

The count of basic operations reveals the advantage of the improved hybrid method. To calculate 4 partial products using the simple Comba method would require 4 times the operations in 3.4 (i), that is 8 loads, 4 multiplies, 4 additions and 8 additions-with-carry. However for the same work carried out, the improved hybrid method requires 4 loads,

Table 3.2: *Comparison of instruction counts for ATmega128*

		New method		Uhsadel *et al.*		Gura *et al.*		Classic Comba	
Instruction	CPI	Instr	Cycles	Instr	Cycles	Instr	Cycles	Instr	Cycles
add/adc	1	1263	1263	986	986	1360	1360	1200	1200
mul	2	400	800	400	800	400	800	400	800
ld	2	200	400	238	476	167	334	800	1600
st	2	40	80	40	80	40	80	40	80
mov	1	70	70	355	355	355	355	81	81
other			38		184		197	44	44
Totals			**2651**		**2881**		**3106**		**3805**

4 multiplies, 4 additions and 8 additions-with-carry, from figure 3.4 (ii). The saving of 4 load instructions is particularly significant as the load instruction takes 2 clock cycles (the same as the multiply instruction).

The ATmega128 processor has enough unused registers to extend the improved hybrid idea to $d = 4$ with further savings. Using this method, the number of clock cycles for a full 160×160 bit multiplication can be reduced to 2651 clock cycles. This compares favourably with the 2881 clock cycles recently reported in [116], and the 3106 clock cycles reported in [57] (approximately 15% faster). This is despite the fact that the above implementations use $d = 5$ rather than the $d = 4$. The improved method cannot use $d = 5$ on ATmega128 due to the extra requirement for carry-catcher registers. Table 3.2 summarizes the results for multiprecision multiplication on the ATmega128.

3.3.2.3 The MSP430 CPU

In order to achieve the best performance for multiprecision multiplication on the 16-bit MSP430, it is important to use the hardware multiplier. This device is a peripheral and is not implemented in every member of the large MSP430 family of microprocessors. It is accessed via memory mapped I/O registers. The MSP430F1611 model which is used in sensor devices includes this hardware multiplier.

Listing 3.5 (i) shows the calculation of a typical partial product on this architecture using the standard Comba method. Registers `r13` and `r14` store the memory addresses of the first 16-bit digits of the multiplicand and the multiplier. Indexed addressing mode is used to find the address of a particular 16-bit digit. In this case the third digit of the multiplicand and the second digit of multiplier are loaded into the hardware multiplier

registers using the memory-to-memory `MOV` instruction. Writing the first operand to the `MPY` register selects unsigned multiply mode but does not start any operation. Loading the second operand to `OP2` initiates the multiply operation. The lower and the higher 16-bit parts of the 32-bit result are stored in `RESLO` and `RESHI` registers respectively and can be read by the next instruction. One `ADD`, one `ADDC` and one `ADC` (add carry to destination in the MSP430 instruction set) instructions are used to add the new product to the triple register which consists of `r9:r10:r11`.

```
01: MOV   4(r13),&MPY
02: MOV   2(r14),&OP2
03: ADD   &RESLO,r9
04: ADDC  &RESHI,r10
05: ADC   r11
```

(i)

```
01: MOV   20(r10),r8
02: MOV   22(r10),r9
03: MOV   12(r11),&MPY
04: MOV   r8,&OP2
05: ADD   &RESLO,r4
06: ADDC  &RESHI,r5
07: ADC   r13
08: MOV   r9,&OP2
09: ADD   &RESLO,r5
10: ADDC  &RESHI,r6
11: ADC   r14
12: MOV   14(r11),&MPY
13: MOV   r8,&OP2
14: ADD   &RESLO,r5
15: ADDC  &RESHI,r6
16: ADC   r14
17: MOV   r9,&OP2
18: ADD   &RESLO,r6
19: ADDC  &RESHI,r7
20: ADC   r15
```

(ii)

Figure 3.5: *The processing of (i) A single partial product (Comba method) and (ii) four partial products (improved hybrid method) on the MSP430*

Due to the limited number of available registers on the MSP430 the improved hybrid method can be implemented only with $d = 2$ (listing 3.5 (ii)). Eleven registers are required to calculate the 4 partial products in every step of this algorithm. This time `r4:r5:r6:r7` are the column registers, and `r13:r14:r15` are the carry-catchers. Registers `r10:r11` are used as pointers to particular digits in the multiplicand and multiplier. In the MSP430 instruction set, the number of clock cycles per instruction depends both on the instruction mnemonic and the type of operands being used. Because memory-to-memory `MOV` instructions require the most cycles, two extra registers `r8` and `r9` are used as temporary storage to save on the overall cycle count (memory-to-register operations are much less expensive).

At a first glance it might appear that the cost of both methods (hybrid and standard) is exactly the same on the MSP430. Calculation of 4 partial products using the simple

Table 3.3: *Comparison of instruction counts for the MSP430*

Instruction	CPI	New method		Classic Comba	
		Instructions	Cycles	Instructions	Cycles
add &label,reg	3	100	300	100	300
addc &label,reg	3	100	300	100	300
adc reg	1	109	109	100	100
mov x(reg),&label	6	45	270	180	1080
mov reg,x(reg)	4	20	80	20	80
mov reg,reg	1	27	27	38	38
mov reg,&label	4	100	400		
mov x(reg),reg	3	45	135		
other			125		167
Totals			**1746**		**2065**

Comba method requires 8 `MOV`, 4 `ADD`, 4 `ADDC` and 4 `ADC` instructions. The hybrid method uses also 20 instructions with exactly the same mnemonics. However on closer inspection its clear that the advantage of the hybrid method lies in the total number of clock cycles. The 8 `MOV` instructions in the simple algorithm takes 48 clock cycles in total. The hybrid method can take advantage of register to memory operations and saves 14 cycles using only 34 cycles for the same work carried out. The benefit in clock cycles of the improved hybrid method is very similar on both the MSP430 and the ATmega128-L in the case where $d = 2$.

The improved hybrid method on the MSP430 is limited to $d = 2$ as it already uses all 12 available registers. Using this algorithm, 160×160 bit multiplication takes 1746 clock cycles, which is an improvement when compared to the 2065 cycles for the standard Comba method (Table 3.3). Assuming that the MSP430 processor is clocked at the standard frequency of 8MHz, the result of the multiplication of two 160-bit numbers is calculated at 0.22ms. This result compares favourably with the 0.95ms reported for the same operation on a MSP430 in [118]. In [56] Guajardo *et al.* tested multiprecision multiplication of 128-bit integers on the TI MSP430x33x family of processors, which differs slightly in design from the MSP430F1611. The common features are the same instruction set and the presence of the hardware multiplier. Once again the improved hybrid method proved to be more efficient using only 1154 clock cycles against the 1425 reported in [56].

3.3.2.4 The ARM processor

Figure 3.6 (i) shows the calculation of a typical partial product on a 32-bit ARM processor using the standard Comba method. Registers `r5` and `r6` store the memory addresses of the first 32-bit digits of the multiplicand and the multiplier. Indexed addressing mode is used to find the address of a particular 32-bit digit. The `UMULL` instruction calculates the 64-bit product in registers `r8` and `r9`. These are then added-with-carry to the triple register `r2:r3:r4`.

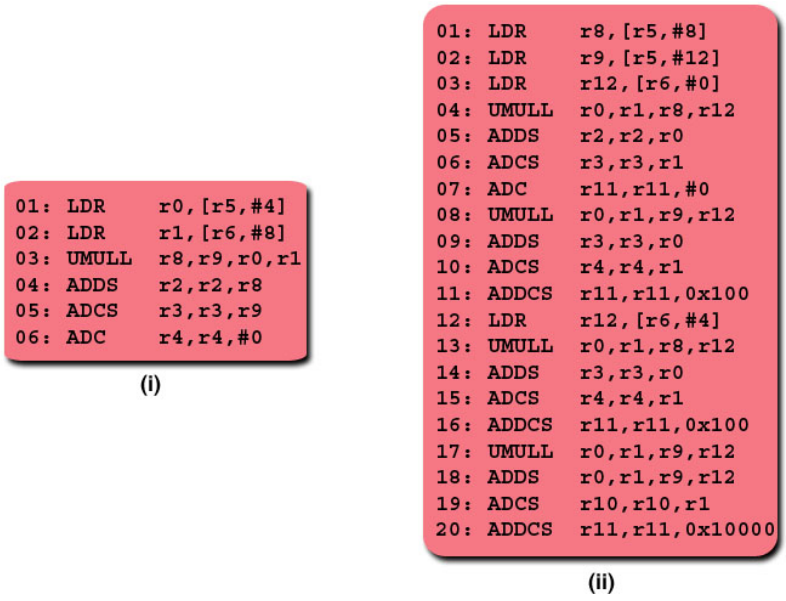

Figure 3.6: *The processing of (i) A single partial product (Comba method) and (ii) four partial products (improved hybrid method) on an ARM processor*

A naive attempt to implement the improved hybrid method on the ARM with $d = 2$ will fail, due to a lack of registers. For the multiplication algorithm 3 registers are required to hold the addresses of the operands, 4 more are required as column registers, 3 as carry-catchers, and 2 more for the row elements, plus another for the multiplier. Furthermore the multiplication instruction requires 2 registers to hold the 64-bit product of a pair of 32-bit registers, for a total of 15 registers.

This problem can be solved by exploiting the novel features of the instruction set. The basic idea is to use just one carry-catcher register instead of three. As the carry-catcher registers are used to catch and accumulate the sum of individual carry bits, they do not require the 32-bit precision of a full register. In fact for any reasonable application with big number operands of a useful size, a byte would be sufficient. This observation is used to compress the carry-catcher requirement into a single register, without incurring

any extra cost. The solution is presented in figure 3.6 (ii).

The `ADDCS` instruction (`ADD` if Carry Set) can be used to add a carry bit to a specific byte in the shared carry-catcher register `r11`. Here `r2:r3:r4:r10` are the column registers. The hybrid approach saves one load instruction on every single partial product calculation. The foot-of-column processing is a little complex, as the individual carry-catcher components must be masked and shifted before being added to the column registers. However, a register argument can be shifted at no extra cost using the ARM's barrel shifter, so the overall cost is small. The method was implemented and tested in the context of a 192×192 bit multiplication using the ARM RealView simulator [4]. The modified hybrid method required only 487 clock cycles, when compared with the 580 cycles of the basic Comba method.

3.3.3 Squaring and modular reduction

In order to save code space, the squaring of multiprecision integers can be calculated by the same algorithm that performs multiprecision multiplication. However, a dedicated routine would significantly improve the performance of the squaring operation. Optimized squaring of large integers can take advantage of the fact that partial products $a_i a_j$, $i \neq j$ occur twice during calculations. These partial products need only be calculated once, and then accumulated twice, with savings in instruction cycles. Figure 3.7 shows how the new hybrid method ($d = 2$) can be used to improve the squaring of multiprecision integers. The diagonal partial products are calculated only once and accumulated twice in registers `r0:r2:r3:r4`. Using this method the number of required single precision multiplications can be reduced by roughly half. This gives squaring an approximately 21-27% decrease in execution time when compared with a standard multiplication routine (see Table 3.4).

After the multiprecision multiplication (or squaring) in $\mathbb{F}_p$, the resulting value has the size of $2m$ bits ($m = \lceil log_2 p \rceil$) and must be reduced modulo p. In case where modulus p is not of a special form, the reduction operation can be an expensive part of modular multiplication. Generic methods like Barret or Montgomery reduction replace expensive divisions in classical reduction methods with cheaper operations. Nonetheless, they are still expensive and require roughly $t^2 + t$ single-precision multiplications, where $t = \lceil m/W \rceil$ [58].

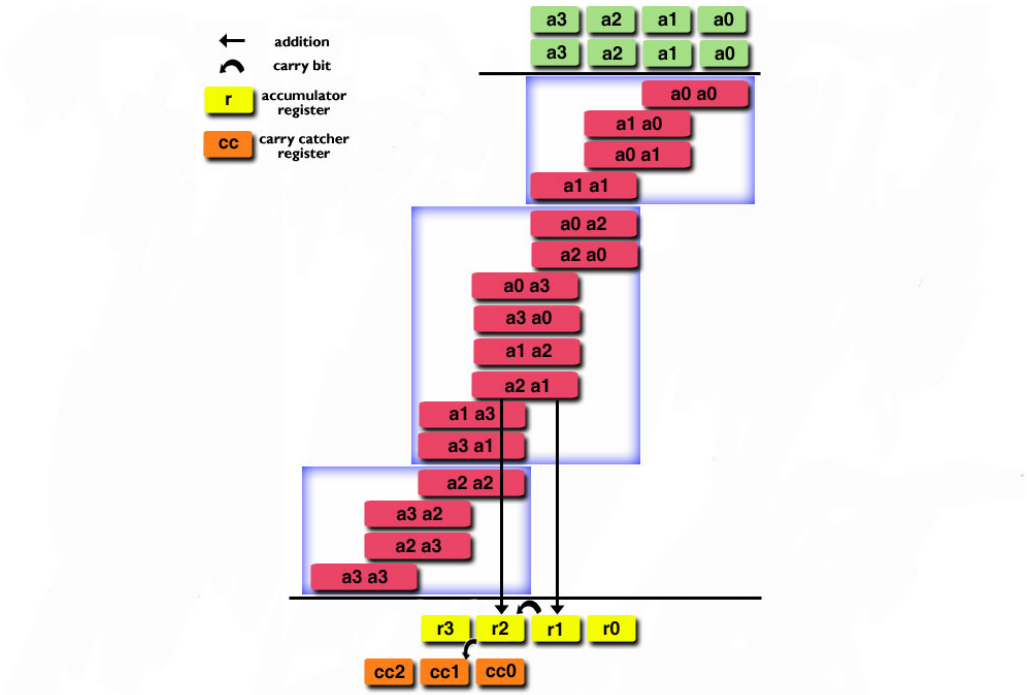

Figure 3.7: *Multiprecision integer squaring using the improved hybrid method*

Since the performance of Elliptic Curve Cryptography schemes depends heavily on the speed of field multiplication, it is beneficial to select moduli that permit fast reduction. Direct reduction techniques for moduli of special form give significant savings in execution time, especially on embedded processors with small word size. All NIST and most SECG recommended primes are chosen as pseudo-Mersenne primes to facilitate optimized reduction. These primes can be represented as $p = 2^m - \omega$ where ω is the sum of a few powers of two and $\omega \ll 2^m$. Pseudo-Mersenne primes are sparse and the powers of 2 in ω are chosen to be divisible by the specific processor word size W. Reduction of a $2m$-bit multiplication result c can be computed based on the congruence $2^m \equiv \omega$. The following example presents the reduction modulo $p = 2^{160} - 2^{112} + 2^{64} + 1$ which is optimized for a 16-bit processor.

The result of a multiprecision multiplication of two integers can be represented using the base 2^{16} with $c_i \in [2^{16} - 1]$:

$$c = c_{19}2^{304} + c_{18}2^{288} + c_{17}2^{272} + \cdots + c_2 2^{32} + c_1 2^{16} + c_0 \tag{3.4}$$

The higher powers of 2 in 3.4 can be reduced using the congruences:

$$
\begin{aligned}
2^{160} &\equiv 2^{112} - 2^{64} - 1 \pmod{p} \\
2^{176} &\equiv 2^{128} - 2^{80} - 2^{16} \pmod{p} \\
2^{192} &\equiv 2^{144} - 2^{96} - 2^{32} \pmod{p} \\
2^{208} &\equiv -2^{64} - 2^{48} - 1 \pmod{p} \\
2^{224} &\equiv -2^{80} - 2^{64} - 2^{16} \pmod{p} \\
2^{240} &\equiv -2^{96} - 2^{80} - 2^{32} \pmod{p} \\
2^{256} &\equiv -2^{112} - 2^{96} - 2^{48} \pmod{p} \\
2^{272} &\equiv -2^{128} - 2^{112} - 2^{64} \pmod{p} \\
2^{288} &\equiv -2^{144} - 2^{128} - 2^{80} \pmod{p} \\
2^{304} &\equiv -2^{144} - 2^{112} - 2^{96} + 2^{64} + 1 \pmod{p}
\end{aligned} \tag{3.5}
$$

The use of congruences from 3.5 in 3.4 give:

$$
\begin{array}{lllllllllll}
c \equiv & -c_{19}2^{144} & & -c_{19}2^{112} & -c_{19}2^{96} & & +c_{19}2^{64} & & & & +c_{19} \\
& -c_{18}2^{144} & -c_{18}2^{128} & & & -c_{18}2^{80} & & & & & \\
& & -c_{17}2^{128} & -c_{17}2^{112} & & & -c_{17}2^{64} & & & & \\
& & & -c_{16}2^{112} & -c_{16}2^{96} & & & -c_{16}2^{48} & & & \\
& & & & -c_{15}2^{96} & -c_{15}2^{80} & & & -c_{15}2^{32} & & \\
& & & & & -c_{14}2^{80} & -c_{14}2^{64} & & & -c_{14}2^{16} & \\
& & & & & & -c_{13}2^{64} & -c_{13}2^{48} & & & -c_{13} \\
& +c_{12}2^{144} & & & -c_{12}2^{96} & & & & -c_{12}2^{32} & & \\
& & +c_{11}2^{128} & & & -c_{11}2^{80} & & & & -c_{11}2^{16} & \\
& & & +c_{10}2^{112} & & & -c_{10}2^{64} & & & & -c_{10} \\
& +c_9 2^{144} & +c_8 2^{128} & +c_7 2^{112} & +c_6 2^{96} & +c_5 2^{80} & +c_4 2^{64} & +c_3 2^{48} & +c_2 2^{32} & +c_1 2^{16} & +c_0 \\
& \pmod{p} & & & & & & & & &
\end{array} \tag{3.6}
$$

From the equation 3.6, c modulo p can be obtained by five additions (subtractions) of 160-bit integers and repeated subtractions of p until the result is less than p. Equation 3.6 leads to Algorithm 1 which can be efficiently implemented on a 16-bit embedded processor. The same method can be also used to derive the reduction algorithm for an 8-

bit CPU. Table 3.4 presents timings for the fast reduction modulo $p = 2^{160} - 2^{112} + 2^{64} + 1$ on ATmega128 and MSP430.

Algorithm 1 Fast reduction modulo $p = 2^{160} - 2^{112} + 2^{64} + 1$

INPUT: An integer $c = (c_{19}, c_{18}, c_{17}, \ldots, c_2, c_1, c_0)$ in base 2^{16} with $0 \leq c < p^2$.
OUTPUT: $c \mod p$

Form 160-bit integers:

$k_1 = (c_9, c_8, c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0)$,
$k_2 = (c_{12}, c_{11}, c_{10}, 0, 0, c_{19}, 0, 0, 0, c_{19})$,
$k_3 = (0, c_{17}, c_{17}, c_{12}, c_{18}, c_{17}, 0, 0, 0, 0)$,
$k_4 = (c_{18}, c_{18}, c_{16}, c_{16}, c_{11}, c_{10}, c_{16}, c_{12}, c_{11}, c_{10})$,
$k_5 = (0, 0, 0, c_{15}, c_{15}, c_{13}, c_{13}, c_{15}, 0, c_{13})$,
$k_6 = (c_{19}, 0, c_{19}, c_{19}, c_{14}, c_{14}, 0, 0, c_{14}, 0)$.

return $(k_1 + k_2 - k_3 - k_4 - k_5 - k_6 \mod p)$.

Table 3.4: *Timings in instruction cycles for 160-bit integer squaring and modular reduction*

Hardware platform	**Atmega128**	**MSP430**
Hybrid multiplication	2651 (d=4)	1746 (d=2)
Squaring	2193 (d=4)	1373 (d=2)
Modular reduction	1228	990

3.4 Efficient binary field arithmetic for low-end processors

Finite fields of characteristic-two are usually the main alternative to prime fields when implementing Elliptic Curve Cryptography systems. Binary fields are attractive especially for hardware implementations since the operations in $\mathbb{F}_{2^m}$ involve only shifts and bitwise addition modulo 2. The simplicity of basic operations is also attractive for software implementations on general-purpose processors.

Prime fields and binary fields have their own pros and cons when it comes to efficient implementation on low-end processors. The most time-critical operation in $\mathbb{F}_{2^m}$ binary polynomial multiplication - calculates only a few bits at a time and can be very slow without proper optimizations. On the other hand, the arithmetic in prime fields is typically more difficult to implement efficiently, due in part to the propagation of carry bits. Nonetheless, prime fields are usually preferred due to better hardware support on embedded processors.

Most of the low-end CPUs support multiplication of wordsize integers in hardware. This feature significantly accelerates multiprecision multiplication in $\mathbb{F}_p$. Arithmetic on

binary fields is not natively supported in the Instruction Set of typical embedded processors. None of the CPUs used in sensor networks support multiplication in the binary field. Therefore efficient implementation of polynomial multiplication remains as a challenging task.

The elements of a finite field $\mathbb{F}_{2^m}$ can be represented by binary polynomials of degree less than m, where m is a prime number. For security reasons Elliptic Curve Cryptography systems are defined over finite fields where m is usually greater than 160 bits. In software, the coefficients of a binary polynomial may be stored in an array of t W-bit words. The $s = tW - m$ highest order bits in the last word of the array remain unused (always set to 0).

The addition of field elements is performed bitwise, and is very fast requiring only t word operations (XORs) which are supported by all computer architectures. Addition in the binary field is analogous to multiprecision integer addition, without the carries. Other arithmetic operations (beside multiplication) include inversion, squaring, reduction and calculation of square roots.

3.4.1 Binary polynomial multiplication

The simplest way to perform polynomial multiplication is to use the basic schoolbook multiplication. This method for two polynomials $a(z)$ and $b(z)$ of degree $m - 1$ is illustrated in Figure 3.8. The algorithm requires $m - 1$ left shifts of $a(z)$ and a modulo 2 addition of $a(z)$ to the accumulator for every non-zero coefficient of $b(z)$. This "shift-and-add" method generates only one bit of the result at each step. Its implementation in software is usually very slow due to many vector shifts. Large number of these operations makes the implementation infeasible especially on small 8-bit processors where shifts can only by performed one bit at a time. The shift-and-add method is more suitable for hardware where a vector shift can be performed in one clock cycle. Embedded processors require faster methods for field multiplication. The following sections present a description of an optimized hierarchical technique for binary polynomial multiplication. This method combines different multiplication techniques and achieves superior results on constrained WSN platforms.

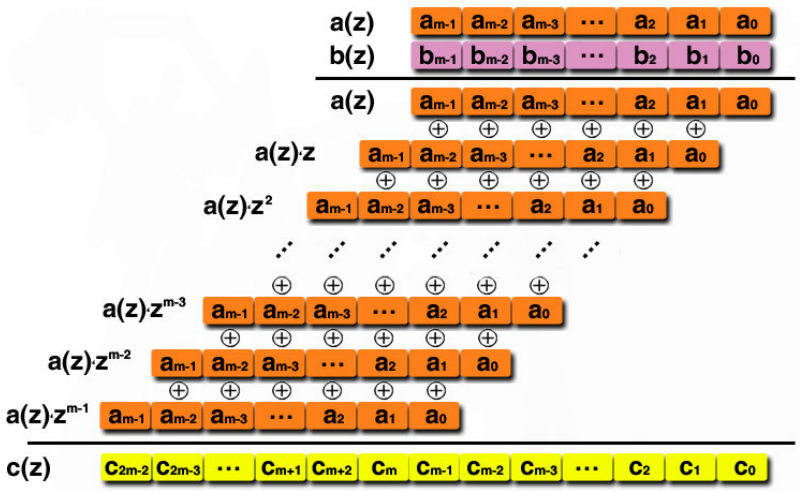

Figure 3.8: *Binary polynomial multiplication using the basic schoolbook method*

3.4.1.1 Optimized hierarchical method for polynomial multiplication

The optimized hierarchical technique for binary polynomial multiplication consists of two stages:

- the first step is the application of the Karatsuba-Ofman technique [74] which divides large polynomials into several smaller polynomials;
- the second stage performs the actual multiplication of binary polynomials using the comb method [82].

For embedded processors with small word sizes the hierarchical multiplication technique is attractive for large fields. In case of smaller values of m the second stage of the technique can be applied directly in order to avoid unnecessary overhead. The execution time of the whole method depends on the efficiency of the comb multiplication. This algorithm should be implemented in assembly language to provide maximum performance on a given hardware platform.

The multiplication of two binary polynomials $a(z)$ and $b(z)$ of degree $m-1$ results in a polynomial of degree at most $2m-2$. In the first step, the Karatsuba-Ofman method divides large $m-1$ polynomials into several polynomials with fewer coefficients. This divide-and-conquer method can be directly adapted for binary polynomials. In the case of a division into two parts (depth $d=2$), the polynomial multiplication $c(z)=a(z)b(z)$ can be represented as:

$$a(z) = A_1(z)x^n + A_2(z) \qquad b(z) = B_1(z)z^n + B_2(z) \tag{3.7}$$

$$c(z) = A_1B_1z^{2n} + ((A_1 + A_2)(B_1 + B_2) + A_2B_2 + A_1B_1)z^n + A_2B_2 \tag{3.8}$$

where $n = \lceil \frac{m}{2} \rceil$ and A_1, A_2, B_1, B_2 are binary polynomials of degree less than n.

For $d = 2$ the above technique substitutes one full multiplication with three half-size multiplications. One multiplication is saved at the cost of several extra additions which are very fast in binary fields. The most time-consuming part of the equation 3.8 is the computation of the sub-products A_1B_1, A_2B_2 and $(A_1 + A_2)(B_1 + B_2)$. In the second stage of the hierarchical multiplication these products are calculated using the optimized comb method with a window width of w-bits (Algorithm 2 based on [82]).

Algorithm 2 Comb method with windows of width w

INPUT: Binary polynomials $A(z)$ and $B(z)$ of degree at most $n - 1$
OUTPUT: $C(z) = A(z) \cdot B(z)$.

1: Compute $U[i] = U(z) \cdot B(z)$ for all polynomials $U(z)$ of degree at most $w - 1$
2: $C \leftarrow 0$
3: **for** $k \leftarrow (W/w) - 1$ **downto** 0 **do**
4: **for** $j \leftarrow 0$ **to** $t - 1$ **do**
5: $C \leftarrow C + U[i] \cdot z^{Wj}$ where
6: $i = (a_{wk+w-1+Wj}, \ldots, a_{wk+1+Wj}, a_{wk+Wj})_{10}$
7: **end for**
8: If $k \neq 0$ then $C \leftarrow C \cdot z^w$
9: **end for**
10: **return** C

For software implementation Algorithm 2 is faster than the basic shift-and-add method since it requires fewer vector shifts (multiplications by z). Additionally it processes w coefficients of $a(z)$ at a time by pre-computing 2^w polynomials. In line number 5 the correct address of the pre-computed polynomial is calculated based on the values of w consecutive bits in a. After this selection the polynomial can be added to specific words of the accumulator. Figure 3.9 shows the order in which the bits of a are processed in Algorithm 2. In the first loop w bits of a are processed in columns from top to bottom. For the consecutive loops the window is shifted w bits from left-to-right.

The size of the window w decides the amount of pre-computed data. The comb method with window size of w-bits requires pre-computation of $t \cdot 2^w$ computer words. On embedded processors, the choice of w is very important and should provide a reasonable trade-

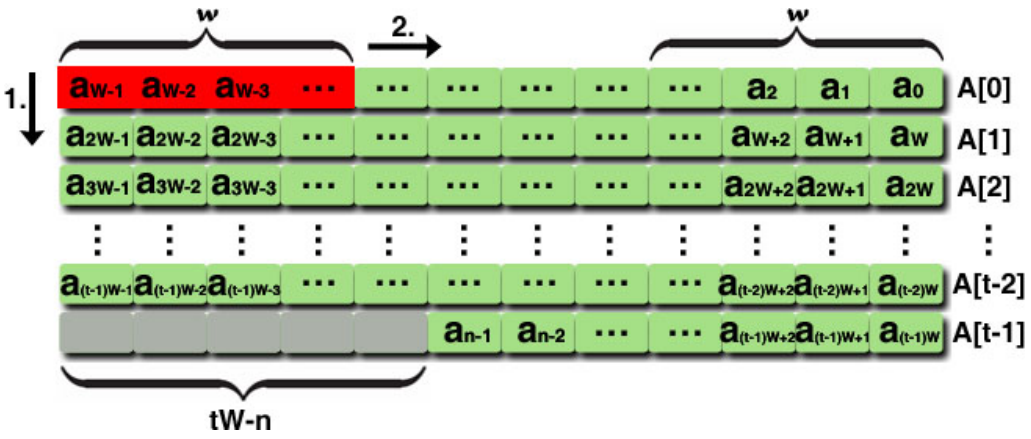

Figure 3.9: *The order of bits processing in the comb method with window width w*

off between speed and storage. When the window is too small the performance of the algorithm does not improve much. If the window is too big the improvement in execution time is overshadowed by a large memory overhead. For finite fields of practical interest, a window size of four bits provide optimal performance on low-end processors. The key to efficient implementation of the comb method lies also in the proper handling of the accumulator C. The best performance can be achieved by using the maximum number of registers to minimize unnecessary store/load operations (as in the hybrid multiplication). The implementation is of course processor specific and is discussed in the context of three standard WSN processors. The implementations were performed over $\mathbb{F}_{2^{271}}$ - a binary field that can be used for pairing-based cryptography. On all three platforms, the assembly language optimization of the hierarchical multiplication achieved significant performance improvement (60-75%) over the standard C implementation.

3.4.1.2 The ATmega128 CPU

The optimized hierarchical multiplication of binary polynomials of degree less than 271 can be performed in two steps and results in a polynomial of degree at most 540. According to formula 3.8, the multiplication can be rewritten as:

$$c(z) = A_1B_1z^{272} + ((A_1 + A_2)(B_1 + B_2) + A_2B_2 + A_1B_1)z^{136} + A_2B_2 \qquad (3.9)$$

The sub-products A_1B_1, A_2B_2 and $(A_1 + A_2)(B_1 + B_2)$ can be efficiently calculated with a comb multiplication with window width $w = 4$ (Algorithm 2). On an 8-bit processor, the computations can be divided into two loops and the multiplication by z^{Wj} can

be replaced by an address computation. In the first loop, all pre-computed polynomials $U[i] \cdot z^{8j}$, $0 \leq j \leq t-1$ are summed up in the polynomial $C(z)$. The correct address i of the polynomial can be calculated by converting the four bits $(a_{7+8j}, a_{6+8j}, a_{5+8j}, a_{4+8j})$ into a decimal value. After the first loop $C(z)$ is multiplied by z^4 (shifted left by 4 bits) and the polynomials $U[i] \cdot z^{8j}$, $0 \leq j \leq t-1$, are added to $C(z)$ again. This time $i = (a_{3+8j}, a_{2+8j}, a_{1+8j}, a_{8j})_{10}$. The multiples $U(z) \cdot B(z)$, $\deg(U) < 4$ are pre-computed and stored in a lookup table of 288 bytes.

The most important problem, which impacts significantly the performance of the multiplication, is the handling of the polynomial $C(z)$. Clearly, the temporary result (36 bytes) cannot be kept in the CPU registers for the entire duration of the multiplication. However, it is important to notice that nearly all registers that are affected by the addition of a particular polynomial $U(z) \cdot B(z)$, are altered again in subsequent iterations of the loop. Hand-crafted assembly implementation can take advantage of this fact to use the CPU registers in an optimal way. The coefficients of $C(z)$ can be rotated through the available registers of the microcontroller, to minimize the number of memory accesses.

Figure 3.10 shows the content of registers `R0–R17` throughout the first loop of the comb algorithm. Bytes $U_0 - U_{17}$ denote the 18 bytes of the particular pre-computed polynomial $U[i]$, which corresponds to bits $(a_{7+8j}, a_{6+8j}, a_{5+8j}, a_{4+8j})$. The bytes in yellow are loaded into registers and bytes in light blue are added to registers in each iteration of the loop (j). The bytes in red mark the data which is stored in RAM after the addition.

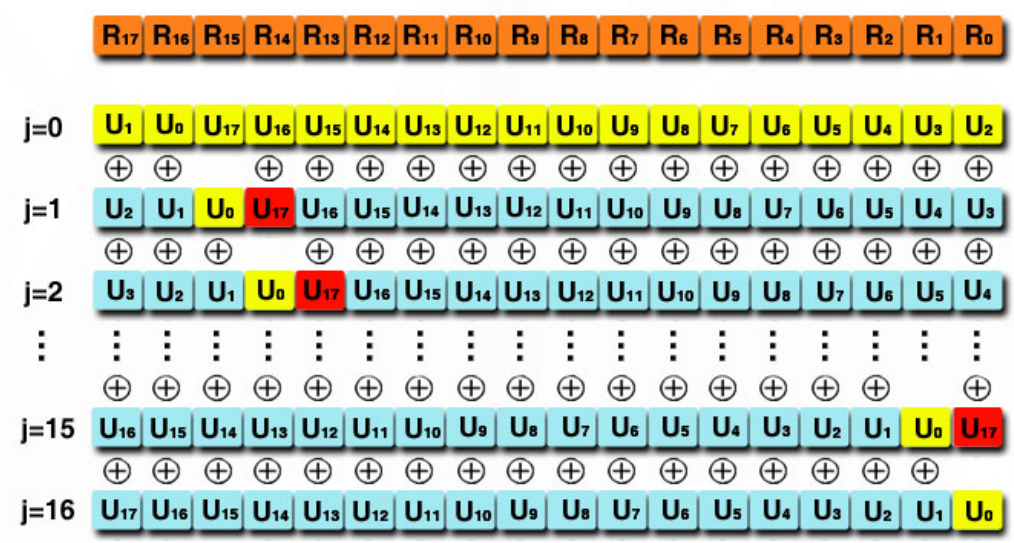

Figure 3.10: *First part of the Comb multiplication on the ATmega128 processor*

Table 3.5: *Instruction counts for polynomial multiplication in $\mathbb{F}_{2^{271}}$ on ATmega128*

Algorithm	Stage	Load	Store	XOR	Shift	Other	Total
Comb $w = 4$	Precomp	190	288	126	126	77	807
	Loop 1	391	16	272	0	143	822
	Shift by 4 bits	18	19	0	148	20	205
	Loop 2	407	34	306	0	158	905
	Total	1006	357	704	274	398	2739
	Clock cycles	1898	714	704	274	457	4047
Karatsuba	Total	258	102	136	0	52	548
	Clock cycles	496	204	136	0	91	927
Hierarchical (Karatsuba + $3\times$ **comb)**	Total	3276	1173	2248	822	1246	8765
	Clock cycles	6190	2346	2248	822	1462	**13068**

The rotation illustrated in Figure 3.10 minimizes the number of copy instructions, but can only be implemented when the code is unrolled. Table 3.5 lists the instruction counts for the Comb algorithm ($w = 4$) and the Karatsuba technique. It shows also the total number of instructions that are necessary for the optimized hierarchical multiplication in $\mathbb{F}_{2^{271}}$. The precomputation phase, the two loops as well as the multiplication of the polynomial by z^4 is separated to provide more details on the implementation.

Using the Karatsuba method the number of smaller field multiplications is reduced to three. This allows us to save more than 3000 clock cycles (20%) on a single multiplication in $\mathbb{F}_{2^{271}}$. For smaller fields the comb technique should be used directly to minimize the overhead. Direct application of this method is more efficient when the whole polynomial $U(z) \cdot B(z)$ fits in the available registers of the CPU. For the ATmega128 with 32 registers this happens when $m \leq 184$. The optimized hierarchical multiplication is recommended for larger values of m. The hierarchical approach also saves code space by reusing the fragments responsible for multiplication in the smaller field.

The binary polynomial multiplication on embedded processors was implemented also in the work of Seo, Han and Hong [110]. The authors reported that the modular multiplication in $\mathbb{F}_{2^{163}}$ took around 2.9ms on the ATmega128 CPU. This corresponds to 21410 clock cycles and is around 40% slower than the optimized hierarchical method (see Table 3.5). The improvement in performance is significant, especially given that the finite field used in [110] is a lot smaller than that used here.

3.4.1.3 The MSP430 microcontroller

On a 16-bit architecture, a single step method for binary polynomial multiplication might be considered. The comb method with window width of 8 instead of 4 would be much faster because shifting by a byte can be performed more efficiently on a 16-bit CPU. This approach, however, is not practical on sensor network devices due to very large memory requirements (both in RAM and ROM). Therefore, the implementation on the MSP430 should use a similar two stage technique as for the ATmega128. The optimized hierarchical method can significantly decrease the memory footprint by using the "divide-and-conquer" technique with consecutive calls to the same multiplication routine.

Efficient implementation of the comb method on the MSP430 is more difficult than on the ATmega128 CPU. The main problem in the MSP430 case is that there are only 12 available registers instead of 32. In the field $\mathbb{F}_{2^{271}}$ one 17 word polynomial is divided into two smaller 9 word polynomials in order to apply the Karatsuba method. An extra last word containing all zeros for the second polynomial is added to maintain the symmetry. The sub-products A_1B_1, A_2B_2 and $(A_1 + A_2)(B_1 + B_2)$ can be then calculated using the comb multiplication with window width of 4. The sub-product calculations require a pre-computed lookup table of 320 bytes. The use of 16-bit words and 4-bit scanning in the comb algorithm requires 4 loops. The polynomial C(z) is multiplied by z^4 in between the loop iterations.

Listing 3.11 shows one iteration of the first loop implemented in assembly language on the MSP430 CPU. The code is optimized to take advantage of all 12 available registers (`r4-r15`). Register `r14` holds 16 coefficients of the polynomial $A(z)$. The first loop scans the 4 most significant bits in every word of A. In order to derive the required 4 bits the 2 bytes of A are swaped and a `0xF0` mask is applied. The result is rotated right by 3 bits (to get an even value) and added to the address of the table that holds the pre-computed polynomials $U(z) \cdot B(z)$. The combined value in `r13` gives the correct address of the polynomial $U[i]$. Then the polynomial is added to the 9 registers of the accumulator `r5-r12:r15`. One word from the accumulator is stored in the memory and one new word is loaded for the next loop iteration. The rotation of words in the registers in subsequent loop iterations is performed in a similar manner as in Figure 3.10.

Table 3.6 lists the instruction counts for all the operations performed during the polynomial multiplication in $\mathbb{F}_{2^{271}}$. The values were obtained using the IAR Embedded Work-

```
01: MOV   @r14+,r13
02: SWPB  r13
03: AND   #0x0F0,r13
04: RRA   r13
05: RRA   r13
06: RRA   r13
07: ADD   #U[0]_tab,r13
08: MOV   @r13,r13
09: XOR   @r13+,r5
10: XOR   @r13+,r6
11: XOR   @r13+,r7
12: XOR   @r13+,r8
13: XOR   @r13+,r9
14: XOR   @r13+,r10
15: XOR   @r13+,r11
16: XOR   @r13+,r12
17: XOR   @r13+,r15
18: MOV   @r13+,r4
19: MOV   r5,c_tab+2
```

Figure 3.11: *Implementation of the comb algorithm on the MSP430 processor*

bench simulator for the MSP430 (v4.10). The instruction set does not include load and store operations, but instructions that move the data from the memory to registers and vice versa can be treated as such. The instruction counts for each loop are given separately as they slightly differ from each other. The application of the Karatsuba technique saves around 20% of the total clock cycles. The implementation of the optimized hierarchical multiplication in assembly language improves the execution time by 70% in comparison with the standard C implementation (optimization flag -O0). Assuming 8 MHz clock speed on the MSP430 the complete multiplication of binary polynomials in $\mathbb{F}_{2^{271}}$ takes only 1.18 ms.

3.4.1.4 The ARM processor

The implementation of binary polynomials multiplication in $\mathbb{F}_{2^{271}}$ is straightforward on a 32-bit ARM processor. The hierarchical method of multiplication is not necessary in this case. The whole polynomial fits in 9 registers, so using Karatsuba here would only result in additional overhead. The comb method can be applied directly, although the size of the window should be chosen with care. A window of size 4 is not appropriate because it requires shifts by 4 bits which are not efficient on a 32-bit architecture. The number of loop iterations in Algorithm 2 depend also on the window size. When $W = 32$ and $w = 4$ the main loop has eight iterations. This number however can be reduced by using $w = 8$

Table 3.6: *Polynomial multiplication in $\mathbb{F}_{2^{271}}$ on the MSP430 CPU*

Algorithm	Stage	Load	Store	XOR	Shift	Other	Total
Comb $w = 4$	Precomputation	21	160	70	70	22	343
	Loop 1	36	10	72	27	27	172
	$3\times$ Shift by 4 bits	10	10	0	84	2	106
	Loop 2	26	10	90	9	27	162
	Loop 3	26	10	90	27	18	171
	Loop 4	26	18	90	9	23	166
	Total	165	238	412	394	123	1332
	Clock cycles	384	952	824	394	317	2871
Karatsuba	Total	38	72	70	0	12	192
	Clock cycles	118	398	282	0	41	839
Hierarchical (Karatsuba + $3\times$ **comb)**	Total	533	786	1306	1182	381	4188
	Clock cycles	1270	3254	2754	1182	992	**9452**

with 2-bit scanning. This modification achieves the same amount of pre-computation as in the case of a window size of 4 (576 bytes). The 2-bit scanning decreases also the number of necessary loop iterations from 8 to 4 and the number of polynomial shifts of $C(z)$ from 7 to 3. These savings, however, are made at a cost of extra polynomial additions and table lookups in each iteration of the four loops.

The ARM processor has 14 available registers. The temporary result is held in nine registers while the remaining five are used for storing operand addresses and choosing the right lookup table element. All four loops can be unrolled to achieve significant savings in execution time. Table 3.7 lists the instructions for the comb multiplication in $\mathbb{F}_{2^{271}}$. The ARM instruction set does not specify shifts explicitly, but these operations can be performed as a part of ordinary instructions (without additional cost). Also, shifting the temporary result vector by 8 bits instead of 4 optimizes this operation for a 32-bit architecture. This can be efficiently implemented without any actual shifts. One loop can be used with 2 pointers that simply load and store appropriate bytes. The multiple load/store instructions and 32-bit processing makes the ARM processor far more efficient than the constrained 8 and 16-bit platforms. Full polynomial multiplication in $\mathbb{F}_{2^{271}}$ takes only 4706 clock cycles when implemented in assembly language.

Table 3.7: *Polynomial multiplication in $\mathbb{F}_{2^{271}}$ on an ARM processor*

Algorithm	Stage	Load	Store	XOR	Shift	Other	Total
Comb $w = 8$	Precomputation	28	16	9	117	14	184
	$4\times$ Loop	370	9	320	0	218	917
	$3\times$ Shift by 8 bits	67	68	0	0	137	272
	Total	1709	256	1289	117	1297	4668

3.4.2 Squaring and reduction of binary polynomials

One of the advantages of binary fields over prime fields is much more efficient squaring of field elements. Squaring of binary polynomials is a linear operation which is much faster than multiplying two arbitrary polynomials. In the case when $a(z) = \sum_{i=0}^{m-1} a_i z^i$ then $a(z)^2 = \sum_{i=0}^{m-1} a_i z^{2i}$. The binary representation of $a(z)^2$ is obtained by spreading the coefficients of $a(z)$ and inserting zeroes in between (see Figure 3.12).

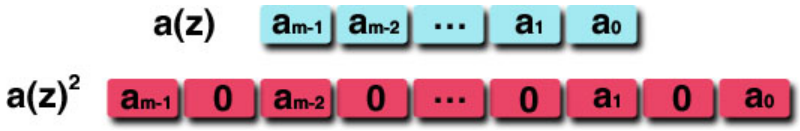

Figure 3.12: *Binary polynomial squaring*

The spreading of coefficients can be optimized using a lookup table for small polynomials. On constrained 8-bit processors (like the ATmega128), the lookup table should map four bits to one byte. Such a table requires only 16 bytes of RAM. Other embedded processors with more memory and 16 or 32-bit processing can use larger lookup tables. A typical solution uses a 512 bytes of pre-computation converting 8-bit polynomials into their expanded 16-bit counterparts. Simple formula for squaring of binary polynomials makes this operation around 8 to 10 times faster than multiplication. Table 3.8 compares the results of modular multiplication and squaring in $\mathbb{F}_{2^{271}}$ on three embedded processors.

Table 3.8: *Modular multiplication and squaring in $\mathbb{F}_{2^{271}}$ on embedded processors*

Hardware platform	ATmega128	MSP430	PXA271
Hierarchical multiplication (assembly)	13557	10116	4926
Squaring (assembly)	1581	1363	499
Hierarchical multiplication (C)	66271	40666	13183
Squaring (C)	4711	3667	2375

The timings presented in Table 3.8 include the reduction step which must be applied to limit the number of coefficients from $2m-2$ to $m-1$. One of the advantages of using methods based on $\mathbb{F}_{2^m}$ fields is that squarings are potentially so fast. But if the reduction algorithm is slow, this advantage cannot be fully utilized. Reduction is performed modulo the irreducible polynomial $f(z)$. The generic reduction methods reduce the result of multiplication $c(z)$ one bit at a time and can be slow, especially on constrained embedded processors.

The reduction operation can be accelerated if $f(z)$ is a trinomial, or a pentanomial. For the field $\mathbb{F}_{2^m}$ it is in practise always possible to choose as an irreducible polynomial either a trinomial $z^m + z^a + 1$ or a pentanomial $z^m + z^a + z^b + z^c + 1$. In both cases the optimal reduction algorithm is linear and fast because the reduction of $c(z)$ modulo $f(z)$ can be performed one word at a time. The following example presents a fast reduction method in $\mathbb{F}_{2^{271}}$ which is optimized for a 16-bit embedded processor. The method is similar to the reduction technique for the NIST parameters proposed in [44].

In the finite field $\mathbb{F}_{2^{271}}$, the irreducible polynomial can be chosen as a trinomial: $f(z) = z^{271} + z^{201} + 1$. The multiplication (or squaring) of polynomials in $\mathbb{F}_{2^{271}}$ results in a polynomial $c(z)$ of degree less than 540. In the binary form, $c(z)$ can be represented as an array of words $C = (C[33], C[32], \ldots, C[1], C[0])$. When reducing the word $C[20]$ which represents the polynomial $c_{335}z^{335} + \cdots + c_{321}z^{321} + c_{320}z^{320}$ the following congruences occur:

$$\begin{aligned}
z^{320} &\equiv z^{250} + z^{49} \quad (mod\, f(z)) \\
z^{321} &\equiv z^{251} + z^{50} \quad (mod\, f(z)) \\
&\vdots \quad \vdots \quad \vdots \quad \vdots \qquad \vdots \\
z^{334} &\equiv z^{264} + z^{63} \quad (mod\, f(z)) \\
z^{335} &\equiv z^{265} + z^{64} \quad (mod\, f(z))
\end{aligned} \tag{3.10}$$

By considering the two columns on the right side of congruences 3.10, the reduction of $C[20]$ can be performed by adding $C[20]$ to bits $(c_{64}, \ldots, c_{50}, c_{49})$ and $(c_{265}, \ldots, c_{251}, c_{250})$ of C. This observation leads to a fast reduction algorithm in $\mathbb{F}_{2^{271}}$ (see Algorithm 3).

Algorithm 3 can be efficiently implemented on a 16-bit embedded processor such as the MSP430. For optimal performance the code should be written in assembly language

Algorithm 3 Fast reduction modulo $f(z) = z^{271} + z^{201} + 1$

INPUT: A binary polynomial $c(z)$ represented as 34 16-bit words C[.]
OUTPUT: $c(z) \mod f(z)$, represented as 17 16-bit words.

1: **for** $i \leftarrow 33$ **downto** 17 **do**
2: $\quad S \leftarrow C[i]$
3: $\quad C[i-17] \leftarrow C[i-17] \oplus (S \ll 1)$
4: $\quad C[i-16] \leftarrow C[i-16] \oplus (S \gg 15)$
5: $\quad C[i-5] \leftarrow C[i-5] \oplus (S \ll 10)$
6: $\quad C[i-4] \leftarrow C[i-4] \oplus (S \gg 6)$
7: **end for**
8: $S \leftarrow C[16] \gg 15$
9: $C[0] \leftarrow C[0] \oplus S$
10: $C[12] \leftarrow C[12] \oplus (S \ll 9)$
11: $C[16] \leftarrow C[16]$ & 0x7FFF
12: **return** $(C[16], C[15], ..., C[1], C[0])$

with all loops unrolled. Listing 3.13 shows the first iteration of the loop (line 1 in Algorithm 3) implemented on the MSP430 processor. Register `r13` holds the address of C, `r6` is equal to 0 and `r7` holds a bit mask `0xFF00`. During the first loop iteration, `r4` and `r5` are loaded with the most significant word of C. One bit rotation to the left through carry is used to derive the necessary words $S \ll 1$ and $S \gg 15$, which are then added to C. The shifts $S \ll 10$ and $S \gg 6$ can be efficiently implemented with a byte swap, bit mask application and 2 rotations left through the carry bit. After the addition of `r4` and `r5` to C, register `r6` is cleared for the next loop iteration. The whole reduction in $\mathbb{F}_{2^{271}}$ takes 664 clock cycles on the MSP430 CPU.

```
01: MOV     66(r13),r4
02: MOV     r4,r5
03: RLA     r4
04: ADC     r6
05: XOR     r4,32(r13)
06: XOR     r6,34(r13)
07: SWPB    r5
08: MOV.B   r5,r4
09: AND     r7,r5
10: RLA     r5
11: RLC     r4
12: RLA     r5
13: RLC     r4
14: XOR     r5,56(r13)
15: XOR     r4,58(r13)
16: CLR     r6
```

Figure 3.13: *A reduction of a single word in $\mathbb{F}_{2^{271}}$ on the MSP430*

Similar reduction methods to those in Algorithm 3 can be also developed for other

hardware architectures. Each reduction algorithm should be optimized for a specific irreducible polynomial and the wordsize of a given processor. On small 8-bit and 16-bit processors, the irreducible polynomial should be chosen carefully to minimize the required word shifts in the reduction algorithm. For example, on the ATmega128 CPU, a shift by 1 bit is always faster than a shift by 6 bits. In an ideal case, the irreducible trinomial should have $m - a \equiv 0 \mod W$ and pentanomial $m - a \equiv m - b \equiv m - c \equiv 0 \mod W$ with all m, a, b, c being odd. This minimizes shift operations on a processor with wordsize W. The above conditions, however, cannot be met on all occasions. In some cases an irreducible polynomial which is optimal for one architecture may not be as good for another. For example $(m - a)$ may be a multiple of 16, but not of 32. On some processors a pentanomial may also be a better choice than the trinomial. For Algorithm 3 one may consider using the pentanomial $z^{271} + z^{207} + z^{175} + z^{111} + 1$ instead of the trinomial $z^{271} + z^{201} + 1$. This would eliminate the shifts $S \ll 10$ and $S \gg 6$ at a cost of an extra XOR instruction (a saving of 51 clock cycles per reduction operation on the MSP430). More guidance on the optimal irreducible polynomials in $\mathbb{F}_{2^m}$ for different hardware architectures can be found in [108].

3.4.3 Square root computation

Recently there have emerged applications in which it is important to calculate the square roots of field elements. Efficient square rooting is especially important in case of pairing-based cryptography over binary fields [7]. Fast calculation of square roots in $\mathbb{F}_{2^m}$ requires that all exponents in the irreducible polynomial (a, b and c) are odd [45]. Such irreducible polynomials are easy to find, but unfortunately most of the standard polynomials (e.g. irreducible polynomials recommended by NIST) are not of this form.

As pointed out by Fong *et al.* [45], if using an irreducible polynomial with all odd exponents, for example the trinomial $z^m + z^a + 1$, the field square root can be expressed as:

$$\sqrt{a(z)} = A_{even} + (z^{(m+1)/2} + z^{(a+1)/2}) \cdot A_{odd} \tag{3.11}$$

where A_{even} and A_{odd} are the even and odd indexed bits of $a(z)$ concatenated into a half-sized bit arrays. The multiplication by the term $z^{(m+1)/2}$ is simple and is performed by storing A_{odd} as the higher order bits of the final result. The multiplication of A_{odd} by the

rest of the terms in $f(z)$ can be performed with some xor operations and shifts when necessary. The amount of calculations required for this step depends on the particular irreducible polynomial. The resulting polynomial $c(z) = \sqrt{a(z)}$ is of a degree less than $m - 1$, therefore the reduction step is not necessary.

In order to extract the odd and even coefficients of $a(z)$, two lookup tables are required. Both tables `evens[.]` and `odds[.]` contain reduced polynomials that are formed by extracting appropriate bits. For example the byte 11111000 needs to be split into even bits 1100 and odd bits 1110, so the tables should contain `evens[248]=12` and `odds[248]=14`. This requires two tables with 256 bytes each. The pre-computation of 512 bytes may not be significant for a 32-bit processors, but it certainly is significant on small embedded CPUs, and should be minimized if possible. Notice that tables `evens[.]` and `odds[.]` actually contain only numbers from 0-15 and can be compressed to 16 bytes each. This can be achieved at the cost of some extra processing when establishing the indices of the tables. Listing 3.14 presents the C code for calculating a field square root using $f(z) = z^{271} + z^{201} + 1$ on an 8-bit processor (such as ATmega128). The input is an array A[.] of 34 bytes and the square root is an array C[.] also of 34 bytes (and initially cleared to zero).

```
01: for (i=0;i<34;i++)
02: {
03:   n=i/2;
04:   w=A[i];                                   /* get byte i of A[] */
05:
06:   we=evens[((w&0x5)+((w&0x50)>>3))];       /* extract 4 even bits */
07:   wo=odds[((w&0xA)+((w&0xA0)>>5))];        /* extract 4 odd bits */
08:   i++;
09:   w=A[i];                                   /* get next byte of A[] */
10:   we|=evens[((w&0x5)+((w&0x50)>>3))]<<4;   /* create 8 bits of evens */
11:   wo|=odds[((w&0xA)+((w&0xA0)>>5))]<<4;    /* create 8 bits of odds */
12:
13:   C[n]^=we;                                 /* xor evens */
14:   C[n+17]=wo;                               /* store odds */
15:   C[n+13]^=wo>>3;                           /* multiply by z^(a+1)/2 */
16:   C[n+12]^=wo<<5;
17: }
```

Figure 3.14: *Square root calculation in $\mathbb{F}_{2^{271}}$ on ATmega128*

The calculation of square roots in $\mathbb{F}_{2^m}$ is most efficient when implemented in assembly language and optimized for a certain reduction polynomial. Table 3.9 shows the clock cycles for square root implementation on three embedded processors. The standard C im-

plementation (Listing 3.14) is on average 83% slower than the hand-crafted assembly implementation. Optimization for speed (flag -O3) reduces the difference by only 10% and proves that the overhead generated by the C-compilers affects the runtime negatively. The overhead is not crucial on an ARM processor where the overall cycle count is low, but it definitely is crucial on constrained 8 and 16-bit platforms.

Table 3.9: *Timings for square root calculation in $\mathbb{F}_{2^{271}}$ on embedded processors*

Hardware platform	**ATmega128**	**MSP430**	**PXA271**
Square root (assembly)	1730	1644	546
Square root (C)	12021	11212	2496
Decrease	86%	85%	78%

3.5 Extension field arithmetic

The arithmetic in the extension fields is not required for ordinary Elliptic Curve Cryptography operations but is necessary in case of pairing-based cryptography. When using pairing-based protocols, it is necessary to perform arithmetic in fields of the form $\mathbb{F}_{q^k}$, for moderate values of k, so it is important that the field is represented in such a way that the arithmetic can be performed as efficiently as possible. The pairing evaluates as an element over the extension field $\mathbb{F}_{q^k}$, and can be represented as a polynomial with coefficients in $\mathbb{F}_q$, modulo an irreducible polynomial of degree k.

For the case where $k = 2$ and $q = p = 3 \mod 4$, a suitable irreducible polynomial has the form $f(z) = z^2 + 1$. The number -1 is a quadratic non-residue modulo p, and so the polynomial $f(z)$ does not factor over the base field. Elements in $\mathbb{F}_{p^2}$ can be represented as polynomials $a + zb$ with $a, b \in \mathbb{F}_p$. Such elements can be multiplied as normal, and then reduced modulo $z^2 + 1$. In fact $z = \sqrt{-1} \mod p$ and z can be considered as the imaginary root of the irreducible polynomial. In this case the elements of the extension field can be represented as $a + ib$ where $i = \sqrt{-1}$. This gives a straightforward analogy with complex numbers and arithmetic operations in such extension fields can be performed using efficient methods for complex arithmetic.

The addition and subtraction of elements in $\mathbb{F}_{q^k}$ can be performed efficiently by a simple addition/subtraction of the subsequent coefficients. Multiplication and squaring, however, are more expensive and can be very slow when not optimized properly. The

naive way to do multiplication in the quadratic extension field is:

$$(a + ib)(c + id) = ac - bd + i(bc + ad) \tag{3.12}$$

This operation costs 4 base field modular multiplications, but can be significantly accelerated by using the well known Karatsuba-Offman technique [107]:

$$(a + ib)(c + id) = ac - bd + i[(a + b)(c + d) - ac - bd)] \tag{3.13}$$

which only requires three modular multiplications. This method can be also used for higher extension fields where it can be even more efficient. For example in the quartic extension field $\mathbb{F}_{2^{4m}}$, elements can be represented as polynomials with 4 coefficients in $\mathbb{F}_{2^m}$: $az^3 + bz^2 + cz + d$. In this setting the Karatsuba method decreases the number of necessary multiplications to 9 from 16 at a cost of many extra additions. Additions are much cheaper in $\mathbb{F}_{2^m}$ than modular multiplications and give significant improvement in execution time.

In the case of squaring in $\mathbb{F}_{p^2}$, the Karatsuba method can be applied as well for a cost of two modular squarings and one modular multiplication. However, a better idea would be to use the well known identity from complex numbers arithmetic:

$$(a + ib)(a + ib) = (a + b)(a - b) + i \cdot 2ab \tag{3.14}$$

This requires just two modular multiplications. For binary fields, squaring is much cheaper than multiplication (see Section 3.4.2). Therefore, in $\mathbb{F}_{2^{m \cdot k}}$ it is better to replace multiplications with squarings where possible. For example a squaring in $\mathbb{F}_{2^{4m}}$ can be efficiently calculated by using only 4 modular squarings and few additions in $\mathbb{F}_{2^m}$.

Another important operation in extension field arithmetic is raising an extension field element to the power of the modulus. This operation can be calculated very efficiently using the Frobenius endomorphism. For example in $\mathbb{F}_{q^2}$:

$$(a + ib)^q = (a^q + i^q \cdot b^q) = (a - ib) \mod q \tag{3.15}$$

Equation 3.15 can be proven in the following way. The expansion of $(a+ib)^q$ produces many terms which are multiplies of q and hence are equal to $0 \mod q$. The only terms left

are a^q and $i^q \cdot b^q$. The first value can be reduced as $a^q \mod q = a$ (Fermat's little theorem) and in the second term $i^q = (i^2)^{(q-1)/2} \cdot i = -i$.

In some settings the value of the extension degree k might be much greater than 2 or 4, making the direct polynomial representation more complex. For higher extension fields, a tower of extensions can be used, where one layer builds on top of the last. For example, $\mathbb{F}_{p^{12}}$ can be implemented as a quadratic extension, of a cubic extension, of $\mathbb{F}_{p^2}$. The details on constructing tower extensions of finite fields can be found in [10]. This thesis is focused mainly on pairings over the extension fields $\mathbb{F}_{q^k}$, where k is equal to 2 or 4. Small values of k minimize the code space for Pairing-Based Cryptography implementations on embedded processors. The overall performance of extension field arithmetic relies on the operations in the base field. This fact once again highlights the importance of efficient implementation of finite field arithmetic in $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$.

3.6 Hardware perspective

The arithmetic operations presented in this chapter were implemented on general purpose embedded processors which were not designed to perform multiprecision operations. The performance of finite field arithmetic can be improved by introducing changes in the hardware architecture of these devices. In many cases small changes to instruction sets can provide better support for software implementation of arithmetic operations. Such an approach is also a more attractive option than addition of an external coprocessor as it requires fewer resources, and provides a seamless programming interface. The addition of a single instruction can significantly improve the execution time of many routines. Multiprecision multiplication is the operation which can benefit the most from the instruction set extension.

As noted in [55] multiprecision integer multiplication can usually be accelerated by introducing a special multiply-and-accumulate instruction. In the context of the improved hybrid method, one multiplication and two additions with carry are repeated many times in the algorithm. These three instructions can be combined together to form one new instruction (called `MACC` from now on). On an 8-bit ATmega128 CPU this can be implemented as (`MACC rX, rZ`), where `rX` is the first operand of the multiplication and `rZ` is the lower part of the accumulated result. `MACC` multiplies `rX` with a register `rY`, which always contains the second operand for the multiplication (e.g `r4` in Fig. 3.4). The lower

part of the 16-bit result is added to register `rZ` and the higher part is added with carry to the consecutive register `r(Z+1)`. Notice that `MACC` acts like an ordinary multiply-and-accumulate instruction when invoked twice with the same register `rZ`.

The application of the `MACC` instruction in the improved hybrid method can bring significant improvements in execution time. For multiplication of 160-bit integers with $d = 4$ it reduces the number of necessary additions by around 800. When implemented as a 2-cycle instruction, `MACC` improves the execution time of the hybrid multiplication by more than 28%. `MACC` can be also used to accelerate squaring by around 18%.

In case of the MSP430 processor an ordinary multiply-and-accumulate instruction is already implemented in the hardware multiplier unit. However, it cannot be used in the hybrid multiplication because it always accumulates the result in the fixed multiplier unit registers `RESLO` and `RESHI`. Additionally the loading of operands into the multiplier is expensive (4-6 clock cycles), because the unit operates on peripheral registers. The multiply-and-accumulate operation would be more efficient if implemented as a CPU instruction that operates on variable registers. In such cases the result of the multiplication would be accumulated in specific general purpose registers (similarly to the `MACC` instruction proposed for ATmega128). The application of a special multiply-and-accumulate operation would eliminate the need for separate additions and save 600 clock cycles (34%) for 160-bit integer multiplication on the MSP430.

Instruction set extensions can particularly improve the arithmetic in the case of binary fields. Embedded processors do not currently have a direct counterpart to the integer multiplication instruction when it comes to multiplication in $\mathbb{F}_{2^m}$. From an implementation point of view, polynomial multiplication can be seen as a simplification of integer multiplication when the carries in the partial product accumulation are not propagated. Multiplication in $\mathbb{F}_{2^m}$ accumulates partial products using XOR operations rather than ordinary additions, and hence can be denoted as XOR multiplication. The implementation of such an instruction on an embedded processor would be simple as it is very similar to the standard `MUL` operation.

In the case when XOR multiplication is available on the target platform, the improved hybrid method can be reused for binary polynomial multiplication. This would allow multiplication in $\mathbb{F}_{2^m}$ to achieve similar performance as multiplication in $\mathbb{F}_p$. In [41] the authors stated that the addition of an XOR multiplication to the ATmega128 instruction

set would significantly improve the performance of arithmetic in $\mathbb{F}_{2^m}$. They reported that the binary field multiplication in $\mathbb{F}_{2^{163}}$ would even outperform 160-bit integer multiplication by 27%. Nevertheless, the optimized hierarchical method proposed in this thesis provides the best performance for binary polynomial multiplication on embedded processors which do not have any support for multiplication in $\mathbb{F}_{2^m}$.

For future hardware designs it would be beneficial to include XOR multiplication in the default instruction set of the processor. Intel recently proposed an extension to the x86 instruction set that would perform arithmetic on binary polynomials. The new range of Westmere processors include a `PCLMULQDQ` instruction, which performs carryless multiplication of two 64-bit quadwords [53]. A similar instruction should be also included on low-end CPUs.

3.7 Summary

The two main types of finite fields that are suitable for implementation of elliptic curve systems are prime fields $\mathbb{F}_p$ and binary fields $\mathbb{F}_{2^m}$. Arithmetic in finite fields is the fundamental building block in every Elliptic Curve Cryptography implementation. The performance of the whole system depends in large degree on the efficiency of the arithmetic operations. On a constrained low-end processor, optimal performance can be only achieved with optimized assembly language implementations of time-critical arithmetic routines.

The most important operation in the prime field is the multiprecision integer multiplication. This chapter proposes a new method for calculating multiplication on low-end processors. One major advantage of the new technique is that, due to its simplicity, it can be relatively easily extended to other processor architectures. The improved hybrid method takes advantage of extra registers to avoid unnecessary load operations and becomes more efficient with the number of registers used. The general applicability of the idea has been demonstrated on three different 8, 16 and 32-bit architectures. In all of these cases, significant performance improvements have been achieved. The approach is also scalable to larger field sizes. The assembly language code for different field sizes can be automatically generated from user defined macros. This, however, has to be performed before generating the arithmetic library code.

Most of the arithmetic implementations on embedded devices are mainly focused on prime finite fields. The choice of the field was dictated by the fact that basic arithmetic

operations can be effectively optimized if a pseudo-Mersenne primes are used in $\mathbb{F}_p$. Binary fields were not favoured because binary polynomial arithmetic (multiplication in particular) is insufficiently supported by current CPU's. One of the main contributions of this chapter is the proposal of a new hierarchical method for polynomial multiplication in $\mathbb{F}_{2^m}$. This new technique is optimized for fields of large characteristic and takes full advantage of available CPU registers. Assembly implementation of the algorithm achieves significant improvement in execution time when compared with a standard C implementation. The method is also scalable and can be applied for different field sizes. However, the use of hand-crafted assembly language routines requires modification in the source code for different values of m. In addition to multiplication methods fast squaring, reduction and square rooting in $\mathbb{F}_{2^m}$ are also presented in this chapter. The implementation results prove that arithmetic in the binary field can be efficiently implemented on different embedded processors and that it is an attractive alternative, when it comes to Elliptic Curve Cryptography systems implementation.

In the context of low-end processors, the fastest binary polynomial multiplication is around 50% slower than integer multiplication in a field of similar size. Despite this disadvantage Elliptic Curve Cryptography over binary fields achieves similar (or better) performance, than over prime fields (see Chapter 4). Multiplication in $\mathbb{F}_{2^m}$ would be significantly faster if a carry-less multiplication instruction was available on constrained architectures. Additionally multiprecision arithmetic operations on embedded CPU's can be improved with an increase in the number of available registers.

CHAPTER 4

Elliptic Curve Cryptography for Wireless Sensor Networks

Until recently it used to be thought that public key cryptography was impractical on resource-constrained sensor nodes and that cryptosystems must depend only on symmetric primitives. As mentioned earlier, symmetric key techniques are more efficient than Public Key Cryptography but they do not solve the key distribution problem and assume that the nodes already share symmetric keys. This fact has motivated work on how to compute Public Key Cryptography efficiently on sensor nodes. The problem is challenging because constrained devices have very limited computational capabilities and short battery lifespan. Chapter 2 shows that elliptic curve cryptography demands considerably less resources than more conventional public key systems for a given security level. In addition results presented demonstrate that Elliptic Curve Cryptography is more efficient than RSA on small 8-bit processors. Faster execution time, smaller memory consumption and savings in bandwidth have made Elliptic Curve Cryptography an attractive public key solution in a constrained environment.

One of the main Elliptic Curve Cryptography primitives is scalar point multiplication. This operation is the base for many Elliptic Curve Cryptography mechanisms such as ECDSA and ECDH. In the context of pairing-based systems, point multiplication is one of the basic building blocks in many security protocols. Point multiplication is an expensive

primitive and a fast execution time for it is also necessary for efficient identity based cryptography.

The implementation of Elliptic Curve Cryptography on constrained devices such as WSN nodes is not an easy task. Software implementations have to be small enough to leave room for the operating system and for the core protocols. The computation of basic primitives needs to be fast enough to minimize energy consumption and maximize the network lifetime. Optimal implementation of Elliptic Curve Cryptography primitives requires several important selections. The choices include the following issues:

- size of domain parameters;
- type of elliptic curve;
- point representation system;
- algorithms for performing curve arithmetic.

For every implementation, there are many variables that influence the choices made. The main factors are the level of security that is required, the communication environment and the actual application platform with its capabilities and constraints. All three of these factors must be considered simultaneously to find the best possible solution for a given application. Sensor network devices are extremely limited in terms of resources and all the Elliptic Curve Cryptography implementation choices must be made with this aspect in mind. Certainly, the optimal choices for sensor network implementation will be quite different than for a workstation application. In many cases, a single best set of choices is almost impossible to decide on. A more realistic scenario would assume different Elliptic Curve Cryptography parameters for various classes of WSN applications. In this situation, an implementation with fixed parameters has limited use and is not the best option. A better solution would require a cryptographic library which allows developers to choose their own domain parameters, elliptic curves and optimization levels. Such flexibility makes it easier to find a good fit for a given combination of hardware platform and application type.

This chapter presents a software package called NanoECC. This cryptographic library provides elliptic curve-based Public Key Cryptography operations that were specifically optimized for different sensor network platforms. The library can be flexibly configured and easily integrated into sensor network applications to provide protocols such as

ECDSA and ECDH. NanoECC provides efficient implementation of Elliptic Curve Cryptography primitives for a range of sensor devices and can be used as a building block in constructing identity-based cryptography protocols for WSNs. The results presented in this chapter prove that Elliptic Curve Cryptography implementation in software is not only viable, but also efficient on constrained sensor nodes. The following sections present a short mathematical introduction to elliptic curves and a brief overview in the state-of-the-art of elliptic curve cryptography implementations on sensor devices. This is be followed by a detailed description and evaluation of the NanoECC library.

4.1 Mathematical background

The implementation issues of Elliptic Curve Cryptography in WSNs cannot be discussed without a proper understanding of the basic mathematical concepts of elliptic curves. This section will present basic information on elliptic curves that are used in cryptography. More detailed information about mathematical aspects of Elliptic Curve Cryptography can be found in [14], [13], [58] and [27]. Most of the information in this section is based on those references.

4.1.1 Elliptic curves

Elliptic curves used in cryptography are typically defined over two types of finite fields: $\mathbb{F}_p$ (where $p > 3$ is a large prime number) and $\mathbb{F}_{2^m}$. In general an elliptic curve E over a field $\mathbb{F}_q$ is defined by the Weierstrass equation:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{4.1}$$

where $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}_q$ and the discriminant $\Delta \neq 0$, where Δ is a discriminant of E and is defined as follows:

$$\Delta = -{d_2}^2 d_8 - 8{d_4}^3 - 27{d_6}^2 + 9 d_2 d_4 d_6$$
$$d_2 = {a_1}^2 + 4a_2$$
$$d_4 = 2a_4 + a_1 a_3$$
$$d_6 = {a_3}^2 + 4a_6$$
$$d_8 = {a_1}^2 a_6 + 4a_2 a_6 - a_1 a_3 a_4 + a_2 {a_3}^2 - {a_4}^2 \tag{4.2}$$

This condition ensures that the elliptic curve is "smooth", meaning that there are no points at which the curve has two or more distinct tangent lines. There is also an additional point on the curve known as the point at infinity, $\mathcal{O}$. The curve is sometimes denoted as $E(\mathbb{F}_q)$ to emphasize that E is defined over the underlying field $\mathbb{F}_q$. It is important to note that if E is defined over $\mathbb{F}_q$, then E is also defined over any extension field $\mathbb{F}_{q^k}$.

Figure 4.1 [58] presents an example of two elliptic curves defined over the real number field $\mathbb{R}$. The point at infinity ($\mathcal{O}$) is not included in the graph. All the points (x, y) graphed in Figure 4.1 satisfy the following equations:

$y^2 = x^3 - x$ curve (i)

$y^2 = x^3 + \frac{1}{4}x + \frac{5}{4}$ curve (ii)

4.1.1.1 Supersingular and non-supersingular elliptic curves

Elliptic curves used in cryptography can be divided into two broad classes. This division is made based on the order of the finite field q. If E is an elliptic curve defined over $\mathbb{F}_q$, then the number of points in the curve $\#E(\mathbb{F}_q)$ can be approximated according to the Hasse theorem:

$$q + 1 - 2\sqrt{q} \leq \#E(\mathbb{F}_q) \leq q + 1 + 2\sqrt{q} \tag{4.3}$$

The value $|t| \leq 2\sqrt{q}$ is called the trace of the Frobenius. The value $2\sqrt{q}$ is relatively small comparing to q, so $\#E(\mathbb{F}_q) \approx q$. If the trace of Frobenius t is divisible by the characteristic of $\mathbb{F}_q$ then the elliptic curve is known as supersingular. A supersingular curve

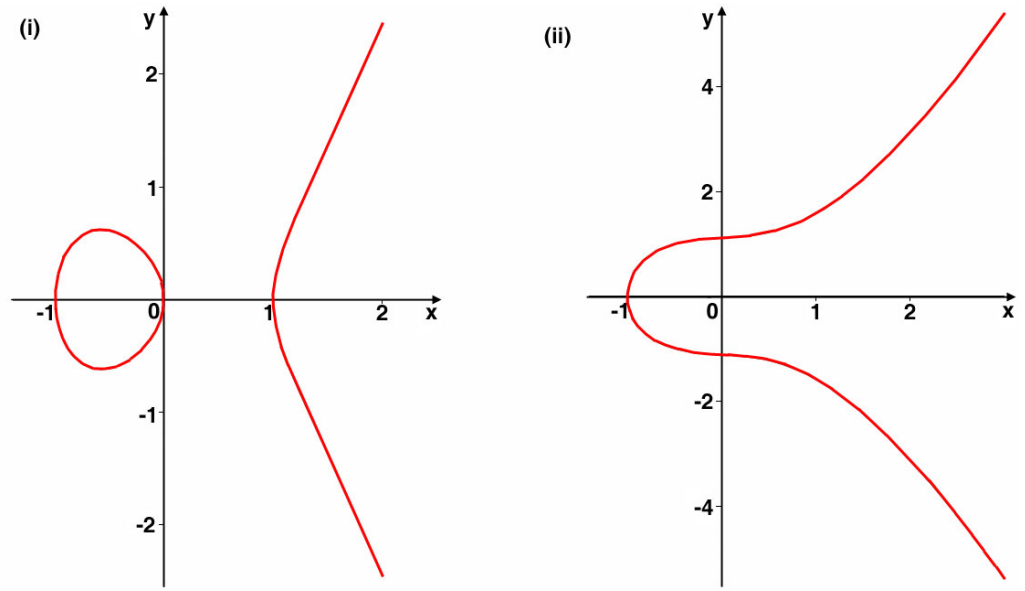

Figure 4.1: *Elliptic curves over* $\mathbb{R}$

has a group order of $q + 1$. If the trace is not divisible by the characteristic of $\mathbb{F}_q$ then the elliptic curve is known as non-supersingular.

The Weierstrass equation (equation 4.1) can be simplified considerably by applying admissible changes of variables. If the characteristic of the field $\mathbb{F}_q$ is prime and $q > 5$, then the curve equation is simplified to:

$$y^2 = x^3 + ax + b \tag{4.4}$$

where $a, b \in \mathbb{F}_q$ are constants such that $4a^3 + 27b^2 \neq 0$. The curve equation can also be simplified in the case where the characteristic of the underlying field is two. If $a_1 \neq 0$ the equation of the curve defined over $\mathbb{F}_{2^m}$ is as follows:

$$y^2 + xy = x^3 + ax^2 + b \tag{4.5}$$

where $a, b \in \mathbb{F}_{2^m}$. Such a curve is said to be non-supersingular and has discriminant $\Delta = b$. Also $(x, y) \in \mathbb{F}_{2^{mk}} \times \mathbb{F}_{2^{mk}}$, where m, k are integers. A supersingular curve is defined when $a_1 = 0$ with the discriminant $\Delta = c^4$ and is given by the equation:

$$y^2 + cy = x^3 + ax + b \tag{4.6}$$

where $a, b, c \in \mathbb{F}_{2^m}$. Supersingular curves are especially interesting due to their direct

application in pairing-based cryptography.

4.1.1.2 Koblitz curves

Koblitz curves belong to elliptic curves defined over $\mathbb{F}_{2^m}$ and are also known as anomalous binary curves. Koblitz curves are given by the equation:

$$y^2 + xy = x^3 + ax^2 + 1 \tag{4.7}$$

where $a \in \{0, 1\}$. The main advantage of these curves is that the point multiplication operation can be calculated more efficiently than on ordinary binary curves. This makes them especially attractive for implementation on constrained sensor devices. A detailed description of Koblitz curves and their properties can be found in [114].

4.1.2 Curve operations

The main arithmetic operations on elliptic curves include point addition, point doubling and scalar point multiplication. This section will describe these operations in detail.

4.1.2.1 Point addition and doubling

If E is an elliptic curve defined over the field $\mathbb{F}_q$, there exists a chord-and-tangent rule for adding two points in $E(\mathbb{F}_q)$. This operation results in a third point in $E(\mathbb{F}_q)$. The set of points $E(\mathbb{F}_q)$ and the point addition operation forms an abelian group with $\mathcal{O}$ as its identity. This group is the main construct that is used in building Elliptic Curve Cryptography schemes. This addition operation can be explained using a geometrical representation. If the curve is defined over the set of real numbers $\mathbb{R}$ then the addition rule can be represented as in Figure 4.2 (i) [58]. The points $P = (x_1, x_2)$ and $Q = (x_2, y_2)$ are added together and the result is point R with affine coordinates (x_3, y_3). The whole rule works as follows. First a straight line is drawn through P and Q. This line intersects the elliptic curve at a third point R'. Then R is the reflection of this point about the x-axis.

In case when $P = Q$ the point has to be doubled (Figure 4.2 (ii)). This operation requires a different technique. The rule for calculating a double of a point P can be explained as follows. First a line is drawn which is tangent to the elliptic curve at point P. This line intersects the elliptic curve at a particular second point R'. The equation for this

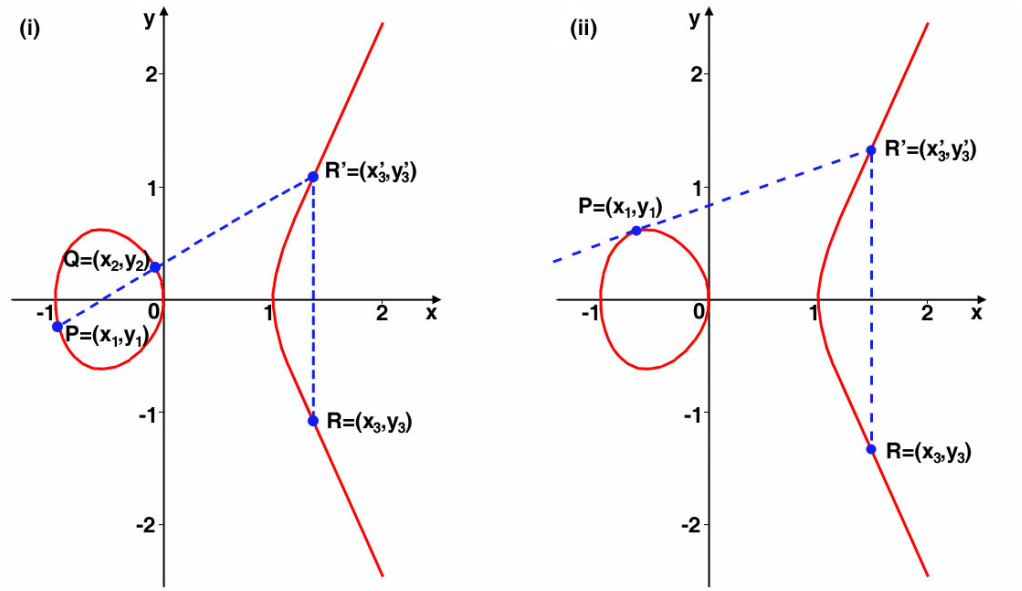

Figure 4.2: *Addition and doubling rules for elliptic curve points*

line in entered into the equation of the curve and is solved for x. This value, is entered into the equation of the line to obtain the $-y_3$ value. To get the coordinates of the R point (result of doubling P) a reflection of R' is calculated about the x-axis.

The algebraic formulas for the above two operations can be derived directly from the geometric description. Each of the formulas differ depending on the characteristic of the underlying field. However, in all cases two properties must hold. There must be an identity element and an inverse of a point. Without those properties points on the curve do not form a group. The identity element is defined as the point at infinity $\mathcal{O}$ such that $P+\mathcal{O}=P$. The inverse is the reflection of the point in the x-axis such that $P+(-P)=\mathcal{O}$. Also, $-\mathcal{O}=\mathcal{O}$.

For the field $\mathbb{F}_q$ with characteristic not equal to 2 or 3 the formulas in standard affine coordinates are equal to:

$$x_3 = \lambda^2 - x_1 - x_2 \qquad y_3 = \lambda(x_1 - x_3) - y_1 \tag{4.8}$$

for point addition $P+Q=R$, where $\lambda = \frac{y_2-y_1}{x_2-x_1}$ and $P \neq \pm Q$. λ is the gradient of the line which joins P and Q. For point doubling, $2P=R$, $\beta = \frac{2{x_1}^2+a}{2y_1}$ and $P \neq -P$. β is the gradient of the tangent line at point P obtained by implicit differentiation of the curve, the equations are as follows:

$$x_3 = \beta^2 - 2x_1 \qquad y_3 = \beta(x_1 - x_3) - y_1 \tag{4.9}$$

In the case of non-supersingular curves in characteristic two, the formulas are as follows:

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \qquad y_3 = \lambda(x_1 + x_3) + x_3 + y_1 \tag{4.10}$$

for point addition $P + Q = R$, where $\lambda = \frac{y_1+y_2}{x_1+x_2}$ and $P \neq \pm Q$.

$$x_3 = \beta^2 + \beta + a \qquad y_3 = {x_1}^2 + \beta x_3 + x_3 \tag{4.11}$$

for point doubling $2P = R$, where $\beta = x_1 + \frac{y_1}{x_1}$ and $P \neq -P$. When working with supersingular curves over $\mathbb{F}_{2^m}$ the formulas are defined as:

$$x_3 = \lambda^2 + x_1 + x_2 \qquad y_3 = \lambda(x_1 + x_3) + y_1 + c \tag{4.12}$$

for point addition $P + Q = R$, where $\lambda = \frac{y_1+y_2}{x_1+x_2}$ and $P \neq \pm Q$.

$$x_3 = \beta^2 \qquad y_3 = \beta(x_1 + x_3) + y_1 + c \tag{4.13}$$

for point doubling $2P = R$, where $\beta = \frac{{x_1}^2+a}{c}$ and $P \neq -P$.

4.1.2.2 Point multiplication

The multiplication of an elliptic curve point P by a value k is a fundamental operation that is underlying Elliptic Curve Cryptography. This arithmetic operation is called scalar point multiplication and dominates the overall execution time of elliptic curve cryptographic schemes. If P is a point on an elliptic curve $E(\mathbb{F}_q)$, then the result of the scalar multiplication kP is a different point Q on the same curve. If the scalar is denoted as $k = (k_{t-1}, \ldots, k_1, k_0)_2$, where $t = \lceil \log_2 q \rceil$ and P is unknown, then the point multiplication $kP = Q$ can be calculated by a right-to-left binary method.

Algorithm 4 (based on [74]) processes the bits of k one-by-one from right to left. As seen in lines 3 and 4, point multiplication consists of a sequence of point additions and point doublings and its cost depends on the number of ones in the binary representation of k. Obviously large number of zeroes in k guarantees faster execution of Algorithm 4. The expected number of ones in k is around $t/2$, but can be reduced by using different

Algorithm 4 Right-to-left binary method for point multiplication

INPUT: $k = (k_{t-1}, \ldots, k_1, k_0)_2$, $P \in E(\mathbb{F}_q)$
OUTPUT: $Q = kP$

```
1: Q ← ∞
2: for i ← 0 to t − 1 do
3:     if k_i = 1 then
4:         Q ← Q + P
5:     end if
6:     P ← 2P
7: end for
8: return Q
```

representation of k. For example, $255P$, or $(11111111)_2P$, requires 8 point additions. But when transformed into $(10000000 - 1)_2P$, which is $256P - P$, only one addition (and one extra subtraction) is required.

There are several techniques that can be used to accelerate the calculation of point multiplication. In some cases, point addition is more expensive than point doubling. In this situation it is beneficial to use more point doublings than point additions to compute the scalar multiplication. To facilitate this a signed digit representation of numbers can be used. One such representation is the Non-Adjacent Form (NAF). It is an effective way to achieve the lowest Hamming weight[1] for scalar k in point multiplication kP.

The non-adjacent form of a positive integer k can be represented as $k = \sum_{i=0}^{l-1} k_i 2^i$ where $k_i \in \{0, \pm 1\}$ and $k_{l-1} \neq 0$. The NAF representation of k has the fewest nonzero digits of any signed digit representation of k. The digits of NAF(k) can be calculated by repeated division of k by 2 with remainders of 0 and ± 1. This procedure results in NAF(k) where no two consecutive digits k_i are equal to ± 1. The number of nonzero digits is reduced on average to 1/3. For example when $k = 27$ $(11011)_2$ then NAF(k) is equal to $(10 - 10 - 1)$. It is important to notice that NAF(k) can have at most one digit more than k. The usage of NAF representation of k in Algorithm 4 reduces the number of necessary point additions by 33%.

In the case when extra memory is available, the running time of the scalar point multiplication algorithm can be further decreased by using a window method which processes w digits of k at a time. A method that combines the window technique with a NAF representation of the scalar is also possible to apply. This method requires generation of a width-w NAF representation of k. For a positive integer $w > 2$, NAF$_w(k)$ can be repre-

[1] Hamming weight of a string is the number of symbols that are different from the zero-symbol of the alphabet used.

sented as $k = \sum_{i=0}^{l-1} k_i 2^i$ where each nonzero coefficient k_i is odd, $|k_i| < 2^{w-1}$, $k_{l-1} \neq 0$ and at most one of any w consecutive digits is nonzero. The standard NAF(k) is equal to NAF$_2$(k) in the width-w representation. The digits of NAF$_w$(k) can be obtained by repeated division of k by 2 with remainders in $[-2^{w-1}, 2^{w-1}-1]$. For example when $k = 189$ $(10111101)_2$ then NAF$_4$(k) is equal to $(300000-3)$. The windowed NAF method for point multiplication requires online pre-computation and can be calculated using Algorithm 5 (based on [114]).

Algorithm 5 Window NAF method for point multiplication

INPUT: Window of w bits, positive integer k, point $P \in E(\mathbb{F}_q)$
OUTPUT: $Q = kP$

1: Calculate NAF$_w$(k)= $\sum_{i=0}^{l-1} k_i 2^i$
2: Compute $P_i = iP$ for $i \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$
3: $Q \leftarrow \infty$
4: **for** $i \leftarrow l - 1$ **downto** 0 **do**
5: $\quad Q \leftarrow 2Q$
6: $\quad$ **if** $k_i \neq 0$ **then**
7: $\quad\quad$ **if** $k_i > 0$ **then**
8: $\quad\quad\quad Q \leftarrow Q + P_{k_i}$
9: $\quad\quad$ **else**
10: $\quad\quad\quad Q \leftarrow Q - P_{-k_i}$
11: $\quad\quad$ **end if**
12: $\quad$ **end if**
13: **end for**
14: **return** Q

The windowed NAF method provides $\frac{1}{w+1}$ of nonzero digits density at the expense of a precomputation table in RAM containing $(2^{w-2} - 1)$ precomputed points (plus the original point). Algorithm 5 reduces the number of necessary point additions by around 60% when compared with Algorithm 4. The windowed NAF method can be also modified to employ a "sliding window" technique. In this modification the window of width w moves left-to-right over the digits in NAF(k), with placement so that the value in the window is odd. The sliding window technique requires more precomputation than the width-w NAF method, but needs fewer point operations in the main loop of the algorithm. However, the difference in performance of both methods is fairly small.

The windowed NAF methods work well when performing point multiplication on prime field curves and ordinary binary curves. On Koblitz curves the point multiplication operation can be further accelerated, as no expensive point doublings are required. Elliptic curve point doublings can be replaced with a Frobenius map $\tau(x, y) = (x^2, y^2)$ where $\tau(\infty) = \infty$. The value of $\tau(x, y)$ can be efficiently calculated since squaring in

$\mathbb{F}_{2^m}$ is relatively inexpensive. Because it is known that $(\tau^2 + 2)P = \mu\tau(P)$ holds for all points on the Koblitz curve, where $\mu = (-1)^{1-a}$, the Frobenius map can be regarded as a complex number τ satisfying $\tau + 2 = \mu\tau$.

The strategy for computing a scalar multiplication over Koblitz curves is to convert a scalar k to a radix τ expansion such as $k = \sum_{i=0}^{l-1} \mu_i \tau^i$, where $\mu_i \in \{0, \pm 1\}$. Such a τ-adic representation can be obtained by repeated divisions of k by τ. To decrease the number of point additions in a scalar multiplication, the τ-adic representation for k should be sparse and short. This can be achieved by applying τ-adic NAF (TNAF), which can be viewed as a τ-adic analogue of the ordinary NAF. As before, the number of calculations can be decreased by applying a window method for TNAF representation, which processes w digits at a time at the expense of extra memory. Width-w TNAF requires the same amount of precomputation as the window NAF method and the algorithm has the same number of point additions. However, the elimination of point doublings makes the window TNAF method significantly faster than other point multiplication algorithms.

All the above techniques assume that the point P is unknown before the calculation of kP (random point multiplication). In the case when P is known a priori, point multiplication can be significantly accelerated. In instances where P is fixed, for example in elliptic curve digital signature generation, point multiplication algorithms can exploit precomputed data that depends only on P (and not on k). For example, if the points $2P, 2^2P, \ldots, 2^{t-1}P$ are precomputed, then the right-to-left binary method (Algorithm 4) has expected running time of approximately $t/2$ additions (t is the bit length of k). This technique eliminates completely the need for point doublings, but requires a large amount of precomputation.

Brickell *et al.* in [21] optimized the above method by using windows of width w to limit the number of precomputed points. The algorithm described in [21] can be accelerated using the exponentiation methods due to Lim and Lee [78]. The resulting fixed-base comb multiplication lowers the number of necessary point additions at the cost of some extra point doublings. The method is illustrated in Figure 4.3. The scalar k is first padded on the left with $dw - t$ zeroes (if necessary), where w is the width of the precomputation window and $d = \lceil t/w \rceil$. Then k is divided into w bit strings $(K^{w-1}, \ldots, K^0)$ each of the same length d. During the calculation of kP, one bit of every bit string is processed and forms a short scalar for P multiplication. This value is precomputed off-line for every

possible combination of w bits (2^w precomputed points). At each stage of the comb multiplication, the required precomputed point is added to the accumulator and doubled. This step is repeated d times. The whole procedure is summarized in Algorithm 6. The comb method is one of the fastest algorithms for calculating point multiplication when P is fixed. This method is especially attractive for embedded devices as the points can be precomputed off-line and stored in program memory to save on precious RAM.

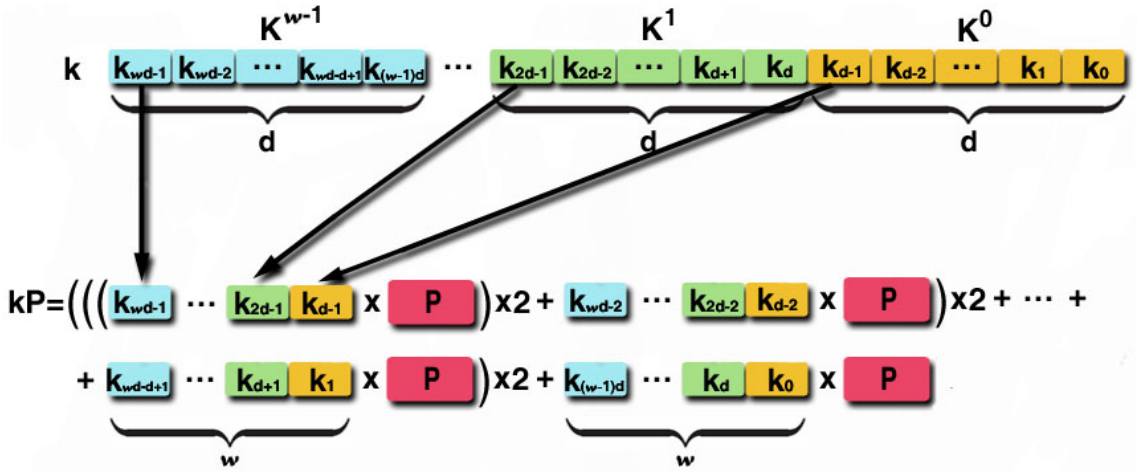

Figure 4.3: *Point multiplication using the fixed-base comb method*

Algorithm 6 Fixed-base comb method for point multiplication

INPUT: Window of w bits, $d = \lceil t/w \rceil$, $k = (k_{t-1}, \ldots, k_1, k_0)_2$, point $P \in E(\mathbb{F}_q)$
OUTPUT: $Q = kP$

1: Precompute $(a_{w-1}, \ldots, a_1, a_0)P$ for all bit strings $(a_{w-1}, \ldots, a_1, a_0)$ of length w.
2: Divide k into w bitstrings $k = K^{w-1}|| \ldots ||K^1||K^0$ each of the same length d. K_i^j denotes the ith bit of K^j.
3: $Q \leftarrow \infty$
4: **for** $i \leftarrow d-1$ **downto** 0 **do**
5: $\quad Q \leftarrow 2Q$
6: $\quad Q \leftarrow Q + (K_i^{w-1}, \ldots, k_i^1, k_i^0)P$
7: **end for**
8: **return** Q

4.1.3 Point representation

The points on an elliptic curve can be represented in many different ways. The selection of the point coordinate system has a big influence on the performance of the curve arithmetic operations. Point multiplication algorithms described in the previous section perform many point additions and doublings, which have different formulas depending on the chosen coordinate system and the type of the curve. Fast execution of these

operations is not possible without efficient representation of elliptic curve points. The following lists the main coordinate systems used in elliptic curve cryptography:

Affine coordinates are standard (x, y) coordinates commonly used in mathematics. All the formulas presented in previous sections use this type of coordinate system.

Projective coordinates may be used in some cases to accelerate certain arithmetic operations. The point (X, Y, Z) on E corresponds to the affine point $(X/Z, Y/Z)$ when $Z \neq 0$ and to the point at infinity $\mathcal{O} = (0, 1, 0)$. It has been shown that projective coordinate systems are more efficient than affine systems when calculating point multiplication.

Jacobian coordinates is another system that improves the timings for basic arithmetic operations on a curve. The point (X, Y, Z) on E in jacobian coordinates corresponds to the affine point $(X/Z^2, Y/Z^3)$ when $Z \neq 0$ and to the point at infinity $\mathcal{O} = (1, 1, 0)$. In general for Jacobian coordinates, doublings are faster and additions slower than for the projective coordinates.

López-Dahab coordinates are designed for curves defined over fields of characteristic two. A point (X, Y, Z) on an elliptic curve in the LD coordinates is equal to the point $(X/Z, Y/Z^2)$ in affine coordinates where $Z \neq 0$. The point at infinity corresponds to $\mathcal{O} = (1, 0, 0)$. López-Dahab coordinates are the optimal choice when calculating point multiplication on binary curves [81].

Mixed coordinates use a combination of different coordinate systems defined above. Some improvements can be made when adding an affine point to one in another system. In general, one can add points expressed in two different systems and give the result in a third one. The most efficient combination of coordinate systems should be chosen for each arithmetic operation on the curve.

Other coordinate systems are also possible for representing points on an elliptic curve. Formulas for point doubling and point addition from Section 4.1.2.1 are based on affine coordinates. Some of them require a field inversion and several field multiplications. In many cases, field inversion reduces the efficiency of elliptic curve cryptosystems drastically, because the ratio of a field inversion to the field multiplication is rather high.

Usually if inversion in $\mathbb{F}_q$ is more expensive than eight multiplications, then it is advantageous to represent points using different coordinates (e.g. projective or mixed coordinates). The underlying field operation counts for point addition and doubling in various coordinate systems are listed in Table 4.1. In all cases, using mixed coordinates for point addition is more efficient than using the same representation for both points. For curves defined over prime fields, Jacobian coordinates yield the fastest point doubling, while mixed Jacobian-affine coordinates yield the fastest point addition [2]. In case of binary curves, López-Dahab coordinates outperform all other point representation systems.

When designing key distribution protocols it is important to consider the transmission of point coordinates. This issue is important in wireless sensor networks, where we want to minimize the number of bytes transmitted in the network. For transmission purposes affine coordinates are the most suitable, because they only include two values. In addition, one can send compressed affine coordinates, which include the value of x and a single bit of y. Based on the curve equation the receiver can calculate two possible values of y. The single bit of y decides, which value is the correct one.

Table 4.1: *Operation counts for point addition and doubling*

Curve type	Coordinate system	Addition	Addition (mixed coord.)	Doubling
$y^2 = x^3 - 3x + b$ defined over $\mathbb{F}_p$	Affine	I+2M+S	–	I+2M+S
	Projective	12M+2S	9M+2S	7M+3S
	Jacobian	12M+4S	8M+3S	4M+4S
$y^2 + xy = x^3 + ax^2 + b$ binary curve	Affine	I+2M+S	–	I+3M+S
	Projective	14M+S	11M+S	7M+5S
	Jacobian	14M+5S	10M+4S	5M+5S
	López-Dahab	14M+6S	8M+5S	4M+5S

M - multiplication in $\mathbb{F}_q$, S - squaring in $\mathbb{F}_q$, I - inversion in $\mathbb{F}_q$

4.2 Elliptic Curve Cryptography for WSNs

The first implementations of Elliptic Curve Cryptography primitives in WSNs proved that it is possible to compute point multiplication on standard 8-bit sensor nodes. The feasibility of Elliptic Curve Cryptography on sensor networks was first proven by Malan *et al.* [85] with the introduction of the EccM library. This implementation provided el-

[2] Recently discovered Edwards coordinates beat these speed records using 3M+5S for point doubling and 7M+4S for point addition (see [11])

liptic curve cryptography over $\mathbb{F}_{2^m}$ with 163-bit key sizes. The results presented in this work confirmed the possibility of using Public Key Cryptography in sensor nodes, but the running time of Elliptic Curve Cryptography primitives was far too high for practical applications. Early implementations of elliptic curve cryptography on sensor devices ([85], [15]) were not promising and it was unclear if Elliptic Curve Cryptography could be attractive for sensor networks without the use of hardware accelerators.

The performance of Elliptic Curve Cryptography primitives in WSNs can be improved by using some of the techniques presented in Section 4.1.2. Projective or mixed coordinates can be used rather than affine coordinates, in order to avoid expensive inversions. More efficient point multiplication algorithms can be used to accelerate cryptographic protocols that are based on elliptic curves. These and other optimizations were implemented in the TinyECC library [80] and in the Wang and Li implementation on the TelosB mote [118]. Both libraries used TinyOS as the underlying operating system and provided improved results on computing Elliptic Curve Cryptography in WSNs. The following sections will present the details on the above implementations and will discuss the security parameters required to apply Elliptic Curve Cryptography in a sensor network environment.

4.2.1 Security parameters

Elliptic Curve Cryptography requires multiple parameters that are very important for the overall security of the system. It is not enough to rely only on big key sizes. Other domain parameters like the selected curve E, an appropriate finite field $\mathbb{F}_q$ and the right base point P are equally important. All these parameters should be chosen in a way that the Elliptic Curve Discrete Logarithm Problem is resistant to all known attacks. There may also be other additional constraints due to implementation reasons. One may chose a particular type of a finite field due to strong support for arithmetic in that field on a given platform. The basic set of domain parameters for Elliptic Curve Cryptography systems includes the following [2]:

- S - seed if the elliptic curve was randomly generated;
- q - the order of the finite field;
- FR - representation used for the field elements;

- a, b - two curve coefficients in $\mathbb{F}_q$ that define the equation of the elliptic curve $E(\mathbb{F}_q)$;
- x_P, y_P - two field elements in $\mathbb{F}_q$ that define a finite point $P = (x_P, y_P) \in E(\mathbb{F}_q)$;
- n - the order of the point P;
- h - the cofactor equal to $\#E(\mathbb{F}_q)/n$.

The number of points in the curve is a very important parameter in elliptic curve systems. In order to avoid the Pohlig-Hellman attack [99] and the Pollard's ρ attack [100] on the Elliptic Curve Discrete Logarithm Problem, it is necessary that $\#E(\mathbb{F}_q)$ is divisible by a sufficiently large prime n. The hardest Elliptic Curve Cryptography scheme publicly broken to date using the Pollard's ρ algorithm had a 112-bit n in $\mathbb{F}_p$ and a 109-bit m in $\mathbb{F}_{2^m}$. For the prime field case, this was broken in July 2009 using a cluster of over 200 PlayStation 3 game consoles over a period of three and a half months [20]. For the binary field case, the record was set in April 2004 using 2600 computers over 17 months [122]. The process of breaking a 131-bit key in $\mathbb{F}_{2^m}$ is currently underway [123]. With the current state-of-the-art in cryptanalysis, Elliptic Curve Cryptography systems based on standard 160-bit keys should be secure until the year 2020 [20]. Such level of security should be sufficient for the majority of sensor network applications.

Using cryptographic keys of appropriate size is not the only security measure required in Elliptic Curve Cryptography systems. In the case when the finite field $\mathbb{F}_q$ is fixed, the security can be improved by choosing proper elliptic curves. The maximum resistance to the Pohlig-Hellman and Pollard's ρ attacks can be achieved by selecting such E that $\#E(\mathbb{F}_q)$ is prime or near-prime[3].

Another threat to elliptic curve systems is the range of isomorphic attacks. Isomorphism attacks reduce the Elliptic Curve Discrete Logarithm Problem to the Discrete Logarithm Problem in groups for which subexponential time algorithms are known. These attacks result in methods that are faster than Pollard's ρ algorithm, but only for a special classes of elliptic curves. When selecting the prime field, one should check that $\#E(\mathbb{F}_q) \neq q$ to avoid this attack. The Weil and Tate pairing attacks on elliptic curves can also be harmful if the right precautions are not exercised. For this reason, n should not divide $q^k - 1$ for all $1 \leq k \leq C$. In this case, C a large enough number that makes the Discrete

[3] this is in case where $\#E(\mathbb{F}_q) = hn$ with n prime and h small (1,2,3 or 4)

Logarithm Problem in $\mathbb{F}^*_{q^C}$ intractable (for $n > 2^{160}$ the value of C should be at minimum 20). Finally, to ensure resistance to the Weil descent attack, one may consider using a binary field $\mathbb{F}_{2^m}$ only if m is prime.

One solution to defend against attacks on special classes of curves that may be discovered in the future is to select the elliptic curve E at random. However, the condition that $\#E(\mathbb{F}_q)$ is divisible by a large prime should still hold. The probability that a random curve is prone to one of the known isomorphism attacks is negligible. In this way all the known attacks are also prevented. The coefficients of an elliptic curve can be selected at random as outputs of a one-way hash function. Of course the chosen one-way function should be secure and not invertable. This provides some assurance that the elliptic curve was not intentionally constructed with hidden weaknesses which could thereafter be exploited.

4.2.2 Existing implementations

Implementing Elliptic Curve Cryptography on wireless sensor nodes became a popular research topic in recent years. Many authors tried to apply Elliptic Curve Cryptography in software on standard WSN nodes, but the efficiency of these implementations is still not satisfactory enough. The next sections presents the main Elliptic Curve Cryptography implementations developed for sensor networks so far.

4.2.2.1 MICA2 implementation

The work described in [85] is the first implementation known of Elliptic Curve Cryptography on sensor devices. It targets the 8-bit MICA2 platform and provides Elliptic Curve Cryptography primitives over $\mathbb{F}_{2^m}$ with 163-bit keys. The authors proposed Elliptic Curve Cryptography as a method for secure distribution of 80-bit symmetric TinySec keys. For this purpose they have developed the EccM library that was implemented in the nesC language and evaluated under TinyOs.

In [85], the Elliptic Curve Diffie-Hellman key exchange protocol is proposed for the nodes to exchange secret keys in a secure manner. ECDH is an elliptic curve variant of the classic Diffie-Hellman protocol (see Figure 4.4). Initially all the nodes must be preloaded with the same domain parameters $(S, FR, q, a, b, P, n, h)$. Also, each node must generate a key pair suitable for Elliptic Curve Cryptography, consisting of a private key

d (a randomly selected integer in the interval $[1, n-1]$) and a public key Q (where $Q = dP$). In the case when node A key pair is (d_A, Q_A) and node B key pair is (d_B, Q_B) both parties can agree upon a shared secret. This operation begins with an exchange of public keys. When node A receives public key Q_B, it calculates the value $d_A Q_B$. At the same time node B computes $d_B Q_A$ and from now on shares the same secret value as node A, because $d_A Q_B = d_A d_B P = d_B d_A P = d_B Q_A$. The shared secret key is usually set as the x coordinate of the calculated point.

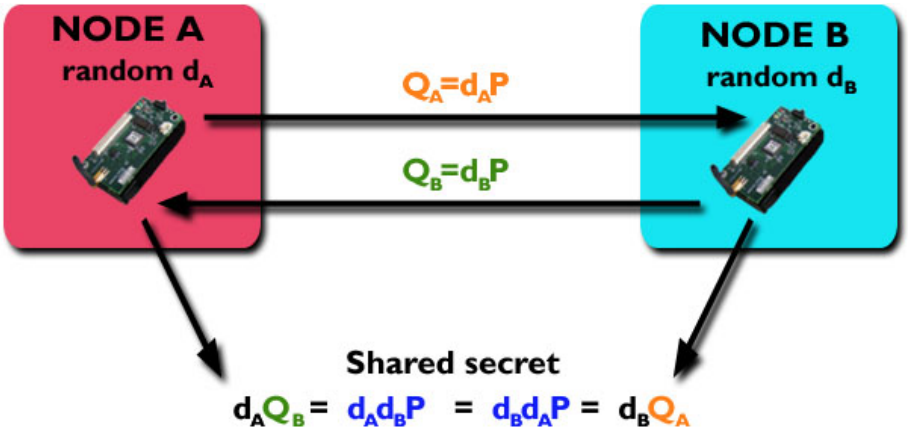

Figure 4.4: *Elliptic curve Diffie-Hellman key exchange protocol in WSNs*

For the field elements in $\mathbb{F}_{2^m}$, the polynomial basis was chosen as binary polynomials can be effectively represented in vector form. The authors also reported that with this basis, the point multiplication implementation was more efficient on the MICA2 nodes. The first version of the EccM library was based on a random elliptic curve with an arbitrary chosen base point. Its performance was low as it could not generate the keys in a reasonable amount of time (below one minute).

EccM 2.0 introduced many improvements in comparison with the previous version. It used a $y^2 + xy = x^3 + x^2 + 1$ Koblitz curve and a fixed base point to achieve better performance. The reduction polynomial $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$ was chosen to facilitate fast reduction of binary polynomials. The time required to generate a private and public key pair with the EccM 2.0 module, averaged over 100 trials, was equal to 34.2s. The memory requirement for this implementation was 1.1KB RAM and 34.3KB in ROM. The authors in [85] argued that such performance is sufficient for infrequent distribution of TinySec keys. However, from the energy budget point of view, a running time of 34 seconds for the point multiplication operation is unacceptable for the majority

of WSN applications.

4.2.2.2 TelosB implementation

Wang and Li in paper [118] proposed an Elliptic Curve Cryptography implementation over the prime field $\mathbb{F}_p$ for the 16-bit TelosB platform. This sensor device is very similar to the Tmote Sky node and is also powered by the same MSP430 microprocessor.

As opposed to the previous work, the authors in [118] chose the prime field as the base finite field for their implementation. They argued that multiplication and modular reduction in $\mathbb{F}_p$, can be more effectively optimized than in $\mathbb{F}_{2^m}$, if pseudo-Mersenne primes are used for the elliptic curves. Their whole implementation was based on the SECG recommended 160-bit elliptic curve secp160r1 [24].

The work presented in [118] implements a suite of large integer arithmetic operations in the underlying finite field $\mathbb{F}_p$. These operations include addition, subtraction, shift, multiplication, division and modular reduction. The authors also used a hybrid multiplication method that was proposed in [57]. To achieve better performance and enable flexible control over registers, they implemented the hybrid multiplication in assembly language. All the points on the secp160r1 elliptic curve were represented using mixed coordinates with modified Jacobian coordinates (X, Y, Z, aZ^4) and Affine coordinates (x, y). The usage of the mixed coordinate system led to improvements in the performance of the point addition and doubling operations. Mixed coordinates resulted also in a 6% improvement for point multiplication when compared with Jacobian coordinates.

The efficiency of the point multiplication routine was further improved with the use of a NAF representation of the scalar. The authors also used the sliding window method for the point multiplication operation. Furthermore, additional memory was utilized for storing precomputed points. Experimental results showed that the sliding window technique is more effective than the window NAF method for fixed point multiplication. The performance of the sliding window method was more than 10% better than that of NAFs.

The implementation of point multiplication on the TelosB mote was considered in two cases. The first one was when multiplying a large integer with a fixed point (base point) and the other was with a random point. Fixed point multiplication allowed for optimizations by precomputation. The results for fixed and random point multiplication were equal to 3.13s and 3.51s respectively. Memory utilization was at the level of 17.8KB

of program space and 1.6KB of data storage.

In addition to the point multiplication operation, the Elliptic Curve Digital Signature Algorithm was also implemented. When signing a message, one fixed point multiplication is the dominant operation. The performance of the signature generation was equal to 3.35s which is close to that of the fixed point multiplication. The verification step of the Elliptic Curve Digital Signature Algorithm consisted mainly of one fixed point multiplication and one random point multiplication and its timing was equal to 6.78s. The program storage for the Elliptic Curve Digital Signature Algorithm protocol was equal to 42.3KB with a large part of space occupied by the SHA-1 implementation.

Paper [118] shows that Elliptic Curve Cryptography primitives and protocols can be executed in a reasonable amount of time on the TelosB platform. The presented results suggest also that Elliptic Curve Cryptography over prime fields is more efficient than over binary fields.

4.2.2.3 TinyECC

TinyECC v1.0 [80] is an elliptic curve cryptography library which provides Public Key Cryptography operations optimized for constrained sensor network platforms. It provides three different Elliptic Curve Cryptography-based security protocols: a digital signature scheme (ECDSA), a key exchange protocol (ECDH), and a public key encryption scheme (ECIES). TinyECC uses different optimization switches to adjust the performance of cryptographic operations. Each of those switches can be turned on or off separately, based on the developer's needs.

This library is intended for sensor platforms running TinyOS. The current version, 1.0, is implemented in nesC, with additional platform-specific optimizations in inline assembly language for popular sensor platforms. So far it has been tested on MICAz, TelosB, Tmote Sky, and Imote2. TinyECC 1.0 supports SECG recommended 128-bit, 160-bit and 192-bit elliptic curve domain parameters. In particular, the following curves are used: sec128r1, secp128r2, secp160k1, secp160r1, secp160r2, secp192k1, and secp192r1 [24].

TinyECC adopts several well known optimization techniques that accelerate Elliptic Curve Cryptography operations on sensor platforms. These methods include [80]:

Projective coordinate system. Many small processors do not support the division instruction (ATmega128, MSP430) so the inverse operation is significantly more ex-

pensive than multiplication. It is efficient to implement elliptic curve operations in projective coordinates instead of affine coordinates. TinyECC uses weighted projective representation (Jacobian coordinates) to speed up point addition, point doubling and scalar point multiplication.

Curve-specific optimizations. For all NIST and most SECG curves, the underlying field primes p were chosen as pseudo-Mersenne primes to allow optimized modular reduction. This optimized modular reduction algorithm was implemented in order to speed up modular multiplication and modular squaring.

Sliding window method. This algorithm was used to speed up scalar point multiplication by reducing the total number of point additions. One disadvantage of this technique is that extra memory is required as some points need to be precomputed.

Hybrid multiplication. In TinyECC, the natural number operations are based on the RSA cryptographic toolkit RSAREF 2.0. This implementation is platform independent, but not efficient. To facilitate this, TinyECC uses inline assembly code to speed up critical operations such as multiplication and squaring for MICAz, TelosB/Tmote Sky, and Imote2 motes. In particular, hybrid multiplication [57] was implemented to accelerate the arithmetic in $\mathbb{F}_p$.

Table 4.2: *Performance analysis of ECC implementations on sensor motes*

Implementation	**EccM2.0**	**Wang-Li**	**TinyECC**	**TinyECC**	**TinyECC**
Platform	MICA2	TelosB	MICA2	TelosB	Imote2
CPU	Atmega128	MSP430	Atmega128	MSP430	PXA271
Clock	7.38MHz	4MHz	7.38MHz	4MHz	13MHz
Curve	sect163k1	secp160r1	secp160r1	secp160r1	secp160r1
Finite field	$\mathbb{F}_{2^m}$	$\mathbb{F}_p$	$\mathbb{F}_p$	$\mathbb{F}_p$	$\mathbb{F}_p$
Point mul.	34.2s	3.5s	2.1s	4.2s	0.4s
ROM	34.3KB	17.8KB	16.0KB	11.4KB	12.9KB
RAM	1.1KB	1.6KB	1.8KB	1.8KB	2.1KB

A performance evaluation of point multiplication in TinyECC and a comparison with other Elliptic Curve Cryptography implementations is presented in Table 4.2. The numbers for the Imote2 platform were achieved at 13MHz clock speed. As shown in Table 4.2, all implementations use the prime field $\mathbb{F}_p$ apart from EccM2.0 which performs arithmetic in the binary $\mathbb{F}_{2^m}$. EccM2.0 is the slowest implementation with the largest memory

footprint. The Wang-Li implementation performs the fastest point multiplication on the TelosB platform, whereas TinyECC provides the best results on the MICA2 and Imote2 devices. In all cases TinyECC is more efficient when it comes to memory utilization. Despite the slightly lower performance on the TelosB mote, TinyECC is the most useful library for sensor networks as it supports many different platforms and allows a great degree of customization.

4.3 NanoECC: optimizing ECC for WSN platforms

Efficient implementation of Elliptic Curve Cryptography primitives in sensor networks is a difficult task. Small memory size, limited CPU capabilities and scarce battery resources create a difficult environment for applying elliptic curve cryptography. Program code needs to be highly optimized to meet all of the above demands. This is the reason why there is some confusion in the literature concerning timings of basic elliptic curve operations on constrained WSN platforms. The results presented in the previous section vary from each other in terms of execution time and memory consumption. It is not certain which base field gives the best performance for the crucial point multiplication operation. Most of the existing implementations focus on optimizing Elliptic Curve Cryptography for a particular hardware platform and forget about the portability of the code. Many available libraries use fixed parameters without permitting the possibility to change elliptic curves, domain parameters or security levels. The energy consumption of cryptographic primitives is not also investigated in many cases.

The NanoECC library addresses all the above issues and offers efficient Elliptic Curve Cryptography primitives that can be used to build complete security protocols. The library can be easily integrated into existing sensor network applications to provide Public Key Cryptography-based security solutions. NanoECC is optimized for different sensor network platforms and can be flexibly configured to find the best set of parameters for a given application type. It is compatible with MICA2, MICA2DOT, MICAz, TelosB and Tmote Sky motes, but can be also easily ported to other hardware platforms. The evaluation results of NanoECC show that the software implementation of Elliptic Curve Cryptography is not only feasible, but in fact attractive for tiny sensor nodes. The results can be also treated as a comprehensive benchmark of Elliptic Curve Cryptography performance in WSNs.

4.3.1 NanoECC overview

When developing software for sensor networks, certain design choices must be made to specify the features and functionality of the code. NanoECC follows four design principles:

1) Portability. The library is compatible with a broad range of existing sensor platforms and can be easily ported to other 8, 16 and 32-bit constrained devices. Most of the procedures use standard C, which favors speed and permits re-use of the code on numerous other WSN platforms. Even though some functions use inline assembly, the code development process is as portable as possible. Assembler routines are generated automatically by special utility program from user defined macros. In this simple and convenient way, appropriate assembler code can be quickly developed for new platforms and processors that are not yet supported.

2) Performance. NanoECC is optimized for speed and energy efficiency. Memory usage is a secondary concern as optimizing for code size lowers the functionality and portability of the library. The memory footprint of the library can be reduced after achieving satisfactory performance by deleting unnecessary functions. In order to speed up the execution of particularly time-critical arithmetic routines, the standard C code is replaced with some assembly language specific for each platform. The high performance of NanoECC is achieved also through many different optimizations for elliptic curve operations.

3) Security. High level of security in Elliptic Curve Cryptography systems can only be achieved through careful selection of algorithms and domain parameters. NanoECC takes into consideration all of the guidelines described in Section 4.2.1. It also supports the elliptic curve parameters recommended by NIST [22] and SECG [24].

4) Versatility. Large set of available library functions in NanoECC gives a lot of possibilities in writing Elliptic Curve Cryptography based programs. The parameters can be flexibly configured and the code can be easily integrated into WSN applications to provide the building blocks for standard Elliptic Curve Cryptography schemes (ECDSA, ECDH) and more advanced Identity-Based Cryptography protocols. NanoECC is the only library for WSNs (so far) that provides Elliptic Curve Cryptog-

raphy operations over both the prime field $\mathbb{F}_p$ and the binary field $\mathbb{F}_{2^m}$. The library can be also used with conjunction with different embedded operating systems.

The prime finite field was the main focus of many different Elliptic Curve Cryptography implementations for sensor networks [56], [80], [118], [15]. The choice of the field was dictated by the fact that basic arithmetic operations can be effectively optimized if pseudo-Mersenne primes are used in $\mathbb{F}_p$. The authors in [80] stated, that it is difficult to obtain efficient Elliptic Curve Cryptography implementation over $\mathbb{F}_{2^m}$ on typical sensor platforms. Binary fields were not favoured because polynomial arithmetic (multiplication in particular) is insufficiently supported by current CPU's. Results achieved by NanoECC using both types of finite fields show for the first time that in some cases in this constrained environment, Elliptic Curve Cryptography operations over the binary polynomial field $\mathbb{F}_{2^m}$ outperform those in $\mathbb{F}_p$. Moreover, the timings for the binary field case would be even faster if a binary polynomial multiplication instruction was available on the architectures considered.

The core of NanoECC is based on MIRACL (Multiprecision Integer and Rational Arithmetic C/C++ Library) [109], which provides all the necessary tools to perform operations on elliptic curves. MIRACL supports Elliptic Curve Cryptography over both standard finite fields, $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$. It also handles all of the operations on big integers and large polynomials. MIRACL was originally designed for desktop class computers, but can be also optimized to work with constrained 8 and 16-bit platforms. On a PC with large memory space, all the memory for big integers is allocated dynamically from the heap. In an embedded environment, the memory resources are very limited and dynamic memory allocation is often not available. MIRACL had to be optimized to allocate all the memory exclusively from the stack. One downside of this is that the maximum size of big variables has to be set at compile time, when the library is being created. Nevertheless, static memory allocation allows maximum use and re-use of memory, and avoids fragmentation of precious RAM. The MIRACL library was extensively optimized to provide the best Elliptic Curve Cryptography performance on constrained WSN motes.

4.3.2 Implementation of NanoECC

NanoeECC was implemented initially on two popular WSN motes: the MICA2 [30] platform developed by Crossbow Technology and the Tmote Sky [92] node developed by the

Moteiv corporation. The library is also compatible with other sensor nodes (MICA2DOT, MICAz, TelosB), which use the same processors (ATmega128 and MSP430F1611). Experimental results have shown that the point multiplication operation is the most time consuming operation in Elliptic Curve Cryptography protocols. For example, more than 90% of the execution time in the Elliptic Curve Diffi-Hellman key exchange protocol is spent on performing point multiplication operations. That is why the primary focus of NanoECC was on optimizing the performance of fixed-point and random-point multiplication routines.

4.3.2.1 Curve arithmetic optimization

The selection of point multiplication algorithms is complicated and depends on platform characteristics, coordinates selection, interoperability, security requirements and memory constraints. Another difficulty in using Elliptic Curve Cryptography is that of finding suitable elliptic curves. Curve parameters have to be carefully chosen to allow efficient computations and provide a reasonable level of security. NIST recommends using at least 160-bit keys in Elliptic Curve Cryptography systems to achieve a security level equivalent to that offered by standard RSA based solutions with 1024-bit keys.

In the case of the binary field $\mathbb{F}_{2^m}$, random point multiplication achieves best performance when using Koblitz curves. The elimination of point doublings gives a significant speed up when performing point multiplication on this type of curve. For the example programs, the NIST k163 Koblitz curve was selected over $\mathbb{F}_{2^{163}}$. Ordinary binary curves (e.g NIST b163) can be also applied, if required. Nevertheless, Koblitz curves are recommended for constrained embedded devices due to better performance. The random point multiplication was implemented using the window TNAF method. The size of the window was selected in an experimental way. The best trade-off between speed and memory was achieved using a window size of 4 bits. This results in a precomputation of 4 points in RAM at the beginning of every point multiplication calculation (online precomputation).

ECDH and ECDSA protocols require multiplication by a scalar of a fixed base point on the selected curve in the first phase of each scheme. This can be carried out more quickly using precomputation. The example programs implement a fixed point multiplication method using additional storage to accelerate the calculations. With this, approach point multiplication is a tradeoff between memory space and computation time. For the

fixed point multiplication in the binary field, the comb method (Algorithm 6) was used. Precomputation was performed with window size $w = 4$ resulting in 16 elliptic curve points stored in ROM (off-line precomputation). The precomputed points were generated automatically by a special utility program based on the curve parameters and the generator point. This approach provides additional support for curves and domain parameters which are not standardized.

The overall performance of point multiplication depends not only on the algorithm but also on the efficiency of the curve operations such as point addition and doubling. The selection of the point coordinate system has a big influence on the performance of the above mentioned operations. For binary curves, point addition and doubling operations were implemented using López-Dahab coordinates, as they require the least number of field multiplications and squarings in $\mathbb{F}_{2^m}$ (see Table 4.1).

In case of the prime field $\mathbb{F}_p$ it has been shown that projective Jacobian coordinates $(X/Z^2, Y/Z^3)$ are more efficient than Affine coordinates (x, y), when it comes to point doubling operation. Rules for point doubling in Affine coordinates require inversion in the underlying field, which is usually much more expensive than multiplication. The same operation in projective coordinates uses a greater number of cheaper multiplications and squarings in place of an expensive inversion. On embedded devices, field multiplication is much faster than inversion (especially when implemented in assembly language). Hence it is beneficial to replace the standard Affine coordinates. For curves defined over prime fields, point addition is fastest using mixed Affine-Jacobian coordinates (Table 4.1).

When using elliptic curves defined over prime fields, significant performance optimizations can be achieved with pseudo-Mersenne primes. Reduction modulo a pseudo-Mersenne prime can be performed by a few modular multiplications and additions without any division operation. As a result, the time for modular reduction can be reduced significantly. For the example programs, a $y^2 = x^3 - 3x + 157$ curve was selected with $p = 2^{160} - 2^{112} + 2^{64} + 1$. Selecting the curve parameter a as -3 minimizes the number of field multiplications and squarings for point doubling. The random point multiplication on the above curve was performed using the window NAF method (Algorithm 5) with a sliding window of size 4. Fixed point multiplication over $\mathbb{F}_p$ was implemented using the same comb method as in the case of binary curves.

4.3.2.2 Finite field arithmetic optimization

Modular arithmetic routines are fundamental operations in every elliptic curve system. The overall performance of Elliptic Curve Cryptography depends greatly on the speed of those primitives. Optimal techniques for performing finite field arithmetic on low-end processor are presented in Chapter 3. This section summarizes only the methods used in the example programs, which demonstrate the performance of NanoECC in WSNs.

In $\mathbb{F}_p$, big integer multiplication and reduction modulo p of the result are the most time-critical operations and must be performed as quickly as possible. On small processors, multiprecision multiplication of large integers not only involves arithmetic operations, but also a significant amount of data transport to and from memory. To limit the number of unnecessary data accesses the improved hybrid method was implemented on the target platforms.

This new multiplication method minimizes the number of operations on memory and uses additional CPU registers for catching and storing the carry bits. The improved hybrid method takes advantage of extra registers to avoid unnecessary load operations and becomes more efficient with the number of registers used. The method can be also used to accelerate squaring in $\mathbb{F}_p$. The implementation details of the improved hybrid method on several embedded processors can be found in Section 3.3.2.1. For modular reduction, fast algorithm was implemented that takes advantage of a special form of the modulus $p = 2^{160} - 2^{112} + 2^{64} + 1$ (Algorithm 1). In the case when p is not of a special form the Montgomery reduction can be applied. NanoECC is optimized for speed. Therefore the basic arithmetic operations in $\mathbb{F}_p$ such as multiplication, squaring, reduction, addition and subtraction were all implemented using the assembly language of a given hardware platform. On the MICA2 mote, which is clocked at 7.3828MHz, 160-bit integer multiplication was performed in 0.36ms. The same operation on the Tmote Sky platform took 0.21 ms at 8.192MHz.

The field $\mathbb{F}_{2^m}$ is usually constructed using a polynomial basis representation. In this case, binary polynomials multiplication and reduction modulo an irreducible binary polynomial are the crucial operations. For the C implementation of the polynomial multiplication, the Karatsuba-Ofman technique was adopted. This divide-and-conquer technique reduces the code size and the multiplication complexity by using word size polynomial multiplication and extra additions (which are very fast in $\mathbb{F}_{2^m}$). If more memory is avail-

able, multiplication can be implemented in assembly language. NanoECC includes an assembly implementation of the comb method for polynomial multiplication (Algorithm 2) with window size $w = 4$. In order to speed up the reduction routines on both platforms, a pentanomial $f(z) = z^{163} + z^7 + z^6 + z^3 + 1$ was used as the irreducible polynomial. Based on the specific form of $f(z)$, a fast reduction algorithm was implemented (similar to Algorithm 3). The squaring of binary polynomials was implemented using the technique described in Section 3.4.2.

4.3.2.3 Integration with TinyOS

In order to check the performance of the NanoECC library on standard sensor nodes, an example Elliptic Curve Diffie Hellman program was developed for the TinyOS operating system [77]. TinyOS is an open source OS written in nesC language [51]. It features a component-based architecture which enables quick development time and reuse of already existing modules, such as sensor drivers, distributed services, data acquisition tools, and network protocols. Although TinyOS is open source and thus it is possible to modify its internal components, NanoECC can be integrated without any kernel modifications. The library is written almost entirely in C and can be integrated with any embedded operating system (such as Contiki [38] or Mantis [12]). TinyOS was chosen as the target OS due to its popularity and strong support from the research community.

When developing for TinyOS, it is possible to use the build-in standard components to create fully functional network applications. TinyOS is especially useful when developing applications for different hardware platforms. The use of nesC and TinyOS permits the same applications to run on completely different devices. Porting the code from one device to another is a lot easier, with unified components such as timers, I/O interfaces, LED's, etc. This approach hides most of the hardware dependencies for different platforms and simplifies the programming.

The example Elliptic Curve Diffie Hellman application was written in the nesC language and used the generic TinyOS network stack. Figure 4.5 shows the configuration of the Ecdh2m application, which implements the ECDH key exchange algorithm in WSNs. Each node in Figure 4.5 represent a different application component. The edges represent interface wiring and triangles are labeled with the corresponding interface name. The following describes the main components used in the ECDH program:

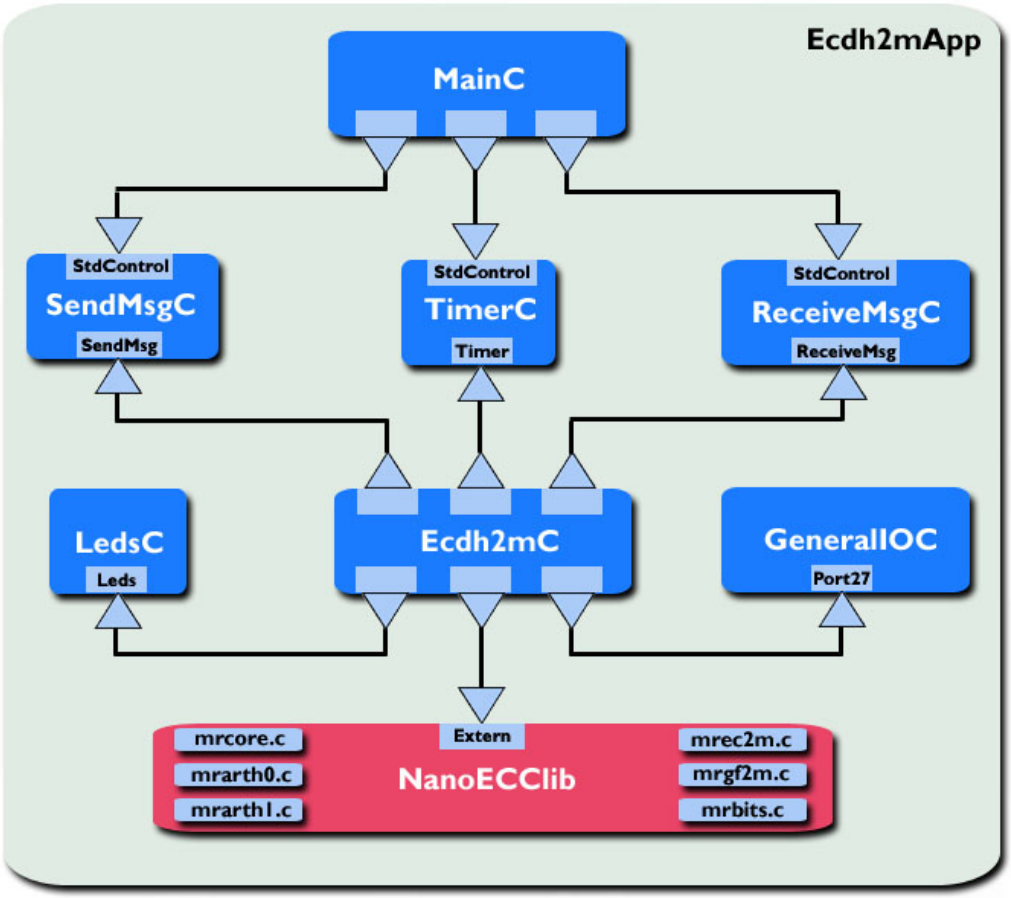

Figure 4.5: *An example ECDH program based on NanoECC implemented in TinyOS*

Ecdh2mApp. Is the overall application configuration file. It defines the wiring between particular components in the entire program.

Ecdh2mC. Is the core component of the program, which implements the actual ECDH protocol over the binary finite field. This component is connected with all the other modules which provide the required functionality. It generates a private/public key pair and processes the received packets (containing public keys of other nodes). It also embeds the node's public key in the messages that are sent over the radio interface. Finally it calculates the shared secret key between two nodes.

MainC. This component is executed first in a TinyOS application. Its common StdControl interface is used to initialise and start the connected TinyOS components.

SenMsgC. Provides multiple abstractions of message senders. It handles the sending of the messages defined by the Ecdh2mC component (which contain the node's public key).

ReceiveMsgC. Provides the interface for receiving packets. It forwards the decrypted packet to the Ecdh2mC component.

TimerC. This component creates multiple instances of timers which can be used by application components. The timers are used to set the time of different events in the system.

LedsC. Controls the mote's leds, which are used for visual notification of particular events such as sending the message over the radio or calculating the shared key. This component is not necessary for proper functioning of the program.

GeneralIOC. Provides the interface to control different input/output operations. In this implementation it is used to set the voltage on a particular pin of the device (port27). This pin is used to pass the trigger signals to the measuring equipment. This component was used during application testing.

NanoECClib. Contains all the cryptographic modules required for the ECDH protocol. It is build out of several components. The first one `mrcore.c` includes library initialization code. The next two `mrarth0.c` and `mrarth1.c` perform large integers arithmetic. The module `mrbits.c` generates random big numbers, and finally `mrec2m.c` and `mrgf2m.c` provide arithmetic operations over $\mathbb{F}_{2^m}$.

4.3.3 NanoECC performance evaluation

The example ECDH programs were compiled under the TinyOs operating system and run on the MICA2 and Tmote Sky motes, so all measurements were taken on actual devices. The three most important parameters for sensor nodes were measured: computation time, memory usage and energy consumption. The devices had to be slightly modified to facilitate data acquisition. A precise one ohm resistor was soldered between the mote and its battery pack to measure the exact amount of current drained during program execution. Input/Output ports on MICA2 and Tmote Sky were used to pass trigger signals to the measuring equipment. The details concerning the energy measurements and the experimental setup are described in Appendix A.

The above way of obtaining measurements achieves exact timings and precise power consumption information without using the mote's timers and other features which increase computation overhead. Figure 4.6 shows an example graph of Tmote Sky voltage

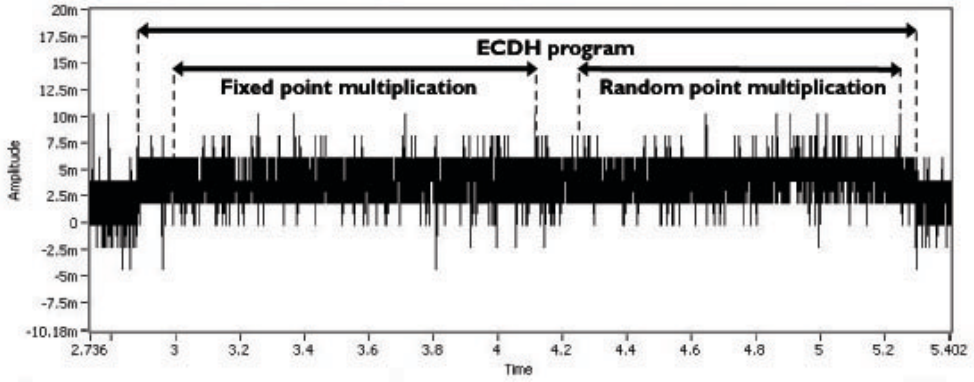

Figure 4.6: *Voltage levels on Tmote Sky during example ECDH program execution*

levels during the ECDH program execution. The calculation of the fixed point multiplication and the random point multiplication takes almost the whole time required for the ECDH protocol (92%). Therefore the performance evaluation of NanoECC is focused on the above operations.

4.3.3.1 Point multiplication results

The cost of point multiplication in terms of basic operations on the curve and in the finite field is presented in Table 4.3. All the values were taken as an average over 100 point multiplications, because operation counts differ slightly depending on the value of k in kP. The number of operations was exactly the same in the case of both sensor nodes. The figures for point additions and doublings depend on the chosen point multiplication algorithm. The numbers for field operations depend on the chosen coordinate system and the number of point additions/doublings. As seen in Table 4.3, fixed point multiplication requires a similar number of operations over both $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$. This is due to the fact that for both fields the same comb multiplication method was used. The advantage of the binary field is evident in case of the random point multiplication. The use of a Koblitz curve and the window TNAF multiplication method eliminates point doublings and at the same time significantly reduces the number of necessary field multiplications. Random point multiplication over $\mathbb{F}_p$ requires more field operations as the window NAF method performs a point doubling in every iteration of the loop (see Algorithm 5).

The results for the point multiplication operations over the binary and the prime field on MICA2 and Tmote Sky nodes are listed in Table 4.4 and Table 4.5 respectively. Precomputation was used in all cases to speed up the point multiplication routines. In order to illustrate the difference in performance, C implementations are listed separately from

Table 4.3: *Cost of point multiplication in basic operations on sensor nodes*

	Fixed point multiplication		**Random point multiplication**	
Field	$\mathbb{F}_{2^m}$, $m = 163$	$\mathbb{F}_p$, p-160 bits	$\mathbb{F}_{2^m}$, $m = 163$	$\mathbb{F}_p$, p-160 bits
Point additions	36	39	35	36
Point doublings	40	38	0	162
Field multiplications	450	463	250	940
Field squarings	340	270	643	742

Table 4.4: *Performance evaluation of point multiplication on MICA2 mote*

	Fixed point multiplication				**Random point multiplication**			
Field	$\mathbb{F}_{2^m}$, $m = 163$		$\mathbb{F}_p$, p-160 bits		$\mathbb{F}_{2^m}$, $m = 163$		$\mathbb{F}_p$, p-160 bits	
Language	C	C+asm	C	C+asm	C	C+asm	C	C+asm
Timing [s]	2.97	0.66	1.20	0.60	2.16	0.87	2.59	0.99
Current [mA]	7.86	7.86	7.88	7.88	7.86	7.86	7.88	7.88
Energy [mJ]	70.0	15.6	28.4	14.3	50.9	20.5	61.2	23.4
ROM [KB]	32.4	38.7	57.8	40.1	32.4	38.7	57.8	40.1
RAM [KB]	1.7	2.1	1.8	1.9	1.7	2.1	1.8	1.9

the mixed C/assembly language implementations. Point addition and point doubling operations were not considered independently as their computation time is insignificant comparing to point multiplication.

For both platforms the results for fixed and random point multiplication over $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$ are comparable. These results question the common opinion that Elliptic Curve Cryptography over binary fields has much lower performance on constrained devices. Actually, when looking at the results of random point multiplication, binary fields yield even better performance on sensor nodes due to the use of Koblitz curves. Fixed point multiplication is faster than random point multiplication in most cases, especially when assembly routines are used. It is important to note that using a Koblitz curve makes off-line precomputation less efficient and in some cases it might by better to use regular point multiplication instead. The use of a mixed C/assembly language implementation significantly improves the performance of all point multiplication operations without increasing the code size in most cases. For multiplication in the prime field, the introduction of assembly routines also decreased the library size. Both point multiplication routines on each platform can be calculated in under one second when using arithmetic routines written in assembly language. The experiments showed also that 75-90% of the time re-

Table 4.5: *Performance evaluation of point multiplication on Tmote Sky mote*

	Fixed point multiplication				**Random point multiplication**			
Field	$\mathbb{F}_{2^m}$, $m = 163$		$\mathbb{F}_p$, p-160 bits		$\mathbb{F}_{2^m}$, $m = 163$		$\mathbb{F}_p$, p-160 bits	
Language	C	C+asm	C	C+asm	C	C+asm	C	C+asm
Timing [s]	1.21	0.53	0.74	0.45	1.04	0.65	1.15	0.72
Current [mA]	3.45	3.45	3.68	3.68	3.45	3.45	3.68	3.68
Energy [mJ]	12.5	5.49	8.17	4.97	10.8	6.73	12.7	7.95
ROM [KB]	32.1	33.2	36.9	31.3	32.1	33.2	36.9	31.3
RAM [KB]	1.8	2.1	1.7	1.9	1.8	2.1	1.7	1.9

quired for the Elliptic Curve Diffie Hellman protocol is spent on performing modular multiplications and squarings. This fact highlights once again the importance of efficient arithmetic routines in the base field.

The values for current in Tables 4.4 and 4.5 were measured based on the voltage levels on each device. The energy consumption was calculated as the multiplication of current, voltage (3V using two AA batteries) and program execution time. When calculating field multiplication on the Tmote Sky node, the hardware multiplier on the MSP430 CPU was used to improve the performance. This fact also influenced the average current consumption, which was slightly higher when the multiplier unit was turned on. Operations in the binary field did not require the hardware multiplier. On the MICA2 the average current drawn was almost the same using both fields. The energy consumption was decreased by 40-75% on both platforms when mixed C/assembly language implementations were used. In all experiments the Tmote Sky node was more efficient than MICA2 mote in terms of power consumption. In some cases it used five times less energy for the same work carried out.

The program size figures given in Tables 4.4 and 4.5 include only the NanoECC implementation without counting additional storage for TinyOS modules. The numbers for the RAM memory requirement were not taken directly from the TinyOS output, because they did not include stack usage. Simulation environments such as AVR Studio and IAR Embedded Workbench for MSP430 were used to achieve precise information about RAM usage and stack size at any given time during ECDH programs execution. The figures for RAM in the above tables show the maximum stack usage for particular programs. The average RAM utilization was usually much lower.

4.3.3.2 Performance comparison

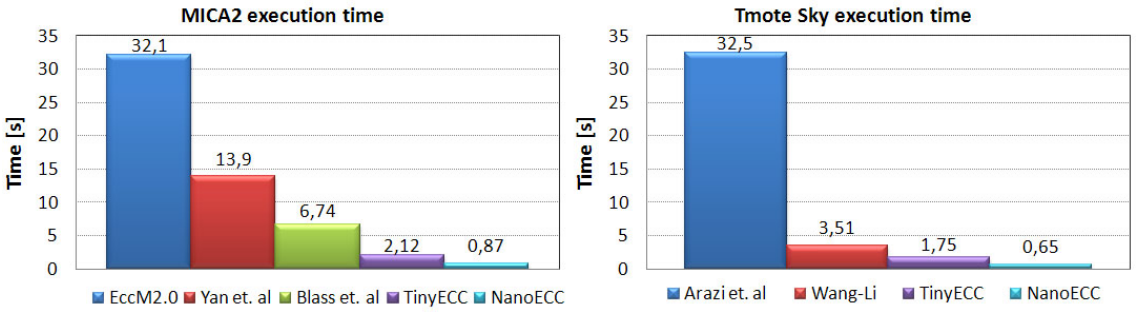

Figure 4.7: *Point multiplication timings comparison on MICA2 and Tmote Sky*

The performance of NanoECC on MICA2 and Tmote Sky nodes compares favourably with the results presented in the literature (Arazi et al [3], Blaß et al [15], EccM2.0 [85], TinyECC [80], Wang-Li [118], Yan *et al* [125]). Figure 4.7 compares the execution time of random point multiplication using different implementations. In all cases the size of the finite field elements was 160 or 163-bits (except from [15] where $\mathbb{F}_{2^{113}}$ was used). On both platforms, NanoECC performs the fastest point multiplication from all available implementations. In [57], Gura *et. al* calculated the scalar multiplication on a `secp160r1` curve in 0.81s on the ATmega128 processor (CPU used in the MICA2 node). However, the implementation was performed entirely in assembly language and is not clear how well it can be integrated into sensor network applications. The performance of many libraries over $\mathbb{F}_{2^m}$ ([85], [3], [125], [15]) is relatively low when compared with NanoECC. The main reason why NanoECC is the fastest among software implementations is that it applies the window TNAF method for point multiplication and implements efficient base field arithmetic in assembly language.

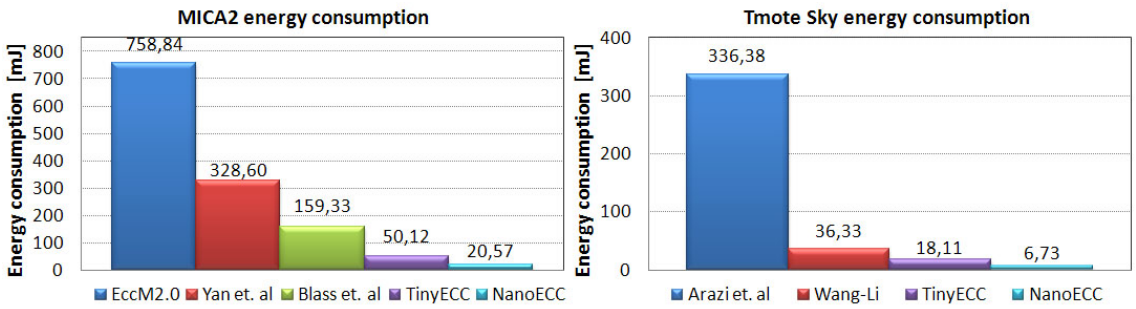

Figure 4.8: *Point multiplication energy comparison on MICA2 and Tmote Sky*

The difference in performance between NanoECC and other Elliptic Curve Cryp-

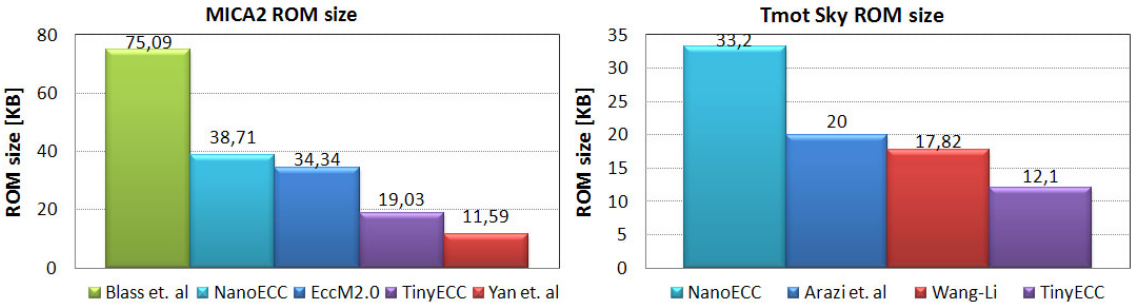

Figure 4.9: *Point multiplication ROM size comparison on MICA2 and Tmote Sky*

tography implementations is even more visible in the case of the energy consumption (Figure 4.8). Faster execution time of random point multiplication in NanoECC achieves significant savings in energy usage. On the Tmote Sky node, NanoECC can compute 50 point multiplications using the same energy as one multiplication in [85]. In comparison with TinyECC, the implementation presented here is around 2.5 times more energy efficient on both hardware platforms. The energy efficiency of NanoECC permits the calculation of many public key operations on sensor devices and extends the overall lifetime of the network.

Figure 4.9 illustrates the code size of different Elliptic Curve Cryptography implementations on sensor nodes. On both platforms, NanoECC takes a considerable amount of ROM. Because asymmetric systems are much slower than symmetric systems, the main focus for NanoECC was on speed rather than code size. However, the size of the library on both devices can be significantly reduced by manual removal of unnecessary functions. TinyECC has a smaller memory footprint than NanoECC, but was written in nesC and cannot be easily implemented on embedded devices that run operating systems other than TinyOS. NanoECC is not dependent on the operating system, hence it can be used in various embedded systems. On the MICA2 node the standard version of NanoECC takes a similar amount of memory to EccM2.0, which is a much slower implementation. The code size is not a big problem on the MICA2 platform, because 38.71KB accounts for 30% of the total space. The library size may be more problematic in case of the Tmote Sky node which has only 48KB of ROM. In this situation it would be beneficial to optimize more on size at a cost of slightly lower performance and loss of some functionality.

Different Elliptic Curve Cryptography implementations on sensor nodes have vari-

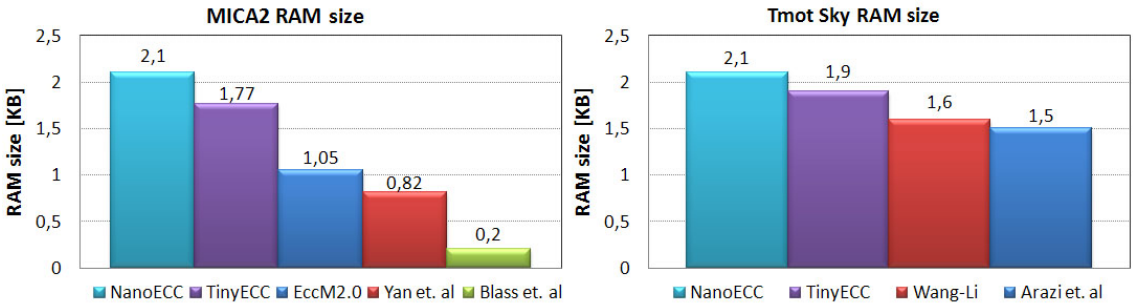

Figure 4.10: *Point multiplication RAM size comparison on MICA2 and Tmote Sky*

ous RAM size requirements. Figure 4.10 summarizes RAM usage of the current elliptic curve cryptography libraries developed for WSNs. NanoECC has similar RAM requirements to the second fastest implementation (TinyECC). In addition, the RAM size for NanoECC also includes the maximum stack usage during the program execution. In other implementations, the value for RAM corresponds only to the `data` and `bss` segments of the memory. This numbers are usually much lower than the total RAM usage. One such example is visible in [15], where the authors state that their implementation use only 200B in RAM. NanoECC uses 2.1KB of RAM on both platforms. On the Tmote Sky node this constitutes 21% of the overall memory. In the case of the MICA2 mote, RAM is limited to only 4KB which corresponds to 52% of total space. If the sensor application requires more RAM, then NanoECC can use a smaller window or even perform point multiplication without online precomputation. This of course has an impact on performance and would increase energy consumption (longer execution time).

4.4 Summary

The possibility of using Public Key Cryptography on resource-constrained devices is very promising for wireless sensor networks. Public Key Cryptography can provide secure broadcasts with digital signature schemes and authenticated key exchange algorithms. All these services are highly desirable in a WSN environment to improve the overall level of security. The main obstacle in using asymmetric cryptography in WSNs is the high resource utilization of standard public key protocols. It is certainly possible to use dedicated hardware accelerators for Public Key Cryptography primitives on WSN platforms. However none of the sensor devices currently available on the market supports public key operations in hardware, so it makes sense to explore different software ap-

proaches for Public Key Cryptography support.

One such alternative method is Elliptic Curve Cryptography, which requires considerably less resources than standard techniques based on RSA. Despite this advantage, the implementation of Elliptic Curve Cryptography in WSNs is still challenging and requires many optimization. This chapter presents the implementation of basic Elliptic Curve Cryptography primitives on different sensor platforms. It presents also a short mathematical background on elliptic curves and curve arithmetic. Of all operations on elliptic curves, point multiplication is the most important one in Elliptic Curve Cryptography. Scalar multiplication is the basic building block not only for standard Elliptic Curve Cryptography protocols (ECDH, ECDSA) but also for more advanced security schemes, which use identity based cryptography. Therefore efficient implementation of this operation is crucial to the overall performance of Public Key Cryptography schemes in wireless sensor networks.

There are many Elliptic Curve Cryptography software implementations on sensor nodes, but most of them have significant limitations and low performance. Many of the implementations were developed with fixed parameters as independent applications, that cannot be easily ported to different hardware architectures. As a result, developers may found it difficult, and sometimes impossible, to integrate such libraries with existing WSN applications. Moreover, the point multiplication operation can be optimized in many ways to significantly reduce the execution time and energy consumption. The results presented in existing implementations vary a lot from each other and it is not clear which optimizations should be used and how they should be combined to achieve the best performance.

This chapter presented the NanoECC library, which solves the shortcomings of existing Elliptic Curve Cryptography implementations in sensor networks. This library is specifically optimized for different sensor platforms and can be flexibly configured for easy integration with existing WSN applications. The code is portable and can be integrated with different embedded operating systems. The evaluation results of NanoECC give a clear answer to the question of how long Elliptic Curve Cryptography primitives take on standard WSN nodes. The library revisits also for the first time the common assumption that Elliptic Curve Cryptography over prime fields is much more efficient than over binary fields. The fastest point multiplication results were achieved on Koblitz

curves over the binary field. Efficient implementation of the windowed TNAF algorithm gives the best performance of the random point multiplication on all sensor network platforms. Therefore binary fields are recommended for elliptic curve cryptography implementations on low-end architectures.

NanoECC has the best performance among existing implementations on different sensor devices. There are libraries which have a smaller memory footprint than NanoECC, but they are not as versatile and have much more limited functionality. The high speed of the library is achieved through efficient implementation of point multiplication algorithms and arithmetic routines optimization in assembly language. The results presented in this chapter showed that Elliptic Curve Cryptography is not only viable on sensor devices, but is in fact attractive for different WSN platforms. Using NanoECC it is now more practical to create Public Key Cryptography-based security solutions on resource constrained nodes such as MICA2 and Tmote Sky.

CHAPTER 5

Pairings for Embedded Devices

Point multiplication is an important operation on elliptic curves and is a basic building block for many Elliptic Curve Cryptography protocols. In the same way that scalar multiplication is the base for Elliptic Curve Cryptography, pairings are the main building block in Identity-Based Cryptography and other pairing-based cryptosystems. Cryptography using pairings (PBC) is an emerging field related to Elliptic Curve Cryptography which has been attracting the interest of the international cryptography community, since it enables the design of original cryptographic schemes, and makes well-known cryptographic protocols more efficient. Pairing-Based Cryptography is a fairly new idea in cryptography, but has already gained a lot of attention.

One of the first usees of pairings to build cryptosystem is due to Joux [67]. He proposed a simple three-party one-round key agreement protocol which was based on bilinear pairings. After the publication of [67] researchers started to further investigate the use of pairings in cryptography. In 2001 Boneh and Franklin [18] proposed an identity-based encryption scheme based on the Weil pairing. Another important application of pairings was the discovery of a short signature scheme by Boneh, Lynn and Shacham [19]. Since then, many research papers have been published on the design of cryptographic protocols using pairings. Pairings have been accepted in the cryptographic community as an interesting tool to design security protocols. The creation of the IEEE P1363.3 standard [65] may serve as a proof that the technology is viable and can be used in practical ap-

plications. The standard defines different pairing-based cryptographic techniques and is currently available in a draft format.

The calculation of cryptographic pairings is the most computationally intensive operation in Identity-Based Cryptography schemes. Therefore efficient implementation of bilinear pairings is the key to practical identity based systems. It can be achieved by using:

- suitable "pairing friendly" elliptic curves;
- fast finite field arithmetic routines;
- efficient pairing algorithms.

The choice of appropriate parameters for pairing-based systems is even more complicated than in the case of elliptic curve cryptography. There are several types of pairing-friendly curves with various embedding degrees and different algorithms for calculating the pairing function. It is not at all obvious which set of parameters is the best for any given application. In many cases the code for one type of pairing cannot be re-used to implement a bilinear mapping of a different type. The optimal pairing to use depends not just on the security level, but also on the protocol to be implemented. For maximum efficiency, each implementation must be highly optimized according to its parameters. This is true especially in the case of constrained embedded devices.

There has been a tremendous amount of work on the realization and efficient implementation of bilinear pairings on standard computers. Although much research has been carried out on Pairing-Based Cryptography, very little attention has focused on implementing those operations on resource-constrained devices. Apparently, software implementations of pairings were considered as too heavyweight for WSN nodes and the application of Pairing-Based Cryptography to sensor networks have not been investigated so far.

This chapter presents the first in-depth study on the application of pairing-based cryptography to wireless sensor networks. The first two sections introduce the mathematics behind bilinear pairings and explain different pairing types. After that the security parameters of Pairing-Based Cryptography are discussed together with various pairing algorithms. Finally Micro-pairings is presented which is an efficient implementation of pairings on a range of sensor network platforms. Micro-pairings was designed in order

to enable different identity-based cryptography schemes in WSNs. The evaluation results show that certain pairings can be calculated in a short amount of time, even on the most constrained sensor devices. Micro-pairings prove that software implementations of pairings are viable in WSNs and can be used to build complete Public Key Cryptography cryptosystems.

5.1 Mathematical background

The implementation of cryptographic pairings is more complicated than the implementation of basic Elliptic Curve Cryptography primitives. Hence it is important to introduce the main mathematical concepts behind Pairing-Based Cryptography. The best known implementations of bilinear pairings (e.g. Weil pairing) involve fairly complex mathematics. However, most of the pairings can be dealt with abstractly, using only the group structure and mapping properties. The abstract notation of bilinear maps is sufficient to build complete Pairing-Based Cryptography security protocols. This section explains only the basic issues related to pairings. More detailed information on the topic can be found for example in [83] or [48].

5.1.1 Bilinear pairings

A pairing is a bilinear map between two groups. The Weil or Tate pairings on elliptic curves are examples of such a map. A bilinear pairing can be defined as follows. Let r be a positive integer. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be additively-written groups of order r with identity element $\mathcal{O}$, and let $\mathbb{G}_T$ be a multiplicatively-written group of order r with identity 1.

A bilinear pairing is a computable, non-degenerate function

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T. \tag{5.1}$$

The map must satisfy the following three properties:

Bilinear: We can say that a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is bilinear if

$$e(aP, bQ) = e(P, bQ)^a = e(aP, Q)^b = e(P, Q)^{ab} \tag{5.2}$$

for all $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$ and all $a, b \in \mathbb{Z}_r$.

Non-degenerate: For a given point $Q \in \mathbb{G}_1$, $\hat{e}(Q, R) = 1$ is the identity element in $\mathbb{G}_T$ for all $R \in \mathbb{G}_1$ if and only if Q is an identity element in $\mathbb{G}_1$. From this and bilinearity, we can find that if P is a generator[1] of $\mathbb{G}_1$, then $\hat{e}(P, P)$ is a generator of $\mathbb{G}_T$.

Computable: There is an efficient algorithm to compute $e(P, Q)$ for any $P \in \mathbb{G}_1$ and any $Q \in \mathbb{G}_2$.

In practice, the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are implemented using a group of points on certain special elliptic curves and the group $\mathbb{G}_T$ is implemented using a multiplicative subgroup of an extension of the underlying finite field. For certain families of supersingular elliptic curves we have $\mathbb{G}_1 = \mathbb{G}_2$.

Most of the pairing applications rely on the hardness of the following problem for their security [48]: given P, aP, bP, and cP for some $a, b, c \in \mathbb{Z}_q$, compute:

$$e(P, P)^{abc}. \tag{5.3}$$

This computational problem is known as the Bilinear Diffie-Hellman Problem (BDHP). The hardness of the BDHP depends on the hardness of the Diffie-Hellman problems (DHP) both on $E(\mathbb{F}_q)$ and in $\mathbb{F}_{q^k}$. The DHP on $E(\mathbb{F}_q)$ is a problem of calculating abP when P, aP and bP are given for some values $a, b \in \mathbb{Z}_r$. If the DHP in $\mathbb{G}_1$ can be efficiently solved, then one could solve an instance of the BDHP by computing abP and then $e(abP, cP) = e(P, P)^{abc}$. Therefore the main hard assumption in Pairing-Based Cryptography is considered to be just as hard as the DHP in $\mathbb{G}_1$ and $\mathbb{G}_T$.

The most important parameters in Pairing-Based Cryptography are the base field size q, the group size r and the embedding degree k. A subgroup $\mathbb{G}$ of $E(\mathbb{F}_q)$ is said to have an embedding degree k with respect to r if k is the smallest integer such that $r \mid q^k - 1$. Embedding degree is an important parameter in pairing based cryptography, which decides on the security of the whole system. This parameter is often denoted as the security multiplier. In general k has to be small for the pairing implementation to be practical.

5.1.2 Pairing types

The properties of a particular pairing depends on the selected groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$. In general pairings can be divided into three basic groups [49]:

[1] element P is a generator of the group when written multiplicatively, every element of the group is a power of P (a multiple of P when the notation is additive).

- Type 1 pairing where $\mathbb{G}_1 = \mathbb{G}_2$;
- Type 2 pairing where $\mathbb{G}_1 \neq \mathbb{G}_2$ but there is an efficiently computable homomorphism $\phi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$;
- Type 3 pairing where $\mathbb{G}_1 \neq \mathbb{G}_2$ and there are no efficiently computable homomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$.

Each pairing type has slightly different properties. It is not possible to simultaneously have all the frequently-used properties using just one type of pairing. The distinction into types is relevant for the design of cryptographic schemes. Some protocols are more suited to particular types of pairings. In practice, Type 2 pairings (denoted as $\check{e}(P,Q)$) are not very useful, because it is not possible to hash efficiently onto elements in the group $\mathbb{G}_2$. In addition, P and Q must be manipulated as points over the full extension field $\mathbb{F}_{q^k}$. These features makes Type 1 and 3 pairings preferred over Type 2 pairings.

From the implementation point of view, Type 1 pairings exist only on supersingular elliptic curves (see Section 4.1.1.1). For all supersingular curves, the embedding degree is one of the values $k = \{1, 2, 3, 4, 6\}$. If the supersingular curve is defined over a prime field $\mathbb{F}_q$ with $q > 3$, then $k = 1$ or $k = 2$. All supersingular elliptic curves with $k = 4$ are defined over characteristic two finite fields, while those with $k = 6$ are defined over characteristic three finite fields. The choice of k is very restricted on supersingular curves, and limits the number of possible pairing-friendly curves.

On supersingular curves, $\mathbb{G}_1 = \mathbb{G}_2$ and a distortion map Ψ exists which maps a point from $E(\mathbb{F}_q)$ to $E(\mathbb{F}_{q^k})$. Hence, a Type 1 pairing can be defined as $\hat{e}(P,Q) = e(P, \Psi(Q))$, where P and Q belong to the curve $E(\mathbb{F}_q)$. For Type 1 pairings, P and Q are linearly dependent, but P and $\Psi(Q)$ are not, hence $\hat{e}(P,P) \neq 1$. Such pairing have also an additional property of symmetry, namely $\hat{e}(P,Q) = \hat{e}(Q,P)$. When $Q = xP$, then $\hat{e}(P,Q) = \hat{e}(P,xP) = \hat{e}(P,P)^x = \hat{e}(xP,P) = \hat{e}(Q,P)$. This is a useful property that can be used to simplify security protocols.

There is an alternative to the supersingular elliptic curves, with their limited choice of embedding degree k. Special families of non-supersingular elliptic curves $E(\mathbb{F}_p)$ can also be pairing-friendly. The condition for being pairing-friendly with an embedding degree of k, is that k is the smallest positive integer such that $r|(p^k - 1)$, and there is no reason that non-supersingular curves cannot be found to meet this condition. Pairings evaluated on

such curves belong to Type 3 pairings. One of the features of non-supersingular curves is that they do not support the distortion map Ψ. Hence all Type 3 pairings do not have the property of symmetry, $e(P,Q) \neq e(Q,P)$. In $e(P,Q)$ point P can be defined on $E(\mathbb{F}_p)$ but Q cannot be set on the same curve. The best that can be done is to put Q on a lower order twisted curve namely $E'(F_{p^{k/w}})$, where w is at least 2 for even k. Despite this inconvenience, Type 3 pairings can be still used to build complete pairing-based cryptosystems.

For most non-supersingular elliptic curves, the embedding degree k is large and $k \approx r$. Hence it is not possible to find suitable pairing-friendly curves by random selection. Fortunately, there are techniques that allow generation of ordinary elliptic curves with low embedding degree. All these techniques are based on the complex multiplication method [87]. In [90] Miyaji, Nakabayashi and Takano used the complex multiplication method to generate MNT curves of low embedding degree where $k = \{2, 4, 6\}$. In 2005, Barreto and Naehrig (BN) [9] discovered a method for constructing elliptic curves E of prime order ℓ over prime fields $\mathbb{F}_p$ with embedding degree $k = 12$. Finally, Cocks and Pinch [48] described a method for generating elliptic curves for any embedding degree. However, one downside of this method is that curves have $p \approx r^2$, which results in a large base field $\mathbb{F}_p$. There are also many other families of ordinary elliptic curves that are suitable for Pairing-Based Cryptography and it is not difficult to find curves that match particular security levels. A comprehensive taxonomy of pairing friendly elliptic curves can be found in [47].

5.1.3 Pairing algorithms

There is only one known setting for cryptographically useful bilinear maps, namely elliptic curves with an efficiently computable Weil or Tate pairing. It has been shown that the Tate pairing is simpler and more efficient then the Weil pairing [48]. Therefore this chapter is focused only on implementing the Tate pairing on embedded devices.

The Tate pairing, denoted $e(P,Q)$, where the points P and Q are linearly independent points on an elliptic curve $E(\mathbb{F}_{q^k})$, evaluates as an element of an extension field $\mathbb{F}_{q^k}$. If P is of prime order r, then the pairing evaluates as an element of order r. The Tate pairing can be calculated using different algorithms. The most basic one is Miller's algorithm [89] (see Algorithm 7).

Miller's algorithm consists of two parts: the main loop and the final exponentiation.

Algorithm 7 Computation of $e(P, Q)$ with Miller's algorithm

INPUT: $Q \in E(\mathbb{F}_{q^k})$, $P \in E(\mathbb{F}_{q^k})$, where P has order r with $r = (r_t, \ldots, r_0)_2$ and $t = \lfloor \log_2 r \rfloor - 1$
OUTPUT: $e(P, Q)$
1: $T \leftarrow P$, $f \leftarrow 1$
2: **for** $i \leftarrow t$ **downto** 0 **do**
3: $\quad f \leftarrow f^2 \cdot \frac{l_{T,T}(Q)}{v_{2T}(Q)}$
4: $\quad T \leftarrow 2T$
5: $\quad$ **if** $r_i = 1$ **then**
6: $\quad\quad f \leftarrow f \cdot \frac{l_{T,P}(Q)}{v_{T+P}(Q)}$
7: $\quad\quad T \leftarrow T + P$
8: $\quad$ **end if**
9: **end for**
10: $f \leftarrow f^{(q^k-1)}/r$
11: **return** f

During the main loop, point P is multiplied by its group order r using a classic double-and-add line-and-tangent algorithm. The l function denotes a line through point T or points T and P. The v function marks a vertical line through point $2T$ or point $T+P$. The values of the line and vertical functions $l_{T,P}(Q)$ and $v_{T+P}(Q)$ respectively, are distances calculated between the fixed point Q and the lines that arise when adding P to T on the elliptic curve in the standard way. If the point T has coordinates (x_j, y_j), the point $T + P$ has coordinates $(x_j + 1, y_j + 1)$, the point Q has the coordinates (x_Q, y_Q), and the line through T and P has a slope of λ_j, hence:

$$l_{T,P}(Q) = (y_Q - y_j) - \lambda_j(x_Q - x_j) \tag{5.4}$$

$$v_{T+P}(Q) = (x_Q - x_{j+1}) \tag{5.5}$$

Equations 5.4 and 5.5 use affine coordinates, although in most cases projective coordinates achieve better performance. Algorithm 7 may fail if one of the intermediate lines l passes directly through Q or a vertical function v evaluates as zero. Therefore P and Q should not be chosen at random. However, this is not a concern in most pairing-based protocols, because P and Q are usually chosen as $P \in E(\mathbb{F}_q)$ and $Q \in E(\mathbb{F}_{q^k})$. Miller's algorithm has $O(\log r)$ iterations, each requiring a constant number of arithmetic operations in $\mathbb{F}_{q^k}$. In order to implement it efficiently on embedded devices, several optimizations need to be applied.

One of the basic optimizations to Algorithm 7 is to choose r to have a low Hamming weight. In this way the execution of operations in lines 6 and 7 will be limited. In some

cases (e.g. for the MNT curves) the choice of a low Hamming weight order may not be practical, in which case the optimal strategy might be to represent r in a NAF format, and use a standard windowed NAF method (similar to Algorithm 5). Further optimizations can be applied in case of particular pairing types.

5.1.3.1 Type 3 pairing optimizations

In the case of non-supersingular elliptic curves of prime characteristic ($q = p$), the embedding degree k is always chosen to be even. For even values of k, the extension field $\mathbb{F}_{p^k}$ is built as a quadratic extension on top of an implementation of $\mathbb{F}_{p^d}$ where $k = 2d$. In this case the exponent in the final exponentiation can be written as $(p^k - 1)/r = (p^d - 1)[(p^d + 1)/r]$. Elements in $\mathbb{F}_{p^k}$ can be represented as $w = a + ib$ where $a, b \in \mathbb{F}_{p^d}$. The conjugate of w is equal to $\bar{w} = a - ib$. From the Frobenius endomorphism one can derive:

$$(a + ib)^{p^d} = (a - ib) \quad \text{and} \quad (1/(a + ib))^{p^d - 1} = (a - ib)^{p^d - 1} \tag{5.6}$$

The above property can be used to eliminate the extension field divisions in the main loop of Algorithm 7 (line 3 and 6). The output of the main loop in Type 3 pairings has to be raised to the power of $p^d - 1$. However, based on equation 5.6, after raising to the power of $p^d - 1$, the inverse and the conjugate give the same result. Hence it is possible to replace expensive divisions by $v_{2T}(Q)$ and $v_{T+P}(Q)$ with cheaper multiplications by conjugates $\bar{v}_{2T}(Q)$ and $\bar{v}_{T+P}(Q)$.

Algorithm 7 can be further optimized by appropriate selection of point Q. Point $Q = (x_Q, y_Q)$ is defined on $E(\mathbb{F}_{p^k})$, where $x_Q = a + ib$ and $y_Q = c + id$ with $a, b, c, d \in \mathbb{F}_{p^d}$. By selecting $b = c = 0$ the functions $\bar{v}_{2T}(Q)$, $\bar{v}_{T+P}(Q)$ evaluate as elements in the field $\mathbb{F}_{p^d}$. These functions are eliminated by the final exponentiation, which always includes $p^d - 1$ as a factor of the exponent. This optimization is known as the denominator elimination technique [8].

Another optimization can be applied because of the group order r, which is always odd. In this case the last iteration of the main loop in Algorithm 7, always contains $r_0 = 1$. However, the last point addition results in a line value which is always eliminated by the final exponentiation. Hence this last step can be omitted. The final exponentiation itself can be also simplified. In many cases the exponent $p^d + 1$ can be further factored. For example $p^3 + 1 = (p + 1)(p^2 - p + 1)$, where $p^2 - p + 1$ is the sixth cyclotomic polynomial

$\Phi_6(p)$. Based on the above factoring, the final exponentiation can be broken down into three parts: easy exponentiations to the power of $p^d - 1$ and $(p^d + 1)/\Phi_k(p)$ and more expensive exponentiation to the power of $\Phi_k(p)/r$. All the above modifications lead to Algorithm 8 (based on [107]).

Algorithm 8 Optimized algorithm for computation of $e(P, Q)$

INPUT: $Q \in E'(\mathbb{F}_{p^d})$, $P \in E(\mathbb{F}_p)$, where P has order r
OUTPUT: $e(P, Q)$

1: $T \leftarrow P$, $f \leftarrow 1$
2: $s \leftarrow \lfloor \log_2(r-1) \rfloor$
3: **for** $i \leftarrow s-1$ **downto** 0 **do**
4: $\quad f \leftarrow f^2 \cdot l_{T,T}(Q)$
5: $\quad T \leftarrow 2T$
6: $\quad$ **if** $s_i = 1$ **then**
7: $\quad\quad f \leftarrow f \cdot l_{T,P}(Q)$
8: $\quad\quad T \leftarrow T + P$
9: $\quad$ **end if**
10: **end for**
11: $f \leftarrow f^{(p^d-1)}$
12: $f \leftarrow f^{(p^d+1)/\Phi_k(p)}$
13: $f \leftarrow f^{\Phi_k(p)/r}$
14: **return** f

In some Pairing-Based Cryptography protocols the first parameter to the pairing, point P, may be a fixed public value or a fixed private key. If P is fixed, then all of the T values (which are multiples of P) can be precomputed and stored in memory to accelerate the calculation of $e(P, Q)$. In this situation, it is preferable to use affine coordinates for all the points. This approach, however, requires a lot of space for precomputed values and should be applied only on embedded devices with a large amount of available memory.

Additional useful optimizations of Type 3 pairings can be applied when particular pairing-friendly curves are chosen. For example, when using BN [9] or MNT [90] curves, the parameter w is always greater than one. In these cases it is possible to use a truncated loop variant of the Tate pairing, which is called the Ate pairing [61]. This method is very similar to Algorithm 8, but this time points P and Q change sides. Now P is on the twisted curve ($P \in E'(\mathbb{F}_{p^d})$) and Q is on the curve over the base field ($Q \in E(\mathbb{F}_p)$). In this setting the main loop of Algorithm 8 can be truncated by a factor of w and a viable pairing can still be calculated. For example, for the BN curves, the loop has half the number of iterations than in Algorithm 8. The fact that P is over the extension field, introduces extra

computation when evaluating subsequent values of T. This, however, can be offset if the point P is fixed and by using the precomputation technique described earlier.

5.1.3.2 Type 1 pairing optimizations

Type 1 pairings can be evaluated only on supersingular elliptic curves. It has been shown that supersingular curves lead to some of the most efficient pairing implementations in terms of processing speed and bandwidth requirements [7], [8]. These features make this type of pairing especially attractive for low power environments like wireless sensor networks where available resources are very constrained. One drawback of supersingular curves is the limited choice of embedding degree k, which limits the range of security levels that can be efficiently achieved.

For the supersingular curves of low characteristic, the basic Miller algorithm can be drastically simplified by integrating the distortion map (Ψ), the point multiplication, and the Frobenius endomorphism directly into the main loop. The fastest known method to compute pairings on supersingular curves is called the η_T pairing [7]. This pairing can be evaluated very efficiently, especially in fields of characteristic 2. For the η_T pairing, the main loop is truncated to half the length of the related η pairing (which is in turn closely related to the Tate pairing), and the final exponentiation cost is small.

If $q = 2^m$, then on a supersingular elliptic curve $E(\mathbb{F}_q)$, the Tate pairing $e(P, Q)$ evaluates as an element in $\mathbb{F}_{q^k}$, where in this case k is equal to 4. The parameters P and Q are points of order r on the curve, where for the supersingular curve r is a large prime divisor of $2^m \pm 2^{(m+1)/2} + 1$. This simple group order can be used to optimize the final exponentiation operation. On a supersingular $y^2 + y = x^3 + x + B$ curve over $\mathbb{F}_{2^m}$ where $B \in \{0, 1\}$ and $m = 3 \mod 8$, the η_T pairing can be calculated using Algorithm 9.

Algorithm 9 (based on [7]) presents an optimized scheme for the calculation of the η_T pairing. Pairs (x_P, y_P) and (x_Q, y_Q) are the coordinates of points P and Q represented as binary polynomials in $\mathbb{F}_{2^m}$. Values f and g are elements evaluated over the extension field $\mathbb{F}_{2^{4m}}$ and they can be efficiently represented as polynomials with four coefficients in $\mathbb{F}_{2^m}$. Elements $s, t \in \mathbb{F}_{2^{4m}}$ can be derived from the distortion map (Ψ) which is defined as:

$$\Psi(x, y) = (x + s^2, y + sx + t) \tag{5.7}$$

The above distortion map maps a point from $E(\mathbb{F}_{2^m})$ to $E(\mathbb{F}_{2^{4m}})$.

Algorithm 9 Computation of $\eta_T(P,Q)$ on a $y^2+y=x^3+x+B$ curve over $\mathbb{F}_{2^m}$

INPUT: P, Q
OUTPUT: $\eta_T(P,Q)$
1: **let** $P=(x_P,y_P)$, $Q=(x_Q,y_Q)$
2: $u \leftarrow x_P+1$
3: $f \leftarrow u\cdot(x_P+x_Q+1)+y_P+y_Q+1+(u+x_Q)s+t$
4: **for** $i \leftarrow 1$ **to** $(m+1)/2$ **do**
5: $\quad u \leftarrow x_P, x_P \leftarrow \sqrt{x_P},\ y_P \leftarrow \sqrt{y_P}$
6: $\quad g \leftarrow u\cdot(x_P+x_Q)+y_P+y_Q+x_P+(u+x_Q)s+t$
7: $\quad f \leftarrow f\cdot g$
8: $\quad x_Q \leftarrow x_Q^2,\ y_Q \leftarrow y_Q^2$
9: **end for**
10: **return** $f^{(2^{2m}-1)(2^m-2^{(m+1)/2}+1)}$

5.2 Pairing-Based Cryptography for WSNs

As mentioned earlier, pairing based cryptography is an emerging trend in cryptography that is strictly related to Elliptic Curve Cryptography. Although much research has been carried out on pairings on standard desktop class computers, not much attention has focused on implementing those complex operations on small and constrained devices. Pairing-based systems have become more and more popular in Public Key Cryptography schemes but it may seem that these operations are far too complex to be calculated in reasonable amount of time on tiny architectures like WSN nodes.

In the literature there are some papers [33], [94] that envisage WSNs as a good application space for Pairing-Based Cryptography. However, the publications presented so far do not specify any implementation details and the viability of Pairing-Based Cryptography in WSNs remains unknown. There are only a few results available which are related to practical deployments of pairings on sensor nodes. So far the only software implementation of pairings was presented by Oliveira *et al.* in [96]. The TinyTate implementation used TinyECC as the underlying library and targeted the MICAz mote. The following sections provide details of the above implementation and discuss the security parameters required to implement pairings on sensor nodes.

5.2.1 Security parameters

The security levels in pairing-based cryptography depends on the value of the embedding degree k. Other important parameters are the group order r and the size of the underlying finite field $\mathbb{F}_q$. In general, for most pairing applications the parameters q, r

and k must satisfy the following security requirements:

1) The group order r must be large enough so that solving the ECDLP in an order-r subgroup of $E(\mathbb{F}_q)$ is infeasible (e.g. using Pollard's ρ algorithm);

2) The embedding degree k should be sufficiently large so that solving the DLP in $\mathbb{F}_{q^k}$ is infeasible (e.g. using the index-calculus method).

With the current state-of-the-art in cryptanalysis setting, r as a 160-bit prime guarantees that any attack on the Elliptic Curve Discrete Logarithm Problem will take at least 2^{80} steps. In a similar manner choosing $k \cdot \log_2 q \approx 1024$ sets a limit of minimum 2^{80} steps for any Discrete Logarithm Problem attack over the extension field. Such values for k, q and r guarantee a security level of the equivalent of 80-bits of security in a symmetric key system. Based on the above conditions, one could use a $k = 2$ supersingular curve (r 160 bits) with a 512-bit prime p and an extension field size of 1024-bits for a security level of 80-bits. Similar level of security can be also achieved using a $k = 6$ MNT curve with a 160-bit prime p and an extension field size of 960 bits (r also 160-bits long). Both implementations, however, will have different performance depending on the speed of arithmetic operations in $\mathbb{F}_q$ and $\mathbb{F}_{q^k}$ on a given platform.

Table 5.1: *Key size comparison in Pairing-Based Cryptography*

Symmetric key size	**Group size (r)**	**$\mathbb{F}_{q^k}$ size ($k \cdot \log_2 q$)**	**Embedding degree (k)**
80	160	960-1280	2-8
128	256	3000-5000	12-18
256	512	12000-18000	24-36

Usually on embedded devices, a smaller size of $\mathbb{F}_q$ results in a faster pairing, as arithmetic operations over large finite fields are very slow on constrained CPU's. On the other hand, larger values of k require more operations in higher extension fields, which can be expensive without proper optimizations. In any case, the selection of parameters for Pairing-Based Cryptography is not straightforward and should be made with a particular security protocol in mind. Table 5.1 presents a comparison of different security levels for Pairing-Based Cryptography together with the required security parameters [107]. The security level of 80 bits should be enough for the majority of sensor network applications. Therefore all the implementations presented in this chapter conform to this level.

Looking at the values in Table 5.1, it is evident that the efficiency of systems using pairings scales more like RSA rather than Elliptic Curve Cryptography. Nevertheless, unique properties of pairings makes pairing-based protocols more interesting in the context of sensor networks than the above cryptosystems.

5.2.2 TinyTate

The authors in [96] implemented the Tate pairing on a MICAz sensor node, which embeds the ATmega128 8-bit CPU. They used the following parameters in their implementation:

Finite field. The Tate pairing in TinyTate was evaluated over the prime field $\mathbb{F}_p$. The choice of the finite field was motivated by the use of TinyECC as the underlying library, which only supports operations over $\mathbb{F}_p$. The authors claimed also that the Discrete Logarithm Problem in prime fields is harder than the Discrete Logarithm Problem in binary fields.

Curve selection. TinyTate used a supersingular $y^2 = x^3 + x$ curve over the prime field $\mathbb{F}_p$ with p is a 256-bit prime.

Coordinate system. Projective coordinates were chosen instead of the affine system to eliminate the expensive inversion in point addition and doubling operations.

Pairing parameters. The main security parameters were chosen as $k = 2$, q is a 256 bit prime and r is a 128 bit prime. For r, a Solinas prime was chosen in order to decrease the number of point additions in the main loop of Miller's algorithm. The choice of $k = 2$ permitted the application of the denominator elimination optimization and made arithmetic operations in $\mathbb{F}_{q^k}$ easier to implement. The security level for the pairing implementation was relaxed in order to achieve better performance.

Table 5.2 presents the performance evaluation of TinyTate. Although the memory consumption figures are reasonable, the execution time of 30s is significant for a MICAz mote with very limited energy resources. The execution time of TinyTate is high especially given that the parameters used by the authors are below the 80-bit level of security. The group order r was set to 128 bits and the extension field size was $k \cdot \log_2 q = 512$. Both values are below the current records for breaking the Discrete Logarithm Problem [73] and below the security levels presented in Table 5.1. All these features makes TinyTate unsuitable for practical wireless sensor network applications.

Table 5.2: *TinyTate implementation results*

Tate pairing implementation on MICAz		
Execution time	**ROM size**	**RAM size**
30.21s	18.384KB	1.831KB

5.3 Micro-pairings: efficient Pairing-Based Cryptography for sensor nodes

The small number of pairing implementations on sensor nodes in the literature indicates that Pairing-Based Cryptography was considered to be too heavyweight for constrained devices. However many recent security papers in WSNs propose protocols which use pairings to secure the communication in sensor networks [33], [94], [126]. The only way to make such protocols practical is to provide efficient implementations of pairings on sensor nodes.

This section presents a comprehensive study on the application of parings to sensor nodes. It describes Micro-pairings which is an efficient implementation of different pairing types on a broad range of sensor platforms. Micro-pairings was designed in order to make existing and future Pairing-Based Cryptography protocols practical on embedded devices. The software package targets typical sensor nodes like MICA2/MICAz, Tmote Sky and Imote2. It can be also easily ported to other sensor nodes. The evaluation results of Micro-pairings prove that pairings with full-size security parameters are viable on different hardware platforms. It also shows for the first time that certain pairings can be calculated quickly and efficiently even on small and constrained devices such as MICA2 or Tmote Sky.

5.3.1 Micro-pairings overview

The main idea when developing Micro-pairings was to provide fast implementations of different pairing algorithms for a broad range of sensor platforms. The key design choices were performance, high security level and diversity in pairing types. All pairings were implemented according to the 80-bit security level from Table 5.1. The main task was to find the optimal combination of pairing type, domain parameters and arithmetic routines which gives the best pairing performance on an embedded device. Similar to

NanoECC, Micro-pairings was optimized for speed and energy efficiency rather than memory utilization. High performance of Micro-pairings was achieved through efficient implementation of pairing algorithms and fast arithmetic operations over $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$.

All the pairings implementation presented in this chapter are based on the MIRACL [109] library, which provides all of the basic tools to perform operations on elliptic curves. All memory allocation in Micro-pairings was taken directly from the stack. This means that after the pairing calculation, almost all of the RAM memory could be re-used for different purposes. The MIRACL library was optimized to work on three completely different 8, 16 and 32-bit architectures. Each of those processors has specific features that needed to be exploited in order to achieve maximum efficiency.

The code for Micro-pairings was first developed and tested on simulation environments of the different processors. For the ATmega128 processor and the MSP430 CPU, the AVR Studio 4.13 and the IAR Embedded Workbench v4.10 were used respectively. Development for the PXA271 processor was performed using the IAR Embedded Workbench v5.0 for the ARM platform. These user-friendly environments allowed the integration of the main C code with different assembly routines that accelerated arithmetic operations on a given CPU. The timings of all the programs could also be estimated with cycle accurate precision using these tools. After initial code testing and simulation, Micro-pairings was implemented on the MICA2, Tmote Sky and Imote2 nodes. The code was integrated with the TinyOS operating system in a similar way as it was performed for the NanoECC library. The overall results of these implementations are described in Section 5.3.4.

5.3.2 Type 1 pairings implementation

As mentioned earlier, Type 1 pairings can be evaluated on supersingular curves. These curves can be separated into three sub-classes: curves over binary fields ($q = 2^m$ with $k = 4$), curves over fields of characteristic 3 ($q = 3^m$ with $k = 6$) and curves over fields of large prime characteristic ($q = p$, $p > 3$ with $k = 2$). The representation of field elements in $\mathbb{F}_{3^m}$ is not very straightforward on a binary system. The case where $k = 2$ requires p to have at least 512 bits (due to security reasons). Arithmetic operations in such a big finite field would be very slow on a constrained 8-bit processor. Therefore, the most suitable curve to implement Type 1 pairings on sensor nodes is the binary curve with the embedding degree $k = 4$.

In order to achieve an adequate security level, the binary field can be chosen as $\mathbb{F}_{2^{271}}$. The equation of the supersingular curve over $\mathbb{F}_{2^{271}}$ is as follows:

$$y^2 + y = x^3 + x \tag{5.8}$$

In this case the number of points on the curve is known to be $2^{271}+2^{136}+1 = 487805 \cdot r$ where r is a large prime, large enough to make any Pohlig-Hellman-like attack [58] on the Elliptic Curve Discrete Logarithm Problem infeasible. The next thing is to consider the security of the discrete logarithm over $\mathbb{F}_{(2^{271})^4}$, where it is vulnerable to an index calculus type of attack. The current state of knowledge of the Discrete Logarithm Problem over extension fields $\mathbb{F}_{2^{m \cdot k}}$ is not well studied. However, it is believed to be roughly the same as that of the Discrete Logarithm Problem over $\mathbb{F}_{2^m}$, for prime m, where $m \approx 4 \cdot 271 = 1084$. In this setting the current record is for $m = 613$ [68], so there is (for now) a relatively wide margin of safety.

From the previous sections, it is known that the fastest pairing algorithm on supersingular curves is the η_T pairing [7]. Supersingular curves lead to more efficient implementations in terms of bandwidth, memory usage and processing speed, hence they are more suitable for wireless sensor networks. The η_T pairing was implemented according to Algorithm 9. Elements in the extension field $\mathbb{F}_{2^{4m}}$ were represented as polynomials with 4 coefficients in $\mathbb{F}_{2^m}$. So, for example, element $b(z) = B_3z^3 + B_2z^2 + B_1z + B_0$ in $\mathbb{F}_{2^{4m}}$ was represented as a vector $[B_3, B_2, B_1, B_0]$. The elements $s, t \in \mathbb{F}_{2^{4m}}$ were equal to $[0, 1, 1, 0]$ and $[0, 0, 1, 0]$ respectively. Both values were derived from the distortion map Ψ.

Looking at Algorithm 9, one can see that the most expensive part is the *for* loop which needs to be executed $(m + 1)/2$ times. All the operations inside this loop have to be optimized in terms of execution speed in order to achieve good performance of the algorithm. The most time-consuming operation inside this loop is the polynomial multiplication, and its implementation has a major impact on the overall execution time of the η_T pairing. In particular, multiplication of $\mathbb{F}_{2^{4m}}$ values (line 7) is very time-critical, but consists of multiplications in $\mathbb{F}_{2^m}$, which again emphasizes the importance of the base field multiplication. Using the Karatsuba method [74] for $\mathbb{F}_{2^{4m}}$ multiplication, decreases the number of necessary operations to nine modular multiplications and some additions which are very cheap in binary fields (just bitwise exclusive-or of the coefficients of the polynomials). Inside the main loop there are also other important operations like squar-

Table 5.3: *Cost of the η_T pairing on $y^2 + y = x^3 + x$ curve*

	ModMuls	**ModSquares**	**Square roots**
Main loop	1904	544	544
Final exp.	114	139	0

ing and calculation of the square root of the field element. However, these functions are not as complex as binary polynomial multiplication and when efficiently implemented can be performed much faster.

The last stage of the η_T algorithm is the final exponentiation. This step is not very time consuming as it can be performed for the relatively inexpensive cost of $(m + 1)/2$ extension field squarings, four extension field multiplications, one extension field division and some other cheaper arithmetic operations.

5.3.2.1 Binary field arithmetic

The total cost of the η_T pairing ($m = 271$) in terms of basic arithmetic operations is given in Table 5.3. Additions are not taken into consideration because they are fast in $\mathbb{F}_{2^m}$ (being just XOR operations). The key to efficient η_T implementation lies in the performance of binary polynomial multiplication. Therefore, Micro-pairings used assembly language routines for all the basic arithmetic operations on binary polynomials.

The binary polynomial multiplication in $\mathbb{F}_{2^{271}}$ was implemented using the optimized hierarchical method described in Section 3.4.1.1. The optimizations for particular hardware platforms were performed according to Section 3.4.1.1 as well. Other polynomial arithmetic operations like squaring, reduction and calculation of the square root were also implemented in assembly language on all three target platforms. Operations like modular reduction and calculation of the square root were strictly optimized for a specific form of the irreducible polynomial $f(z) = z^{271} + z^{201} + 1$. Reduction modulo $f(z)$ was implemented based on Algorithm 3, and squaring of binary polynomials was performed as described in Section 3.4.2. Calculation of the square root in $\mathbb{F}_{2^{271}}$ was implemented using the techniques from Section 3.4.3. Table 5.4 summarizes the performance of the main arithmetic routines in $\mathbb{F}_{2^{271}}$ on all three target processors. All values are in clock cycles of a given CPU and the multiplication and squaring routines also include the reduction operation.

Table 5.4: *Timings in clock cycles for modular arithmetic routines in* $\mathbb{F}_{2^{271}}$

	Atmega128			MSP430			PXA271		
Operation	**Mul**	**Sqr**	**Sqrt**	**Mul**	**Sqr**	**Sqrt**	**Mul**	**Sqr**	**Sqrt**
Assembly	13557	1581	1730	10147	1363	1644	4926	499	546
C code	66271	4711	12021	40666	3667	11212	13183	2375	2496
Decrease	80%	66%	86%	75%	63%	85%	62%	79%	78%

All the figures in Table 5.4 were obtained on simulation environments like AVR Studio (Atmega128) and IAR Embedded Workbench (MSP430 and PXA271). There were significant differences for the same optimization levels when different compilers were used (for example gcc and the IAR compiler). That it is why the same settings for the compilers in all cases were used. Optimization flag -O0 was set during all simulations, so the results in Table 5.4 can be directly compared with other implementations no matter which compiler is used. Results achieved with the assembly language routines are compared with a C-only implementation to show the savings in execution time.

As can be seen in Table 5.4, the difference between the standard C code functions and specially optimized assembly routines is quite significant. Handcrafted code gave a nice improvement in execution time on all tested hardware platforms. All the operations timings were decreased by between 60% and 85%. Square root computation was around four to seven times faster and, (of the most significance for the η_T algorithm), polynomial multiplication improved up to five times. Field-specific assembly code gives the maximum speed up for the η_T pairing algorithm. The timings in clock cycles for the η_T pairing together with memory occupation on all three processors are presented in Table 5.5 and Table 5.6.

Table 5.5: *Performance of the* η_T *pairing on Atmega128 and MSP430*

	Atmega128			MSP430		
	Cycles	**ROM**	**Stack**	**Cycles**	**ROM**	**Stack**
Assembly	19,660,993	47.41KB	3.17KB	14,097,304	23.66KB	4.17KB
C code	80,608,843	41.23KB	3.17KB	50,684,686	23.01KB	4.17KB
Decrease	76%	-15%	0%	72%	-3%	0%

With the introduction of specially optimized arithmetic routines, Micro-pairings calculate the η_T pairing 65-76% quicker. In the best case (the ATmega128 CPU) the execution is four times faster than in the case of the C-only implementation. It appears that careful

Table 5.6: *Performance of the η_T pairing on the PXA271*

	PXA271		
	Cycles	**ROM**	**Stack**
Assembly	6,002,134	29.55KB	4.12KB
C code	16,974,044	37.24KB	4.12KB
Decrease	65%	20%	0%

optimization of critical routines in assembly language leads to a large performance increase on embedded microcontrollers. Usually on standard desktop computers, savings of around 20-30% are possible to achieve when using assembly language.

The results presented in Tables 5.5 and 5.6 are especially significant because in almost all cases, the same level of memory usage was achieved. The memory requirements for the η_T pairing on the three platforms tested are reasonable when taking into consideration the complexity of the operations. Stack usage in all implementations remained at the same level, as assembly routines did not use any additional variables. RAM utilization may seem high, but the memory is reserved only for the duration of the pairing calculation. After that, all of that RAM memory is released and can be reused for different purposes. Stack size values presented in Tables 5.5 and 5.6 were also the peak numbers during program execution. Average stack utilization was usually 60% of those values. The increase in memory overhead is considerable only on the ATmega128 platform, but provides the best performance results. For the MSP430 processor, the 3% increase in ROM utilization is negligible, as it leads to 72% improvement in execution time. On the PXA271 microcontoller the assembly routines resulted in a 20% decrease in program code.

5.3.3 Type 3 pairings implementation

Type 3 pairings give an attractive alternative to Type 1 pairings with their limited choice of pairing-friendly curves. Special families of non-supersingular elliptic curves $E(\mathbb{F}_p)$ can also be pairing friendly, and the Tate pairing now evaluates as an element in $\mathbb{F}_{p^k}$, for some reasonable value of k. Type 3 pairings have all the properties of Type 1 pairings, except that of symmetry: $e(P,Q) \neq e(P,Q)$. However, the lack of symmetry is not a major downfall and Type 3 pairings can also be used in Pairing-Based Cryptography.

From amongst the many possible families of non-supersingular pairing-friendly elliptic curves [46], Micro-pairings implements the MNT curve [90] with $k = 4$:

$$y^2 = x^3 - 3x + B \tag{5.9}$$

The prime p was chosen to be 160 bits in length, and the number of points on the curve was $34 \cdot r$, where r was a large prime. To be specific:

$$p = \texttt{E3F367D542C82027F33DC5F3245769E676A5755D}$$

$$B = \texttt{DABC0397E45200C4DF4CF67714DB64EB866BA034}$$

$$r = \texttt{6B455E0A014F1E30EAEF7300BD4BB4258290FC5}$$

Since r is 155 bits in length, there is an adequate resistance to a Pohlig-Hellman type of attack on the discrete logarithm problem [58], which would require around 2^{78} steps to succeed. However, the resistance to an index calculus attack [58] is more problematical. Given that the embedding degree is 4, the number of bits of DLP difficulty is $4 \cdot 160 = 640$. The current record is 531 bits [73], so again the safety margin is sufficient for the time being. The level of security here can be considered similar to that for the previously used supersingular curve, given the current state of knowledge [76].

For the implementation of Type 1 pairings the optimized Miller's algorithm was used (Algorithm 8). However, one additional optimization was applied, which accelerated the calculation of the final exponentiation. On a $k = 4$ MNT curve, the exponent can be divided into three parts. Using the Frobenius endomorphism, the first two parts of the final exponentiation can be easily calculated. The only hard part is the calculation of f to the power of $(p^2 + 1)/34r$. The number of points on the base field curve is equal to $r = p + 1 - t$, where t is the trace of Frobenius. Based on this, the final exponent can be calculated as $(p^2 + 1)/34(p + 1 - t) = p + \delta$, where δ is equal to:

$$\delta = (p^2 + 1)/(34r) - p = \texttt{F19192168B16C1315D34}$$

Therefore the expensive part of the final exponentiation can be calculated as $f^p \cdot f^\delta$, which requires one application of Frobenius endomorphism and a half-length exponentiation. The final method for computing the Tate pairing on a $k = 4$ MNT curve is shown in Algorithm 10.

Algorithm 10 Computation of $e(P,Q)$ with Tate pairing

INPUT: $Q \in E'(\mathbb{F}_{p^2})$, $P \in E(\mathbb{F}_p)$, δ
OUTPUT: $e(P,Q)$
1: $T \leftarrow P$, $f \leftarrow 1$
2: $s \leftarrow \lfloor \log_2(r-1) \rfloor$
3: **for** $i \leftarrow s-1$ **downto** 0 **do**
4: $\quad f \leftarrow f^2 \cdot l_{T,T}(Q)$
5: $\quad T \leftarrow 2T$
6: $\quad$ **if** $s_i = 1$ **then**
7: $\quad\quad f \leftarrow f \cdot l_{T,P}(Q)$
8: $\quad\quad T \leftarrow T + P$
9: $\quad$ **end if**
10: **end for**
11: $f \leftarrow f^{(p^2-1)}$
12: $f \leftarrow f^{34}$
13: $f \leftarrow f^p \cdot f^\delta$
14: **return** f

The line functions $l_{T,T}(Q)$ and $l_{T,P}(Q)$ in Algorithm 10 were calculated according to equation 5.4. In addition some, ideas, were used from [26] to recycle calculations from the elliptic curve point doubling and point addition formulas. In effect the calculation of lines 4 and 5, and lines 7 and 8 were combined. It is important to note that the calculation of f^p is effectively for free, using the Frobenius endomorphism[2]. So, for example, the calculation of f^{p^2-1} requires only two applications of the Frobenius, and a single inversion.

5.3.3.1 Prime field arithmetic

The most time consuming operation in the calculation of $e(P,Q)$ is the modular multiplication (ModMul), and to a lesser extent modular addition/subtraction (ModAdd). Modular inversion (ModInv) is expensive, but rare. For maximum efficiency Micro-pairings used the Montgomery representation of numbers modulo p [91], so that modular reduction required no divisions. The code for multiplication, reduction, addition and subtraction was in the form of automatically generated loop-unrolled assembly language routines. For large integer multiplication and squaring, the improved hybrid method was implemented (see Sections 3.3.2.1 and 3.3.3). The memory for all variables was allocated from the stack and some effort was made to keep the code size to a minimum by removing unnecessary functions.

The Tate pairing evaluates as an element of an extension field $\mathbb{F}_{p^k}$. So for $k = 4$, special functions for multiplication, addition, exponentiation and inversion in $\mathbb{F}_{p^4}$ had to

[2]The Frobenius endomorphism over finite fields can be defined as $\Phi(f) = f^p$

be developed. These operations were constructed mainly on the basis of standard base field arithmetic primitives. More details about the implementation of these routines can be found in Section 3.5. The overall cost of the Tate pairing in terms of the basic arithmetic operations is given in Table 5.7.

Table 5.7: *Cost of the Tate pairing on MNT $k = 4$ curve*

	ModMul	**ModAdd**	**ModInv**
Main loop	5730	15006	2
Final exp.	571	2642	1

Type 3 pairing implementations were tested in the same simulation environments as Type 1 pairings. A summary of results for finite field arithmetic over $\mathbb{F}_p$, assuming 160-bit integers can be found in Table 5.8. The number of clock cycles for modular addition was taken as an average value because the reduction step was not always executed. Modular multiplication of big integers was the most important operation that had a big influence on the overall pairing performance. This was also the most time-critical routine because it was repeated over 6000 times throughout the pairing calculation. Modular inversion was the most expensive arithmetic operation on all embedded platforms, but luckily it had to be performed only three times during the pairing calculation. When looking at the results in Table 5.8, the differences between assembly language and C-only implementations are apparent. Modular multiplication improved up to three times and other operations like modular addition and modular inversion were also quicker. On the Atmega128 the improved hybrid multiplication with column size $d = 4$, gave a times three improvement, whereas on the MSP430, the same method with $d = 2$ was more than twice as fast as the standard multiplication technique. In the case of the PXA271 differences in timings between assembly language and C routines were not as significant as they were for other platforms. That is mainly because the ARM architecture was designed from the beginning for fast integer arithmetic operations and there is not much space for improvements when using hand-crafted assembly code. Despite that, Micro-pairings still achieved a 42% improvement in execution time for modular multiplication on the PXA271 processor.

Tables 5.9 and 5.10 present the overall results for the calculation of the Tate pairing on different embedded processors. The introduction of assembly language routines gave

Table 5.8: *Timings in clock cycles for modular arithmetic routines in $\mathbb{F}_p$ using 160-bit integers*

	Atmega128			MSP430			PXA271		
Operation	**Mul**	**Add**	**Inv**	**Mul**	**Add**	**Inv**	**Mul**	**Add**	**Inv**
Assembly	7547	404	364291	4734	386	229724	843	155	49223
C code	22493	596	419812	11148	533	269768	1463	200	53318
Decrease	66%	32%	13%	57%	27%	15%	42%	22%	8%

up to 62% reduction in the total execution time of the pairing. However, the timings for the Tate pairing calculation were significantly slower than for the η_T algorithm on all processors. On the PXA271, an improvement of 36% was achieved, which proves that integer arithmetic can be very efficiently executed on an ARM-based machine with 32-bit processing, even without the use of assembly language routines. The results from Tables 5.9 and 5.10 prove that field-specific assembly language code gives the maximum speedup for the Tate pairing algorithm.

Memory usage is sometimes equally important as execution time and this is certainly the case for small and constrained embedded devices like sensor nodes. Tables 5.9 and 5.10 show the memory overhead of all Tate pairing implementations. In all cases the code size was decreased with the introduction of assembly language implementations. On the Atmega128 processor, the optimized pairing implementation saved over 30KB in comparison with the standard C version. Memory usage results were also satisfactory on the MSP430 CPU, where 10KB of program space were saved. This achievement is especially important for the MSP430F1611 platform where only 48KB of ROM are available. Despite these optimization efforts, the memory overhead for the Tate pairing was still quite significant on these devices. Looking from the memory point of view, the η_T pairing should be recommended over the standard Tate pairing on the most limited WSN platforms.

Table 5.9: *Performance of the Tate pairing on Atmega128 and MSP430*

	Atmega128			MSP430		
	Cycles	**ROM**	**Stack**	**Cycles**	**ROM**	**Stack**
Assembly	54,800,077	60.91KB	3.39KB	37,739,040	34.88KB	3.39KB
C code	143,888,874	94.41KB	3.39KB	77,411,534	46.15KB	3.39KB
Decrease	62%	35%	0%	51%	24%	0%

Table 5.10: *Performance of the Tate pairing on the PXA271*

	PXA271		
	Cycles	**ROM**	**Stack**
Assembly	8,055,473	44.40KB	3.75KB
C code	12,573,931	51.50KB	3.75KB
Decrease	36%	14%	0%

5.3.3.2 The Ate pairing

In order to compare different pairing algorithms on non-supersingular elliptic curves, the Ate pairing [61] was implemented on the Atmega128 and MSP430 processors. The Ate pairing is briefly described in Section 5.1.3.1 and is the counterpart of the η pairing for ordinary curves. This pairing has a truncated main loop (similar to the η_T pairing) where the number of loop iterations is smaller than r. The Ate pairing can be calculated with Algorithm 11 (based on [107]).

Algorithm 11 Computation of $e(P, Q)$ with Ate pairing

INPUT: $P \in E'(\mathbb{F}_{p^d})$, $Q \in E(\mathbb{F}_p)$, t is the trace of Frobenius
OUTPUT: $e(P, Q)$

1: $T \leftarrow P$, $f \leftarrow 1$
2: $s \leftarrow \lfloor \log_2(t-1) \rfloor$
3: **for** $i \leftarrow s-1$ **downto** 0 **do**
4: $\quad f \leftarrow f^2 \cdot l_{T,T}(Q)$
5: $\quad T \leftarrow 2T$
6: $\quad$ **if** $s_i = 1$ **then**
7: $\quad\quad f \leftarrow f \cdot l_{T,P}(Q)$
8: $\quad\quad T \leftarrow T + P$
9: $\quad$ **end if**
10: **end for**
11: $f \leftarrow f^{(p^d-1)}$
12: $f \leftarrow f^{(p^d+1)/\Phi_k(p)}$
13: $f \leftarrow f^{\Phi_k(p)/r}$
14: **return** f

On the MSP430 platform, the method was implemented according to Algorithm 11, but on the Atmega128 more program space was available and the multiplies of point P were precomputed. In both cases, the parameters were set as $k = 4$, p a 256-bit prime and a fixed point P on the MNT curve. Table 5.11 shows the performance results of the Ate pairing on the Atmega128 and MSP430.

As can be seen in Table 5.11, the Ate pairing was slower than the standard Tate pairing on both hardware platforms. This was mainly due to the fact that the Ate pairing was

Table 5.11: *Performance of the Ate pairing on Atmega128 and MSP430*

Atmega128			MSP430		
Cycles	**ROM**	**Stack**	**Cycles**	**ROM**	**Stack**
132,373,604	71.9KB	2.5KB	96,829,440	47.0KB	3.0KB

calculated with larger security parameters. In particular, the finite field was larger by almost 100 bits. This makes a large difference in performance especially on constrained 8-bit and 16-bit processors. The Ate pairing also used more memory both in ROM and RAM. This was due primarily to precomputation data. This fact was especially visible for the ATmega128 CPU where precomputed values of T took 28KB of a total 71.9KB of program memory. Precomputation was not applied in the case of the MSP430 processor due to the 48KB memory limit. Otherwise the pairing program on this platform would have been a bit faster.

5.3.4 Overall results

The overall results for Micro-pairings, assuming a 7.3828MHz clock rate on the MICA2, 8.192MHz on Tmote Sky and two different clock speeds for Imote2, are presented in Tables 5.12 and 5.13. As can be seen, the pairing calculation can be performed in as little as 2.66s on a tiny 8-bit MICA2 mote. The timings for other more powerful devices are much faster. The η_T pairing over $\mathbb{F}_{2^{271}}$ is the fastest pairing on all target platforms. The difference is a lot smaller on the Imote2 platform where arithmetic operations over $\mathbb{F}_p$ can be executed much more efficiently. The program code was also noticeably smaller for the Type 1 pairing in all cases. These features favour the usage of the η_T pairing, especially on the two more constrained sensor nodes where available memory is very limited.

Table 5.12: *Overall results for pairing implementation over $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$ on MICA2 and Tmote Sky nodes*

	MICA2 (7.38MHz)			Tmote Sky (8.19MHz)		
	η_T	Tate	Ate	η_T	Tate	Ate
Timing	2.66s	7.43s	17.93s	1.71s	4.61s	11.82s
ROM	47.4KB	60.9KB	71.9KB	23.7KB	34.9KB	47.0KB
RAM	3.1KB	3.3KB	2.5KB	4.1KB	3.3KB	3.0KB
Current draw	7.86mA	7.88mA	7.88mA	3.45mA	3.68mA	3.68mA
Energy usage	62.73mJ	175.65mJ	423.87mJ	17.70mJ	50.89mJ	130.49mJ

The energy consumption of Micro-pairings was also calculated on all three platforms.

An experimental setup was used for the MICA2 and the Tmote Sky motes to measure the current drawn from the batteries (see Appendix A) during the example programs execution. In the case of the Imote2 platform, the energy consumption values were taken from the manufacturer data sheet [29]. As can be seen in Tables 5.12 and 5.13, the η_T pairing was the most efficient in terms of energy consumption on all devices. Due to faster computation time and the use of power management unit, the total energy usage for η_T pairing on Imote2 was the lowest among all the platforms. According to the Imote2 data sheet [29], the core voltage on Imote2 is only 0.85V when clocked at 13MHz with the radio transceiver turned off. The Tmote Sky node was also very efficient in terms of energy consumption. The Imote2 was surprisingly the most energy efficient device even though it was the most powerful of the test platforms. All calculations on Imote2 can be significantly accelerated with an increase of the clock speed to 104MHz. According to the current draw values in [29], switching the clock speed to 104MHz decreases the total energy consumption during the pairing calculation.

Table 5.13: *Overall results for pairing implementation over $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$ on Imote2 node*

	Imote2 (13MHz)		**Imote2 (104MHz)**	
	η_T	Tate	η_T	Tate
Timing	0.46s	0.62s	0.06s	0.08s
ROM	29.55KB	44.40KB	29.55KB	44.40KB
RAM	4.12KB	3.75KB	4.12KB	3.75KB
Current draw	31mA*	31mA*	66mA*	66mA*
Energy usage	12.12mJ	16.34mJ	3.76mJ	5.02mJ

* - manufacturer data

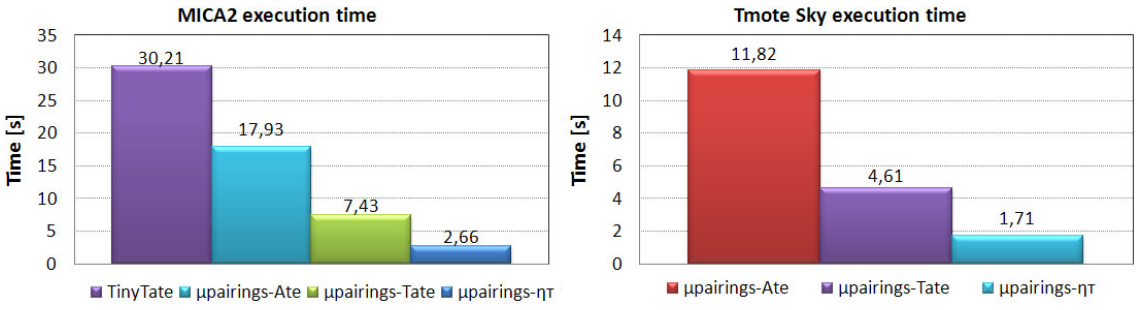

Figure 5.1: *Pairing timings comparison on MICA2 and Tmote Sky*

The pairings implementations presented in this chapter are the fastest yet reported on all three sensor network platforms. Figure 5.1 shows a comparison of execution time of

different pairing implementations on low-end sensor nodes. The Tate pairing on MICA2 in Micro-pairings takes only 7.43 seconds to complete when clocked at 7.3828MHz. This compares very favourably with the timing of TinyTate [96] (30.21s). The improvement in pairing calculation is even more evident in the case of the η_T algorithm. On the MICA2 mote, the η_T pairing can be calculated in only 2.66s. This is around eleven times faster than TinyTate. The timings for Tmote Sky are not compared with the related work as there are no other pairing implementations available on that platform.

The energy consumption of different pairing algorithms is compared in Figure 5.2. As can be seen the Micro-pairings implementation is the most efficient on both the MICA2 and the Tmote Sky. In the best case on MICA2, the η_T pairing uses 91% less energy than the Tate pairing in [96]. The difference is pretty significant, especially when the mote's limited energy resources are taken into consideration. The Tmote Sky platform is more efficient in terms of energy consumption than MICA2, while calculating the same pairing types.

On the MICA2 platform, the TinyTate implementation has the smallest memory footprint, but at the same time offers a limited security level with much smaller parameters that are considered as insecure in today's security systems. On the Tmote sky node, the η_T pairing is the most efficient algorithm in terms of memory consumption. Nevertheless, the memory usage is considerable and takes 23.7KB of ROM, which is almost half of the available memory on the Tmote Sky platform. RAM utilization is similar for all algorithms in Micro-pairings implementation. A comparison of ROM and RAM occupation for different pairing implementations is illustrated in Figures 5.3 and 5.4.

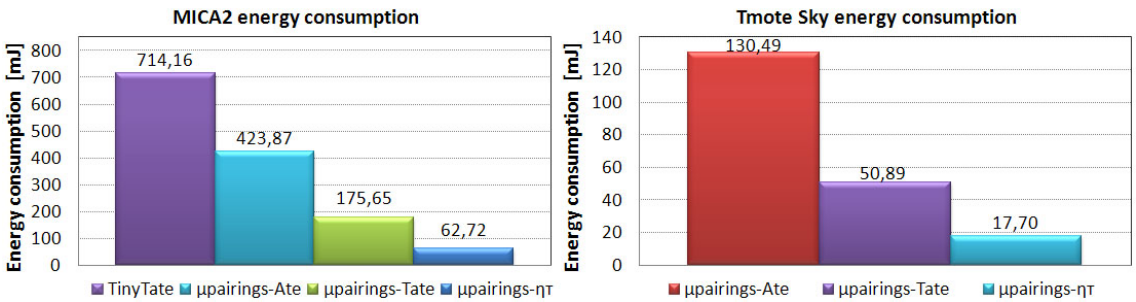

Figure 5.2: *Pairing energy consumption comparison on MICA2 and Tmote Sky*

All of the above results show that pairings can be implemented efficiently in software and that they can be calculated in a reasonable amount of time, even on small and con-

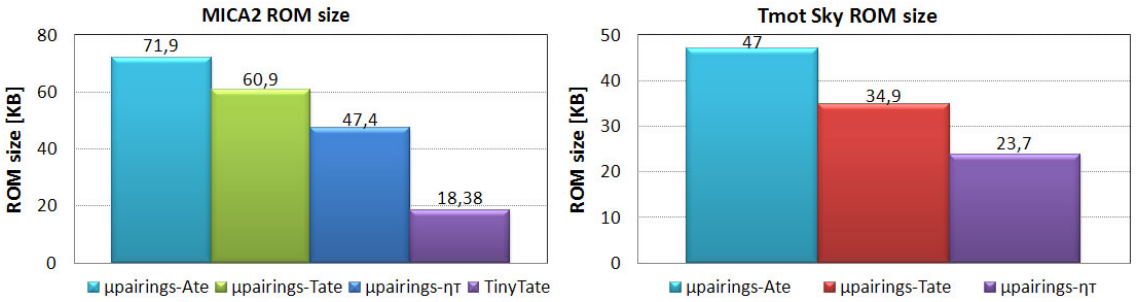

Figure 5.3: *Pairing ROM size comparison on MICA2 and Tmote Sky*

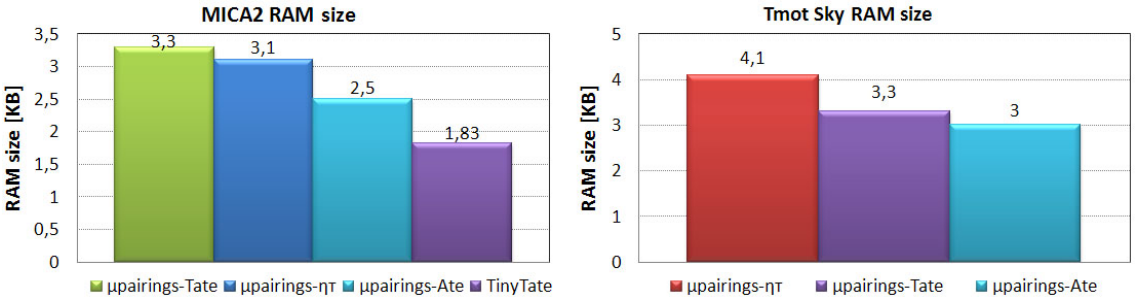

Figure 5.4: *Pairing RAM size comparison on MICA2 and Tmote Sky*

strained devices. From the above performance comparisons, it is clear that the η_T pairing over binary fields gives the best trade-off between speed, energy consumption and memory occupation. Pairings offers a new and interesting cryptographic primitive that can be used to design many novel security protocols, and Micro-pairings facilitates the development of such mechanisms.

5.4 Pairings implementations in hardware

This chapter is focused on software implementations of pairings for wireless sensor networks, but it is also possible to design a hardware unit for computing the pairing. A hardware implementation of pairings for sensor nodes would be several orders of magnitude quicker and would save a lot of program space, and consume less energy. This would be especially beneficial for very constrained devices with critical amounts of memory. The main drawback of this approach is that a dedicated hardware accelerator would lift the total cost of a mote and would complicate its design. This of course, goes against the low-cost/small size philosophy which is a priority for sensor network devices.

A hardware accelerator for computing the pairing can be implemented on a Field Programmable Gate Array (FPGA) or can be designed as an Application Specific Integrated

Circuit (ASIC). Most of the papers in the literature used the first option, due to flexibility and smaller implementation cost. Shu *et al* [113] implemented the Tate pairing using the η and η_T approach. The second algorithm was faster when implemented on a FPGA, but at a considerable cost in area of the design. In [103] Ronan *et al* implemented a dedicated processor for the Tate pairing on a FPGA using the η_T algorithm. He used a pipelining technique to improve the latency of the design. This architecture, however, was larger and slower than the version presented in [113].

In [70] Keller *et al* designed an interesting hardware unit that could calculate, not only the Tate pairing, but also the scalar point multiplication. The implementation presented in [70] was based on the optimized Miller algorithm (Algorithm 8). The authors managed to achieve a good performance of the hardware accelerator, whilst maintaining a relatively small area size. There were also a couple of attempts to implement pairings over fields of characteristic three. Kerins *et al* [71] used the Duursma–Lee algorithm to calculate the Tate pairing over $E(\mathbb{F}_{3^{97}})$ on a FPGA. The authors in [54] also calculated the Tate pairing in characteristic three, but instead of having a dedicated hardware solution they used a general purpose processor to control the ALU co-processor. In both papers the authors achieved results comparable with previous implementations over $\mathbb{F}_{2^m}$.

All the above designs were implemented using FPGA's. The obvious constraint of such an approach is that the hardware accelerator uses a lot more energy than an ASIC. The above implementations were optimized for speed and area size rather than energy efficiency. Hence they are not directly applicable to low power environments like sensor networks. From hardware accelerators targeted specifically at WSNs, McCusker [86] designed an ASIC for computing the η pairing. He focused on energy efficiency of the overall design and was able to calculate the η pairing in 1.75ms using 0.08mJ. This implementation, however, is the only one available that was designed with sensor networks in mind. Hence the topic of pairings implementation in hardware for sensor netwroks needs to be further investigated.

Currently there are no pairings hardware accelerators available for off-the-shelf sensor nodes. Hence the developers can only relay on software based solutions. This chapter presented an extensive study of Pairing-Based Cryptography implementations in software and showed their practicality on different hardware platforms. Software implementations have many advantages, such as flexibility, fast development time, and low cost.

Deploying software-based solutions such as Micro-pairings demonstrates the validity of Pairing-Based Cryptography in the WSN arena, and should motivate the incorporation of hardware support for Pairing-Based Cryptography in future mote designs.

5.5 Summary

Bilinear pairing is a flexible crypto primitive that can be used to build many interesting security schemes in wireless sensor networks. Pairing-based cryptography is more complex than methods based on integer factorization and the discrete logarithm. Nevertheless, Pairing-Based Cryptography has paved the way for a new wide range of cryptographic protocols. It has also allowed many long-standing open problems to be solved in an elegant way. Perhaps the most impressive among those applications is identity-based encryption, which in turn has allowed complete identity-based cryptographic schemes.

In the literature, many pairing implementations for standard desktop computers are reported. However, very little attention is focused on implementing those operations on small embedded devices. Efficient computation of pairings is essential to the large and ever growing area of pairing-based cryptosystems. Fast pairing implementations on sensor nodes are necessary to enable different identity-based cryptography schemes in sensor networks.

This chapter has investigated in detail the application of Pairing-Based Cryptography to wireless sensor networks. It presents the mathematical concepts behind cryptographic pairings and discusses various pairing types. It shows also efficient algorithms for calculating bilinear pairings on different curves and over different finite fields. All this research resulted in the development of Micro-pairings, which is an efficient implementation of pairings on a wide range of sensor nodes. Micro-pairings provides the fastest available pairing implementations on popular sensor nodes. The software package can also be easily ported to other embedded devices. Micro-pairings was tested on a wide range of different WSN processors and it showed that pairings can be performed in a very quick and efficient way, even on the most constrained devices. The results achieved with Micro-pairings have shown that the η_T pairing over binary fields gives the best performance on all hardware platforms. The η_T pairing is a Type 1 pairing and according to the achieved results it has a lot better performance than any Type 3 pairing, on different sensor nodes. Therefore Type 1 pairing is recommended for all low-end architectures. On

the most constrained 8-bit MICA2 node, the η_T pairing took 2.66s using only 62.73mJ of energy. This result is comparable with the calculation of the point multiplication operation on the same platform. The above result is significant, especially as pairings are considered far more complex than Elliptic Curve Cryptography primitives. Micro-pairings goes a step forward to show that Pairing-Based Cryptography can be now used as an attractive alternative to Elliptic Curve Cryptography in many different security schemes for wireless sensor networks.

The above study is not only relevant to sensor networks but also to different classes of embedded systems where energy and computation power are limited. Examples of those might be *ad hoc* and other distributed networks where security issues are still open problems. ARM processors are also used in palmtop class devices and Micro-pairings can be used to secure mobile systems that are build upon this platform. The efficiency of such security mechanisms would be much greater due to the improved capabilities both in terms of available memory and processing power.

Software implementations of pairings have some advantages when compared with hardware solutions. They are more flexible and can be quickly implemented on existing sensor nodes. Hence they allow easy evaluation of new Pairing-Based Cryptography protocols at lower cost. Micro-pairings provides all the necessary primitives to implement such protocols on different hardware platforms and thus opens new ways for achieving security in sensor networks. High performance of pairings over $\mathbb{F}_{2^m}$ fields makes binary field the recommended finite field for software implementations of Elliptic Curve Cryptography and Pairing-Based Cryptography.

CHAPTER 6

Identity-Based Security Protocols for WSNs

In Section 2.3.4 it was suggested that Identity-Based Cryptography can be used as a solution for the key distribution problem in sensor networks. Section 2.3.5 presents all of the building blocks that are necessary to make Identity-Based Cryptography protocols practical in a sensor network environment. Chapters 3, 4 and 5 describes in detail how to efficiently implement these modules on constrained sensor nodes. This chapter utilizes all the information presented in previous chapters to build identity-based security protocols for sensor networks.

While there is a whole family of security protocols that use identities (IBE, ID-based signcryption, ID-based signatures, threshold cryptography schemes), the most relevant ones to sensor networks are key agreement schemes and Identity-Based Encryption. These schemes cannot be applied directly as a universal solution for the key distribution problem in sensor networks. Wireless sensor networks can have different network models and communication patterns, and hence the key distribution mechanism has to be tailored to a specific network organization. Such an approach allows the designer to minimize the resource utilization and achieve optimal performance of the network.

Sensor network models can be organized into three broad groups. Based on this division, three new key distribution protocols for WSNs are proposed. The first two investi-

gate the application of non-interactive key exchange schemes in the flat and hierarchical network scenarios. The last key agrement scheme uses identity-based encryption to secure communication within a heterogeneous sensor network. This chapter also includes a comprehensive security analysis of all the proposed solutions, together with possible security threats and defensive measures.

From the implementation point of view, the pairing calculation is the most expensive operation of every Identity-Based Cryptography scheme both in terms of computational complexity and memory overhead. The implementation of a complete Identity-Based Cryptography system on sensor devices is quite difficult and expensive in terms of resources utilization. However, such systems are envisioned mainly as security bootstrapping mechanisms in sensor networks. For link layer security, cheaper symmetric key encryption schemes should be used.

6.1 Identity based key distribution

In the literature, there are papers that propose the use of identities to distribute keys in sensor networks. Many of them [23], [34] are based on symmetric cryptosystems due to Blundo *et al.* [17] and Blom [16]. These schemes do not provide perfect resilience, as after a certain percentage of nodes have been captured, the whole network is compromised. The authors in [33], [126], [94] envisaged the use of IBE based on pairings as a security solution in sensor networks. All three papers proposed the Boneh and Franklin identity-based encryption scheme [18] to distribute keys in the network.

In [126], the Identity-Based Encryption scheme was evaluated through a simulation on a desktop class computer. The relevance of the results achieved to sensor networks is not clear, as simulation details were not presented. Doyle *et al* in [33] showed the energy consumption results for the Tate pairing calculation on an ARM7 processor. This platform, however, is considerably more powerful than most of the devices that are currently in use in WSNs. Finally Oliveira *et al* [94], [96] proposed the Identity-Based Encryption scheme for sensor networks and presented the Tate pairing implementation figures for the MICAz mote. Despite all this efforts, none of the above proposals have actually implemented a complete identity-based encryption scheme in a sensor network. Hence, the feasibility of such a security solution had not yet been proven . In addition, all of the authors proposed the Boneh and Franklin Identity-Based Encryption scheme, which is not

the most efficient identity-based protocol. Proper security mechanism for WSNs should consider the network topology and the communication patterns between the nodes. The Identity-Based Cryptography scheme should be designed for a particular model of a sensor network.

6.1.1 Wireless sensor network models

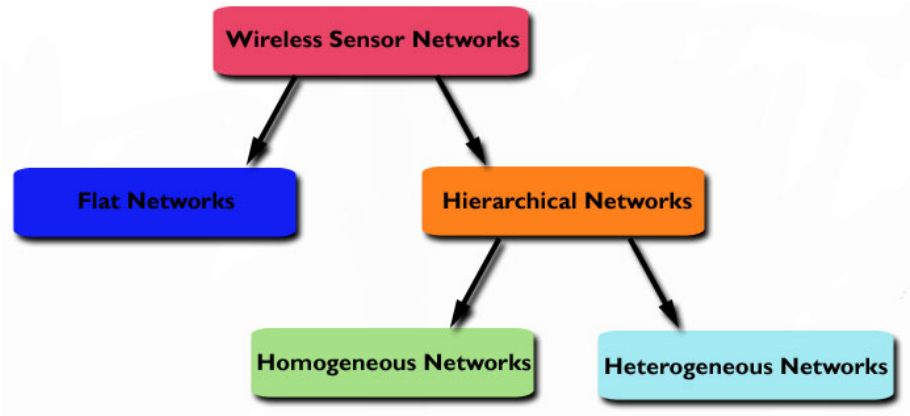

Figure 6.1: *Wireless sensor networks classification*

Sensor networks can be organized in different ways. Figure 6.1 presents a broad classification of wireless sensor network models. In flat sensor networks, all nodes play similar roles and there is no hierarchy between them. All motes participate in sensing, data processing and routing. The communication is performed on a peer-to-peer basis in a mesh type of network (see Figure 6.2).

A flat network presents a simple network organization where all the nodes have similar capabilities and operate using the same software and protocols. The main communication pattern is from the sensors to the base station. In such a network topology the nodes in the closest vicinity of the base station have the highest load and may run out of energy a lot quicker than the rest of the nodes. In this case, the routing protocol should control the proper load balancing in the network. In flat sensor networks each of the nodes should be able to establish a secure connection with the majority of remaining nodes.

In hierarchical WSNs the network is typically organized into clusters, with ordinary cluster members and specially designated cluster heads. While ordinary cluster members are responsible for sensing, the cluster heads have additional tasks such as aggregating and processing the data from nodes within their cluster. They are also responsible for

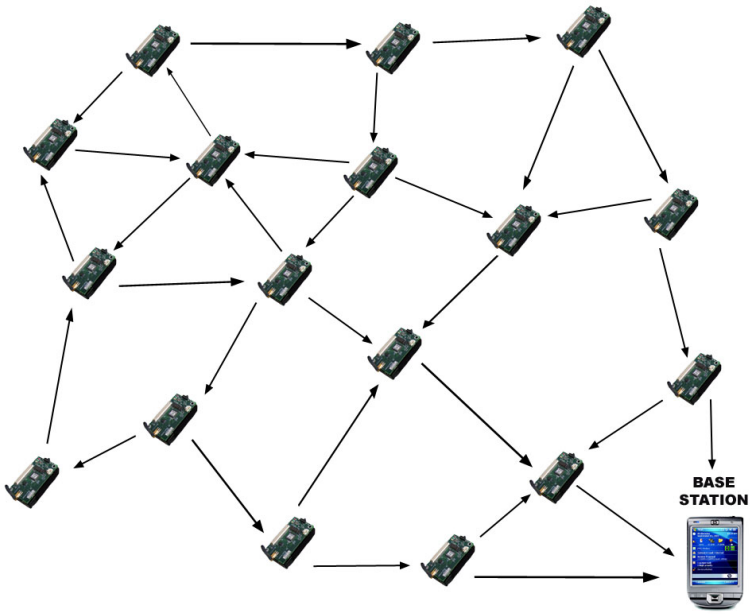

Figure 6.2: *Flat wireless sensor network*

forwarding the sensor data to the base station. Hierarchical networks can be divided into homogeneous and heterogeneous networks. In the former, all devices except the base station are of the same type and have the same capabilities. Heterogeneous networks include different kinds of devices where usually the cluster heads are more powerful than other sensor nodes. In general, hierarchical networks can have more than two levels of hierarchy. In this case, the cluster heads on one level are treated as ordinary cluster members by the nodes that are one level up in the hierarchy.

Figure 6.3 illustrates a typical homogeneous sensor network. The whole network is divided into three clusters each having its own cluster head. All the nodes have similar capabilities and similar radio range. The nodes join the clusters based on the hop count to the nearest cluster head. The cluster formation is not usually fixed due to the limited energy resources of ordinary nodes. After a certain amount of time (or processed packets) new cluster heads must be elected from the remaining sensor nodes. The selection can be made randomly (like in the LEACH protocol [60]) or using more elaborate methods. After a new cluster head election the cluster formation process is repeated. In homogeneous networks each sensor node must be able to establish a secret key with every other node within the cluster. In addition cluster heads should have secure connectivity with each other and the base station.

A Heterogenous Sensor Network (HSN) is a hierarchical WSN that is typically orga-

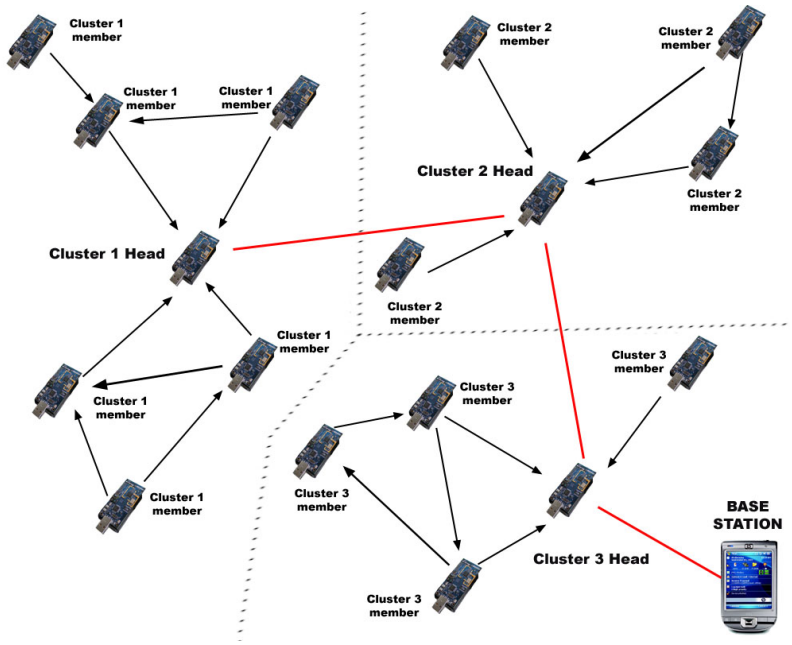

Figure 6.3: *Homogeneous sensor network*

nized into clusters that consists of two types of nodes. More powerful devices (H-sensors) usually take the role of cluster heads and low-end nodes (L-sensors) become ordinary cluster members. L-sensors are mainly responsible for sensing, whereas H-sensors perform additional tasks such as data collection, information processing and reporting to the base station. The role of a cluster head is fixed in this kind of a network (as oppose to cluster heads in LEACH [60] based sensor networks). The dominating traffic pattern in heterogeneous sensor network is many-to-one, where L-sensors send data to their cluster heads. Hence each L-sensor has to encrypt the data for its cluster head and cluster heads must be able to securely deliver the information to the base station. Figure 6.4 presents the cluster formation in a typical heterogeneous.

One of the biggest advantages of sensor networks is their versatility on many fronts. They can have different number of devices, many configuration modes and various types of sensors that make them suitable for a wide range of applications. This versatility, however, is also one of the main challenges in WSNs when it comes to network protocol design. Sensor networks may be organized in a variety of ways. The above classification does not include all possible sensor network organizations. There are also other sensor network models with different topologies. However, the three main groups described earlier, demonstrate various communication patterns that might be used to optimize the key distribution mechanisms in each case. A solution designed specifically for a flat net-

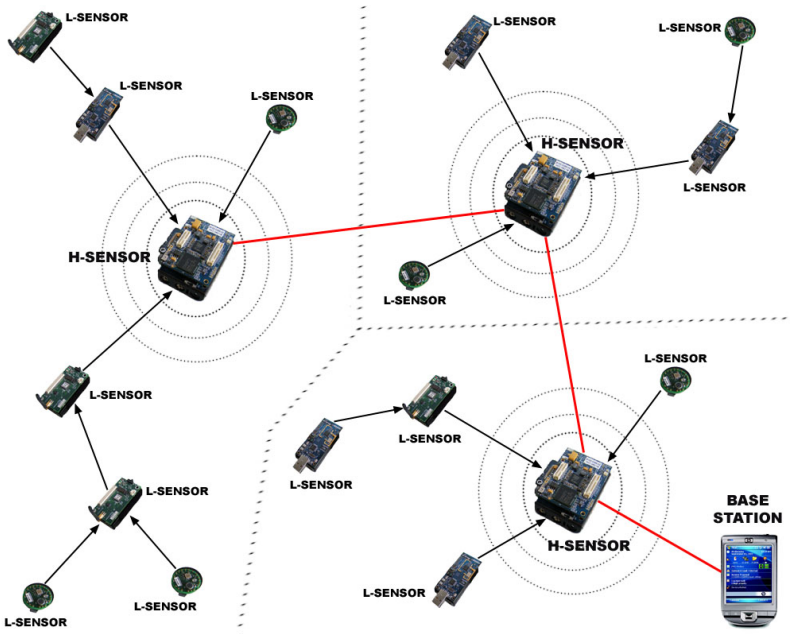

Figure 6.4: *Heterogeneous sensor network*

work topology is unlikely to be the right choice in a clustered network scenario. Hence an optimal solutions should be tailored to a specific network organization. The following sections present three different key distribution protocols designed specifically for the afore-mentioned network models.

6.2 New approach to key distribution in flat WSNs

Key distribution in sensor networks is challenging because the system has to be simple, energy efficient and at the same time offer a high level of security. One simple approach for key distribution in flat sensor networks would be to issue just one shared bootstrap crypto key to all nodes. From this initial state, the network could self-organize into groups which, under cover of the shared key, might then go on to negotiate new group keys for use in subsequent communication. This solution scales nicely to large networks, and facilitates the addition of new nodes to the network. However, it does not provide any kind of node-to-node authentication. Also it fails catastrophically to an active attacker, who captures just one node during the early deployment stage, and with it the shared master key. An alternative approach is to issue each node with a list of $n-1$ keys, one for each of the other $n-1$ nodes in an n-node network. This does provide authentication, but it requires a lot of memory, and it is not at all trivial to add new nodes to an existing

network without recalling existing nodes for a refit.

This section shows that the key distribution problem in flat sensor networks can be addressed by using an identity-based cryptography approach. Using the scheme described here, any two nodes in the network can establish a shared symmetric key without exchanging any messages. This scheme simplifies key management in a network by reducing the number of keys required , making the system scalable to large numbers of nodes. The evaluation results prove that the software implementation of the scheme is practical on popular sensor platforms and different CPU architectures. Comparison with the state-of-the-art shows that identity-based key distribution is a superior security solution for sensor networks with better resistance against known attacks.

6.2.1 Protocol overview

The key agreement scheme for flat networks is based on the simple key exchange procedure that was first introduced by Sakai, Ohgishi and Kasahara in [106]. The protocol proposed here is also based on secret key pre-distribution using the secure channel that exists during the network pre-deployment phase.

6.2.1.1 Pre-deployment phase

The original network deployer takes the role of a trusted authority in the system. It needs to load secret keys into each node's memory together with all the public parameters. Initially it generates a master key s which has to be kept secret. It also assigns a unique identity to each node that will participate in the network. For this purpose it can use TinyOS or IP addresses (in the case of an IPv6 addressing scheme).

In the next step, the trusted authority calculates each node's private key. This operation can be performed by the use of Elliptic Curve Cryptography primitives. The publicly available hash function H, is used to derive a hash value based on the node's identity $N_X = H(ID_X)$, where ID_X is the identity of node **X**. The N_X value is then mapped to an elliptic curve point via a mapping function. The point multiplication operation uses the master key as a scalar (sN_X). The result of this operation is a node's private key. From the description of security levels in elliptic curve systems (Section 2.2.1), it is known that it is not feasible to derive the s value based on sN_X, when the size of N_X in bits is bigger than 160. This operation would require the solution of an intractable discrete logarithm

problem in an elliptic curve setting.

Each node is issued with a single secret key sN_X, its identity ID_X, hashing function H, mapping function and a key derivation function that is based on a one-way hash function. The key derivation function is required to derive the session key of a size that is suitable for a particular symmetric cipher. All these parameters are preloaded into each node's memory before the deployment phase.

6.2.1.2 Identity based key agreement

After the setup process, nodes are ready for deployment. During network operation two nodes **A** and **B** that know each other's identities can exchange information in a secure way without any prior interaction. Node **A** has a private key sA and node **B** a key sB. Both sides can independently obtain the required public keys A and B by calculating $A = H(ID_A)$ and $B = H(ID_B)$.

When node **A** wants to setup a pair-wise session key $K_{A,B}$ with node **B**, it calculates the bilinear pairing function $\hat{e}(sA, B)$. The Key Derivation Function (KDF) is used to derive the session key from the calculated pairing value $K_{A,B} = KDF(\hat{e}(sA, B))$. Now **A** can use the key $K_{A,B}$ to encrypt the message and send it over the radio channel to **B**. Node **B** receives the message and obtains the decryption key $K_{B,A} = KDF(\hat{e}(sB, A))$ to read the packet payload. Both **A** and **B** will end up with the same key since:

$$K_{A,B} = KDF(\hat{e}(sA, B)) = KDF(\hat{e}(A, B)^s)$$
$$= KDF(\hat{e}(A, sB)) = KDF(\hat{e}(sB, A)) = K_{B,A}. \quad (6.1)$$

This follows from the bilinearity and symmetry properties of the pairing function. It is important to notice that this protocol can only be implemented using a Type 1 pairing, as the symmetry property is necessary for the key agreement to work.

The above protocol allows two nodes to agree upon a common pair-wise key without any prior interaction with each other. There is no extra bandwidth overhead associated with the cryptography. The scheme presents a simple way to bootstrap the security in a sensor network. However, the above protocol on its own is not sufficient to secure the network against any given attack. Other security mechanisms (e.g. an intrusion detection

scheme) might be required as well. The main purpose of this protocol is to provide a secure and practical key agreement mechanism for flat sensor networks that will be a foundation for other network protocols built on top of it.

6.2.2 Implementation details

The key distribution is performed mainly at the beginning of network operation to establish session keys with neighbouring nodes, and thus the expensive pairing calculations are very infrequent. It is also expected that the network will use an energy efficient routing protocol, which performs load-balancing in the network. In this case, the majority of nodes will have to perform the key agreement only a few times, when establishing common keys with the neighbouring nodes.

In elliptic curve systems it is not enough to rely only on large key sizes to provide a high level of security. Other domain parameters such as the selected curve E and an appropriate finite field $\mathbb{F}_q$ are equally important. All of these parameters should be chosen so that the Elliptic Curve Discrete Logarithm Problem is resistant to all known attacks. For the Type 1 pairing over a field of characteristic two ($\mathbb{F}_{2^m}$), the security parameters should be chosen as $k \cdot m > 1024$ [76]. These countermeasures would make any index calculus attack [58] on the Bilinear Diffie Hellman Problem infeasible for the time being. According to the above security policy, a supersingular elliptic curve $y^2 + y = x^3 + x$ was chosen over the binary field $\mathbb{F}_{2^{271}}$ with the embedding degree $k = 4$. In this case, the number of points on the curve is known to be $2^{271} + 2^{136} + 1 = 487805 \cdot r$ where r is a large prime.

In the beginning, the setup process needs to be performed and all the necessary information has to be pre-load onto the sensor nodes. The evaluation presented here does not consider the operations performed by the trusted authority in the pre-deployment phase. This is done off-line, and so is not time-critical, and it does not have any influence on the network performance.

The overall performance of the identity-based key distribution mechanism relies on the efficiency of the pairing calculation $\hat{e}(P, Q)$. The η_T pairing [7] is one of the fastest known pairings that can be evaluated very efficiently. It uses a variant of Miller algorithm to calculate the pairing. One of its advantages is that it only requires half the number of iterations of the Miller loop compared with typical pairing algorithms. Hence it was

chosen as the pairing function for the implementation of the key agreement protocol.

The η_T pairing evaluates as an element in the $\mathbb{F}_{2^{4 \cdot 271}}$ extension field. The result of this operation is a 1084-bit value that can be used to calculate the mutual session key. The key derivation function derives the appropriate key by hashing the pairing result. In the implementation presented here, a 128-bit session key was used in combination with the AES block cipher. In general any kind of symmetric algorithm can be used and the key derivation function should generate a key of an appropriate size for that cipher.

To evaluate the efficiency of the scheme, there is a need to consider also the cost of mapping node identities to elliptic curve points. One viable method is to hash the identity to the x-coordinate, and then solve a quadratic equation to find y. The following fast algorithm to solve the quadratic equation $y^2 + y = c$ can be used for that purpose (based on [58]).

Algorithm 12 Solve $y^2 + y = c$ (basic version)

INPUT: $c = \sum_{i=0}^{m-1} c_i z^i \in \mathbb{F}_{2^m}$ where m is odd, trace $Tr(c) = 0$ and $H(z^i)$ is a half-trace function
OUTPUT: A solution s of $y^2 + y = c$
1: Precompute $H(z^i)$ for odd i, $1 \leq i \leq m-2$
2: $s \leftarrow 0$
3: **for** $i \leftarrow (m-1)/2$ **to** 1 **do**
4: **if** $c_{2i} = 1$ **then do**: $c \leftarrow c + z^i$, $s \leftarrow s + z^i$
5: **end for**
6: $s \leftarrow s + \sum_{i=1}^{(m-1/2)} c_{2i-1} H(z^{2i-1})$
7: **return** s

When the quadratic equation has no solution, the x value needs to be incremented and the whole process described in Algorithm 12 needs to be repeated. Otherwise one of the two solutions for y is selected and the elliptic curve point (x, y) is multiplied by a large cofactor, to obtain a point of an appropriate order. This point multiplication is the most expensive operation in the whole ID mapping scheme.

The non-deterministic nature of this mapping operation is a little unsatisfactory. However, the identities of the nodes are assigned by the network deployer. Hence the trusted authority might only assign those identities for which the quadratic equation is known to have a solution, at the negligible cost of adding approximately one bit to the overall length of the node identities.

In order to check the performance of the key agreement scheme, the whole protocol was implemented on typical sensor nodes. Three popular hardware platforms used in

real-life deployments were chosen: the MICAz [30] and Imote2 [29] platforms developed by Crossbow Technology, and the Tmote Sky node [92] developed by Moteiv corporation. For the pairing implementation, the Micro-pairings package was used on all hardware platforms. As the one-way hash function, the popular SHA-1 algorithm was chosen. The code of the programs was integrated with the TinyOS operating system to measure the timings for the main protocol operations.

6.2.3 Experimental results and analysis

Tables 6.1 and 6.2 present the evaluation results for all the basic operations that are performed by a single node in the key exchange protocol. The results are summed up at the bottom of the table to give the total values for the whole key agreement scheme. The pairing calculation is the most expensive operation in terms of time consumption and memory utilization on all three platforms. ID mapping also takes a considerable amount of time to complete. Other operations include setup routines, hashing, and session key derivation.

Table 6.1: *ID-based key agreement evaluation on MICAz and Tmote Sky.*

Operation	**MICAz (7.38MHz)**			**Tmote Sky (8.19MHz)**		
	Time	**ROM**	**Energy**	**Time**	**ROM**	**Energy**
Pairing	2.66s	47.41KB	62.73mJ	1.71s	23.66KB	17.70mJ
ID mapping	1.55s	0.72KB	36.55mJ	1.07s	0.48KB	11.07mJ
Other	29ms	2.78KB	0.68mJ	19ms	1.81KB	0.20mJ
Total	4.27s	50.91KB	99.96mJ	2.80s	25.95KB	28.97mJ

Table 6.2: *ID-based key agreement evaluation on the Imote2 node.*

Operation	**Imote2 (13MHz)**		
	Time	**ROM**	**Energy**
Pairing	0.46s	29.55KB	12.12mJ
ID mapping	0.19s	0.66KB	5.01mJ
Other	3.4ms	2.23KB	90μJ
Total	0.65s	32.44KB	17.22mJ

The fact that the implementation was set for a fixed field size ($\mathbb{F}_{2^{271}}$) permits us to greatly optimize the code. Software implementation was performed using C, nesC and assembly language. Assembly language was used only to accelerate the most time crit-

ical arithmetic routines (in particular binary polynomial multiplication). The whole key agreement takes as little as 0.65s on the Imote2 and 4.27s in the worst case on the most constrained MICAz mote. All of the results presented in Tables 6.1 and 6.2 assume a 7.38MHz clock rate on the MICAz, 8.192MHz on the Tmote Sky and 13MHz on the Imote2. The timings for the key agreement protocol are acceptable because the key exchange is performed very rarely during the network operation.

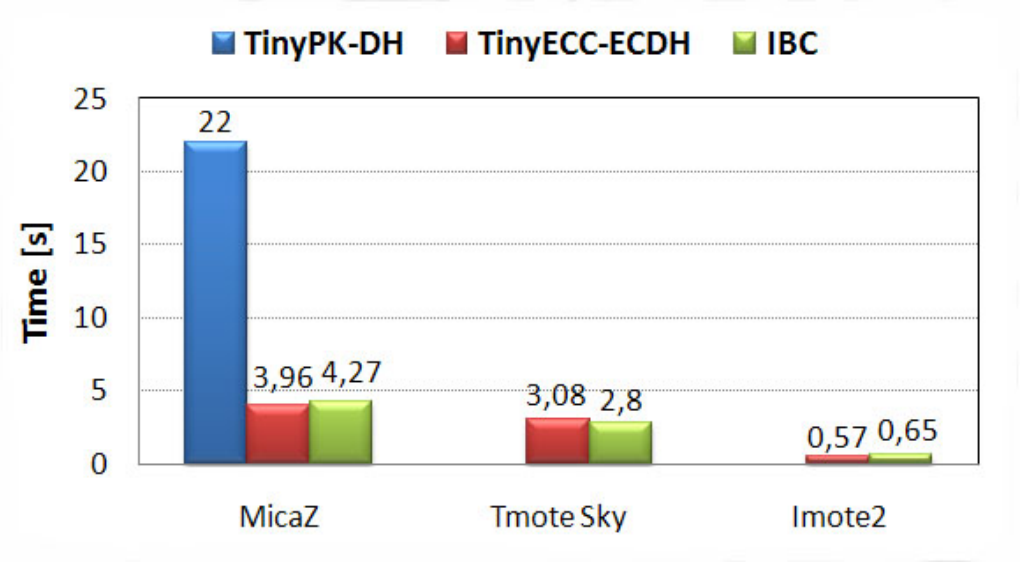

Figure 6.5: *Execution time comparison for different key agreement schemes.*

Figure 6.5 shows how the Identity-Based Cryptography scheme performs against the state-of-the-art. For comparison, the TinyPK scheme [120] and the TinyECC [80] library were chosen. The first method implements the Diffie-Hellman key exchange (DH) on the MICA2 platform (the MICAz mote has the same CPU). The second one uses a configurable Elliptic Curve Cryptography library for sensor networks. This library implements the Elliptic Curve Diffie-Hellman key exchange protocol (ECDH) on a broad range of sensor devices. In case of TinyPK, n was a 512 bit prime, and x, y were 112-bit exponents (see Figure 2.6 for an explanation). TinyECC had all the optimization switches enabled and was configured for maximum speed and computational efficiency. Figure 6.5 shows that the traditional Diffie-Hellman key exchange is the slowest method requiring, 22s, even though its security level is a lot lower than for other solutions. This is due to the fact that modular exponentiation is very expensive on 8-bit architectures, especially with a large modulus.

The timings for the Identity-Based Cryptography scheme are comparable with the

TinyECC implementation of the Elliptic Curve Diffie-Hellman protocol. However the performance of the TinyECC is actually worse due to the fact that it operates over a 160-bit finite field compared to the 271-bit required in the identity-based scheme. In addition the Elliptic Curve Diffie-Hellman key exchange in [80] is not authenticated, like the key agreement scheme proposed here. It requires also an exchange of information before the joint key can be established. The Identity-Based Cryptography scheme presented here works in a non-interactive manner, which is especially beneficial in large distributed networks.

One of the most important issues for sensor devices is efficient memory utilization. Looking at the numbers in Tables 6.1 and 6.2 one can notice that the pairing calculation takes a significant amount of ROM on all platforms. This is mainly due to the large size of the Elliptic Curve Cryptography library (more than 40KB on the MICAz, around 20KB on the Tmote Sky and 25KB on the Imote2). The code needed to perform point multiplication for ID mapping is also included in this library. Nevertheless further improvements to the code size on all platforms are possible through manual removal of unnecessary library functions.

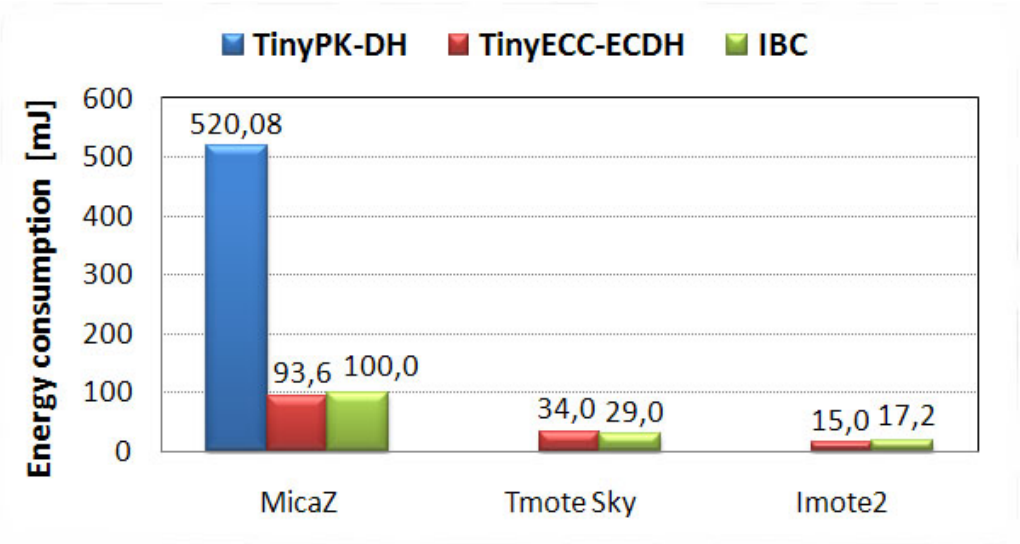

Figure 6.6: *Energy consumption comparison for different key agreement schemes.*

The energy consumption of the Identity-Based Cryptography key agreement was also evaluated. According to Table 6.1, the Tmote Sky node is far more efficient in terms of energy consumption than the MICAz platform. The Imote2 uses even less energy thanks to its low power operation mode (31mA at 13MHz). Assuming two new AA alkaline

batteries with 2850mAh capacity, around 0.6 million key agreement operations can be performed on the MICAz mote. Using the same energy, the Tmote Sky can calculate more than 2 million shared keys.

Figure 6.6 presents the energy consumption of different key agreement schemes. The TinyPK uses around 520mJ during the Diffie-Hellman key exchange. The TinyECC and the Identity-Based Cryptography scheme are more efficient and use five times less energy. This result is quite reasonable, especially given that key exchange operations are not very frequent in a static sensor network environment. The values in Figure 6.6 relate only to CPU usage and do not include the communication overhead. The actual energy consumption for the Diffie-Hellman and Ellioptic Curve Diffie-Hellman schemes would be higher due to packet transmission and reception. The ID-based key agreement scheme does not require any information exchange and can save energy on expensive radio communication.

6.3 A novel key distribution technique for cluster-based WSNs

In a clustered sensor network scenario, each cluster head acts like a sink for data. All nodes within the cluster send the sensor readings directly to their cluster head. It is envisaged that most nodes act as sources of information, most of the time, and that relatively few would be sinks (cluster heads). The task of the cluster head is to aggregate the data from the cluster and send it to the base station in an appropriate form. As mentioned earlier, in a homogeneous sensor network, all nodes have similar capabilities and same energy resources. Hence, the role of a data collator should be passed around, to equalize the energy consumption of nodes in the network. Many routing algorithms for sensor networks (e.g [60], [79]) use such clustering and data aggregation techniques.

One of the main problems in wireless sensor network is that the networks are often deployed over wide areas, which in many cases are open to the public. This environment poses a threat of physical node capture. Typical low-cost devices will not have secure storage for cryptographic keys or tamperproof hardware. If a node is captured by an attacker, its secrets can be extracted, and the node converted to meet the needs of its new owner. A particularly insidious attack is for a subverted node to announce itself as a data sink, hence collecting data from the network and passing it on to the adversary. There is not much that can be done to prevent this. One solution to limit the problem

might be to issue each node with two private keys: a source key and a sink key. A node whose battery is running low and who is unlikely (due to its low battery or its physical placement within the network) to be required to operate as a data sink, should then delete its own sink key. Now the captured node can only operate as a source of data, and so is likely to be of much less use to the adversary.

A security scheme that uses a pair of private keys can be designed with the help of identity-based cryptography techniques. This approach leads to a key agreement protocol which is authenticated and does not require any interaction between sensor devices. The following section presents a non-interactive identity-based key agreement scheme designed specially for hierarchical sensor networks.

6.3.1 Protocol description

The identity-based key agreement protocol for clustered sensor networks should have similar properties to the scheme proposed for flat sensor networks (Section 6.2). It should also have an additional property of using two private keys during the key exchange operation. The protocol proposed here is based on the scheme by Dupont and Enge [39]. This scheme is equivalent to the non-interactive ID-based key exchange proposed by Sakai, Ohgishi and Kasahara, but uses Type 3 pairings instead. However, on ordinary curves, due to the loss of symmetry, the scheme is slightly more complicated. The application of the scheme in sensor networks requires also secret key pre-distribution. For Type 3 pairings, one parameter of the pairing may be a point on $E(\mathbb{F}_p)$. The best that can be done for the other is that it should be a point on $E'(\mathbb{F}_{p^d})$, where d is a divisor of the embedding degree k, and E' is some twist of the original elliptic curve.

As in the previous protocol, the trusted authority assigns identities and secrets to each node. This is performed off-line in a secure environment. The node identities are mapped to points on the elliptic curve. The node represented by the identity ID_A is hashed and mapped to a point on $E(\mathbb{F}_p)$ of order r via $A_{sr} = H_1(ID_A)$, and to a point on $E'(\mathbb{F}_{p^d})$ via $A_{sk} = H_2(ID_A)$. Here H_1 and H_2 are functions which hash arbitrary strings to points on the elliptic curves $E(\mathbb{F}_p)$ and $E'(\mathbb{F}_{p^d})$ respectively. Node **A** is then issued with the pair of secrets sA_{sr} (its source key) and sA_{sk} (its sink key). If a node **B** is issued with a similar key pair (sB_{sr}, sB_{sk}), then both parties can calculate a mutual key as:

$$KDF(e(sA_{sr}, B_{sk}) \cdot e(B_{sr}, sA_{sk})) = KDF(e(sB_{sr}, A_{sk}) \cdot e(A_{sr}, sB_{sk})) \tag{6.2}$$

where node **A** calculates the left-hand-value, and node **B** calculates the right-hand-value. They are clearly the same because of bilinearity. It is important to note that there are fast ways to calculate products of pairings as required here.

As one can notice in equation 6.2, both nodes need to calculate a product of pairings which is more expensive than a single pairing calculation. This calculation might introduce a large computational overhead especially on the constrained 8 and 16-bit platforms. The extra complication is due to the necessity to restore symmetry to the calculation and to maintain the non-interactive nature of the protocol. However, if the nodes can agree that one will use its source key, and the other its sink key, then a mutual key can be determined in two ways:

$$KDF(e(sA_{sr}, B_{sk})) = KDF(e(A_{sr}, sB_{sk})) \tag{6.3}$$

$$KDF(e(B_{sr}, sA_{sk})) = KDF(e(sB_{sr}, A_{sk})) \tag{6.4}$$

In the first case, node **A** is an ordinary cluster member, whereas node **B** is the cluster head. In the second case, both nodes play the opposite roles. An observation from equations 6.3 and 6.4 is that both sides must use different types of keys to calculate the pairing. The computation of $e(sA_{sr}, B_{sr})$ always equals to one, because both points are linearly dependent. For the protocol to work, one of the nodes must use its source key and the other its sink key.

The protocol described here is little bit more complicated than the scheme proposed for flat WSNs. However, it offers some additional protection against node compromise. The captured node can only operate as a source of data and due to pairing properties it cannot decrypt the packets that are addressed to other nodes. These features make such a node far less useful for a possible attacker. The security of the above protocol is higher in the case when most of the nodes are deployed with source keys, and only small proportion of devices are sink-enabled. This, however, depends on a particular application of the sensor network and the clustering scheme.

6.3.2 Performance evaluation

As mentioned earlier, the key exchange protocol proposed in this section can be implemented using a Type 3 pairing. For implementation purposes, the Tate pairing was chosen over a $y^2 = x^3 - 3x + B$ MNT pairing-friendly curve with $k = 4$. For this pairing, the first parameter to the pairing can be a point on $E(\mathbb{F}_p)$, and the second parameter can be a point on the quadratic twist $E'(\mathbb{F}_{p^2})$. The parameter p was chosen as a 160-bit prime and the number of points on the curve was $34 \cdot r$, where r denotes a large prime. More details on the implementation of the Tate pairing together with values of parameters p, r and B can be found in Section 5.3.3.

In order to map identities to curve points, the method described earlier can be used. The identity string should be hashed to x and a quadratic equation should be solved to find y. Again, only identities that are known to have solutions can be chosen. In this case the process is a little more time-consuming, requiring the calculation of a modular square root. Furthermore, there is a requirement to hash identities to points of order r on $E(\mathbb{F}_p)$, which requires a multiplication by the cofactor 34. For the second parameter to the pairing, a multiplication by a larger cofactor would be necessary to map to a point of order r on $E'(\mathbb{F}_{p^2})$. However, a certain property of the Tate pairing can be utilized here to simplify the calculations. In the original Tate pairing, the second parameter need not be of order r for bilinearity to hold, and so this potentially expensive operation can be avoided [107]. This makes the cost of mapping identities to elliptic curve points similar to the cost of the same operation in the protocol from Section 6.2. The above optimization makes the Tate pairing more suitable in this protocol than the Ate pairing (which requires the first parameter on $E'(\mathbb{F}_{p^2})$ of order r).

The performance evaluation of the proposed key agreement scheme was performed in exactly the same way as in case of the previous protocol. The same three hardware platforms were used: MICAz, Tmote Sky and Imote2. Pairings were implemented using Micro-pairings and functions H_1 and H_2 were based on the SHA-1 algorithm.

Tables 6.3 and 6.4 present the total results for the key agreement protocol in clustered-base sensor networks. As in the previous protocol, the pairing calculation is the most expensive operation on all the platforms. However, the use of a Type 3 pairing makes the protocol this time more complex in terms of calculations. This is especially visible on more constrained platforms such as the MICAz and the Tmote Sky where the total

scheme duration is twice as long as in the previous case. At the same time, the energy consumption is doubled as well. The protocol described here uses also more memory than the scheme proposed earlier. The difference in performance is not that visible on the Imote2 platform where both protocols have a similar running time.

Table 6.3: *Evaluation of the key agreement in cluster-based WSNs.*

Operation	**MICAz (7.38MHz)**			**Tmote Sky (8.19MHz)**		
	Time	**ROM**	**Energy**	**Time**	**ROM**	**Energy**
Pairing	7.43s	60.9KB	175.65mJ	4.61s	34.9KB	50.89mJ
ID mapping	1.41s	0.85KB	33.34mJ	0.94s	0.54KB	10.38mJ
Other	41ms	2.91KB	0.97mJ	28ms	1.94KB	0.31mJ
Total	8.88s	64.67KB	209.96mJ	5.58s	37.38KB	61.58mJ

Table 6.4: *Evaluation of the key agreement in cluster-based WSNs.*

Operation	**Imote2 (13MHz)**		
	Time	**ROM**	**Energy**
Pairing	0.62s	44.4KB	16.34mJ
ID mapping	0.15s	0.69KB	3.95mJ
Other	4.2ms	2.34KB	110μJ
Total	0.77s	47.43KB	20.29mJ

As one can notice, the key exchange protocol based on the Type 3 pairing is a little bit more complicated than the simple scheme described earlier. It would seem that a Type 1 pairing protocol should be preferred over a Type 3 pairing scheme for the majority of wireless sensor network applications. Type 1 pairings are also much faster to compute and use less memory. Nevertheless, the Type 3 pairing protocol achieves higher security especially in a cluster network scenario. The extra protection against node compromise may be beneficial for applications where security has the highest priority and resources utilization is less important. Clearly the capture of a node with source-only capabilities will result in less damage than the capture of a node which can masquerade as either a legitimate data source or as a legitimate data sink.

6.4 TinyIBE: identity based encryption for heterogeneous sensor networks

Most of the work in the area of sensor network security has focused on flat and homogenous networks where all sensor nodes have the same capabilities in terms of memory storage, CPU power, transceiver speed and energy supply. In such networks, all nodes perform similar tasks such as sensing, routing and data processing. However, according to research results, this design is not suitable for all kinds of applications. Theoretical calculations and experimental results have shown that homogenous networks have lower performance and scalability than heterogenous networks. They are also less energy efficient and suffer from high communication and storage overheads [37]. Recently deployed sensor networks are starting to follow a heterogenous design [52].

Many of the existing key distribution schemes for sensor networks are designed to set up pairwise keys among nodes without considering the actual communication pattern [43, 98]. In heterogeneous sensor networks L-sensors usually exchange information with only a small number of their neighbours (see Figure 6.4) and they do not need to maintain shared keys with all of them. Based on this observation, a new security bootstrapping mechanism is proposed which takes advantage of H-sensors processing capabilities. The protocol is called TinyIBE, uses identity-based encryption and requires only a small number of keys in the network. The implementation results show that by using efficient public key techniques, it is possible to simplify the key distribution process and significantly reduce the communication overhead. This allows the designer to lower the energy consumption and prolong the lifetime of the sensor network.

The main task of this protocol is to securely distribute cryptographic keys among cluster heads and L-sensors in the network. The scheme is a Public Key Cryptography based algorithm and is much more computationally expensive than standard symmetric key algorithms. However, it is only used as a bootstrapping mechanism to distribute pair-wise keys after network deployment. During normal network operation, cheap link-layer security mechanisms can be used (like Tinysec [69]) to encrypt data with established symmetric keys.

Before the actual security scheme can be designed, there is a need to specify certain assumptions about the network organization. The model of the heterogeneous sensor net-

work used here assumes that the network will consist of a large number of L-sensors and only a small number of H-sensors. In this model, nodes are static and they do not move after deployment. L-sensors are cheap devices that do not have tamper-proof hardware. It is possible for an active attacker to compromise any L-sensor and extract all its secrets. The role of a cluster head is fixed and can only be performed by a H-sensor. Cluster heads store valuable data and keying material, which makes them especially attractive for attackers. The assumption that every H-sensor will be tamper-resistant is not realistic, as this would be too expensive in most deployments. The best that can be hoped for is that H-sensors will incorporate some sort of a cost effective memory protection mechanism (as proposed in [59]). The model assumes also that the environment is insecure from the beginning and the attacks are possible right after network deployment (as opposed to [95]).

Another assumption is that each device in the network has a unique node ID. During the cluster formation phase, all H-sensors broadcast *Hello* messages to nearby L-sensors. This message includes the ID of the H-sensor. Typically the transmission range of the broadcast will be large enough so that most low-end nodes can receive *Hello* messages from several H-sensors. To select its primary cluster head each L-sensor measures the strength of the signal received (through the received signal strength indicator) and chooses the one with the best value. Other IDs of H-sensors are stored in the memory as secondary cluster heads in case the primary one fails. The clustering structure formed at the initial setup of the network does not change unless a cluster head becomes unavailable. If this happens all L-sensors try to join other clusters based on their cluster head lists. The communication between an L-sensor and a H-sensor within a cluster can be single or multi-hop depending on the distance.

Sensing reports from L-sensors are sent to corresponding cluster head at regular intervals. Each H-sensor aggregates the data and sends only the result to the base station. This can be done on request or periodically. The base station does not have to be connected to the network at all times. It can be mobile and it is regarded as well-protected from physical attacks. In the scheme described, the base station can establish a secure connection with any node in the network using its master secret. It is also not possible for an attacker to masquerade as a base station without the knowledge of the master key.

6.4.1 TinyIBE overview

Heterogeneous sensor networks are asymmetric in terms of the traffic pattern and nodes capabilities. A security protocol for heterogeneous sensor network should make good use of high-end devices and save the resources of many constrained L-sensors. Previous proposals for using Identity-Based Encryption in sensor networks [94, 126] suggested the Boneh and Franklin scheme [18] for key distribution. However, this protocol is symmetric in nature and requires expensive operations for both encryption and decryption.

The solution proposed here uses a simple variant of the Sakai and Kasahara Identity-Based Encryption scheme [105]. This Identity-Based Encryption method is not as well-known as that of Boneh and Franklin but it has its advantages that perfectly fit the heterogenous sensor network setting. There is no pairing calculation required for encryption and there is no need to hash identities to elliptic curve points. These potentially expensive operations are avoided, which makes the whole scheme more affordable, especially for low-end sensor nodes.

The TinyIBE scheme can be divided into two phases, each consisting of two stages. In the pre-deployment phase, the trusted authority performs *Setup* and *Extract* operations. This is done off-line in a secure environment:

Setup. The base station generates global system parameters that include its own master secret $s \in \mathbb{Z}_r^*$, a pairing-friendly supersingular elliptic curve E over $\mathbb{F}_q$ and random points $P \in E(\mathbb{F}_q)$ and $Q \in E(\mathbb{F}_q)$ of order r, where $Q = sP$. It generates also $g = \hat{e}(P, P)$ and defines two hash functions $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ and $H_2 : \mathbb{F}_q \rightarrow \{0,1\}^n$, for some n. All nodes are pre-loaded with system parameters: $< E(\mathbb{F}_q), Q, P, g, n, H_1, H_2 >$.

Extract. The base station issues a unique identity ID_X for each node. The identity of every H-sensor is hashed to a value $a = H_1(ID_H)$ and a corresponding private key is generated $D = \frac{1}{s+a}P$. Each H-sensor is also pre-loaded with a broadcast key pair $< B_H, E_H >$ and identities of all the L-sensors in the network (ID_L).

In the above notation sP is a point multiplication operation and $\hat{e}(P, P)$ is a Type 1 pairing. The second phase of TinyIBE scheme starts after network deployment and all the operations are run by the nodes to encode and decode messages that are exchanged between them. It consists of two main stages:

Encrypt. Every L-sensor selects its primary cluster head, finds the value $a = H_1(ID_H)$ for a given H-sensor and generates random $w \in \mathbb{Z}_r^*$ and $t \in \mathbb{Z}_q^*$. To encrypt the session key t, the L-sensor creates the ciphertext C_1, C_2:

$$C_1 = w(Q + aP) \qquad C_2 = t \oplus H_2(g^w) \tag{6.5}$$

It includes also its identity in the message.

Decrypt. After receiving the encrypted message, every cluster head checks the identity of the L-sensor. The decryption process can only start in the case of a positive ID check. The H-sensor recovers the session key t by using its private key D:

$$t = H_2(\hat{e}(D, C_1)) \oplus C_2 \tag{6.6}$$

The session key is used with a symmetric cipher to encrypt the broadcast public key B_H.

The decryption process works because of the bilinearity property of the pairing:

$$\begin{aligned} \hat{e}(D, C_1) &= \hat{e}\left(\frac{1}{s+a}P, w(Q + aP)\right) \\ &= \hat{e}\left(P, sP + aP\right)^{\frac{w}{s+a}} \\ &= \hat{e}\left(P, P\right)^{w} \\ &= g^w \end{aligned} \tag{6.7}$$

After the decryption step, every L-sensor can use its session key to securely communicate with a given cluster head. It can also authenticate broadcasts by using the Elliptic Curve Digital Signature Algorithm (ECDSA). Secure broadcasts are important especially in the case when remote network reprogramming is needed.

Figure 6.7 presents all of the TinyIBE operations that are performed after network deployment. The need for identity exchange between nodes cannot be considered an overhead incurred by the key distribution protocol. The majority of existing sensor network routing algorithms already require that nodes know each other's IDs. It is also assumed that H-sensors are equipped with a lot more storage space than L-sensors. This allows

them to store all the L-sensor identities (which could be simple serial numbers) without introducing large overhead. Additionally, ID sizes are negligible when compared to public key and certificate sizes. It is more convenient to maintain a list of eligible L-sensor identities rather than storing all their public keys.

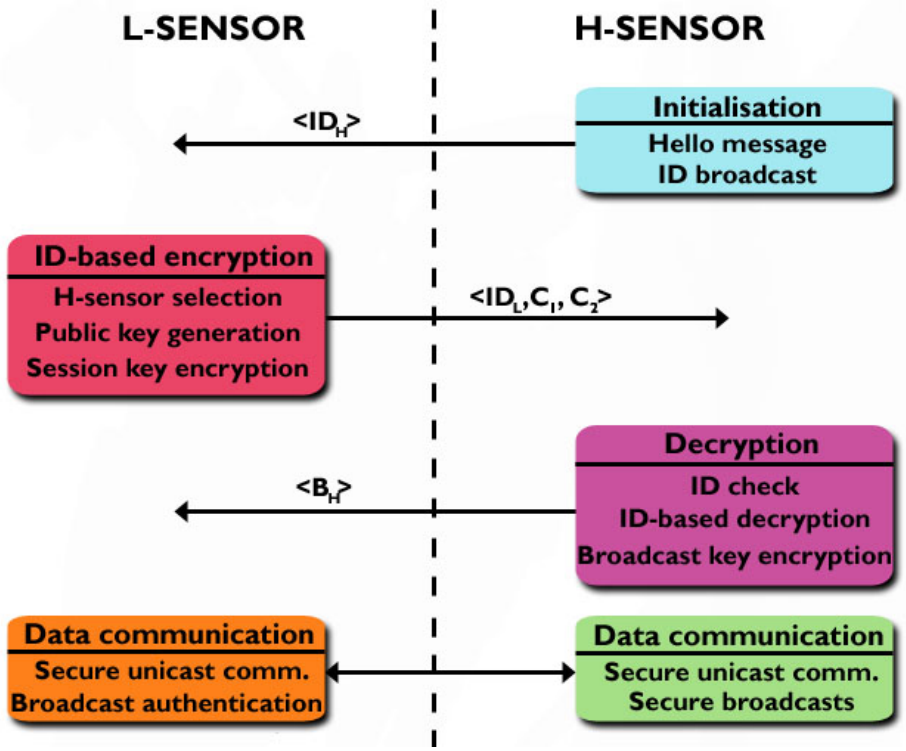

Figure 6.7: *Key distribution during the on-line phase of TinyIBE.*

TinyIBE provides authenticated key distribution without using certificates. The message encrypted for a specific cluster head can only be decrypted with a correct private key D which is in the possession of a single H-sensor and the base station. Every H-sensor authenticates itself to any L-sensor simply by decrypting the message and sending back data encrypted with the session key. This prevents L-sensors from sending sensor readings to fake cluster heads or base stations. Even if an adversary steals an identity of a node or takes fake identities, it still cannot act as a cluster head or a base station without obtaining the proper private key D or the master secret s. Those values are well-protected and the protocol guarantees that the master secret is not revealed, even if an attacker subverts all the nodes in the network[1].

It is also important to prevent fake L-sensors from taking part in the network. A H-sensor can only setup a shared key after checking the L-sensor's identity. If the ID is not

[1] it is protected by a hard discrete logarithm problem in the elliptic curve setting.

on the list the cluster head can contact the base station to clarify if this ID is valid. In the case where two L-sensor claim the same ID, the H-sensor informs the base station of a possible adversary in the network. The communication between the base station and H-sensors is secured using the Encrypt/Decrypt mechanism described earlier. Both parties use ID-based encryption, so the communication is authenticated in both directions. The base station can also perform periodic checks on the lists of nodes connected to different H-sensors. In this way, the base station can detect intruders that claim the same IDs in various parts of the network.

New node addition is easy using TinyIBE and does not require any new keying material for existing nodes. The base station performs the *Setup* and *Extract* operations and deploys a node with a new ID. The H-sensor will contact the base station to validate the new ID during the session key establishment phase. Additionally any external party can deploy its own L-sensors and add them to an existing sensor network. They only have to agree on the identities with the base station using a secure channel. When an L-sensor becomes an orphan, it can quickly join another cluster by encrypting a message for its secondary cluster head. This is done according to the scheme described above. In the case of node compromise the base station can revoke keys by simply removing the ID and the corresponding session key from the list that is held by every cluster head. This prevents an L-sensor from establishing a secure connection with any H-sensor of its choice.

TinyIBE is designed to work with heterogeneous sensor networks and uses more powerful sensor nodes to perform complex cryptographic operations. Each L-sensor performs less expensive point multiplications and is only required to store two keys. The extra bandwidth overhead associated with cryptography is limited to only one encrypted message that the L-sensor sends to its cluster head. Software implementation results show that TinyIBE is feasible on constrained sensor nodes without using any crypto hardware accelerator. In addition, TinyIBE is the first practical implementation of a complete Identity-Based Encryption scheme on sensor devices.

6.4.2 Implementation of TinyIBE

TinyIBE is based on the mathematical theory of elliptic curves which involves operations that are computationally intensive. Two of the most expensive operations in the scheme are the point multiplication and pairing calculation. The implementation of the whole

scheme is quite difficult and requires cryptographic primitives used in Elliptic Curve Cryptography and Pairing-Based Cryptography.

In order to check the feasibility and performance of TinyIBE, the scheme was implemented on three commonly used sensor devices. The Imote2 platform was used a an H-sensor and L-sensors were deployed on the MICAz and the Tmote Sky nodes. All three platforms use the same CC2420 radio transceiver which allows them to communicate within the same frequency band.

In some sensor network scenarios, a 64-bit security level has been used [98] to reduce the security overhead. However, sensor networks should use a minimum 80-bit security level (equivalent to 1024-bit RSA) as it is the current standard in information security. An 80-bit symmetric cipher provides equivalent security to a 160-bit elliptic curve system. In order to achieve a similar security level for pairing-based cryptography there is a need to work with 1024-bit values. This fact makes the pairing computation the most expensive operation in TinyIBE. The pairing calculation in TinyIBE is performed only by the more powerful H-sensors during the decryption process. This allows us to save on code space and execution time on low-end devices.

The scheme can be implemented using a Type 1 pairing. Results presented in Chapter 5 have shown that the η_T algorithm [7] can be efficiently implemented on embedded sensor devices. Hence this pairing type was chosen for the TinyIBE scheme. The η_T pairing was evaluated over a binary field $\mathbb{F}_{2^m}$, where m defines the length of the binary polynomial that represents field elements. In accordance with the 80-bit security level, a supersingular elliptic curve $y^2+y=x^3+x$ was chosen over the binary field $\mathbb{F}_{2^{271}}$ with the embedding degree $k=4$. The details of the pairing implementation can be found in Section 5.3.2. In order to make the TinyIBE scheme practical, the implementation was based on NanoECC and Micro-pairings. A detailed description of both software packages is presented in Sections 4.3 and 5.3 respectively. TinyIBE was integrated with TinyOS v1.15 and evaluated on three different sensor node platforms.

6.4.3 Experimental results

In the first phase of the TinyIBE scheme, the trusted authority pre-loads all the necessary information on to sensor nodes. This is performed off-line before network deployment. The *Setup* and *Extract* steps are not time critical and they do not have any influence on

the network performance. In what follows, the focus is only on the operations that are carried out by the sensor nodes during the session key establishment phase.

The *Encrypt* step involves hashing and calculation of two point multiplications. In TinyIBE, the popular SHA-1 function was used, but in general any other secure hash function can be adopted for this purpose. The calculation of C_1 is the most expensive step during the encryption process. A H-sensor public key can be regarded as the value $(Q + aP)$ which is fixed for a given cluster head. If it is assumed that the encryption step will be performed multiple times for the same H-sensor, this value can be cached to save one point multiplication during subsequent encryptions. However, in most cases the encryption step will be performed only once for a given cluster head. The calculation of aP is a fixed point multiplication with a 160-bit scalar and can be accelerated using the precomputation methods described in Section 4.1.2.2. To compute C_2, the g value exponentiation needs to be performed, but this operation is negligible when compared with the C_1 calculation.

The decryption step is pretty straightforward and requires one η_T pairing calculation and one hashing to retrieve the session key. The pairing takes as the first parameter the H-sensor's private key, which is a constant value. This fixed parameter can be exploited for precomputation, which accelerates the pairing computation at the expense of some storage overhead. In the TinyIBE implementation, the pairing calculation was performed only on more powerful H-sensors, hence additional acceleration was not necessary. However, the precomputation option should be considered in the case where more constrained devices are used as H-sensors. The performance evaluation presented below excludes the ID check operation performed by every H-sensor before the decryption step.

Table 6.5: *TinyIBE evaluation results*

Platform	**Encryption**			
	Time	**ROM**	**RAM**	**Energy**
MicaZ (7.38MHz)	3.93s	39.6KB	2.9KB	92.67mJ
Tmote (8.19MHz)	2.62s	30.3KB	3.2KB	27.12mJ
Platform	**Decryption**			
	Time	**ROM**	**RAM**	**Energy**
Imote2 (13MHz)	462ms	32.87KB	4.12KB	12.12mJ
Imote2 (104MHz)	57.7ms	32.87KB	4.12KB	3.76mJ

Table 6.5 presents the evaluation results for the TinyIBE implementation. All the mem-

ory and timing results were obtained using cycle accurate simulators. The AVR Studio was used for the Atmega128L processor (MICAz) and the IAR Embedded Workbench was used for both the MSP430 (Tmote) and the Xscale (Imote2) platforms. The energy consumption was measured experimentally for the MICAz and the Tmote Sky nodes. Data sheet figures were used in the case of the Imote2 node. As can be seen in Table 6.5, the Tmote Sky node is almost four times more efficient in terms of energy consumption than the MICAz platform. The Imote2 also has low energy consumption, especially when its CPU frequency is set to 104MHz.

The encryption times for L-sensors are acceptable, given that these operations are performed very rarely, and mainly at the beginning of the network operation. After the session key establishment, the nodes will use cheap symmetric key encryption methods. The Imote2 platform handles the decryption process well, and its 32-bit ARM processor facilitates the calculation of complex cryptographic operations. The best performance can be achieved when the maximum frequency is set (416MHz). At this speed the Imote2 can perform around 70 decryption operations per second. So far, TinyIBE is the fastest Identity-Based Encryption scheme for sensor networks.

The memory footprint on L-sensors is considerable and it is mainly due to the large size of the Elliptic Curve Cryptography library. However, this also includes all the necessary library code needed to perform the Elliptic Curve Digital Signature Algorithm during the secure broadcast verification. RAM utilization may seem high for low-end devices, but the memory is reserved only for the duration of cryptographic operations. The values shown in Table 6.5 present the peak numbers for stack usage during the program execution.

6.4.4 Performance analysis

Existing security mechanism for WSNs are mainly based on key-predistribution mechanisms. Most of the key pre-distribution schemes were designed for homogenous sensor networks (e.g. [43]), and once applied in heterogeneous sensor networks they suffer from high communication and storage overhead. Du *et al.* [36] tried to address these issues by proposing an asymmetric key pre-distribution scheme that uses the capabilities of H-sensors. The authors in [95] and [63] also tried to adopt key pre-distribution schemes to fit heterogenous network design. However, all these solutions are based on symmetric

key techniques and they do not provide a perfect trade-off between resilience and storage. Compared with symmetric key cryptography, Public Key Cryptography provides a more flexible and simple interface, which reduces the number of necessary keys in the network.

One of the biggest advantages of TinyIBE when compared with the state-of-the-art is the significant storage saving in terms of cryptographic keys. In the following calculations, the number of H-sensors and L-sensors in a heterogeneous sensor network is denoted as M and N, respectively. In the case of a key pre-distribution scheme proposed by Eschenauer and Gligor (E-G [43]) the total number of pre-loaded keys in the network is equal to: $m(M+N)$, where each sensor is pre-loaded with m keys. The value of m depends on the key pool size (P) and the probability of sharing at least one key between two nodes (p'). Du *et al.* proposed asymmetric pre-distribution (A-P [36]) where H-sensors are pre-loaded with more keys (x) than L-sensors (y). The number of pre-loaded keys can be calculated as $xM + yN$ where $xy = m^2$. The Elliptic Curve Cryptography scheme proposed in [35] decreases the number of necessary keys to $M(N+3)+2N$. The TinyIBE scheme proposed here assumes that each H-sensor is pre-loaded with only three keys and the total number of pre-loaded keys in the network ($3M$) is independent of the number of L-sensors. This feature allows TinyIBE to scale gracefully with the number of nodes in the network.

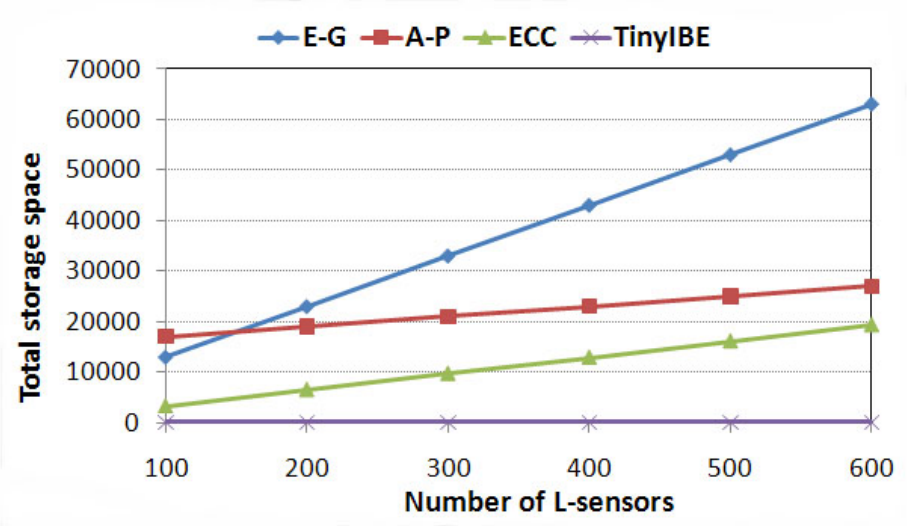

Figure 6.8: *Pre-loaded key storage space.*

Figure 6.8 presents the comparison between TinyIBE and different security solutions proposed for HSNs. The parameters were set as: $M = 30$, $m = 100$, $P = 5000$, $p' =$

0.87, $x = 500$, $y = 20$ and the total number of pre-loaded keys for each scheme was calculated. As can be seen in Figure 6.8, TinyIBE requires much less storage space for the pre-loaded keys (for a constant number of 90 keys) than other solutions. The storage savings of TinyIBE increase drastically with the number of L-sensors in the network. The IBE scheme proposed here uses the smallest key space among all existing key distribution protocols for heterogeneous sensor networks.

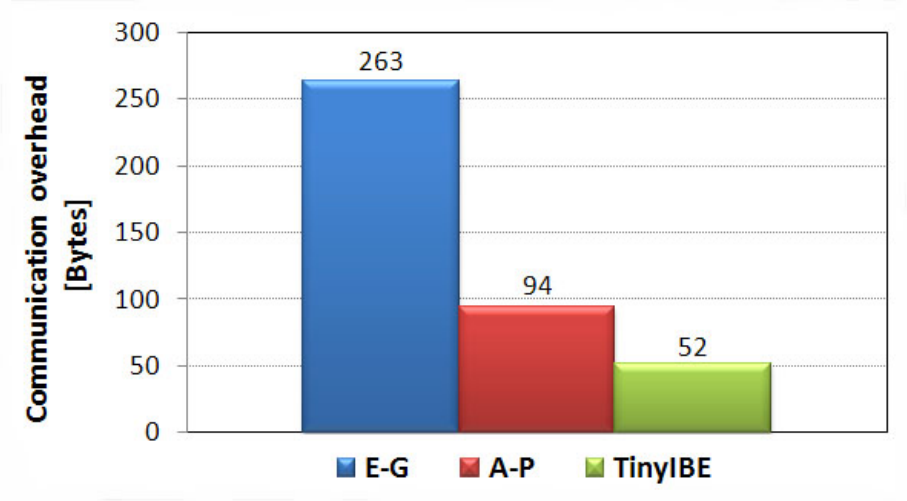

Figure 6.9: *Communication overhead.*

TinyIBE also introduces significant savings in communication overhead. Figure 6.9 shows how much data needs to be exchanged between an H-sensor and a given L-sensor in each scheme, to establish a common session key. This comparison does not include the packet header or any additional overhead incurred by packet fragmentation. In the Eschenauer-Gligor (E-G) scheme each sensor broadcasts the list of identifiers of keys on their key ring. When the key pool $P = 10000$, m needs to be larger than 150 to achieve a high key sharing probability of 0.9 [43]. Each key identifier requires at least 14 bits and the resulting message will have a size of 263 bytes. The A-P scheme [36] uses a similar technique in the key agreement phase. To achieve the same key sharing probability as for the E-G scheme, the parameters need to be set as $x = 450$, $y = 50$, $P = 10000$. Every L-sensor sends a key list message to its H-sensor which includes the L-sensor's ID, the list of key IDs and the node's location. This results in a communication overhead of 94 bytes (assuming 2 bytes for ID and 4 bytes for location data). The TinyIBE scheme requires an exchange of only one message $< ID_L, C_1, C_2 >$ to establish a shared session key. The C_1 value is a point on $E(\mathbb{F}_{2^{271}})$ which can be compressed to 34 bytes and C_2 has the size of

the session key (128-bits). The resulting message has 52 bytes. According to Figure 6.9, TinyIBE is the most efficient scheme in terms of data exchange. The communication overhead in TinyIBE is independent of the network size and fixed for each node pair. In case of the key pre-distribution schemes ([43], [36]), the overhead increases with the number of pre-loaded keys and the key pool size. The results of the Elliptic Curve Cryptography scheme proposed in [35] cannot be compared with TinyIBE as the paper does not specify the public key encryption method used for session key distribution.

6.5 Security analysis of Identity-Based Cryptography protocols

Like any wireless networks WSNs are vulnerable to different kinds of attacks including jamming, eavesdropping and spoofing. There are also additional attacks that are exclusive to sensor networks. Deployments in public areas introduce a risk of a physical attack and limited battery power opens the door for a whole range of DoS (Denial-of-Service) attacks, resulting in a node's energy depletion. Sensor networks that use node identities are also prone to replication and Sybil [93] attacks. In the first type of attack the adversary adds a node to the network by replicating the ID of an existing node, or by generating a fake ID. The second attack relies on presenting multiple IDs by a single node, in order to intercept data from the network.

Perhaps the biggest problem in sensor networks is the node capture attack. Sensors are usually very cheap and do not have any protection against tampering. Recent results show that standard sensor nodes, such as the MICA2, can be compromised in less than one minute [59]. In the case when an adversary manages to compromise a sensor node, it gains access to valuable information and can stage a number of different insider attacks, targeting nodes within the cluster and across the network. Insider attacks are a lot harder to prevent and detect, because they are performed by nodes that are regarded as legitimate members of the network. There is not much that can be done to prevent node compromise attacks. Therefore, the best that can be hoped for is that the network will be immune to passive attacks, and that it can survive in some sense the loss and capture of a certain percentage of its nodes.

The performance of the proposed Identity-Based Cryptography schemes have been evaluated in the previous sections. However, the resistance of these security schemes against different types of attacks have not been discussed in detail. This section presents

a security analysis of Identity-Based Cryptography protocols in wireless sensor networks. It describes also possible attacks on sensor networks and recommended defensive measures.

6.5.1 Possible attacks

Wireless sensor networks are susceptible to more attacks than ordinary networks. A whole range of attacks from passive eavesdropping and active injection of bogus messages, to node physical capture and subversion is possible. Especially dangerous are various denial-of-service attacks [124]. Potential threats include attempts to disrupt, subvert or destroy a network, but in many cases a simple denial-of-service attack would be sufficient for the adversary to succeed. A denial-of-service attack is any event that diminishes or eliminates a network's capacity to perform its expected functions. Hardware and software failures, resource exhaustion, environmental conditions or a mix of these factors, can cause a denial-of-service attack. Such an attack results in a loss of availability and degrades the efficiency and functionality of the whole network. It is extremely hard to defend against these types of attacks. A standard attack is to jam wireless transmission between the nodes by sending a strong radio signal, that interferes with the network operational frequency. Another type of denial-of-service attack might be performed by simply taking out a node's batteries. Constant flooding of a network with packets is also a big threat, as it leads to a quick depletion of nodes energy.

Most of the sensor networks are particulary vulnerable to several key types of attacks. Attacks can be performed in a variety of ways and are not limited to simple denial-of-service attacks. They encompass a variety of techniques that pose a serious threat against sensor devices. The main types of attacks include:

Monitoring and eavesdropping. This is the easiest attack on sensor networks. It may compromise the privacy of information that is exchanged in the network.

Traffic analysis. This attack may lead to identifying certain entities in the network, without even understanding the contents of the packets. In this way valuable information is gained, that may be used to perform a denial-of-service attack.

Replication attack. An attacker adds a node to the network by replicating the ID of an existing node. The adversary gains unauthorized access to the network. Certain

packets can be easily misrouted because of replicated identities.

The Sybil attack. Single node presents multiple identities to other nodes in the network. This reduces the effectiveness of fault-tolerant schemes, such as distributed storage, multi-path routing and topology maintenance.

Routing attacks. By spoofing, altering or replaying routing information, adversaries may be able to create routing loops, attract or repel network traffic, generate fake routing messages, and degrade the network performance.

Selective forwarding. Compromised node may refuse to forward particular packets or even completely limit its role as a router. This may lead to partitioning in the network.

Denial of service. It is a standard attack that limits the availability of a node in a sensor network. This could be achieved in many different ways like: jamming, energy depletion attacks or network flooding with bogus messages.

Node capture. It is a serious threat to sensor network security. Adversaries can tamper with the devices, extract cryptographic keys and subvert the nodes.

Sinkhole attack. This type of attack lures nearly all traffic from a particular area of the network to the compromised node. An adversary may spoof or replay an advertisement for an extremely high quality route to the base station, in order to perform this attack.

Wormhole attack. It is a more sophisticated attack, where two devices conspire with each other, to provide a low latency route in the network (wormhole). Most of the communication between the nodes goes through this wormhole, as it appears to be a short and efficient path in the network. The adversary can exploit this traffic to perform other types of attack.

6.5.2 Defensive measures

Security in sensor networks is a large and complex topic due to the many possible threats in these networks. Many security issues concerning: key establishment, routing, intrusion detection, broadcasting, trust management and data aggregation, must be solved to provide a complete security system.

Well planned denial-of-service attacks on sensor networks might be very hard to defend. Jamming the network with a strong signal is one of the simple attacks, that may disable the whole network without any difficulties. Improved radio hardware including spread spectrum capability with low probability of detection and interception, may someday help resolve this problem. Unfortunately, the currently proposed spread spectrum capability defined in IEEE 802.15.4, is designed to protect against only unintentional interference, rather than denial of service attacks.

The research area of wireless sensor network security is still in its early stages. A clear line of defence is not yet available. The best that can be achieved at the moment, is to use a multi-fence security mechanism that should be embedded into every component of the network. Obviously, a single solution against all security threats in sensor networks does not exist. Instead, a set of solutions can be proposed, to provide the functionality that is needed for a given application. Possible security threats and related defensive measures are summarized in Table 6.6. As can be seen, most of the security solutions rely on a proper key distribution technique that is based on Public Key Cryptography. The Identity-Based Cryptography protocols, described in this chapter, can provide a reliable framework for all the security services built on top of them.

Table 6.6: *Security threats in WSNs and possible defensive measures*

Security threat	**Countermeasure**
Unauthorized access	Public key cryptography, random key distribution
Eavesdropping	Link or network layer encryption (symmetric algorithms), access control
Denial of service	Intrusion detection, improved radio hardware, node redundancy
Routing attacks	Secure routing, message authentication codes (MACs), digital signatures
Node capture	Tamper resistant hardware, robust PKC schemes, node revocation
Intrusion	Intrusion detection, secure group communication, nodes authentication

6.5.3 Non-interactive key agreement security evaluation

The non-interactive identity-based key distribution schemes proposed in this thesis prevent intruders from taking part in the network, injecting messages into the system, as well

as eavesdropping on communication between legitimate nodes. They employ asymmetric cryptography primitives, which makes the effect of a node being compromised strictly local. Subverting one node does not reveal anything about communication between other sensor devices. This allows the network to continue to operate even with a small percentage of compromised nodes. The key distribution schemes proposed here, limit the effect of a node compromise, but cannot fully protect against this attack. Additional protection can be provided by using an intrusion detection mechanism, which spots misbehaving nodes with genuine identities. Many of the possible attacks (sinkhole attack, selective forwarding, wormhole attack) can be staged only after one of the nodes in the network is subverted. Therefore it is important to apply mechanisms, which detect malicious behaviour of sensor nodes.

One of the problems in every ID-based security scheme is proper management of node identities. The assignment and control over nodes IDs must be performed only by the trusted authority, to ensure that each identity is unique in the network. Such an approach allows the network to achieve high resistance against the Sybil attack and the node ID replication attack. In the case when an attacker steals an identity of a node, or takes fake identities it still cannot establish a shared key, because he does not have the master key s and is not issued with a private key sN_X, needed for decryption. The ID-based key agreement schemes also guarantee, that the master secret s, is not revealed even if all the nodes in the network are subverted (only the trusted authority holds the master key). Network access control is also provided, as only the trusted authority issues identities and pre-loads sensors with valid private keys. An active attacker can encrypt messages to given identities, but cannot decrypt any message without a proper private key.

Another problem, that should be also considered, is the whole range of denial-of-service attacks that try to deplete the energy of sensor nodes. The Identity-Based Cryptography protocols proposed here, perform public key operations, which are computationally intensive. Establishing a private key between two nodes requires the calculation of a pairing, which consumes precious energy. Constant requests for a key exchange, from a malicious mote, would quickly deplete node's energy resources. This can be prevented with a simple mechanisms that limits the number of key establishment trials with a certain node. Every request above the limit, which comes from the same source, should

be simply discarded.

6.5.4 TinyIBE security analysis

The security requirements for heterogeneous sensor networks are similar to the requirements in flat networks. TinyIBE provides authenticated key agreement between an L-sensor and a H-sensor. This operation requires an exchange of only one message between the nodes, and does not require digital certificates. Dangerous attacks, such as the Sybil attack and node ID replication, can be easily prevented. Even in the case when an adversary steals an identity of a node, or takes fake identities, it still cannot masquerade as a cluster head or a base station without the knowledge of the proper private key D or the master secret s. Those values are well-protected by H-sensors and the base station. TinyIBE also guarantees that the master secret cannot be calculated based on the private keys of H-sensors. In addition, the ID replication attack gives a chance to spot the intruder that uses the same ID as a legitimate node.

Careful management of identities in the network is especially important in the case of the TinyIBE scheme. Proper ID checks prevent fake L-sensors from taking part in the network. A H-sensor can only setup a shared key after checking the L-sensor's identity. The IDs of L-sensors should be included on the list maintained by every H-sensor. If the ID is not on the list, then the L-sensor is treated like an intruder and the H-sensor (if possible) should clarify with the base station, if the ID is valid. The base station itself can periodically check the lists of node IDs, to update them, and to detect intruders, which use the same IDs in different parts of the network.

Similar to the two previous schemes, TinyIBE applies asymmetric cryptography primitives, which minimizes the effect of node compromise. Subverting one L-sensor does not reveal the session keys used by other nodes in the network. The resilience of TinyIBE against node capture can be measured by calculating the compromising probability. This parameter checks the effect of an L-sensor compromise on the rest of the network. The compromising probability can be defined as the probability that an attacker can decrypt the communication between any two nodes, after capturing c L-sensors.

Paper [35] presents the formula to calculate the probability that two sensors have exactly j common keys in the Eschenauer and Gligor scheme (E-G) [43]:

$$P(j) = \binom{P}{j}\binom{P-j}{2(m-j)}\binom{2(m-j)}{m-j}/\binom{P}{m}^2 \tag{6.8}$$

where P is the key pool size and m is the number of pre-loaded keys in each node. The compromising probability for the E-G scheme can be calculated as:

$$C(m) = \sum_{j=1}^{m}(1-(1-\frac{m}{P})^c)^j P(j)/\sum_{j=1}^{m} P(j) \tag{6.9}$$

Similar formulas can be found in [36] to compute the same probability in the asymmetric pre-distribution scheme (A-P).

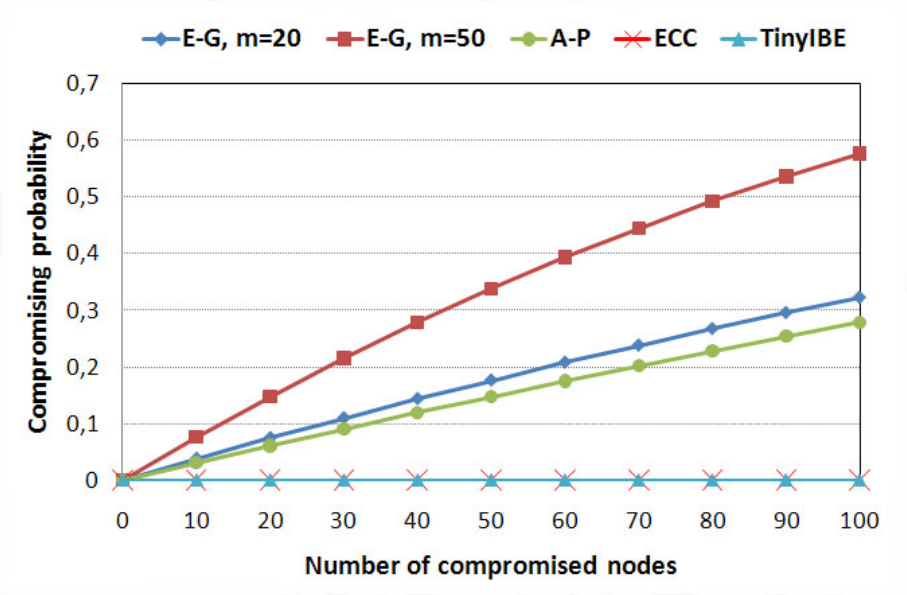

Figure 6.10: *The compromising probability of different key distribution schemes.*

Figure 6.10 compares the compromising probability for different key distribution schemes in heterogeneous sensor networks. The following parameters were used: $P = 5000$, $x = 125$, $y = 20$ (A-P scheme) and two different values of m for the E-G scheme. The Elliptic Curve Cryptography-based protocol [35] and the TinyIBE scheme employ asymmetric cryptography primitives. After the session key distribution phase, each pair of nodes has a different shared key. Hence, compromising c L-sensors does not affect the security of communication among other pairs of nodes. The compromising probability in both schemes is always equal to zero, for all c values. For the key predistribution schemes the compromising probability increases significantly with the number of pre-loaded keys (m or y). Therefore, the resilience to node compromise attack is much higher in public key

protocols. In addition, TinyIBE can easily revoke keys used by compromised L-sensors. The base station needs to simply remove the ID and the corresponding session key from the list that is held by every cluster head. This prevents an L-sensor from establishing a secure connection with any H-sensor of its choice.

6.6 Summary

The problem of distributing symmetric keys within a sensor network is one of the fundamental security issues in WSNs. Existing key distribution mechanisms have many flaws and the level of security provided is not satisfactory for many applications. Hence, there is a need to develop new security solutions that would offer high security levels and simplify key management in the network. This chapter shows that such solutions can be developed with the use of bilinear pairings and identity-based cryptography schemes.

Wireless sensor networks can be divided into three broad groups based on nodes organization: flat, cluster-based and heterogeneous. Each of these groups have specific features and communication patterns that need to be considered when developing key distribution schemes. Optimal security solution should be tailored to a specific network organization.

This chapter proposes three different ID-based key agreement protocols which were developed for specific sensor network models. For a clustered network scenario a novel variant of a key exchange protocol, which is based on a Type 3 pairing, has been proposed. This protocol uses sink and source keys, which limit the effect of node compromise and increase the level of security for more demanding sensor network applications. However, the implementation results of this protocol have shown, that its performance is lower than other Identity-Based Cryptography schemes on sensor devices. This protocol uses a Type 3 pairing, which is much slower than the η_T pairing over binary fields.

The second scheme investigated the application of a simple non-interactive key exchange protocol in flat sensor networks. This scheme uses a Type 1 pairing and allows any two nodes in the network to agree upon a shared secret, in a secure manner. The evaluation results have shown that this key agreement is feasible and practical on different sensor platforms, and can be evaluated in around 4.3s on a tiny 8-bit sensor device. The security analysis proves that appropriately designed ID-based scheme might be a perfect solution for the key distribution problem in sensor networks. This protocol is preferred

over the previous scheme and is especially recommended for use in different sensor network scenarios, due to its performance and simplicity.

The last security protocol proposed in this chapter utilizes identity-based encryption to distribute keys in a heterogeneous sensor network. TinyIBE exploits the enhanced capabilities of high-end cluster heads to secure the communication between different classes of sensor devices. The protocol was designed specially for heterogeneous sensor networks to provide a new and simplified way for key distribution in the network. A common misconception is that identity-based encryption is not a suitable security technique for sensor networks. The argument is that Identity-Based Encryption suffers from identity theft and replication problems, and that it is too heavyweight for sensor devices. The results presented in this chapter prove otherwise. The TinyIBE scheme is protected against fake cluster heads and is feasible to implement on constrained sensor nodes. The encryption procedure takes less than four seconds on an 8-bit MICAz platform. Identity theft and replication issues can be addressed by simple ID check procedures. A comparison with the state of the art, shows that the TinyIBE protocol introduces significant savings in key storage space and communication overhead, compared to existing solutions for heterogeneous sensor networks. Additionally it provides a higher level of security and stronger resilience against node capture attack, than any key pre-distribution technique.

The results presented in this chapter prove for the first time that key distribution schemes based on Identity-Based Cryptography are practical in wireless sensor networks. They can be applied in real-life applications on standard sensor nodes, without any specialized hardware accelerators. Features such as non-interactive key agreement and certificate-less node authentication, makes the proposed schemes superior security solutions in a sensor network environment. The application of ID-based cryptosystems to sensor networks has a very promising future. This direction is especially interesting, as more powerful nodes are being developed that allow more complex security protocols.

CHAPTER 7

Conclusions

Wireless technology is evolving very rapidly nowadays. Large distributed networks comprising of many cheap and tiny sensing devices have become a reality. In the near future it is likely that these wireless sensor networks will be deployed all over the world on a large scale. It is predicted that these pervasive systems will be integrated with the environment to form a bridge between the digital and the physical world.

This revolution in technology brings of course many open questions concerning privacy and data confidentiality. In many cases, sensor networks will handle sensitive and important data in open areas that anyone can access. The wireless nature of communication between the nodes makes it easier to eavesdrop, intercept and inject bogus information into the network. The security problem in this environment is especially important because wireless senors will gather information not only about the environment but also about people.

The range of sensor network applications spans environmental monitoring and home automation to more complex ones like traffic control and health care. These applications require various security levels and services. Simple applications like habitat monitoring need only basic security assurances that can be fulfilled by using symmetric key algorithms. More advanced sensor network applications require more complex methods for applying security. Sensing, monitoring and actuating systems are expected to play a key

role in reducing buildings overall energy consumption. Sensor networks can provide intelligent control based on precise real-time measurements to reduce the energy consumption and greenhouse gas emissions of commercial buildings. This can make a major impact on the issues of climate change and energy use in buildings. Leveraging wireless sensor systems to monitor critical infrastructure brings many novel research challenges especially in the field of security.

Despite many efforts, security in wireless sensor networks still remains an active research field. The devices used in sensor network have limited capabilities in terms of processing power, available storage and energy. It is difficult to design a security solution which has minimal resources utilization and at the same time offers high level of security. Most of the security schemes proposed for wireless sensor networks are based on simple symmetric key cryptography mechanisms. This approach minimizes resource utilization, but does not solve the problem of dynamic key distribution in the network.

In fixed networks, the problem of key distribution is usually solved through Public Key Cryptography methods. Published results showed that traditional Public Key Cryptography solutions (e.g. RSA) are not suitable for constrained devices and thus new security methods are required. These new key distribution methods should offer the same security services as standard public key algorithms, but at a lower cost that is acceptable for sensor devices.

This thesis proposes new mechanisms for key distribution in sensor networks. It showes that Public Key Cryptography can be applied in sensor networks through the use of bilinear pairings and identity-based cryptography schemes. This new approach to wireless sensor network security is promising because sensor networks seem to be a perfect setting for applying Identity-Based Cryptography. One of the main advantages of the proposed methods is that they can provide simple key distribution in the network without the use of expensive certificates. They also introduce significant savings on radio transmission and the storage of cryptographic keys when compared with other solutions. In order to check the feasibility of the proposed schemes, all key distribution protocols were implemented on standard sensor nodes. The evaluation results show that the protocols have short execution time and low energy consumption which makes them suitable even for tiny 8-bit devices. These results prove also for the first time that the use of Identity-Based Cryptography schemes is a practical way of providing Public Key

Cryptography in sensor networks.

7.1 Research objectives achieved

The application of identity-based cryptography protocols in sensor networks is a difficult task that requires execution of cryptographic primitives used in Elliptic Curve Cryptography and Pairing-Based Cryptography. The computational complexity of Identity-Based Cryptography schemes is significant, and implementations on standard sensor nodes need to be highly optimized to be practical. Identity-Based Cryptography schemes require many modules and each of them needs to be implemented efficiently in order to achieve the maximum performance of the protocols. The ID-based key distribution schemes also need to be designed for particular network organization. A solution designed for a heterogeneous sensor network might have low performance when applied in a flat network scenario.

This thesis presents a detailed description on how to provide practical key distribution mechanisms for sensor network. It also proposes three new key distribution protocols tailored for different sensor network organizations and various communication patterns. New contributions to the state-of-the-art are added on many levels of development.

On the lowest level, finite field arithmetic operations on constrained CPU's are described in detail. In particular, new methods for performing multiprecision multiplication are presented. In the case of the prime finite field, an improved hybrid method is proposed, which accelerates the calculations by using extra registers of a given processor. The general applicability of the idea has been demonstrated on three different 8, 16 and 32-bit architectures. In all of these cases, significant performance improvements have been achieved. For the binary field case a new hierarchical method for polynomial multiplication is proposed. This new technique is optimized for fields of large characteristic and achieves a significant improvement in execution time when compared with previous methods.

The thesis also studied the implementation of basic Elliptic Curve Cryptography primitives on different sensor platforms. This research resulted in the development of the NanoECC library. The library has the best performance among existing Elliptic Curve Cryptography implementations on different sensor devices and can be easily integrated

into sensor network applications. The high speed of the library was achieved through efficient implementation of point multiplication algorithms and arithmetic routines optimization in assembly language. The energy consumption of basic Elliptic Curve Cryptography primitives was also measured. The achieved results prove that Elliptic Curve Cryptography is not only feasible on sensor devices, but is also attractive for constrained sensor nodes such as the MICA2 and the Tmote Sky. NanoECC also showed for the first time that Elliptic Curve Cryptography over binary fields can be implemented as efficiently as over prime fields.

Another contribution of this thesis is the first in-depth study on the application of pairing-based cryptography to wireless sensor networks. The research on bilinear pairing implementations on sensor nodes resulted in the development of the Micro-pairings package. Micro-pairings provides the fastest currently available pairing implementations on popular sensor nodes. The results achieved with Micro-pairings show that the η_T pairing over binary fields gives the best performance on the tested hardware platforms. On the most constrained 8-bit sensor nodes, the pairing can be calculated in only 2.66s – a time comparable with the point multiplication operation. The significance of the above result is that pairings were long considered as too heavyweight for small embedded devices. Efficient implementation of pairings opens the door for a whole new range of security protocols in sensor networks. With the use of Micro-pairings, these new security schemes can now be easily implemented and evaluated in a sensor network environment.

One of the main research objectives of this thesis was to design identity-based security protocols for sensor networks. This goal was achieved by proposing three different key distribution schemes that were designed for specific wireless sensor network models. The first protocol was developed with flat sensor networks in mind to solve the key distribution problem among homogeneous sensor nodes. The main advantages of this scheme is simplicity and the non-interactive feature of the key agreement process. This allows the network to save on the energy required for radio communication and makes the whole system highly scalable. The evaluation results showed that the proposed key agreement protocol is feasible and practical on different sensor platforms. Proper management of node identities in the system also guarantees that the scheme is resistant to many possible attacks.

The second scheme proposed here investigates the application of a non-interactive

key exchange protocol in a clustered network scenario. This novel variant of a key agreement scheme uses a Type 3 pairing and assigns a source and a sink key to each node in the network. The captured node can only operate as a source of data and cannot decrypt the packets that are addressed to other nodes. This make such a device far less useful for a possible attacker and limits the effect of a node compromise. This feature might be important especially in high security wireless sensor networks which deal with critical data.

The last security protocol proposed in this thesis implements identity-based encryption to distribute keys in a heterogeneous sensor network. The scheme is called TinyIBE and takes advantage of the enhanced processing capabilities of the cluster head nodes. The proposed Identity-Based Encryption method provides authenticated key exchange between the nodes and fits perfectly in the heterogenous sensor network setting. TinyIBE is an efficient security bootstrapping mechanism that achieves significant savings in communication overhead and key storage space, when compared with the state-of-the-art. Implementation results showed that TinyIBE is a preferable security scheme for heterogeneous sensor networks which provides affordable Public Key Cryptography and achieves stronger resilience against node capture attack than any key pre-distribution technique.

This thesis proves for the first time that Identity-Based Cryptography protocols can be applied to wireless sensor networks not only in theory, but also in practice. All the implementations can be made entirely in software without the use of any specialized hardware accelerators for cryptographic operations. The key distribution protocols presented in this thesis can be implemented efficiently even on the most constrained 8-bit platforms. This means that the technology is viable and can be used to secure real-life WSN applications.

7.2 Future work

Providing security is undoubtedly one of the biggest challenges in the field of wireless sensor networks. This thesis presents novel methods for symmetric key distribution within sensor networks. Nevertheless, some interesting research problems in the area of sensor network security still require further investigation. Many security issues have not yet been fully addressed. Some of these include secure reprogramming of motes “on-the-

fly" and secure node localization. The key distribution schemes proposed in this thesis can be enhanced to offer the above functionality. It would be interesting to evaluate the performance of a combined solution in a real sensor network scenario.

The results presented in this thesis were obtained using a range of sensor devices in order to test the feasibility of the proposed solutions on different hardware platforms. However, it would be interesting to evaluate the proposed methods on large scale sensor networks and already deployed networks. One possible solution would be to deploy the code on different sensor net testbeds. This approach would check the performance and robustness of the solutions developed, but does not provide all the results (e.g. data regarding energy efficiency).

This thesis presents implementation results for different cryptographic primitives used in Elliptic Curve Cryptography and Pairing-Based Cryptography. These areas of cryptography are still evolving and new advances in the state-of-the art are being made. Recently discovered Edward curves [11] and new pairing algorithms (e.g. R-Ate pairing [75]) provide more efficient ways to calculate the basic cryptographic primitives. It would be beneficial to include these new methods in future versions of NanoECC and Micro-pairings. Further code optimization in terms of execution time and memory usage would also be advantageous.

Another opportunity for future research would be to investigate different ways to provide end-to-end authentication of sensor data over the Internet. In many cases, the confidentiality of simple sensor readings is not as important as the authenticity of the data. Many applications need an assurance that the origin of the sensor readings is genuine and that the data is not fabricated. A solution to this problem would require new energy efficient methods for data authentication, especially on the sensor network side.

7.3 Concluding remarks

Recent advances in wireless sensor network technology have opened the door for an array of new and innovative applications. As the applications mature the need for reliable security systems grows significantly and justifies the use of Public Key Cryptography techniques to secure communication in sensor networks. Practical Public Key Cryptography schemes in sensor networks can be applied by using identity based cryptography techniques. This thesis proposes different ID-based key distribution protocols and shows

for the first time that they can be efficiently implemented on popular sensor network devices. Deploying software-based solutions such as those described in this thesis will demonstrate the feasibility of complete public key schemes in sensor networks, and may motivate the incorporation of hardware support for Public Key Cryptography in future mote designs.

BIBLIOGRAPHY

[1] I. F. AKYILDIZ, W. SU, Y. SANKARASUBRAMANIAM, AND E. CAYIRCI, *Wireless sensor networks: a survey*, Computer Networks (Amsterdam, Netherlands: 1999), 38 (2002), pp. 393–422.

[2] ANSI-X9.62, *Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA)*, 1999.

[3] O. ARAZI AND H. QI, *Load-balanced key establishment methodologies in Wireless Sensor Networks*, International Journal of Security and Networks (IJSN). Special Issue on Security Issues on Sensor Networks, 1 (2006), pp. 158–166.

[4] ARM-LTD., *Arm realview tools*, 2007. `http://www.arm.com/products/`.

[5] ATMEL, *ATmega128(L) datasheet*, October 2006. `http://www.atmel.com`.

[6] D. V. BAILEY AND C. PAAR, *Optimal extension fields for fast arithmetic in public-key algorithms*, in CRYPTO '98: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology, London, UK, 1998, Springer-Verlag, pp. 472–485.

[7] P. S. L. M. BARRETO, S. GALBRAITH, C. O'HEIGEARTAIGH, AND M. SCOTT, *Efficient pairing computation on supersingular abelian varieties*, Designs, Codes and Cryptography, 42 (2007), pp. 239–271.

[8] P. S. L. M. BARRETO, H. Y. KIM, B. LYNN, AND M. SCOTT, *Efficient algorithms for pairing-based cryptosystems*, in CRYPTO '02: Proceedings of the 22nd Annual Inter-

national Cryptology Conference on Advances in Cryptology, London, UK, 2002, Springer-Verlag, pp. 354–368.

[9] P. S. L. M. BARRETO, M. NAEHRIG, E. POLITÉCNICA, AND L. F. T. INFORMATIONSTECHNIK, *Pairing-friendly elliptic curves of prime order*, in Proceedings of SAC 2005, volume 3897 of LNCS, Springer-Verlag, 2005, pp. 319–331.

[10] N. BENGER AND M. SCOTT, *Constructing tower extensions for the implementation of pairing-based cryptography*. Cryptology ePrint Archive, Report 2009/556, 2009. http://eprint.iacr.org/.

[11] D. J. BERNSTEIN AND T. LANGE, *Faster addition and doubling on elliptic curves*, in ASIACRYPT'07: Proceedings of the Advances in Crypotology 13th international conference on Theory and application of cryptology and information security, Springer-Verlag, 2007, pp. 29–50.

[12] S. BHATTI, J. CARLSON, H. DAI, J. DENG, J. ROSE, A. SHETH, B. SHUCKER, C. GRUENWALD, A. TORGERSON, AND R. HAN, *MANTIS OS: an embedded multithreaded operating system for wireless micro sensor platforms*, Mobile Network Applications, 10 (2005), pp. 563–579.

[13] I. BLAKE, G. SEROUSSI, N. SMART, AND J. W. S. CASSELS, *Advances in Elliptic Curve Cryptography*, Cambridge University Press, New York, NY, USA, 2005.

[14] I. F. BLAKE, G. SEROUSSI, AND N. P. SMART, *Elliptic curves in cryptography*, Cambridge University Press, New York, NY, USA, 1999.

[15] E.-O. BLASS AND M. ZITTERBART, *Towards Acceptable Public-Key Encryption in Sensor Networks*, in The 2nd Int'l Workshop on Ubiquitous Computing, ACM SIGMIS, May 2005.

[16] R. BLOM, *An optimal class of symmetric key generation systems*, in Proc. of the EUROCRYPT 84 workshop on Advances in cryptology: theory and application of cryptographic techniques, New York, NY, USA, 1985, Springer-Verlag, pp. 335–338.

[17] C. BLUNDO, A. D. SANTIS, A. HERZBERG, S. KUTTEN, U. VACCARO, AND M. YUNG, *Perfectly-secure key distribution for dynamic conferences*, in CRYPTO '92:

Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology, London, UK, 1993, Springer-Verlag, pp. 471–486.

[18] D. BONEH AND M. FRANKLIN, *Identity-based encryption from the weil pairing*, SIAM Journal of Computing, 32 (2003), pp. 586–615.

[19] D. BONEH, B. LYNN, AND H. SHACHAM, *Short signatures from the weil pairing*, in ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security, London, UK, 2001, Springer-Verlag, pp. 514–532.

[20] J. W. BOS, M. E. KAIHARA, T. KLEINJUNG, A. K. LENSTRA, AND P. L. MONTGOMERY, *On the security of 1024-bit rsa and 160-bit elliptic curve cryptography*. Cryptology ePrint Archive, Report 2009/389, 2009. `http://eprint.iacr.org/`.

[21] E. F. BRICKELL, D. M. GORDON, K. S. MCCURLEY, AND D. B. WILSON, *Fast exponentiation with precomputation*, in EUROCRYPT'92: Proceedings of the 11th annual international conference on Theory and application of cryptographic techniques, Berlin, Heidelberg, 1993, Springer-Verlag, pp. 200–207.

[22] M. BROWN, D. HANKERSON, J. LÓPEZ, AND A. MENEZES, *Software implementation of the nist elliptic curves over prime fields*, in CT-RSA 2001: Proceedings of the 2001 Conference on Topics in Cryptology, London, UK, 2001, Springer-Verlag, pp. 250–265.

[23] D. W. CARMAN, P. S. KRUUS, AND B. J. MATT, *Constraints and approaches for distributed sensor network security.*, Technical report, NAI Labs, (September 2000).

[24] CERTICOM, *Standards for efficient cryptography – SEC 2: Recommended elliptic curve domain parameters*, in Certicom Research publications, Certicom, 2000.

[25] H. CHAN, A. PERRIG, AND D. SONG, *Random key predistribution schemes for sensor networks*, in SP '03: Proceedings of the 2003 IEEE Symposium on Security and Privacy, Washington, DC, USA, 2003, IEEE Computer Society, p. 197.

[26] S. CHATTERJEE, P. SARKAR, AND R. BARUA, *Efficient computation of tate pairing in projective coordinate over general characteristic fields*, in Information Security and Cryptology — ICISC 2004, vol. 3506 of LNCS, 2005, pp. 168—-181.

[27] H. COHEN AND G. FREY, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, Chapman & Hall/CRC, 2006.

[28] P. COMBA, *Exponentiation cryptosystems on the IBM PC*, IBM Systems Journal, 29 (1990), pp. 526–538.

[29] CROSSBOW-TECHNOLOGY, *Imote2 datasheet*. `http://tinyurl.com/3a4jwea`.

[30] ——, *MPR/MIB Mote Hardware Users Manual – Document 7430-0021-05*, December 2003.

[31] R. DI PIETRO, L. V. MANCINI, A. MEI, A. PANCONESI, AND J. RADHAKRISHNAN, *Connectivity properties of secure wireless sensor networks*, in SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks, New York, NY, USA, 2004, ACM, pp. 53–58.

[32] W. DIFFIE AND M. E. HELLMAN, *New directions in cryptography*, IEEE Transactions on Information Theory, IT-22 (1976), pp. 644–654.

[33] B. DOYLE, S. BELL, A. F. SMEATON, K. MCCUSKER, AND N. O'CONNOR., *Security considerations and key negotiation techniques for power constrained sensor networks*, The Computer Journal (Oxford University Press), 49 (2006), pp. 443–453.

[34] W. DU, J. DENG, Y. S. HAN, AND P. K. VARSHNEY, *A pairwise key pre-distribution scheme for wireless sensor networks*, in CCS '03: Proceedings of the 10th ACM conference on Computer and communications security, New York, NY, USA, 2003, ACM, pp. 42–51.

[35] X. DU, Y. XIAO, S. CI, M. GUIZANI, AND H.-H. CHEN, *A routing-driven key management scheme for heterogeneous sensor networks*, in ICC '07 IEEE International Conference on Communications, June 2007, pp. 3407–3412.

[36] X. DU, Y. XIAO, M. GUIZANI, AND H.-H. CHEN, *An effective key management scheme for heterogeneous sensor networks*, Ad Hoc Networks, 5 (2007), pp. 24 – 34.

[37] E. J. DUARTE-MELO AND M. LIU, *The effect of organization on energy consumption in wireless sensor networks*, in the proceedings of IEEE Globecom '02, pp. 113–133.

[38] A. DUNKELS, B. GRONVALL, AND T. VOIGT, *Contiki - a lightweight and flexible operating system for tiny networked sensors*, in LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, Washington, DC, USA, 2004, IEEE Computer Society, pp. 455–462.

[39] R. DUPONT AND A. ENGE, *Provably secure non-interactive key distribution based on pairings*, Discrete Applied Mathematics, 154 (2006), pp. 270–276.

[40] DUST-NETWORKS, *Technical overview of time synchronized mesh protocol (TSMP)*, 2006. http://www.dustnetworks.com.

[41] H. EBERLE, A. WANDER, N. GURA, S. CHANG-SHANTZ, AND V. GUPTA, *Architectural extensions for elliptic curve cryptography over GF(2^m) on 8-bit microprocessors*, IEEE International Conference on Application-Specific Systems, Architectures and Processors, (2005), pp. 343–349.

[42] T. EISENBARTH, S. KUMAR, C. PAAR, A. POSCHMANN, AND L. UHSADEL, *A survey of lightweight-cryptography implementations*, IEEE Design and Test of Computers, 24 (2007), pp. 522–533.

[43] L. ESCHENAUER AND V. D. GLIGOR, *A key-management scheme for distributed sensor networks*, in CCS '02: Proceedings of the 9th ACM conference on Computer and communications security, New York, NY, USA, 2002, ACM, pp. 41–47.

[44] FIPS186-2, *Digital signature standard (DSS)*, in Federal Information Processing Standards Publication 186-2, National Institute of Standards and Technology, 2000.

[45] K. FONG, D. HANKERSON, J. LOPEZ, AND A. MENEZES, *Field inversion and point halving revisited*, IEEE Transactions on Computers, 53 (2004), pp. 1047–1059.

[46] D. FREEMAN, M. SCOTT, AND E. TESKE, *A taxonomy of pairing-friendly elliptic curves*. Cryptology ePrint Archive, Report 2006/372, 2006. http://eprint.iacr.org/2006/372.

[47] D. FREEMAN, M. SCOTT, AND E. TESKE, *A taxonomy of pairing-friendly elliptic curves*, Journa of Cryptology, 23 (2010), pp. 224–280.

[48] S. GALBRAITH, *Pairings*, in Advances in Elliptic Curve Cryptography, I. Blake, G. Seroussi, and N. Smart, eds., London Mathematical Society Lecture Notes, Cambridge University Press, 2005, ch. IX, pp. 183–213.

[49] S. GALBRAITH, K. PATERSON, AND N. SMART, *Pairings for cryptographers*. Cryptology ePrint Archive, Report 2006/165, 2006. http://eprint.iacr.org/2006/165.

[50] T. E. GAMAL, *A public key cryptosystem and a signature scheme based on discrete logarithms*, in Proceedings of CRYPTO 84 on Advances in cryptology, New York, NY, USA, 1985, Springer-Verlag New York, Inc., pp. 10–18.

[51] D. GAY, P. LEVIS, J. R. VON BEHREN, M. WELSH, E. A. BREWER, AND D. E. CULLER, *The nesC language: A holistic approach to networked embedded systems.*, in ACM Conf. on Programming Language Design and Implementation, 2003, pp. 1–11.

[52] L. GIROD, T. STATHOPOULOS, N. RAMANATHAN, J. ELSON, D. ESTRIN, E. OSTERWEIL, AND T. SCHOELLHAMMER, *A system for simulation, emulation, and deployment of heterogeneous sensor networks*, in SenSys, 2004, pp. 201–213.

[53] V. GOPAL, E. OZTURK, J. GUILFORD, G. WOLRICH, W. FEGHALI, M. DIXON, AND D. KARAKOYUNLU, *Fast crc computation for generic polynomials using PCLMULQDQ instruction*. Intel whitepapers, December 2009.

[54] P. GRABHER AND D. PAGE, *Hardware acceleration of the tate pairing in characteristic three*, in Cryptographic Hardware and Embedded Systems (CHES - 2005), Springer-Verlag LNCS 3659, August 2005, pp. 398–411.

[55] J. GROSSCHÄDL, *Instruction set extension for long integer modulo arithmetic on risc-based smart cards*, in SBAC-PAD '02: Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing, Washington, DC, USA, 2002, IEEE Computer Society, p. 13.

[56] J. GUAJARDO, R. BLUEMEL, U. KRIEGER, AND C. PAAR, *Efficient implementation of Elliptic Curve Cryptosystems on the TI MSP430x33x family of microcontrollers*, in International Workshop on Practice and Theory in Public Key Cryptography (PKC 2001), 2001.

[57] N. GURA, A. PATEL, A. WANDER, H. EBERLE, AND S. C. SHANTZ, *Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs.*, in Workshop on Cryptographic Hardware and Embedded Systems (CHES'04), 2004, pp. 119–132.

[58] D. HANKERSON, A. MENEZES, AND S. VANSTONE, *Guide to Elliptic Curve Cryptography*, Springer, 2004.

[59] C. HARTUNG, J. BALASALLE, AND R. HAN, *Node compromise in sensor networks: The need for secure systems*, Technical Report CUCS-990-05, (2005).

[60] W. R. HEINZELMAN, A. CHANDRAKASAN, AND H. BALAKRISHNAN, *Energy-efficient communication protocol for wireless microsensor networks*, in HICSS: Hawaii International Conference on System Sciences, 2000.

[61] F. HESS, N. SMART, AND F. VERCAUTEREN, *The Eta pairing revisited*, IEEE Transactions on Information Theory, 52 (2006). http://eprint.iacr.org/2006/110.

[62] J. L. HILL, *System architecture for wireless sensor networks*, PhD thesis, 2003. Adviser-David E. Culler.

[63] S. HUSSAIN, F. KAUSAR, AND A. MASOOD, *An efficient key distribution scheme for heterogeneous sensor networks*, in IWCMC '07: Proceedings of the 2007 international conference on Wireless communications and mobile computing, New York, NY, USA, 2007, ACM, pp. 388–392.

[64] IEEE COMPUTER SOCIETY, *IEEE Standard Specifications for Public-Key Cryptography – IEEE Std 1363:2000*, New York, USA, 2000.

[65] IEEE-P1636.3, *IEEE P1636.3/D1 Draft Standard for Identity-based Public-key Cryptography Using Pairings*, IEEE P1636 working group, 2008.

[66] INTEL-CORPORATION, *Intel Xscale Microarchitecture Datasheet*, 2000. http://www.intel.com.

[67] A. JOUX, *A one round protocol for tripartite diffie-hellman.*, Journal of Cryptology, 17 (2004), pp. 263–276. Proceedings of ANTS-IV, 2000.

[68] A. JOUX AND R. LERCIER, *Discrete logarithms in GF(2^{607}) and GF(2^{613})*, 2005. http://perso.univ-rennes1.fr/reynald.lercier/file/.

[69] C. KARLOF, N. SASTRY, AND D. WAGNER, *Tinysec: A link layer security architecture for Wireless Sensor Networks*, in 2nd ACM SensSys, Nov 2004, pp. 162–175.

[70] M. KELLER, T. KERINS, F. M. CROWE, AND W. P. MARNANE, *FPGA implementation of a GF(2^m) tate pairing architecture*, in ARC, 2006, pp. 358–369.

[71] T. KERINS, W. MARNANE, E. POPOVICI, AND P. BARRETO, *Hardware accelerators for pairing based cryptosystems*, IEE Proceedings - Information Security, 152 (2005), pp. 47–56.

[72] S. KIM, S. PAKZAD, D. CULLER, J. DEMMEL, G. FENVES, S. GLASER, AND M. TURON, *Health monitoring of civil infrastructures using wireless sensor networks*, in IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks, New York, NY, USA, 2007, ACM, pp. 254–263.

[73] T. KLEINJUNG, *Discrete logarithms in GF(p) – 160 digits*, 2007. `http://www.nabble.com/`.

[74] D. E. KNUTH, *The art of computer programming, volume 2*, Addison-Wesley Longman Publishing Co., Inc., 1997.

[75] E. LEE, H.-S. LEE, AND C.-M. PARK, *Efficient and generalized pairing computation on abelian varieties*, IEEE Transactions on Information Theory, 55 (2009), pp. 1793–1803.

[76] A. K. LENSTRA, *Unbelievable security. Matching AES security using public key systems.*, in Advances in Cryptology – Asiacrypt 2001 LNCS, vol. 2248, Springer-Verlag, 2001, pp. 67–86.

[77] P. LEVIS, S. MADDEN, J. POLASTRE, R. SZEWCZYK, K. WHITEHOUSE, A. WOO, D. GAY, J. HILL, M. WELSH, E. BREWER, AND D. CULLER, *TinyOS: An operating system for Wireless Sensor Networks*, in Ambient Intelligence, W. Weber, J. Rabaey, and E. Aarts, eds., Springer-Verlag, New York, NY, 2004.

[78] C. H. LIM AND P. J. LEE, *More flexible exponentiation with precomputation*, in CRYPTO '94: Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology, London, UK, 1994, Springer-Verlag, pp. 95–107.

[79] S. LINDSEY AND C. S. RAGHAVENDRA, *Pegasis: Power-efficient gathering in sensor information systems*, 2002.

[80] A. LIU AND P. NING, *TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks*, in IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks, Washington, DC, USA, 2008, IEEE Computer Society, pp. 245–256.

[81] J. LÓPEZ AND R. DAHAB, *Improved algorithms for elliptic curve arithmetic in GF(2^n)*, in SAC '98: Proceedings of the Selected Areas in Cryptography, London, UK, 1999, Springer-Verlag, pp. 201–212.

[82] ——, *High-speed software multiplication in GF(2^m)*, in INDOCRYPT '00: Proceedings of the First International Conference on Progress in Cryptology, London, UK, 2000, Springer-Verlag, pp. 203–212.

[83] B. LYNN, *On the implementation of pairing-based cryptosystems*, PhD thesis, 2007. Adviser-Dan Boneh.

[84] A. MAINWARING, J. POLASTRE, R. SZEWCZYK, D. CULLER, AND J. ANDERSON, *Wireless sensor networks for habitat monitoring*, 2002.

[85] D. J. MALAN, M. WELSH, AND M. D. SMITH, *A Public-Key Infrastructure for key distribution in TinyOS based on Elliptic Curve Cryptography*, in 1st IEEE Intl' Conf. on Sensor and Ad Hoc Communications and Networks (SECON'04), 2004.

[86] K. MCCUSKER, *Cryptographic Key Distribution In Wireless Sensor Networks: A Hardware Perspective*, PhD thesis, 2008.

[87] A. MENEZES, *An introduction to pairing-based cryptography*. Available on-line, 2005. `http://www.math.uwaterloo.ca/ajmeneze/publications/`.

[88] A. MENEZES, T. OKAMOTO, AND S. VANSTONE, *Reducing elliptic curve logarithms to logarithms in a finite field*, IEEE Transactions on Information Theory, 39 (1993), pp. 1639–1646.

[89] V. S. MILLER, *Short programs for functions on curves*, in IBM Thomas J. Watson Research Center, 1986.

[90] A. MIYAJI, M. NAKABAYASHI, AND S. TAKANO, *New explicit conditions of elliptic curve traces for FR-reduction*, IEICE Transactions on Fundamentals, E84-A (2001), pp. 1234–1243.

[91] P. MONTGOMERY, *Modular multiplication without division*, Mathematics of Computation, 44 (1985), pp. 519–521.

[92] MOTEIV, *Tmote Sky datasheet*, 2006. http://www.moteiv.com.

[93] J. NEWSOME, E. SHI, D. SONG, AND A. PERRIG, *The sybil attack in sensor networks: analysis & defenses*, in IPSN '04: Proceedings of the 3rd international symposium on Information processing in sensor networks, New York, NY, USA, 2004, ACM, pp. 259–268.

[94] L. B. OLIVEIRA, R. DAHAB, J. LOPEZ, F. DAGUANO, AND A. A. F. LOUREIRO, *Identity-based encryption for sensor networks*, in 5th IEEE International Conference on Pervasive Computing and Communications Workshops PERCOMW '07, 2007, pp. 290–294.

[95] L. B. OLIVEIRA, H. C. WONG, A. A. F. LOUREIRO, AND R. DAHAB, *On the design of secure protocols for hierarchical sensor networks*, International Journal of Sensor Networks, 2 (2007), pp. 216–227.

[96] L. OLIVIERA, D. ARANHA, E. MORAIS, F. DAGUANO, J. LOPEZ, AND R. DAHAB, *Tinytate: Computing the tate pairing in resource-constrained sensor nodes*, in 6th IEEE Inernational Symposium on Network Computing and Applications – NCA 2007, 2007.

[97] A. PERRIG, J. STANKOVIC, AND D. WAGNER, *Security in wireless sensor networks*, Communications of the ACM, 47 (2004).

[98] A. PERRIG, R. SZEWCZYK, V. WEN, D. CULLER, AND J. D. TYGAR, *SPINS: Security protocols for sensor networks*, Wireless Networks, 8 (2002), pp. 521–534. Also appeared in MobiCom'01.

[99] S. POHLIG AND M. HELLMAN, *An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance*, IEEE Transactions on Information Theory, 24 (1978), pp. 106–110.

[100] J. M. POLLARD, *Monte carlo methods for index computation (mod p)*, Mathematics of Computation, 32 (1978), pp. 918–924.

[101] R. L. RIVEST, A. SHAMIR, AND L. ADLEMAN, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, 21 (1978), pp. 120–126.

[102] K. RÖMER AND F. MATTERN, *The design space of wireless sensor networks*, IEEE Wireless Communications, 11 (2004), pp. 54–61.

[103] R. RONAN, C. O. HÍIGEARTAIGH, C. MURPHY, M. SCOTT, AND T. KERINS, *Hardware acceleration of the tate pairing on a genus 2 hyperelliptic curve*, Joranl of Systems Architecture, 53 (2007), pp. 85–98.

[104] S. ROUNDY, P. K. WRIGHT, AND J. M. RABAEY, *Energy Scavenging for Wireless Sensor Networks: With Special Focus on Vibrations*, Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[105] R. SAKAI AND M. KASAHARA, *ID based cryptosystems with pairing on elliptic curve*. Cryptology ePrint Archive, Report 2003/054, 2003. `http://eprint.iacr.org/`.

[106] R. SAKAI, K. OHGISHI, AND M. KASAHARA, *Cryptosystems based on pairing*. The 2000 Symposium on Cryptography and Information Security, Okinawa, Japan, 2000.

[107] M. SCOTT, *Implementing cryptographic pairings*, in Pairing 2007, vol. 4575 of Lecture Notes in Computer Science, Springer-Verlag, 2007, pp. 177–196.

[108] ——, *Optimal irreducible polynomials for $GF(2^m)$ arithmetic*. Cryptology ePrint Archive, Report 2007/192, 2007.

[109] M. SCOTT, *MIRACL – Multiprecision Integer and Rational Arithmetic C/C++ Library*, 2009. `http://www.shamus.ie`.

[110] S. C. SEO, D.-G. HAN, H. C. KIM, AND S. HONG, *TinyECCK: Efficient elliptic curve cryptography implementation over GF(2^m) on 8-bit micaz mote*, IEICE - Transactions on Inforamtion Systems, E91-D (2008), pp. 1338–1347.

[111] A. SHAMIR, *Identity-based cryptosystems and signature schemes*, in On Advances in cryptology, proceedings of CRYPTO 84, New York, NY, USA, 1985, Springer-Verlag New York, Inc., pp. 47–53.

[112] E. SHI, A. PERRIG, AND C. M. UNIVERSITY, *Designing secure sensor networks*, IEEE Wireless Communications, 11 (2004).

[113] C. SHU, S. KWON, AND K. GAJ, *FPGA accelerated tate pairing based cryptosystems over binary fields*, in Cryptology Eprint Archive, Report 2006/179, 2006.

[114] J. A. SOLINAS, *Efficient arithmetic on koblitz curves*, Design, Codes and Cryptography, 19 (2000), pp. 195–249.

[115] TEXAS-INSTRUMENTS, *MSP 430F1611 Datasheet*, 2002. http://www.ti.com.

[116] L. UHSADEL, A. POSCHMANN, AND C. PAAR, *Enabling Full-Size Public-Key Algorithms on 8-bit Sensor Nodes*, in Proceedings of ESAS 2007, vol. 4572 of LNCS, Springer, 2007.

[117] J. P. WALTERS, Z. LIANG, W. SHI, AND V. CHAUDHARY, *Wireless sensor network security: A survey, in book chapter of security*, in Distributed, Grid, and Pervasive Computing, Yang Xiao (Eds), CRC Press, 2007, pp. 0–849.

[118] H. WANG, B. SHENG, AND Q. LI, *Elliptic Curve Cryptography based access control in sensor networks*, International Journal of Security and Networks (IJSN). Special Issue on Security Issues on Sensor Networks, 1 (2006), pp. 127–137.

[119] B. WARNEKE, M. LAST, B. LIEBOWITZ, AND K. S. J. PISTER, *Smart dust: Communicating with a cubic-millimeter computer*, Computer, 34 (2001), pp. 44–51.

[120] R. J. WATRO, D. KONG, S. FEN CUTI, C. GARDINER, C. LYNN, AND P. KRUUS, *Tinypk: securing sensor networks with public key technology.*, in 2nd ACM Workshop on Security of ad hoc and Sensor Networks (SASN'04), Washington, DC, October 2004, pp. 59–64.

[121] WEB-RESOURCE. http://webs.cs.berkeley.edu/800demo/.

[122] ——. http://www.certicom.com/index.php/.

[123] ——. http://www.ecc-challenge.info/.

[124] A. D. WOOD AND J. A. STANKOVIC, *Denial of service in sensor networks*, Computer, 35 (2002), pp. 54–62.

[125] H. YAN AND Z. J. SHI, *Studying software implementations of elliptic curve cryptography*, in ITNG '06: Proceedings of the Third International Conference on Information Technology: New Generations, Washington, DC, USA, 2006, IEEE Computer Society, pp. 78–83.

[126] G. YANG, C. RONG, C. VEIGNER, AND H. CHENG, *Id-based key agreement and encryption for wireless sensor networks*, IJCSNS: International Journal of Computer Science and Network Security, 6 (2006).

[127] S. ZHU, S. SETIA, AND S. JAJODIA, *LEAP: efficient security mechanisms for large-scale distributed sensor networks*, in 10th ACM conference on Computer and communication security (CCS'03), ACM Press, 2003, pp. 62–72.

[128] ZIGBEE-ALLIANCE, *Zigbee specification 1.1*, 2006. `http://www.zigbee.org`.

APPENDIX A

Energy consumption measurements

Energy consumption is one of the most important parameters in wireless sensor networks. Almost all of the nodes available on the market today use batteries as the main energy source. Current batteries offer very limited capacity especially for sensor nodes, which are expected to operate for a few years. That is why sensor networks require low duty cycling and energy efficient protocols to prolong the network lifetime.

The energy efficiency requirement applies also to security protocols in wireless sensor networks. The energy consumption is especially important in the case of asymmetric cryptography, which is far more complex in terms of computation than symmetric key solutions. The security protocols proposed in this thesis are based on identity based cryptography, which requires complex arithmetic operations. The key to energy efficient Identity-Based Cryptography operations lies in fast execution of basic cryptographic primitives. During this research much time and effort was spent on accelerating finite field arithmetic, point multiplication operations and bilinear pairings. In each case precise energy measurements were taken to evaluate the energy consumption of sensor nodes. This appendix describes how these energy measurements were performed.

The energy consumption of a sensor node can be measured using different methods. It can be profiled internally in software or externally by measuring devices. For experiments described in this thesis the second method was chosen. To measure the average power consumption of a sensor node a special measuring circuit was built (Figure A.1).

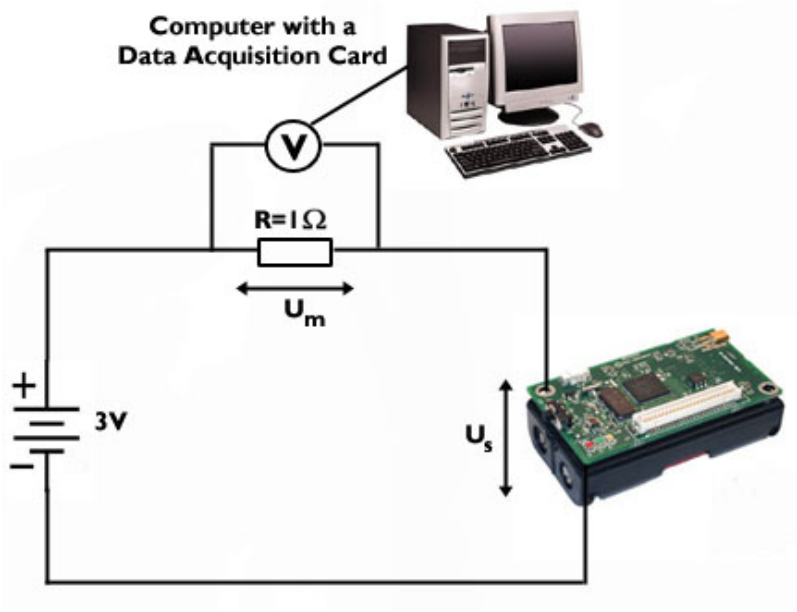

Figure A.1: *Experimental setup for measuring energy consumption on senor nodes*

The experiments were performed on the MICA2 and the Tmote Sky platforms. The nodes were powered from a regulated power supply, which was set to 3V (the voltage of two standard AA batteries). Both devices had to be slightly modified for the experiments. A precise 1Ω resistor was soldered between the main PCB of the node and its battery pack. The current drawn from the batteries (3V power source in this case) was measured based on the voltage level (U_m) on the resistor R. The voltage samples of U_m were measured using a PC with a data acquisition card. The same measurements on both devices were taken using a National Instruments NI 5112 digitizer card. All experiments were carried out using a LabVIEW application, which was developed for this purpose. The program was storing samples and calculating the average voltage level from all the samples within the Elliptic Curve Cryptography program execution period. A typical graph of U_m values on the Tmote Sky node is presented in Figure A.2. The apparent negative samples of voltage are the result of a displaying error in the graphical software, when plotting large number of collected samples (10MHz sampling rate). The interval with a higher average level of voltage, indicates the time when cryptographic operations were calculated.

The total energy consumption of a node can be calculated based on the following formula:

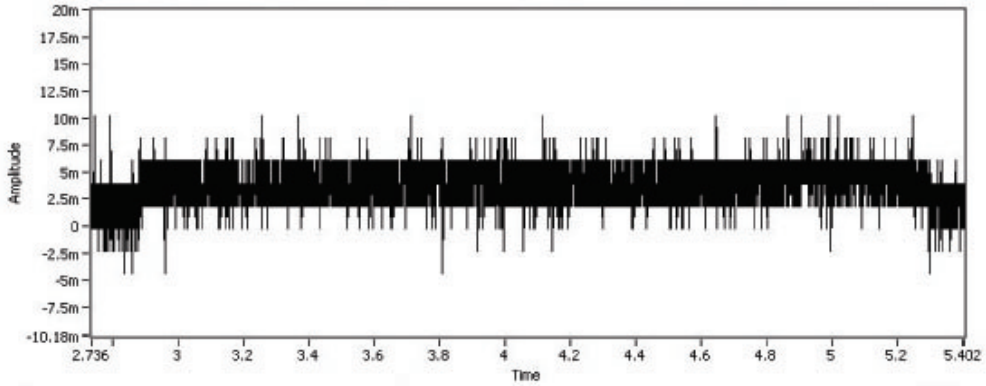

Figure A.2: *Voltage samples measured on the Tmote Sky node*

$$E = U_s \cdot I_m \cdot t \tag{A.1}$$

where U_s is the voltage on the sensor node, I_m is the current running through resistance R and t is the code running time. The average value of current I_m over time t can be obtained as:

$$I_m = \frac{U_m}{R} \tag{A.2}$$

With the approximation that $U_s = 3V$ the total energy consumption of a node can be represented as:

$$E \approx \frac{3 \cdot U_m}{R} \cdot t \tag{A.3}$$

where $R = 1\Omega$. In reality the value of U_s is lower than 3V due to the additional resistance R and is equal to $U_s = 3 - U_m$. The current I_m running through the node can be at a maximum 20mA (according to the motes data sheets), hence the measured value of U_m is never bigger than 20mV (using $R = 1\Omega$). This gives a maximal relative error of 0.67% for U_s. For example if the energy consumption was measured as 50mJ using equation A.3 than the relative error is only 0.33mJ. This makes equation A.3 a good approximation of the node's energy consumption, which can be used in practice.