Efficient Elliptic Curve Cryptography Software Implementation on Embedded Platforms

By

Mohamed Said Sulaiman Albahri

Thesis submitted for the Degree of Doctor of Philosophy

Department of Electronic & Electrical Engineering

The University of Sheffield

September-2019

Abstract

The demand for resources-constrained devices of 8-bit and 32-bit microcontrollers has increased due to the requirements of different applications such as Radio Frequency Identification (RFID), Internet of Things (IoT) and Wireless Sensor Network (WSN). Applying efficient security in these applications and their microcontroller platform is one of the significant concerns for its limited acceptance. In fact, public key cryptography (PKC), RSA and Elliptic Curve Cryptography, are generally considered the most powerful cryptosystems that could provide a high level of security. However, RSA involves very intensive computational arithmetic with a key size of 1024-2048 bits. Therefore, ECC could be a feasible solution to provide a similar level of security with a smaller key size and lesser arithmetic computations. However, the highly effective ECC implementations in microcontroller devices remain as a concern, due to some drawbacks of the microcontrollers.

This thesis illustrates the technique for achieving highly efficient ECC on microcontroller devices that could be used in applications such as IoT, WSN and RFID. We implement an efficient ECC cryptosystem in single-core microcontroller and a homogenous multicore microcontroller. The Elliptic Curve Digital Signature is implemented on an 8-bit and 32-bit microcontrollers and its performance is evaluated for the possible combination of finite field arithmetic, point doubling, point addition and scalar point algorithms. The developed technique reduced the time required for generating EDSA key from 83ms in 32bit microcontroller to 263ms in 8bit microcontroller. The parallelization of the Comba multiplication in $GF(2^{163})$ implemented in a homogenous multicore microcontroller, obtained a performance enhancement of 85% in comparison to a single core microcontroller. The feasibility of the algorithms and the advantages of adopting parallelization is validated by using these algorithms to implement ECC scalar point multiplication over GF(2^m) using the Xmos multi-core microcontroller. Also it is believed that our proposed solutions for a multicore microcontroller that could be used in applications like IoT, WSN and RFID is the first of its kind.

Acknowledgements

First, I would like to express gratitude to my supervisor, Dr. Mohammed Benassia, for his valuable help and support and his continuous encouragement and direction. In fact, there is no doubt that without his effort and support, this thesis would not have been achievable.

Also, I am very pleased to Dr. Luke Seed and Professor. Sakir Sezer for their kind acceptance to evaluate and examine me during my viva. Their discussions were very productive and effective, which added more valuable information to my knowledge as well as my thesis.

Additionally, I would to acknowledge and expression my appreciation to my colleagues for their kind help and support during my PhD studies. Specifically, I would like to thank Zia Uddin Ahmed Khan, a PhD fellow who significantly contributed to achieving parallel implementation of ECC point multiplication presented in this thesis. Also, I offer grateful thanks to my colleague Ahmed Al-Baidhani for his remarkable help and support on finalizing some of administrative work related to my requirements due to circumstances I faced during my PhD course. Special thanks go to Ms. Hilary, and all administrative staff members in the Electrical and Electronic Engineering department, for taking care of my continuous administrative requests throughout my coursework.Last, but not least, grateful thanks go to my wife, AZZA Albahri. Really, I cannot appreciate enough her outstanding effort of sacrificing with me, despite her illness. To all of my children (Marwa, Mariya, Maryam, Ahmed, Maram and Mayar), thank you for your help and support. Actually, you have done a lot to motivate me and to encourage me to continuing my PhD, despite the difficult time we all together are passed through due to the health problems of Mom, Maram and Mariya. Also, I would like to express appreciation to my mother, brothers and sisters for their help and encouragement. Great thanks go to my brother-in-law Said Ahmed Albahri for taking care of my family while I was way to do my PhD.

To the spirit of my dear father, and all those mentioned above, I am very thankful for you.

Mohamed Said Albahri

Sheffield-05/09/2019

Table of Contents

Table of Figures

List of Tables

List of Algorithms

Glossary

AES	Advance Encryption Standard
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
FPGA	Field Programmable Gate Arrays
FLT	Fermat’s Little Theorem
GF	Galois Field
GF2	Galois Field in the field characteristic two
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
MD	Message Digest
MSB	Most Significant Bit
NAF	Non-Adjacent Form
NIST	National Institute of Standards and Technology
PKC	Public Key Cryptography
RFID	Radio Frequency Identification
RSA	Rivest-Shamir-Adleman
SHA-1	Secure Hash Algorithm-1
SECG	Standard for Efficient Cryptography Group
WSNs	Wireless Sensor Nodes
XOR	Exclusive OR (logical Operation)
τNAF	τ-adic Non-Adjacent Forms

Chapter 1 Introduction of Thesis

This chapter provides an overview of the complete thesis. It starts by expressing the demands of cryptography in securing the new technology of WSN and IoT. Then, it illustrates how the aims of the thesis been developed, followed by the main contribution. Finally, a detailed list of the published papers is given.

1.1 Overview

The emergence of new technologies related to the Wireless Sensor Network (WSN), Wireless Body Area Network (WBAN), Internet of Things (IoT) and Radio Frequency Identification (RFID) is based on embedded microcontroller platforms. It is considered that these microcontrollers are resource constrained devices, which can accommodate small sized-code memory and have low-speed processors working with limited battery resources. The main functionality of these microcontrollers in such applications is to aggregate the information produced by the sensors connected to them and transmit the data over a communication channel to its destination. For example, WBAN could be designed to help patients and doctors with real-time data about the vital life parameters related to patients, such as blood pressure, pulse heart rate, body temperature and other parameters. A new advancement in medication related to WBAN goes beyond the scope of merely transmitting the data, and may involve injecting the medicine into the patient, such as an online insulin pump system. Therefore, such highly critical applications related to human health hazards, lead to concern over the security of these applications and their networks.

However, modern cryptography plays a vital role in ensuring the security of these applications. For that, a different type of symmetric cryptography could be used to guarantee confidentiality, integrity, and authenticity for the provided services [13]. However, employing highly effective Public Key Cryptography (PKC), such as RSA or ECC on microcontrollers, can lead to many technical challenges and problems that need to be addressed in advance, like the deployment of ECC schemes delay the processes within the processor due to the complexity of the arithmetic operations associated with this scheme [13].

In this context, there have been many attempts to improve the ECC efficiency of constrained devices, which in turn resulted in the possibility of implementing the ECC algorithm in such devices. As a result, researchers have been encouraged to undertake further evolutionary researches that have also demonstrated the ability of ECC to provide the same level of security provided by RSA with lower key size in these devices. The fundamental approach of ECC allows end users to make ECC implementation more flexible and selectable. For example, ECC could be built based on either over a binary finite field or prime finite field arithmetic. However, the latest research shows possibilities of implementing the ECC as hardware, software or even combination of software/ hardware. This could lead to further investigation for the identification of the best solutions that provide higher performance with low power consumption and high-security level.

1.2 Thesis Aims and Scope

Overall, there is an ongoing debate over the enhancement of the ECC performance without the loss of its strength when executed on the embedded microcontroller of IoT, WSN, and WBAN. In fact, the feasibility of implementing highly-efficient ECC in a microcontroller and integrating them with other functions, such as actual network communication, is still difficult primarily due to two facts (1) designing the architecture of the microcontroller platform with limited power consumption, speed of the processor, and memory size; and (2) ECC cryptosystem is based on complex arithmetic operation with operands size $(\geq 163-bit)$, which may result in decreasing the overall performance of the microcontroller and subsequently having a negative impact on the overall performance of the application.

Therefore, based on the above concerns, we aim to provide solutions that can result in an optimal ECC performance along with maintaining the efficiency of the applications. Therefore, we address such concerns by attempting to enhance the performance with the help of an ECC software implementation. Our selection is based on the flexibility and simplicity of integrating ECC with application architecture. Hence, in this thesis, we used two different approaches, detailed below, to provide directions for our research.

Firstly, evaluating the possibilities of enhancing the overall ECC performance on a software implementation on single core microcontroller, using an open source reconfigurable library. Thereby, importing a Relic Toolkit [14] in such constrained devices helps in understanding and analyzing the behavior of the microcontroller. Also, this methodology addresses the issue of the best combination of a finite field, point doubling, point addition and scalar point multiplication algorithms that could lead to better ECC performance as well as understanding the influence. In particular, through this procedure, we can observe the impact of these factors on the overall performance of the ECC scheme and the Microcontroller.

Furthermore, designing ECC security level is highly dependent on the underlying layer, known as finite field arithmetic operations, and its size. Therefore, a flexible tool that could support different ECC curves over the binary field or prime field is a necessity. Such mechanisms allow the selection of the required level of security. Also, this enables a smooth communication between the microcontroller and applications when an application has different security requirements.

Secondly, research on the possibility of implementing ECC on a homogeneous multi-core microcontroller and exploring its capabilities to boost the ECC performance. Homogeneous multicore microcontrollers are designed to support simultaneous tasks. Different types of microcontroller have been designed with a multi-core processor, including XMOS, Parallax, and Ultra-Reliable Multi-core ARM-based processor.

Among these types of multicore microcontrollers, XMOS is considered the best since it works similar to an ordinary microcontroller and due to its ability to tackle issues beyond the capabilities of a traditional microcontroller. Additionally, it has a multiple core processor that allows simultaneous execution of sequential or multiple tasks. It can also provide timing analysis along with hardware simulation, using a powerful IDE known as Xtimecomposer [15]. Design and implementation on an XMOS multicore microcontroller is very flexible since the parallelization can be invoked in the main function or within all the functions of the programs. Calling for parallelizing multiple tasks from the primary function will also allow the developer to assign a particular core for his parallel tasks while calling for parallelization on other functions to automatically select the core allocation by the system. Thus, for complex implementation where only a particular function is required for conducting parallelization, the latter should be implemented within these functions instead of the main function, and the tasks, logic cores will be dynamically distributed by the system based on the availability of the resources.

In fact, having such features and functionalities makes the development of such platforms more suitable for accommodating complex algorithms. In the later stages this could particularly lead to the integration of some application (such as IoT, WSN, or WSN) with parallel reading from sensors along with the possibility of conducting the parallel tasks of implementation necessary for secure key generation, encryption, and security protocol in parallel approaches.

Further, examining the reconfiguration and scalability within the multi-core microcontroller are the concerns that need to be specifically addressed during implementation. With this knowledge, the National Institute of Standards Technology (NIST) has published different ECC curves, including: curves over binary fields $GF(2^m)$ where m=163, m=233, m=283, m=409 and m=571 as well as curves for ECC implementation over prime field $GF(p)$ where p=192, p=224, p=256, p=384 and p=521[16]. Having said that, parallelizing such complex ECC algorithms using a software implementation approach is not an easy task. However, the possible features and functionalities in the XMOS multicore microcontroller and its powerful Xtimecomposer IDE can help in tackling such challenges. As mentioned before, our selection for such a platform is not only based on implementing the ECC, but also specifically considering the flexibility and simplicity of integrating the ECC cryptosystem with the applications' sensors data collection and overall communication protocols.

Hence, the focus of this thesis is at a different level of ECC cryptosystem layers. Thus, we initially attempt to adopt the well-known algorithms that are assigned to each layer in order to obtain an effective speedup of the sequential and parallel performance. However, some algorithms are considered to improve the overall ECC performance. In a way, the proposed technique will serve the purpose of enhancing the ECC performance on microcontrollers that could be used in different applications, such as RFID, WSN and IoT. The end-to-end encryption and decryption, higher-level protocol communication, power consumption analysis and system failure analysis are not considered in this thesis.

1.3 Thesis Main Contributions

The current published works are mainly focused on enhancing the ECC performance of a single core microcontroller. Many of them attempted to create their own library for their targeted devices. However, we noticed that these works are limited with few algorithms that should be supported by the ECC. ECC is a highly algorithmic-based cryptosystem with many algorithms in place that could either increase or decrease their performance. When the algorithms for implementing the ECC is limited then the speedup efficiency of the ECC processor is not enhanced to the desired level. Hence, in this work, we initially started by importing a highly effective open-source library that allows flexible and reconfigurable features, along with increasing the ECC efficiency by selecting the best combination from a wide range of algorithms provided by the tool.

Also, there are a significant number of published works that are related to improving the ECC efficiency in large scale computer and processors. However, none of the published works make any attempts to improve the ECC in a homogeneous multicore microcontroller. Therefore, to our knowledge, we are the first to propose boosting the ECC on a multicore microcontroller, despite the overall complexity of implementing ECC in such a constrained device.

The initial research indicated the presence of some open-source tools that help in boosting the efficiency of ECC in microcontroller-based platforms. However, some of them are limited to supporting just a specific platform and some of the others could be used to support a wide range of microcontrollers. To overcome this limitation, we managed to import relic tool open-source to our microcontroller and accordingly managed to prove the enhancement in the performance of the ECC. The novelty of our contribution is summarized below:

- Experimental analysis and evaluation for Elliptic Curve Digital Signature (ECDSA) on both an 8-bit and a 32-bit platform (Arduino mega2560 and Arduino Due) has been carried out using Relic library [14], and comparative results of the implementation are provided. To our knowledge, no such analysis and results have been reported till date.
- We are the first to use the configuration features provided by an open source tool for enhancing the performance of ECC, which could be considered as a guidance and benchmark for the developers planning to use the relic tools in Arduino-mega2560 and Arduino-due. In this, we reported ECDSA key generation on Arduino Due can be achieved in (90ms) when compared to (263ms) on the Arduino Mega for m=163.

Accordingly, we consider enhancing the ECC performance based on the second approach mentioned in subsection 1.2. Where we attempted to introduce the concept of a homogeneous multicore microcontroller. Thus, we managed to achieve the following novelties:

- The first-ever novel parallel Comba multiplication over $GF(2^{163})$ was implemented using an XMOS multicore microcontroller. Our implementation showed an improvement in 85% of the measured time in comparison to a single core implementation. This result also considered the performance of Comba multiplication algorithm with and without fast modular reduction for different word sizes (8-bit, 16-bit and 32-bits). Another contribution to this novelty is the fast reduction algorithm to support the 8-bit word size, which was a result of modifying the 32-bit reduction algorithm.

- Our second novelty related to the second approach in the overall proposed improvement in ECC scalar multiplication over the binary field $GF(2^m)$ for m=163, m=233, m=283, m=409 and m=571, using an XMOS homogeneous multicore microcontroller. In this work, we managed to report a 63% improvement in ECC point multiplication, in comparison to its sequential implementation in a single core implementation. Furthermore, in this particular work, three algorithms listed below have been modified and optimized:
  - I. Modified Point Doubling in LD coordinate system by implementing a parallelization principle in it. Accordingly, we managed to reduce the number of algorithm steps from 14 to 9.
  - II. Modified Point Addition in LD coordinate system by implementing a parallelization principle in it. Hence, we were able to reduce the number of algorithm steps from 26 to 20 steps.
  - III. Modified left to right binary method point multiplication algorithm has been proposed. The enhancement is achieved by performing an initial scanning of the most significant bit (MSB) of k in order to track down the first non-zero bit from the MSB. If the non-zero first bit is found, then the coordinates are filled in Q to start the loop operation. P's coordinates will be filled in Q to start the loop operation.
- The third novelty presented relates to the second proposed approach is our contribution towards improving the ECC point multiplication over GF(P), where P=128, P=192, P=256 and P=384 using XMOS homogenous multicore microcontroller. In this work, we obtained an 80% improvement of ECC point multiplication, which is higher when compared to its sequential implementation in a single-core implementation. Also, in this study we modified the algorithms listed below.:
  - I. We were able to parallelize the Comba algorithm proposed by the scholars in [17] and [18] and replaced the original algorithm proposed by the scholars in [19] with this new modified parallelized algorithm.
  - II. We modified (X, Y)- only co-Z conjugate addition with update XYCZ - ADDC algorithm, in turn we were able to reduce the original sequential

operational steps of the algorithm from 19 to 13, only with the cost of 5M+3+16A by involving of 6 field registers in this operation.

III. We modified (X, Y)- only co-Z addition with update XYCZ -ADD algorithm, as a result reduced the original sequential operations steps of the algorithm from 13 steps to only 7 steps.

IV. We modified the Jacobian doubling ($a=-3$) algorithm for reducing the original sequential point doubling operation steps from 18 to 15.

Hence, it can be observed that our effective software design for efficient ECC tackles ECC over the binary field as well as ECC over the prime field. Furthermore, in this work, we considered three different types of data width 8, 16 and 32 bits. This is the first-ever effort that attempts to enhance the performance of ECC by considering curves over the binary as well as the prime fields.

Finally, in this research, we have been able to enhance the performance of the ECC on a microcontroller that could be used for different applications. For that, our thesis contributions point to the fact that PKC could be used in the constrained devices and secure communication could be established easily – taking into account the various types of microcontrollers mentioned in this thesis.

1.4 Thesis Outlines

Immediately after this chapter, the next chapter provides the reader with essential background and historical information about cryptography. In addition to that, a detailed explanation and differentiation between symmetric and asymmetric cryptography is also provided. Furthermore, details about the elliptic curve cryptography group's law and point multiplication algorithms, have been provided. Finally, the chapter 2 presents a discussion about the domain parameters and protocols of ECC.

In Chapter 3, our first effective ECC implementation is discussed. This chapter introduces the concept of ECC and highlights the related research., Further this chapter provides a brief introduction to the Arduino microcontroller architectures, followed by detailed implementation of the proposed solution. This chapter is concluded by presenting an analysis of the obtained results.

Chapter 4 introduces a new concept of emerging technology related to microcontrollers. This chapter presents the implementation of parallelizing the Comba algorithm in a homogeneous multicore microcontroller. First the concept of the algorithm and its relationship and importance in ECC over binary field cryptosystem is described. Then, a detailed parallelization concept for the algorithm is provided. Finally, the chapter is concluded by analyzing the obtained results.

In Chapter 5, presents the concept of parallelizing the ECC point multiplication over a binary field. The chapter begins with the introduction of the overall concept of ECC over a binary field, followed by a related mathematical background. Then the proposed solution to increase the performance of ECC point multiplication is elaborated. The details of implementing the proposed solutions is provided in subsection 5.4. Finally, the performance analysis is presented along with the conclusions drawn from the analysis.

Chapter 6 elaborates the proposed solution to enhance the scalar point multiplication of ECC, but this time for ECC over the prime field. Initially a general description for ECC over a prime field along with its mathematical background is provided. Then the proposed solution is described along with the technical details of its implementation. Finally, the results are analyzed and the conclusions drawn from it are mentioned.

Finally, the Chapter 7, provides a summary of the complete research along with the suggestions for the future work that can be conducted.

1.5 Published Papers

Albahri, M. S., and M. Benaissa. "Parallel comba multiplication in GF (2163) using homogenous multicore microcontroller." In Electronics, Circuits, and Systems (ICECS), 2015 IEEE International Conference on, pp. 641-644. IEEE, 2015.

Albahri, M.S., Benaissa, M. and Khan, Z.U.A., 2016, December. Parallel Implementation of ECC Point Multiplication on a Homogeneous Multi-Core Microcontroller. In Mobile Ad-Hoc and Sensor Networks (MSN), 2016 12th International Conference on (pp. 386-389). IEEE.

Albahri, M. S., and M. Benaissa. "Parallel elliptic Curve Cryptography over GF(P) on a Homogeneous a Multi Core Microcontroller." To be submitted to IEEE Embedded Systems Letters (ESL).

Chapter 2 Background Theory

This chapter discusses historical information about cryptography and its goals. We further provide a preliminary related background and fundamental terminologies for public key cryptography including RSA and Elliptic Curve Cryptography. Hence, details about ECC Field arithmetic over $GF(P)$ *and* $GF(2^m)$ *are given. We also discuss some algorithms related to the Point of Multiplication that is employed in this thesis, along with elliptic curve domain parameters protocols.*

2.1 Cryptography History

In the past, steganography, a Greek originated word meaning “covered writing”, was referred to as a technique for communication of secure messages. In contrast to cryptography, steganography means concealing the message itself by covering it with something else, whereas cryptography means concealing the content of the message by enciphering [20]. Also, the word ‘cryptography’ originally came from the Greek root words *kryptos* and *gráphō,* which together mean “hidden writing” [21]. Cryptography has very long and interesting history, dating back to 4000 years ago. The best description for the history of cryptography can be found in [22]. In this book, the author traced its history from the initial use of cryptography by the Egyptians in 1900 B.C to the 20th century.

As per [1] and [2], the first attempt to cipher a plain text was developed by Roman generals in the 1st century B.C. They ciphered the message by shifting a fixed number of letters down to the alphabet. This procedure of ciphering came to be known as Caesar's Cipher.

However, the principle of substitution ciphers was created by a Greek historian Polybius in the 2nd century B.C. This technique is based on replacing the letter of the alphabet and placing it within a Polybius square using numbers.

It is believed that the first transposition cipher was used by Spartan in 5th century B.C [21]. They used to exchange secret messages by wrapping slender bar parchment and wound it in something called a scytale. To decipher the message, the papyrus needed to be rewound it in the scytale of identical thickness.

In the 9th century, the first code-breaking textbook *Istikhraj al-Kotob Al-Mu'amah* was published by Islamic mathematician named Abū Yūsuf Yaʻqūb ibn Isḥāq al-Kindī. In his book, al-Kindī introduced the alphabetic cipher and frequency analysis techniques [21] .

The Middle Ages witnessed more progress in the cryptographic field. During this time, most of the Western European governments used cryptography to be in touch with their ambassadors. The most significant enhancements were developed in Italy in 1452. Venice established a new organization with three secretaries responsible to cipher and deciphers the government's messages [23].

In 1553, a new concept of a Vigenere cipher was published by Italian Renaissance Leon Basttista Alberti. This new concept was considered as strong as polyalphabetic substitution cipher at that time [21].

The invention of telegraph communication in 1844 triggered a dramatic rise in cryptography. Thus, the Vigenere cipher was used in telegraph communication until Friedrich W. Kasiski developed all periodic polyalphabetic ciphers in 1863 [23].

Furthermore, the historical information shows that cryptography played a vital role in the outcome of both world wars. For example, in 1895, the invention of radio transmission made a remarkable change of using cryptography in telegraphic communication – when the French military managed to intercept German communication during the First World War. This is because the French cryptanalysts managed to break the double columnar transposition created by German military [23]. The Enigma machine is an encryption gadget created and utilized in the mid-twentieth century to ensure business, discretionary and military correspondence. It was utilized widely by Nazi Germany during World War II, in all parts of the German military. But, the Enigma encryption demonstrated to be vulnerable to cryptanalytic attacks by Germany's foes, at first Polish and French and, later, a gigantic effort by the United Kingdom at Bletchley Park. While Germany acquainted a progression of enhancements with Enigma and these hampered efforts of decryption to fluctuating degrees, they didn't decisively keep Britain and its partners from misusing Enigma-encoded messages as a noteworthy source of knowledge during the war. Numerous observers state that this flow of intelligent communication reduced the duration of the war altogether and may even have modified its result.

The growth of computers and communication systems, starting in the 1960s, introduced cryptography as a requirement to secure the digital information. Historically, Data Encryption Standard (DES) is considered the first standard for encrypting unclassified information. It was adopted by the U.S Federal Information Processing Standard as a result of the work conducted by Feistel at IBM 1970s. In fact, DES is the most well-known cryptographic mechanism in history. It continues to be a standard technique for protecting electronic commerce provided by different financial organizations around the world [24].

However, the work published by Diffie and Hellman in 1976 created new directions in cryptography. In fact, this work is considered the most impressive development in the history of cryptography. The authors introduced a new concept of public-key cryptography and an innovative method for key exchange based on a discrete logarithm problem. Despite the fact that there was no practical acknowledgement of public-key cryptography scheme at that time, the idea creates an extensive interest in the cryptography community.

Accordingly, a new practical public-key encryption and digital signature scheme proposed by Rivest, Shamir and Adleman in 1978 is now known as RSA. The idea of RSA is mainly based on the intractability of factoring large integers. This approach of cryptography energized efforts to research for better techniques for factorization.

2.2 Goals of Cryptography

Cryptography has become a hot topic in the existing research due to high demand for applications and computer networks. Currently, cryptographic algorithms are required in all the secure communications and digital data authentication. However, cryptography should not be considered as the only means of securing information, but rather one set of techniques responsible in providing security. In principle, cryptography has primary goals, as summarized below [1, 20, 25]:

- **Data Confidentiality:** Content of a message sent from A to B cannot be read by somebody else and is protected from an unauthorized user.
- **Entity Authentication:** This is a procedure of verifying the user identity to ensure that each arriving message came from a trusted source.
- **Data Origin Authentication:** To enable a received entity and verify that the incoming message has been sent by a trusted entity and the message has not been altered thereafter.
- **Data Integrity:** This method allows the received entity to verify that the inbound message has not been tampered in transit.
- **Non-Repudiation:** This procedure ensures that it is impossible for the sender to turn around later and deny sending the message.

2.3 Private Key Cryptography

Private key cryptography also known as symmetric-key or single-key encryption, was the only used type cryptography scheme until the end of the 1970s. This scheme played a primary role in providing security services in many network devices. It was in use until the development of public-key cryptography, which was developed for tackling the drawbacks of this scheme. The concept of this scheme is based on using a single key during the encryption and decryption processes. It consists of five main components as listed below [20, 26]:

- Plaintext
- Encryption Algorithm
- Secret Key
- Ciphertext
- Decryption Algorithm

Figure 2.1 illustrates the symmetric key cryptography. Here, the plaintext represents the original message, which is fed into the encryption algorithm. The purpose of having an encryption algorithm is to conduct various substitutions and transformation in the plain text.

The secret key *K* used in this scheme is completely independent of the plaintext and

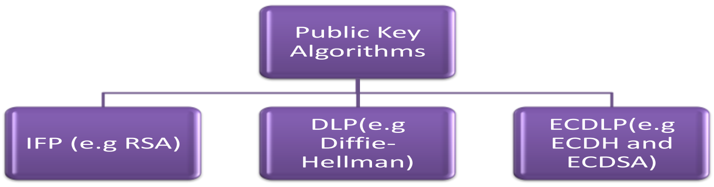


Figure 2.1 Main Families Public Key Cryptography

encryption algorithm. The key is generally selected to be the binary alphabet 0,1. To the plain text X the sender needs to form the cipher Y as a function of *K*. Based on this, the encryption transformation could be written in the form given below:

$$Y = E_k(P)$$

The output of the encryption algorithm, transformation processes, substitution processes is totally dependent on the secret key. The encryption process yields the cipher text, which is mainly a scrambled message and is heavily dependent on the secret key – wherein, if two different secret keys used by the same message then it must result in two different cipher texts. Apparently, the cipher text is a random stream of data that is unintelligible.

The decryption process allows the receiver to retrieve the original message, X, using the below decryption function:

$$X = D_k(Y)$$

During the decryption process, the decryption algorithm must apply the same secret key that has been used in the encryption process. Therefore, the algorithm being used must enable any person to perform the deciphering process, without using the pre-shared key or even figuring out the secret key from the ciphered text. Thus, it is important to note the minimum specifications required while using the encryption algorithm. Accordingly, such an implementation must ensure the secrecy of the secret key, because all the information will be readable to the opponent if he/she knows the secret key and encryption algorithms.

In general, the security of this scheme is maintained by ensuring that the security mechanisms are being used by the sender and receiver. It should be also noted that encryption algorithms are not kept secret for the following reasons:

- To help manufacturers in developing a low-cost implementation of data encryption algorithms
- To help in providing a number of products with different cost allowing end-users to select from varieties.

Symmetric key cryptography has two categories: stream cipher and block cipher. The concept of the former is based on encrypting bit individually by adding a bit from secret keystream to the plaintext. The stream cipher is further divided into two types: synchronous stream cipher and asynchronous stream cipher. The synchronous stream cipher is only dependent on the secret keystream, whereas asynchronous stream ciphering the keystream depends on the cipher text.

The synchronous stream cipher is the most practically-used stream cipher, and an A5/1 cipher, an example of it, particularly used in the GSM mobile phone standard. The concept of the block cipher is based on encrypting the entire block of the plaintext at a time using the secret key.

Advance Encryption Standard (AES) and Data Encryption Standard (DES) or triple (3DES) are standards comes under the block cipher. The AES has a block length of 128 bits (16 Bytes) and DES has a block length of 64 bits. To summarize, the symmetric key cryptography is still playing a major role in providing the security services, such as confidentiality, integrity and authenticity due to its efficiency and short key length. Although it does have powerful advantages, it has some cons which are listed below:

1. It requires a secure transmission channel before exchanging the secret key between the sender and receiver.
2. Setting up shared key manually results in losing control over the secret keys, especially when it is used in the large network which contains a large number of entities.
3. More storage requirements will be needed for storing a large number of key pairs.

These drawbacks can be tackled by using public-key cryptography in line with symmetric key efficiencies and functionalities.

2.4 Public Key Cryptography

Public key cryptography was invented by Diffe Halmen and Markle in 1976. The concept of the public cryptography is based on using two different keys (public or private key) during encryption and decryption processes. Both of the keys are to be generated by the receiver party. Accordingly, the receiver must communicate his public key to the transmitter side. Upon receiving the public key, the transmitter will be in a position to encrypt the data using the public key provided by the receiver party. To decrypt the message, a receiver must use his private key. Although their keys need to be communicated between sender and receiver, the main disadvantage here is that these two keys are transmitted over insecure channels, as shown in Figure 2.1. The use of a public-key algorithm is not limited to exchanging the key, but it can also be used for proving the authentication through the digital signature. Furthermore, there are three basic mechanisms for the public-key algorithm:

1. Digital Signature
2. Encryption
3. Key Establishment Protocol and Key Transport Protocol

In addition to the above mechanisms, the public key schemes allow implementing all

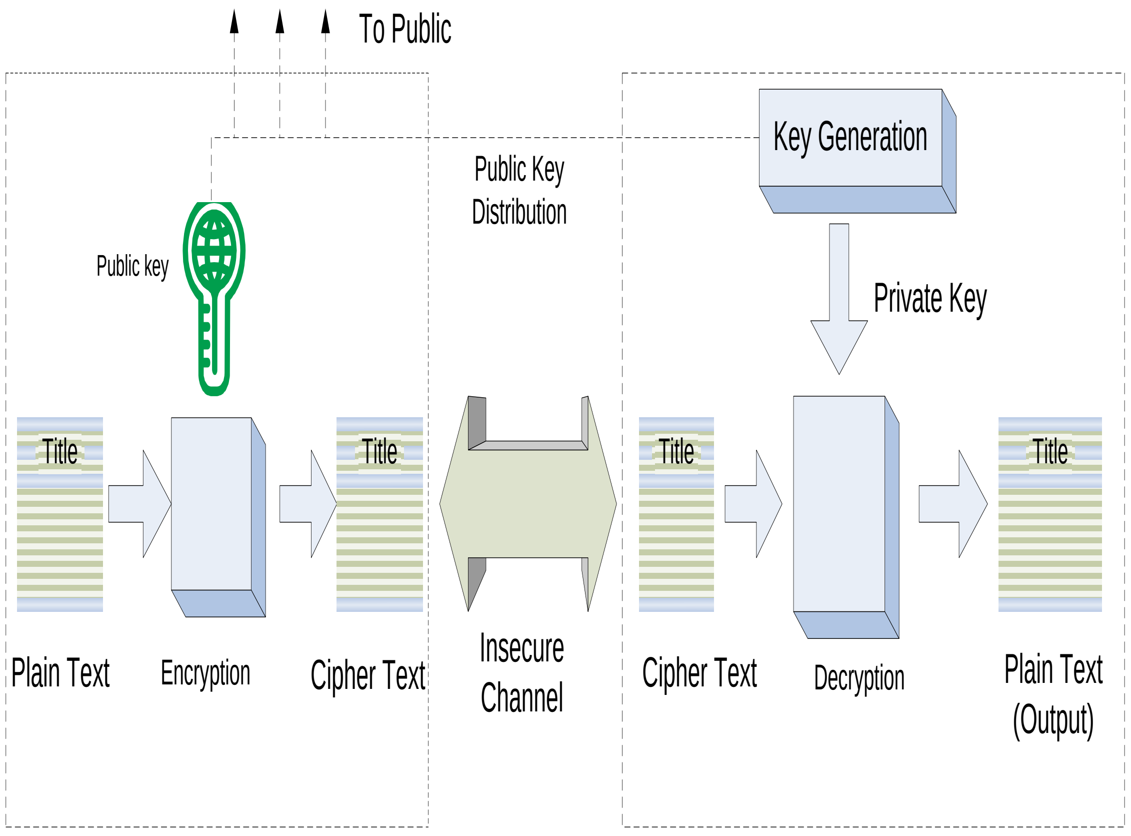
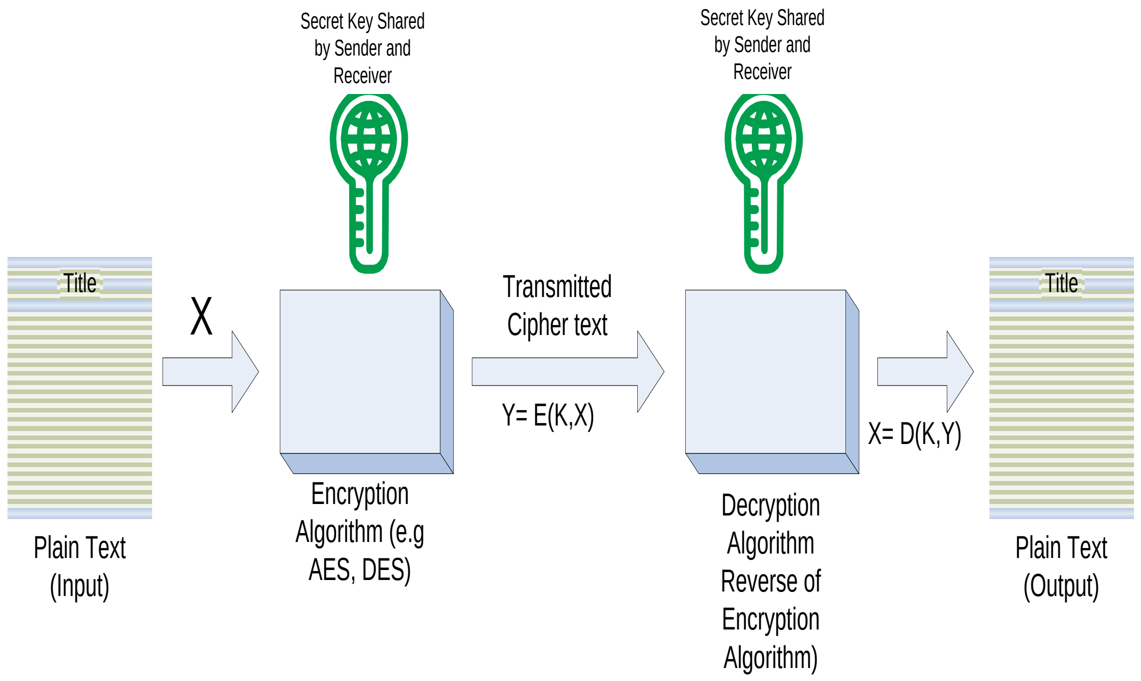


Figure 2.2 Public key cryptography model

required functionalities for modern security protocols, such as SSL/TLS [27]. However, implementing public-key schemes is not an easy task due to its high computational requirements. For that, implementing cryptographic system requires a mixed implementation of symmetric and asymmetric key cryptography, which could be nominated as a hybrid cryptosystem. Hybrid cryptosystem can be achieved by using the public key algorithm for the key establishment, and the symmetric key algorithm will be used to perform the data encryption processes.

There are three main families for public key cryptography listed in Figure 2.3:


Secret Key Shared by Sender and Receiver
Secret Key Shared by Sender and Receiver
Title
X
Transmitted Cipher text
Y= E(K,X)
X= D(K,Y)
Title
Plain Text (Input)
Encryption Algorithm (e.g AES, DES)
Decryption Algorithm Reverse of Encryption Algorithm)
Plain Text (Output)

The first category of the public key algorithms referred to as Integer Based Problem (IFP). The concept of these algorithms is based on determining the prime factors of a given positive integer. RSA, which refers to its developers of the algorithm Rivest, Shamir and Adleman, is one of the famous IFP families [27].

The second type of public key algorithms is called a Discrete Algorithm Problem (DLP). These types of algorithms are based on finding positive integer k of a given α and β such that $\beta = \propto k \mod p$. The two examples of this algorithm are Diffie-Hellman Key exchange protocol and digital key exchange.

The last type of this family is called Elliptic Curve Discrete Logarithm Problem (ECDLP). Its concept is based on finding the positive integer, K, on a given points P and Q in the elliptic curve that is defined over a finite field, such that Q=K.P. Elliptic Curve Diffie Hellman (ECDH) key exchange protocol and Elliptic Curve Digital Signature Algorithm (ECDSA) are examples of ECDLP based algorithms.

2.4.1 Some Definitions From Number Theory

A finite group and field are the primary mathematical constructs used in public-key cryptography. These constructs contain a non-empty set of elements, and they have the ability to generate other set of elements if it is jointly operated with one or more functions. In this thesis, we consider some definitions concerning elementary common algebraic structures. These include *groups, rings and fields*. Seeking further detail, we refer the reader to [28] as it provides in-depth details for a finite field.

Groups

DEF. 2.1 A **group (G)** is defined as a set of elements along with binary operation "*", satisfying four properties. An Abelian group is the most common type of algebraic groups that satisfies the four properties defined below [1, 20, 29]:

1. **Closure:** The group is called closure if a and b are elements of **G**, then $c = a * b$ is also an element of **G** – in which the result obtained after applying the operation on any two elements in the set.
2. **Associatively:** The group is to be considered associative if a, b and c are elements of **G**, then *(a*b)*c=a*(b*c)* – in which the operations can be applied in any order.
3. **Commutativity:** To satisfy a commutative rule all of a and b in **G** should satisfy *a*b=b*a*.
4. **Existence of identity:** For all ‘a’ in **G**, there exists an element *e*, called the identity element, such that e*a=a*e=a.
5. **Existence of an inverse:** The existent element for each a in **G** is a' and known as inverse of *a* and such that $a * a' = a' * a = e$.

Despite the groups that involve one single operation, the properties provided on the operation allow using the pairs of operations. For example, addition and subtraction operations could be supported by defined addition operation in the group, as long as addition is using inverse – in which, if the identity element *e=0,* then the inverse is $a^{-1} = -a$.

DEF. 2.2 If the set has a finite number of elements, it is said to be a finite group; otherwise, it is known as an infinite group [20]. The number of elements in a finite group is referred to as the order of the group, |**G**|. However, if the group is not finite, its order also is infinite; if the group is finite, the order is finite.

DEF. 2.3 A subset ***H*** of a group ***G*** is a subgroup of ***G,*** if ***H*** itself is a group with respect to the operation on ***G***. Therefore, if $\boldsymbol{G} =< S,\bullet>$ is a group, $H =< T,\bullet>$ is a group under the same operation, and ***T*** is a non-empty subset of ***S***, H ***is a subgroup*** of ***G*** [28]. In accordance with this, the definition implies the following [20]:

1. The groups share the same identity element.
2. Each group has a subgroup of itself.
3. The group made of the identity element of ***G***, *H=<[5],•>* is a subgroup of ***G***.
4. If a and b are members of both groups, then c=a*b is also a member of the groups.
5. If a is a member of both groups, the inverse of a is also a member of both groups.

DEF. 2.4 A multiplicative finite group **G** is called cyclic if all elements of the group $a \in G$ can be generated by repeated application of group operation. Thus, if there is an element $a \in G$ such that for any $a \in G$, there is some integer j with $b = a^j$. However, such an element is nominated as a generator of the cyclic group and to be written as $\boldsymbol{G} = \langle a \rangle$ [28].

DEF. 2.5 If a subgroup of a group can be produced by applying the power of an element, then it is known as the cyclic subgroup. The term “power” here stands for repeatedly employing the group process to the element, which is presented below[20] :

$$a^n \rightarrow a * a * \cdots * a(n\ times)$$

Ring

DEF. 2.6 A ring is an algebraic structure having two operations, and denoted as $R =< \{\cdots\},*,>$. All of the abelian group properties must be fulfilled by the first operation. The second operation must satisfy only the first two properties, and it must distribute over the first operation.

DEF. 2.7 A ring is called distributive if all a,b and c elements of ***R*** have $a\square(b * c)$ = *(a□b)*(a□c) and (a*b)□c=(a□c)*(b□c).* A commutative ring is a ring in which the commutative property is also satisfied for the second operation.

Field

DEF. 2.8 A field represented by F=<{⋯},*,□> is a commutative ring in which the second operation satisfies all five properties defined for the first operation, except that the identity of the first operation and (sometimes called the zero elements), which has no inverse.

2.4.2 RSA

RSA, whose concept was first introduced by Diffie-Hellman, was developed by Revest, Adi Shamir and Len Adleman in 1978. Their main objective was the development of a cryptographic algorithm that could meet the requirements of the public-key cryptosystem. The RSA algorithm can be categorized under the block cipher. The format of the plaintext and ciphertext in the RSA, whose typical size is 1024 bits or 309 decimal digits < 2^{1024}, is referred to as integers between 0 and n-1. 2^{1024}.

RSA is involved in various applications like key transport, encrypting small pieces of data. Digital signature is another application of RSA which can be utilized for the digital certificates on the internet. RSA is unable to replace the symmetric cipher since it is very much slower due to higher number of computations in comparison with the AES. The main purpose of having RSA encryption feature is to provide highly secure key exchanges for a symmetric cipher. The primary objective of RSA encryption feature is proving the highest security for key exchanges in a symmetric cipher, which means that the RSA is generally used along with symmetric ciphers like AES since it has the responsibility of performing bulk encryption of the data [20]. The RSA encryption is accomplished by using ring Z_n and modular computation. The functions below describe the RSA encryption and decryption.

RSA Encryption:

Given the public key (n,e) ≡ K_{pub} and Plaintext Y=e. K_{pub} (x) ≡X^e modn where, x,y ϵ Z_n

RSA Decryption:

Given the private key d ≡ $K_{private}$ the ciphertext Y, X ≡ d. $K_{private}$ (Y) ≡ Y^d modn where, X,Y ϵ Z_n

Generally, implementing RSA is more critical in comparison with 3DES or AES since it involves the exponation of large numbers. Additionally it involves algorithms of modular multiplication, squaring and multiply [20]. As the RSA has high computational complexity, the Elliptic Curve Cryptography (ECC) is another option for implementing public key cryptosystem due to its attractive features and reported efficiencies [1]. Some of the merits of ECC are listed below:

- ECC can provide the same level of security as that of RSA with smaller key sizes.
- ECC requires lesser memory size and faster arithmetic operations.

- There exists a high possibility of implementing ECC in the constrained devices like mobiles, as it requires less memory and less power consumption.

Therefore, based on the ECC advantages, we consider it as the main scope of this thesis.

2.4.3 Digital Signature Algorithm (DSA)

The main concept of a digital signature scheme is to provide the same services provided by traditional signature. Normally, a conventional signature is included in the document, whereas a digital signature is a separate entity. To verify the conventional signature, the recipient needs to compare the signature with a signature in the document, whereas to verify a digital signature, the recipient needs to apply a verification process to the documents and signature. Different services that could be provided by digital signature include message authentication, integrity and non- repudiation.

A confidential communication cannot be provided by a digital signature. Thus, if confidentiality is needed, the message and the signature must be encrypted using either secret key or public-key cryptosystem. For example, the RSA digital signature scheme uses the RSA cryptosystem, and also the ElGamal digital signature uses ElGamal cryptosystem. In normal public key cryptography communication, we use the public key and private key of the receiver. In contrast, with digital signature communication, the public and private key of the sender is used.

Besides the RSA digital signature and ElGamal digital signature, Elliptic Curve Digital Signature Algorithm came as an alternative solution for providing confidential service. ECDSA consists of three processes, which are key generation process, signing process and signature process. These processes are explained in detail below and are based on implementing ECDSA between the sender Alice and the receiver Bob:

- **Key Generation process:**
  1. Alice selects elliptic curve $E_P(a, b)$ with p is a prime number.
  2. Alice selects q which to be used in the calculation
  3. Alice selects the private key d
  4. Alice selects the point in the curve $e1(\dots\dots, \dots\dots)$
  5. Alice calculate $e2$ which can be obtained by multiplying $d \times e1 = e2$
  6. Alice is now ready to send the public key which includes $(a, b, p, q, e1, e2)$

- **Signing Process:** The main purpose of the signing process is selection of a random number, creation of a third point, calculation of the signature and sending the message with the signature.
  1. Alice selects random number r between 1 and $q - 1$
  2. Alice calculates the third point $P = r \times e1(\ldots.,\ldots.)P = (U, V)$
  3. Alice calculates the first signature $s1 = Umodq$
  4. Alice calculates the second signature $s2 = (h(M) + d \times s1)r^{-1} mod\ q$ where $(h)M =$ message digest,d = private key r = secret random number
  5. Alice sends $s1, s2$ and the message
- **Verification Processes**
  1. Bo calculates the intermediate result $(A\ and\ B)$as below:
     - $A = h(M)s1^{-1}modq$ and $B = \ s2^{-1}s1\ modq$
  2. Bob constructs the $T\ point\ T(x, y) = A \times e1(\cdots, \cdots)and\ B \times e2(\cdots, \cdots)$ if $x = s1$ then the signature is True.

2.5 Elliptic Curve Key Cryptography

Although the RSA and ElGamal cryptosystems are providing considerably secure symmetric key cryptosystems, their security comes with a price due to their large keys. To overcome this issue, researchers have looked for alternatives that offer the same level of security with smaller key sizes. One of the promising solutions is the Elliptic Curve Cryptography (ECC). The principle of Elliptic Curve, backdated to the mid of 19th century, was discovered by Victor Miller (IBM) and Neil Koblitz (University of Washington) in 1985.

Table 1 clearly exemplifies the fact that ECC requires a much smaller key size in comparison to RSA for providing the same level of security since the security per key bit rate is much higher. For example, the level of security offered by a 3072-bit legacy key (RSA) is

Table 2.1 NIST Guidelines for Public key size [2]

ECC key	RSA/DH key Size	Key-Size Ratio	AES Key Size
163 bit	1024 bit	1:6	N/A
256 bit	3072 bit	1:12	128 bit
384 bit	7680 bit	1:20	192 bit
512 bit	15360	1:30	256 bit

the same as that offered by a 256 bit ECC key, and thus ECC offers better performance with a key size that is 1/12th of RSA key. This type of performance efficiency gets better on increasing the security level. Hence, they can be effectively used in constrained platforms like wireless devices, handheld computers, smart cards, etc. [30]

The Concept of Elliptic Curve Cryptography is very rich with theories and deep arithmetic. Therefore, ECC implementation requires a focus on different arithmetic and operations and algorithms. Figure 2.4 describes the system level of elliptic curve cryptography implementation. The top layer in the pyramid represents the implementation of ECC Protocols levels, such as Elliptic Curve Digital Signature Algorithm (ECDSA), Elliptic Curve Diffie Hellman ECDH and Elliptic Curve Integrated Encryption Schemes (ECIES). Scalar point multiplication is considered as a second level before ECC protocol. The fundamental operation of the elliptic curve cryptography is scalar point multiplication, which is defined in equation (2.1) as follows:

$$Q = k.P \tag{2.1}$$

The scalar point multiplication is based on point addition and point multiplication. The point addition operation for two given points $P, Q \in E$, resulting in point known as the sum of P and Q,$P + Q \in E$, whereas the point multiplication is the process concern of

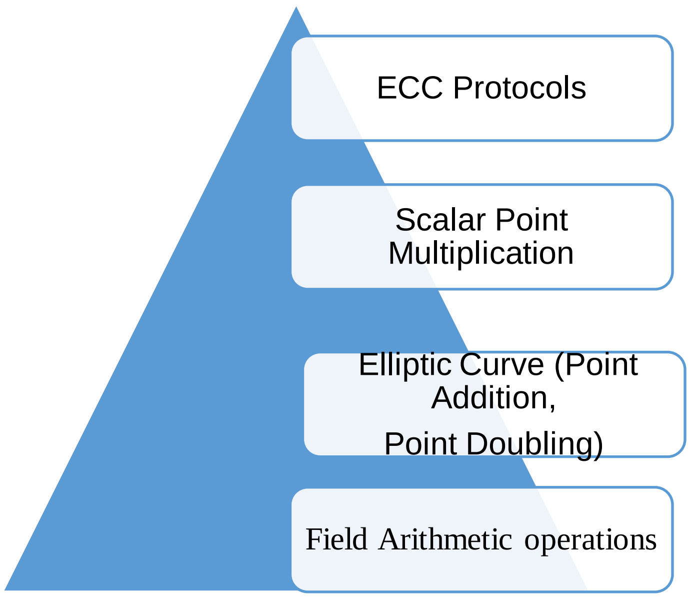

ECC Protocols
Scalar Point Multiplication
Elliptic Curve (Point Addition,
Point Doubling)
Field Arithmetic operations

Figure 2.4 ECC Implementation Pyramid

multiplying two elements in the multiplication group **G**F*(q) for integer modulo a prime P.

Different point multiplication algorithms are presented in [1] to compute (1). The performance of ECC depends on the point multiplication and its associated coordinates systems. The scalar point multiplication for two points $Q, P \in {}^{E}/_{GF(q)}$ which belongs to elliptic curve $E \in \boldsymbol{GF}(q)$ is basically defined over an Abelian group as shown below:

$$Q = K.P = \underbrace{P + P \cdots P + P}_{k-1\ times}$$

The implementation of Elliptic Curve Cryptography is based on scalar point multiplication and Elliptic Curve Discrete Logarithm Problem (ECDLP). The concept of ECDLP based on finding K for a given Q and P, where the parameter of K is called discreet algorithm of Q to the base P and K= log_p Q Despite the availability of this algorithm, it does not have the capability to solve the ECDLP, as it can only be used for factorizing a large number. For this reason, the RSA requires a larger key size, and ECC can provide the same level of security with a shorter key length.

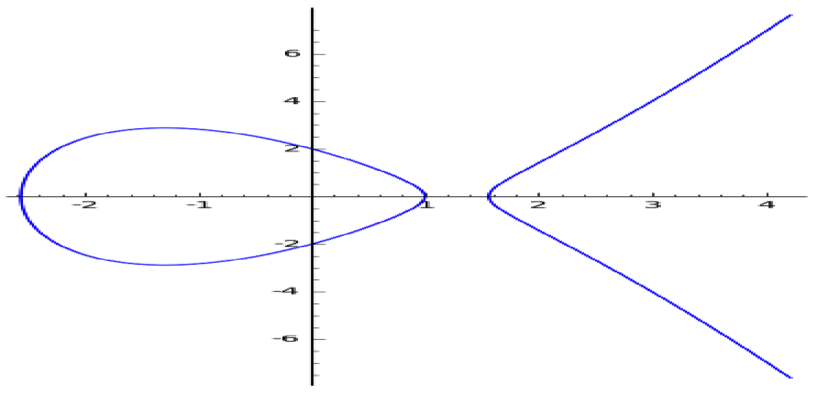


Figure 2.5 The Elliptic Curve $\boldsymbol{y^2 = x^3 - 5x + 4}$ over R

An elliptic curve E over a field K could be defined over either the field R of the real numbers, the field Q of rational numbers, the field C of complex numbers, or finite field F_q of $q = p^r$elements. Figure 2.5 shows an elliptic curve over the rational field Q. Thus, an elliptic curve over K is to be defined as a set of points (x, y) where $x, y \in K$ that satisfies the following equation:

$$E: y^2 = x^3 + ax^2 + b \tag{2.3}$$

where $a, b \in k$ is to be a cubic polynomial with no multiple roots and K is to be a field of characteristic $\neq 2{,}3$.

Therefore, the elliptic curve E over finite field GF is to be defined using the following *long Weierstrass* equation in the projective form [1]:

$$E: Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3 \tag{2.4}$$

where $a_1, a_2, a_3, a_4, a_6 \in GF$.

Weierstrass equation (2.4) represents a smooth elliptic curve in projective coordinates. It has the correspondence in the affine (Euclidean) coordinates, containing the form:

$$E: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.5}$$

where $a_1, a_2, a_3, a_4, a_6 \in GF$. and $\Delta \neq 0$ and Δ is discriminante of E which defined as below:

$$\left.\begin{array}{l} \Delta = -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \\ d_2 = a_1^2 + 4a_2 \\ d_4 = 2a_4 + a_1a_3 \\ d_6 = a_3^2 + 4a_6 \\ d_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2 \end{array}\right\}$$

However, the below simplified *Weierstrass* equations provided by [1] shows that it is not necessary to use whole equations (2.4 and 2.4). In which, the same original *Weierstrass* equation could be rewritten in a simpler way, depending upon the field characteristics.

Simplified *Weierstrass* equations:

1. When characteristics of *K* is not equal to either 2 or 3 then *E* transforms to a curve where $E: y^2 = x^3 + ax + b, a, b \in K \; and \; \Delta = -16(4a^3 + 27b^2)$.
2. When characteristics of *K* is 2 and if $a_1 \neq 0$, then *E* transforms to non-super singular curve where $E: y^2 + xy = x^3 + ax^2 + b, a, b \in K \; and \; \Delta = b$.
3. When characteristics of *K* is 2 and if $a_1 = 0$ then *E* transforms to a supersingular curve where $E: y^2 + cy = x^3 + ax^2 + b, a, b, c \in K \; and \; \Delta = c^4$.

4. When characteristics of K is 3 and $a_1^2 \neq -a_2$, then E transforms to a non-super singular curve where $E\colon y^2 = x^3 + ax^2 + b,\ a,b,c \in K\ and\ \Delta = -a^3b.$
5. When characteristics of K is 3 and $a_1^2 = -a_2$, then E transforms to a supersingular curve where $E\colon y^2 = x^3 + ax^2 + b,\ a,b,c \in K\ and\ \Delta = -a^3.$

Points on the Elliptic Curve

The points in the elliptic curve E over the field L are defined below:

$$E(L) = \{\infty\} \cup \{(x,y) \in L \times L | y^2 + \cdots = x^3 + \cdots\}$$

The points in the elliptic curve are to be known as the set of a collection in the union of the point at infinity where (x,y) that belongs to L, which satisfy the original curve equation (2.5). The point at infinity is kind of a point sitting at the top of y-axis and bottom of the y-axis. Using the Diophantus techniques drawing a vertical line between two points in the elliptic curve, these two points should intersect in the elliptic curve in the third point. However, in the case of the point at infinity, the points will actually intersect again in the elliptic curve at the point of infinity. In fact, this is very useful as it helps in defining the concept of the group, which is very much needed in order to apply the elliptic curve concept for various applications. In principle, the elliptic curve should form an abelian group as it has two points P and Q in a defined field over the elliptic curve, resulting in a third point denoted by P+ Q lays on E the field. Thus, given two points P, Q in $E(F_p)$ yield a third point, denoted by $P + Q$ on $E(F_p)$, which should satisfy the following properties:

- $P + Q = Q + P$ (Commutativity) – in which adding P and Q should be identical if we are adding Q and P.
- $(P + Q) + R = P + (Q + R)$ (Associativity) – which means it does not matter in which order we are performing the addition operation.
- $P + O = O + P = P$ (existence of an identity element). This means there should be an identity element, and if we add P and O, then we should get back as the same as O plus P because of the commutativity property. Also, both of them should actually go back to P as there is O. However, O is generally referred to as the point of infinity, which is assumed to be the identity element in the plus operation.
- Three exists $(-P)$ such that $-P + P = p + (-P) = O$ (Existence of Inverse). This property means that another point exists as $(-P)$, such that if we take $(-P)$ and

added with $(-P)$ that additive is the inverse of P, which should give us back O point on infinity again. Therefore, a minus P is the additive inverse of P.

Point Addition Point Doubling

For example, if we have two points $P\ and\ Q$ in the elliptic curve:

$$y^2 = x^3 - x + 1$$

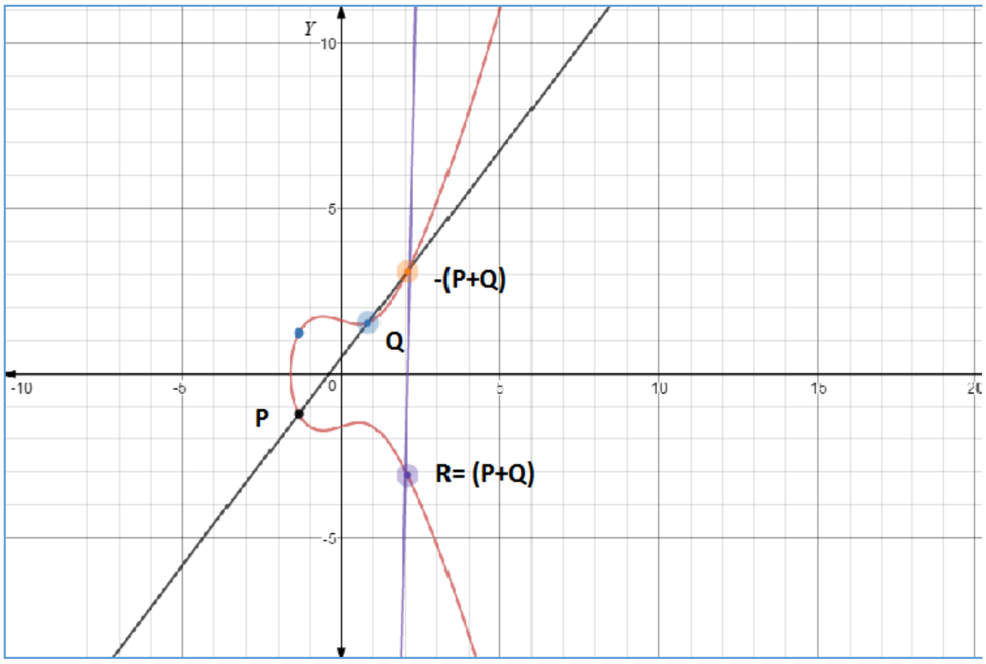


Figure 2.6 The Elliptic Curve Point Addition $\boldsymbol{y^2 = x^3 - x + 1}$

which shows the form of an elliptic curve that we have chosen here, if $P\ and\ Q$ are two points in this curve. Thus, we can define the addition operation for the $P\ and\ Q$ using Diophantus techniques – in which we can draw a straight line through them, and it will intersect in the elliptic curve in the third point. Accordingly, we take the reflection of the third point in x-axis as the sum, and we refer R as the sum of $P and\ Q$. In fact, this has a very close relationship with Diophantus techniques.

2.5.2 Elliptic Curve Parameter Selection

In general, the implementation of ECC requires more focus and a number of decisions taken at different levels are presented in the ECC pyramid (Figure 2.4).

- **At the ECC protocol layer**
    - Appropriate selection of protocols (key exchange or signature)
- **At the Scalar Point Multiplication layer**
    - Selecting the algorithm for scalar point Multiplication $k.P$
- **At the Elliptic Curve layer**
    - To select a proper point addition and point doubling algorithms
    - To choose the type of representation for the points (affine or projective coordinates)
- **At the Field Arithmetic layer**
    - A decision for selecting an underlying field $(GF(2^m), GF(p), GF(p^m))$ need to be taken.
    - A selection of the field representation (e.g polynomial or normal basis)
    - Appropriate selection for the finite field algorithms (Addition, Multiplication, Squaring, Reduction, Inversing)

Having such choices and huge flexibility makes ECC feasible for both constrained devices and high-performance servers. Subsequently, we first provide a fundamental arithmetic for the curve defined over prime field $GF(p)$ (*Section 2.5.3*) and elliptic over the binary field $GF(2^m)$ (*Section 2.5.4*). In general, in this thesis, we only present the algorithms that are applied in our research. Further detail of different algorithms could be obtained in [1, 25].

2.5.3 Elliptic Curve Arithmetic Over $GF(P)$

Finite field or Galois field was invented by (Evariste Galois) in the 19th century [31]. The finite filed is a field included with finite field order. The order of finite field could be represented by the prime or the power of a prime [32]. In practice, the finite field is used in many applications besides the cryptography like number theory, algebraic geometry Galois theory and quantum error correction. ECC could be implemented in constrained devices underlying three types of finite fields which include: elliptic curve over prime field GF(p), elliptic Curve over binary fled $\mathrm{GF}(2^{\mathrm{m}})$ and over a prime extension field $\mathrm{GF}(\mathrm{p}^{\mathrm{m}})$. However, in this thesis, our research is based only on the binary field and prime field. Further details will be provided in the next sections, but they are limited to those related to this thesis.

The elliptic curve over prime field is represented by the following equation:

$$E\colon y^2 = x^3 + ax + b \bmod p,\ \ where\ 4a^3 + 27b^2 mod\ p \neq 0$$

The finite field elements for the above equation are integers number between 0 and $p - 1$. The integers to be involved in all of the Elliptic Curve modular arithmetic operations include addition, subtraction, multiplication, division and multiplicative inverse. The selection of prime number is to be conducted based on SEC specification, where p is rated between 112-521 bits. Figure 2.5 shows points generated by Sagemath tool for the prime field of size $y^2 = x^3 + x$

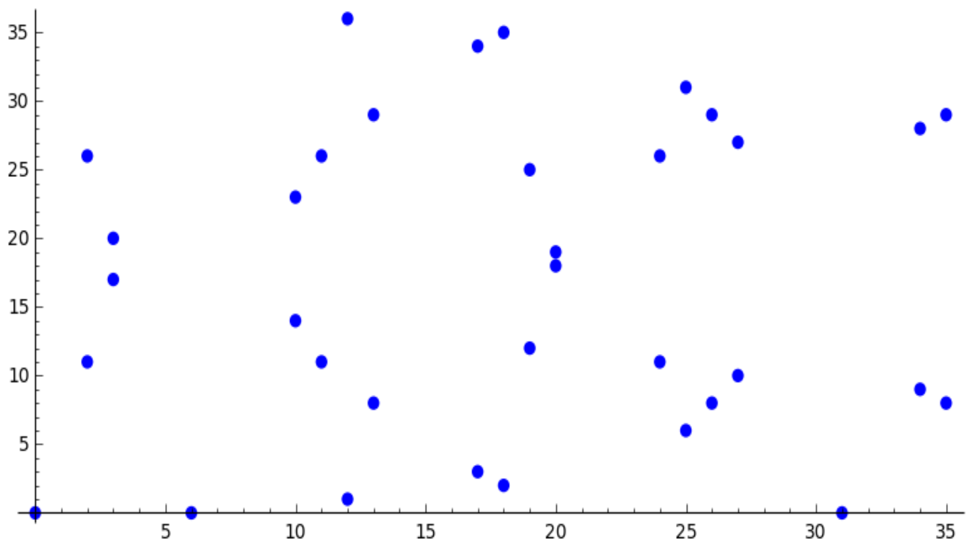


Figure 2.7 Elliptic Curve Point for $y^2 = x^3 + x$

Projective coordinate representations

Mathematically, elliptic curve over prime field consisting of integer P over finite field $GF(p)$ and the elements $a, b \in F_p$ can be defined by the equation below:

$$E\colon y^2 \equiv x^3 + ax + b \pmod p$$

Basically, (x, y) points are to be represented by the coordinate referred to as Affine Coordinates (A). However, it is a very common practice that projective coordinates are used in replacing the Affine Coordinates and represent the points P and Q. This is because Affine Coordinates over the prime field is expensive due to the necessary field inversion operations during the Elliptic Curve Scalar Point Multiplication (ECSPM) computations.

- Using the standard projective coordinates, the affine point $({}^{X}/_{Z}, {}^{Y}/_{Z})$ could be used in correspondence to the projective point $(X\colon Y\colon Z), Z \neq 0$. The point at infinity ∞

corresponds to corresponds to $(0:1:0)$, and the negative of $(\mathrm{X}:\mathrm{Y}:\mathrm{Z})$ is $(\mathrm{X}:-\mathrm{Y}:\mathrm{Z})$. The elliptic curve equation that corresponds to the standard projective coordinates is:

$$\mathrm{Y}^2\mathrm{Z} = \mathrm{X}^3 + \mathrm{aXZ}^2 + \mathrm{bZ}^3$$

- Using the Jacobian projective coordinates, the affine point $({}^{\mathrm{X}}/_{\mathrm{Z}^2}, {}^{\mathrm{Y}}/_{\mathrm{Z}^3})$ could be used in correspondence to the projective point $(\mathrm{X}:\mathrm{Y}:\mathrm{Z}), \mathrm{Z} \neq 0$. The point at infinity ∞ corresponds to $(0:1:0)$, and the negative of $(\mathrm{X}:\mathrm{Y}:\mathrm{Z})$ is $(\mathrm{X}:-\mathrm{Y}:\mathrm{Z})$. The elliptic curve equation corresponding to the Jacobian projective coordinates is:

$$\mathrm{Y}^2\mathrm{Z} = \mathrm{X}^3 + \mathrm{aXZ}^4 + \mathrm{bZ}^6$$

The calculation of point addition and point doubling is highly dependent on the type of the selected projective coordinates. Thus, these representations can be considered as advantageous if the inversion operation is more expensive when compared to multiplication in the finite field. To accomplish the processes of elliptic curve arithmetic, a single inversion at the end of point addition and point doubling is needed. This can be achieved using the Fermat's Little Theorem: $\mathrm{x}^{-1} \equiv \mathrm{x}^{\mathrm{p}-2} \bmod \mathrm{p}$. However, Table 2.2 illustrates different complexity concerning group operations for a different type of coordinates representation on $y^2 = x^3 - 3x + b$. Where A=affine representation, J=Jacobin, P=Standard Projective, I= Field Inversion, M=Field Multiplication, S=Field Squaring.

Table 2.2 Point addition and Point Doubling Operation Counts[1]

Point Doubling		General Addition		Mixed Coordinates	
2A →A	*1I,2M,2S*	*A+A→A*	*1I,2M,1S*	*J+A→J*	*8M,3S*
2P → P	*7M, 3S*	*P+P→P*	*12M,2S*		
2J → J	*4M,4S*	*J+J→J*	*12M,4S*		

2.5.4 Field Arithmetic over $GF(P)$

In this section, we present some of the algorithms functioning as arithmetic in the prime field $GF(P)$. In fact, the elliptic curve cryptography is to be defined on a finite field known as Galois Field *GF*. The selected GF is used to define a set of operations that are used to compute point doubling and point addition. However, in the case of *GF(P)* the performance of modular arithmetic is very essential, as its performance directly affects the overall performance of ECC algorithms on *GF(P)*. Since field arithmetic over *GF(P)* consists of several algorithms, we presented some of these algorithms along with their functionality.

Modular Reduction:

The modular operation is to be used for reducing to modulo P, where P is large. In which, a finite field of order P, *GF(P)* with P prime is to be identified as the set Z_p of integers $\{0,1,\cdots,P-1\}$. The main concept of modular R arithmetic is based on dividing C by M where C, M $\in Z$ such that $M < C$. In which, the modular reduction is a process of computing $R = C \: mod \: M$ – i.e., the remainder R of the division C is to be represented as below:

$$R = C - \left\lfloor \frac{C}{M} \right\rfloor M$$

Having a modular reduction in place would help in avoiding an expensive operation. For example, the expensive operations to be conducted by division operation could be avoided by using modular reduction. Additionally, having a special form of reduction steps for modulo P can result in archiving a considerable acceleration, which would directly enhance the performance of ECC. Performing ECC computation consists of conducting a variety of modulo P arithmetic. Different modular reduction algorithms have been proposed by researchers to help in achieving fast reduction. For example, a classical method could be replaced by Barret methods, due to its powerful mechanism on conducting a reduction with less-expensive operations. Alternatively, when the cost of input and output conversions is cancelled out, the Montgomery's method could be selected because it is capable of reducing the number of intermediate multiplications [1]. In addition to that, there are five recommended elliptic curves by FIPS 186-2 standard with moduli, as shown below, that can yield fast reduction algorithms, especially on word size 32 [1]:

$$P_{192} = 2^{192} - 2^{64} - 1$$

$$P_{224} = 2^{224} - 2^{96} + 1$$

$$P_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$$

$$P_{384} = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$$

$$p_{521} = 2^{521} - 1$$

Considering the above prime properties, a powerful reduction algorithm could be obtained especially with a machine with 32-word size. In fact, these properties could be presented as sum or differences of a small number of power 2. However, these powers appear to be a multiple of 32. For example, a $P = P_{192} = 2^{192} - 2^{64} - 1$ could be reduced using a congruence arithmetic. Thus, let *c* be an integer with $0 \leq c < P^2$

$c = \ c_5 c^{320} + c_4 2^{256} + c_3 2^{192} + c_2 2^{128} + c_1 2^{64} + c_0$

where the base of 2^{64} representation of c where each $c_i \in [0, 2^{64} - 1]$ reduce the higher power of 2 in (2.) using the congruence

$$c \equiv c_5 2^{128} + c_5 2^{64} + c_5 + c_4 2^{128} + c_4 2^{64} + c_3 2^{64} + c_3 + c_2 2^{128} + c_1 2^{64} + c_0 \ (mod\ P)$$

and $c\, modulo\, p$ can obtained by adding the four 192-bit integers $c_5 2^{128} + c_5 2^{64} +$

Algorithm 2.1 Fast Reduction modulo$\boldsymbol{P_{192} = 2^{192} - 2^{64} - 1}$

```
INPUT: An integer $c = (c_5, c_4, c_3, c_2, c_1, c_0)$ in base $2^{64}$ with $0 \leq c < P_{162}^2$ .
OUTPUT: $c\ mod P_{192}$
    1) $Define\ 192 - bit\ integers$

       $s_1 = (c_2, c_1, c_0),\ s_2 = (0, c_3, c_3),$
       $s_3 = (c_4, c_4, 0),\ )\ s_4 = (c_5, c_5, c_5)$

    2) Return $(s_1 + s_2 + s_3 + s_4)\ mod\ P_{192}$
```

$c_5, c_4 2^{128} + c_4 2^{64}, c_3 2^{64} + c_3, c_2 2^{128} + c_1 2^{64} + c_0$ and continually subtracting P until the result is less than p. This procedure can be illustrated on fast modular reduction for modulo $P_{192} = 2^{192} - 2^{64} - 1$ Algorithm 2.1 [1]. However, in chapter 6, we consider using a Generic Generalized Mersenne Reduction procedure as proposed by [19].

Addition and Subtraction: The mechanism for performing the field addition and subtraction is quite straightforward. [1] proposed Algorithm 2.2 and Algorithm 2.3 performs the addition and subtraction respectively of multiword integers where assigning in the form "(ε,z)←w" for

Algorithm 2.2 Multi-precision addition

```
INPUT: Integers $a, b \in [0, 2^{Wt}). c$
OUTPUT: $(\varepsilon, c) where\ c = a + b\ mod 2^{Wt}\ and\ \varepsilon\ is\ carry\ bit.$
    1) $(\varepsilon, C[0]) \leftarrow A[0] + B[0]$.
    2) For $i$ from 1 to $t - 1$ do.
        2.1 $(\varepsilon, (C[i]) \leftarrow A[i] + B[i] + \varepsilon$
    3) Return $(\varepsilon, c)$
```

an integer w means that $z \leftarrow w\ mod 2^{W}$ and $\varepsilon \leftarrow 0\ if\ w \in [0, 2^{W})$, otherwise $\varepsilon \leftarrow 1$.

Algorithm 2.3 Multi-precision subtraction

```
INPUT: Integers a,b ∈ [0, 2Wt).
OUTPUT: (ε, c) where c = a + b mod 2Wt and ε is the carry bit.
    1. (ε,C[0])← A[0] + B[0].
    2. For i from 1 to t −1 do
        2.1 (ε, C[i])← A[i] + B[i] +ε.
    3. Return(ε, c).
```

However, to compute modular addition $((x + y) mod\ P)$ and subtraction $((x - y) mod\ P)$, some modification with additional reduction steps on reduction modulo P are required. These steps start by performing the steps mentioned in Algorithms 2.2 and Algorithm 2.3, then processed with If and Else If statements, as shown below:

Modular Addition:

If $\varepsilon = 1$, then substract $P\ from\ c = (C[t-1], \cdots, C[2], C[1], C[0])$;

Else if $c \geq P\ then\ c \leftarrow c - p$

Modular Subtraction:

If $\varepsilon = 1,$ then add $P\ to\ c = (C[t-1], \cdots, C[2], C[1], C[0]);$

Modular Multiplication:

The efficiency of modular multiplication plays a very important role in the overall performance of ECC. However, to perform a field multiplication of $a, b \in F_P$, the multiplication process of a and b as integers need to be accomplished first. Then, they need to be processed with a reduction process of the result modulo p. Operand-scanning, product scanning, Comba Algorithm, Montgomery Multiplication and Karatusba Multiplication are the most popular modular multiplication algorithms. The operand scanning method and product scanning method are based on obtaining the bit quantity (U V) by concatenating of *W-bit* word U and V. Algorithm 2.4 illustrates the integer multiplication using the operand scanning method. In this algorithm, the main operation is to be executed in step 2.2, known as inner product operation. The calculation process is used to be represented by $C[i+j] + A[i] \cdot B[j] + U$ and operands are w-bit values. In general, the inner product in this algorithm is bounded by $2(2^w - 1) + (2^w - 1)^2 = 2^w - 1$, which can be depicted by *(UV).*

A product scanning Algorithm 2.5 is based on calculating the product $\mathrm{c} = \mathrm{a.b}$ from right to left. However, a (2w) bit of $\mathrm{w-bit}$ operand is required and values of $\mathrm{R_0, R_1, R_2, U\,and\,V}$ are presented by w-bit words. Another form of product scanning algorithm is so-called a Comba algorithm, proposed by [33]. The idea of Comba algorithm is quite similar to the product scanning, since the outer loops move through the words of product P. As shown in Algorithm 2.6, a Comba algorithm mainly consists of two inner loops and two outer loops. The inner loops are responsible for performing a bulk of computation. This includes executing Multiply-Accumulate (MAC) operations (i.e., two bits words are multiplied and $2\ \mathrm{w-bit}$ products to be added to the accumulative sum). Therefore, three w-bit registers are needed for storage purposes, since the same operation can be easily longer than w-bit long. The accumulative sum of the values *(t,U,V)* represent the integer value $t.2^{2w} + U.2^w + V$. As shown in the Algorithm 2.6, it is very clear that line 7 and 14 are just performing right -shift of accumulative sum *(t,U,V)* [17]. After conducting the integer multiplication using the Comba algorithm, a fast modular reduction algorithms proposed by [1] or [19] could be used in order to reduce the result of modulo P.

However, these new techniques of multiplication created a consolidated background for the researchers to select between them during the implementation. Also, it opens trapdoor to them for further enhancement, such as the work conducted by [17].

Algorithm 2.4 Integer Multiplication - Operand Scanning

```
INPUT: INPUT: Integers $a, b \in [0, P-1]$
OUTPUT: $c = a.b$ .
1) $Set\ C[i] \leftarrow 0 for\ 0 \leq i \leq t-1$
2) For i from 0 to $t-1$ do.
        2.1 $U \leftarrow 0$.
        2.2 For j form 0 to $t-1$ do
                $(U\ V) \leftarrow C[i+j] + A[i].B[j] + U$
                $C[i+j] \leftarrow V$.

        2.3 $C[i+t] \leftarrow U$
3) Return $(c)$
```

Algorithm 2.5 Integer Multiplication - Product Scanning

```
INPUT: INPUT: Integers $a, b \in [0, P-1]$
OUTPUT: $c = a.b$ .
    1) $R_0 \leftarrow 0, R_1 \leftarrow 0, R_2 \leftarrow 0$

    2) For $k$ from 0 to $2t-2$ do.

        2.1 For each element of $\{(i,j) | i+j = k, 0 \leq i,j \leq t-1\}$ do
                $(U\ V) \leftarrow A[i].B[j]$
                $(\varepsilon, R_0) \leftarrow R_0 + V$.
                $(\varepsilon, R_1) \leftarrow R_1 + U + \varepsilon$.
                $R_2 \leftarrow R_2 + \varepsilon$.
        2.2 $C[k] \leftarrow R_0,\ R_0 \leftarrow R_1, R_1 \leftarrow R_2, R_2 \leftarrow 0$
    3) C[$2t-1$] $\leftarrow R_0$.
    4) Return $(c)$
```

Algorithm 2.6 Comba Algorithm

INPUT: $A = (a_{s-1}, \cdots, a_1, a_{0,}) \; and \; B = (b_{s-1}, \cdots, b_1, b_{0,})$.

OUTPUT: Product $= A.B \; (P_{2s-1}, \cdots, P_1, P_0)$

```
1: $(t, u, v) \leftarrow 0$
2: for i from 0 by 1 to s do
3:     for j from 0 by 1 to i do
4:         $(t, u, v) \leftarrow (t, u, v) + a_j \times b_{i-j}$
5:     end for
6:     $P_i \leftarrow v$
7:     $v \leftarrow u,\ u \leftarrow t, t \leftarrow 0$
8: end for
9: for i from  s by 1 to 2s-1 do
10:     for j from i+1--s by 1 to s do
11:         $(t, u, v) \leftarrow (t, u, v) + a_j \times b_{i-j}$
12:     end for
13:     $P_i \leftarrow v$
14:     $v \leftarrow u,\ u \leftarrow t,\ t \leftarrow 0$
14: end for
15: $P_{2s-1} \leftarrow v$
```

Modular Squaring:

In general, a field squaring of $a \in F_P$ could be completed by firstly squaring a as an integer, then reducing obtained result modulo P. The principle of squaring long integer A is considered to be faster compared to multiplication operation $A.B$. This is due to the symmetries of the squaring operation. However, the *2w-bit* terms of the form $a_x.a_y$ appears to be once for $x = y$ and twice for $x \neq y$. In fact, as $a_x.a_y$ and $a_y.a_x$ are similar, the squaring computation is to be accomplished by one-time multiplication and performing left shift in accordance.

In addition to the multiplication, the Comba Algorithm could be also used for squaring. This is because the Comba squaring is structured with a nested loop. The nested loop in Comba squaring is to be only iterated by ${s^2}/_2$ compared to $(s^2+s)/2$ in single precision multiplication. In which, the operation in inner loop could be presented by the following form:

$$(t, U, V) \leftarrow (t, U, V) + 2(a_j + a_{i-j})$$

Modular Inversion:

The process of finding the inversion in a prime field can be done using a direct exponentiation technique. Thus, if B is an element of prime field $GF(P)$ and C an inverse of field B, then the inverse could be computed using a direct exponentiation of $C = B^{-1} = B^{P-2}$. However, direct exponentiation is considered to be costly, as it involves modular multiplication, modular squaring and modular reduction. Therefore, a binary extended Euclidean Algorithm could be considered as the most effective way of implementing inversion [1]. This is because the only divisions done are by 2 and accordingly processed with a right-shift. The normal process of computing gcd of positive integers $a\, and\, b$ is implemented through a classical Euclidean Algorithm. The algorithm is based on dividing $b\, by\, a$ and obtaining a quotient and a remainder where $b \geq a$. The overall process should satisfy $b = qa + r$ and $0 \leq r \leq a$. However, to achieve this, $gcd(a, b)$ is to be reduced by computing $gcd(r, a)$ until the argument (r, a) is obtained smaller than the argument (a, b) and the process need to be repeated until one of the argument is 0 with a result of $gcd\ (0, d)\ = d$. Therefore, at this point, the algorithm could be terminated, as there are no negative remainders to be reduced. Hence, this method is very efficient, since division steps could be shown at most $2k$ where k is the length of a. The mechanism above could be extended to Euclidean Algorithm 2.7 to find integers $x\, and\, y$ in which $ax + by = d$ where $d = gcd\ (a, b)$.

However, finding a modular inversion using Extended Euclidean could be achieved by slightly modifying Algorithm 2.7. For that, let P prime, $a \in [1, P-1]$ and Algorithm 2.8 is processed with input (a, P). Thus, the last none zero remainder r encountered in step 3.1 is $r = 1$. In which, the integers $u_1, x\ and\ y_1$ is to be updated in step 3.2 where $ax_1 + Py_1 = u$ with $u = 1$. These results $ax_1 \equiv 1\ (mod\ P)\ and\ a^{-1} = x_1 mod\ P$ considering that $y_1\ and\ y_2$ are not required for determining r_1.

Algorithm 2.7 Extended Euclidean Algorithm for Integers

```
INPUT: INPUT: Positive Integers $a$ and $b$ with $a \leq b$.
OUTPUT: $d = \gcd(a,b)$ and integers $x, y$ satisfying $ax + by = d$
    1) $u \leftarrow a, v \leftarrow b$

    2) $x_1 \leftarrow 1, y_1 \leftarrow 0, x_2 \leftarrow 0, y_2 \leftarrow 1$

    3) While u $\neq$ 0 do
            3.1 $q \leftarrow \lfloor v/u \rfloor, r \leftarrow v - qu, x \leftarrow x_2 - qx_1, y \leftarrow y_2 - qy_1$
            3.2 $v \leftarrow u, u \leftarrow r, x_2 \leftarrow x_1, x_1 \leftarrow x, y_2 \leftarrow y_1, y_1 \leftarrow y$

    4)$d \leftarrow v, x \leftarrow x_2, y \leftarrow y_2$
    5) Return $(d, x, y)$
```

Algorithm 2.8 F_P Inversion using Extended Euclidean

```
INPUT: INPUT: Prime $P\ and\ a \in [1, P-1]$
OUTPUT: $a^{-1}\ mod\ P$.
    1) $u \leftarrow a, v \leftarrow p$

    2) $x_1 \leftarrow 1, y_1 \leftarrow 0, x_2 \leftarrow 0$

    3) While u $\neq$ 0 do
            3.1 $q \leftarrow \lfloor v/u \rfloor, r \leftarrow v - qu, x \leftarrow x_2 - qx_1$
            3.2 $v \leftarrow u, u \leftarrow r, x_2 \leftarrow x_1, x_1 \leftarrow x$

    4) Return $(x_1 mod\ P)$
```

2.5.5 Elliptic Curve Arithmetic over $GF(2^m)$

The general form of Elliptic Curve over binary field $E(2^m)$ is to be presented by the following equation:

$$y^2 + xy = x^3 + ax^2 + b \tag{2.1}$$

where a, b are parameters $\in GF(2^m)$, $b \neq 0$ and $\{(X_i, Y_i), \text{for } X_i, Y_i \in GF(2^m)\}$ are set of solutions for equation (2.1). However, the number of the points in $GF(2^m)$ are denoted by $\#E(2^m)$, whereas the addition inverse point $R_i(X_r, Y_r)$ of $E(2^m)$ is defined as $-R(X_r, X_r - Y_r)$ and the elliptic curve $E(2^m)$ points from addition group are normally satisfying closure, identity and inverse properties. [1] proposed the below set of rules for defending the operations of Elliptic Curve over $GF(2^m)$

Point Addition Rule:

The result of adding two points, P and Q, where $P\colon (X_p, Y_p) \in (2^m)$, $Q\colon (X_q, Y_q) \in (2^m)$, $X_p \neq X_q$ and the coordinates of R is (X_r, Y_r) is given by

$$X_r = s^2 + s + X_p + X_q + a, Y_r = s(X_p + X_r) + Y_p + X_r,\ where\ s = \frac{Y_p + Y_q}{X_p + X_q} \tag{2.2}$$

Point doubling rule:

The result of doubling points P where $P\colon (X_p, Y_p) \in (2^m)$, $X_p \neq 0$ and the coordinates of R is (X_r, Y_r) is given by

$$X_r = s^2 + s + a, Y_r = X_p^2 + (s+1)X_r,\ where\ s = X_p + \frac{Y_p}{X_p} \tag{2.3}$$

Projective coordinate representations

The process of elliptic curve key generation and elliptic curve digital signature contain modular inverse operations. In fact, as mentioned previously, the modular inversion is considered to be too costly in terms of time complexity compared to multiplication computation. Therefore, an alternative way to avoid such cost is to convert the affine coordinates (X, Y) of elliptic curve point to the projective coordinate (X^*, Y^*, Z^*) and to take care of denominator part of the operations with Z^*. The process is to be finalized by returning back the projective coordinates (X^*, Y^*, Z^*) to affine coordinates (X, Y). There are different types of projective coordinates considering the elliptic curve of a non super-singular formula shown:

$$E: y^2 + xy = x^3 + ax^2 + b$$

1. **Standard projective coordinates**

The projective equation of elliptic curve is presented by the following equation:

$$Y^2Z + XYZ = X^3 + aX^2Z + bZ^3$$

where the point of infinity ∞ corresponds to $(0:1:0)$, negative points of $(X:Y:Z)$ is $(X:X+Y:Z)$. The projective point $(X:Y:Z)$ for $Z \neq 0$ corresponds to Affine point $({}^{X}/_{Z}, {}^{Y}/_{Z})$ where $c = 1\ and\ d = 1$.

2. **Jacobian Projective Coordinates**

In this type of projective coordinate, the projective point $(X:Y:Z)$ corresponds to Affine points $(X/Z^2, Y/Z^3)$ where, $Z \neq 0, c = 2, d = 3,$ Point at infinty ∞ corresponds to $(1:1:0)$ and negative points of $(X:Y:Z)$is $(X:X+Y:Z)$. Also, the projective equation of elliptic curve is to be presented as below:

$$Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^6$$

3. **Lopez-Dahab(LD) Projective Coordinates**

The projective equation of elliptic curve is presented by the following equation:

$$Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4$$

where the point of infinity ∞ corresponds to $(1:0:0)$, negative points of $(X:Y:Z)$ is $(X:X+Y:Z)$. The projective point $(X:Y:Z)$ for $Z \neq 0$ corresponds to Affine point $({}^{X}/_{Z}, {}^{Y}/_{Z^2})$ where $c = 1\ and\ d = 2$.

2.5.6 Field Arithmetic over $GF(2^m)$

In practice, there is no practical use of implementing elliptic curve over the real numbers. This is due to the computational limitation and constraints. Therefore, in this subsection, we discuss the mechanism and related computation arithmetic of implementing elliptic curve over binary field $GF(2^m)$, where the order of elliptic curve can be defined up to $m - bit$. Until recently, most of the applications, such as ECDSA over $GF(2^m)$, defines m to be equal or greater than 163 bits. Thus, most of the ECDSA operations over $GF(2^m)$ involve *m-bit* integers. In the other words, the size of elliptic curve coefficients, points and elliptic curve are all *m-bit* numbers. However, performing elliptic curve over binary field $GF(2^m)$ requires many binary arithmetic functions that include modular reduction, addition, multiplication, squaring and inverse.

The practical implementation of the arithmetic operations on embedded processors normally works using *4-bit*, *8-bit*,*16-bit* and *32-bits* words. In this, we normally do not perform *m-bit* arithmetic bit by bit as it is time consuming. Taking into account that, we can handle $GF(2^m)$ field elements most of the time as *32-bit* words. Thus, we can present the 32-bits word elements as $X = (x[n-1], \cdots, x[2], x[1], x[0])$ of $GF(2^m)$, where $n = [m/32]$ and the right most bit $x[0]$ is LSB bit of the *m-bit* field elements. Whereas, the Left most $t = (32n - m)$ bit of $x[n-1]$ are not used and to be set to zero. For example, if $m = 163$ then $n = 6$ words, which can be presented as $(x[5], x[4], x[3], x[2], x[1], x[0])$ with left-most $t = 29$ bits that to be set to zero in $x[5]$.

Modular Reduction with $\mathbf{f(x)}$:

A modulus computation for $f(x)$ based on the output of $GF(2^m)$ can be achieved if $f(x) = x^m + r(x)$ is irreducible binary (Primitives) polynomial of degree of m and if the elements of degree m and if the elements of $GF(2^m)$ is also generated using primitive polynomial $f(x)$, where the elements of $GF(2^m)$ of the degree at most $m-1$. Additionally, the $GF(2^m)$ field elements is an $m - bit$ member, which can be presented in polynomial form as $a(x) = a_{m-1}x^{m-1} + \cdots + a_2x^2 + a_1x + a_0$ or $A = [a_{m-1}, a_{m-x}, \cdots, a_2, a_1, a_0]$ for a vector form representation.

The binary arithmetic for squaring and multiplication with $m-1$ polynomial resulting the output polynomial with a degree of $2m-2$. Therefore, we can compute $Y(x)\ modulo\ f(x)$ if the output of $Y(x)$ is greater than the degree of the primitive polynomial. Having such mechanism, we will be to ensure that the output result $Y(x)$ polynomial is less than m. It is very often that the binary field arithmetic i is to be normally consider true for $x^i = x^{i-m}\, r(x)\big(mod\ f(x)\big)\ if\ i \geq m.$ For example, if we consider $m = 163$ then $2m-2$ degree $Y(x)$. Accordingly, we can use $32 - bit\ word$ by utilizing a 32-bit vector, which could be represented as below:

$$Y = (y[10], y[9] \cdots, y[2], y[1], y[0])$$

Assuming that $f(x)$ is to be trinomial or pent-nominal having middle terms close to each others. Accordingly, we can process a reduction process of $Y(x) modulo\, f(x)$ in very effective way by reducing $32-bit$ at a time. The reduction process of $y[9]$ starts by adding $y[9]$ four times to Y. In detail, the process is to be accomplished by using 0th of LSB belongs to $y[9]$ and added to bit 132,131,128 and 125 of Y. Then, we can add the first LSB of $y[9]$ to bit 133,129 and 126 of $y[9]$ and so on. For example, if $f(x) = x^{163} + x^7 + x^6 + x^3$, then the computation for the modular reduction for $y[9]$ should start from bit 288 to 319 of Y as shown below:

$$x^{288} = x^{132} + x^{131} + x^{128} + x^{125}\ (mod\ f(x))$$

$$x^{289} = x^{133} + x^{132} + x^{129} + x^{126}\ (mod\ f(x))$$

$$\dots$$

$$x^{318} = x^{162} + x^{161} + x^{158} + x^{155}\ (mod\ f(x))$$

$$x^{319} = x^{163} + x^{162} + x^{159} + x^{156}\ (mod\ f(x))$$

For further detail about modular reduction over binary field, that related can be found in Chapter 4 (*Section 4.4.2)* and [1] .

Finite Field Multiplication over Binary Field:

Algorithm 2.9 Pencil and Paper Polynomial Multiplication

```
INPUT: binary Polynomial $a(x) = (A[t-1] \cdots A[1], A[0]) and\ b(x) = (b[t-1], \cdots B[1], B[0])$ of the degree max $m-1$
OUTPUT: Polynomial product (C[2t-1],$\cdots, C[1], C[0]) = A \otimes B$

    1) $for\ (i = 0\ to\ 2(t-1)$
         C[i]$\leftarrow 0$
        End for
    2) for (i= $0\ to\ t-1)$
            For $(j = 0\ to\ t-1)$
                P,Q$\leftarrow A[i] \otimes B[j]$
                C[i+j]$\leftarrow C[i+j] \oplus P$
                C[i+j+1]$\leftarrow C[i+j+1] \oplus P$
            End for
        End for
```

The third field elements known as $c(z) = a(z).b(z)\ mod\ f(z)$. In fact, conducting finite field multiplication could involves two steps: polynomial multiplication and reduction process of modulo using irreducible polynomial. In fact, there are two benefits of using irreducible polynomial: firstly, it simplifies a reduction process, and secondly, it can help to fewer nonzero especially with spare irreducible. However, many algorithms have been proposed by researchers to help implement binary field multiplication, which include: Pencil and Paper Polynomial Multiplication algorithm, Karatusba-Ofman Algorithm, Montgomery Algorithm and Comba Multiplication Algorithm.

The concept of Pencil and Paper Multiplication illustrated in Algorithm 2.9 is basically based on modifying the Shift-and-add multiplication Algorithm proposed by [34]. The method of Pencil and Paper Polynomial Multiplication is denoted by $\otimes$. However, the principle is normally conducted by multiplying individually the word $A[i]$ and $B[j]$ where $a(x) = (A[t-1], \cdots, A[1], A[0])$ and $B(x) = (B[t-1], \cdots, B[1], B[0])$, which accordingly result in two words output. The process of obtaining the word level polynomial product involves the following:

- Scanning the coefficients of $B[j]$ from b_{w-1} to b_0
- Summing the partial product $A[i]b_k$ to the running sum

However, the partial product could be either 0 if $b_k = 0$ or multiplicand $A[i]$ if $b_k = 1$. Accordingly, after each partial addition the product is to be multiplied by x (just one bit left shifting) to make the necessary alignment for the next partial product.

Alternatively the Karatusba- Ofman Algorithm [35] proposed a recursive divide-and conquer approach to multiply two polynomials plus reducing the number of single precision multiplication. This mechanism works on replacing the multiplication operations with many additions operations – due to the fact that addition operation can be accomplished and processed faster on microprocessor compare to the multiplication operations.

The procedure for multiplying $a(x)$ and $b(x)$ of degree at most $m - 1$ using Karatusba- Ofman Algorithm mainly consists of the two steps. The first step is to split $a(x)$ and $b(x)$ into two polynomials of degree at most $(\frac{m}{2}) - 1$. However, in case m is odd, then the polynomials are to be pretended with zeros. Thus, $\mathrm{A(x) = A_1(x)X + A_0(x) - B(x) = B_1(x)X + B_0(x), where\ X = x^{m/2}}$. Accordingly, $a(x).b(x) = A_1B_1X^2 + [(A_1 + A_0)(B_1 + B_0) + A_1B_1 + A_0B_0]X + A_0B_0$ and X products is to be derived from three products of degree $(m/2 - 1)$ per the following steps:

- $m - bit$ multiplication is performed by two $m/2$ bit multiplication.
- one $(m/2 + 1) - bit$ multiplication responsible to handle the output of sum term
- Several multi-precision addition

Accordingly, the results, of $\frac{m}{2} and\ m/2 + 1$ could be recalculated again using Karatusba Ofman Algorithm which could lead to have a recursive multiplication algorithm. However, the practical implementation shows that the number of used recursion levels will be finally dictated by the amount of overhead associated with algorithm implementation. Also, it will be relatively dictated by the performance of the multiplication and addition process. For instance, applying 192-bit binary polynomials with a 32-word-length processor leads to the following recursions [29]:

$$192 \to 96 + 96 \to 32,32,32 + 32,32,32\ or$$

$$192 \to 64 + 64 + 64 \to 32,32 + 32,32 + 32,32$$

The initial attempt of computing $a.b$ in $GF(2^m)$ was proposed by Koc and Acar and proposed in [36]. Their proposal was based on computing $a.b.r^{-1}$ in $GF(2^m)$, where r

is to be considered special fixed elements of $GF(2^m)$. Accordingly, Montgomery in [37] proposed modular multiplication of integers. The recent software implementation of Montgomery proposal shows its capabilities to enhance the overall performance of integer multiplication. However, such proposal is highly dependent on the selection of $r(x) = x^m$, where r is the element of the field denoted by $r(x) mod\, f(x)$ $i.e$ if $f = (f_{m-1}, f_m, \cdots, f_1, f_0)$ then $r = (f_m, \cdots f_1, f_0)$. Additionally, to implement Montgomery multiplication, it necessary that $r(x) and\, f(x)$ are to be relatively prime $i.e$ $gcd\ gcd\ r(x).f(x) = 1$ in which $f(x)$ should not be divisible by x. Whereas, the $f(x)$ is an irreducible polynomial over $GF(2)$ as well as $r(x)$ and $f(x)$ are relatively primes. This resulting two polynomials known as $r^{-1}(x)$ and $f'(x)$ with the propriety:

$$r(x)r^{-1}(x) + f'(x)f(x) = 1$$

where r^{-1} is the inverse of $r(x)$ modulus $f(x)$.

The Extended Euclidean algorithm could be used to compute the $r^{-1}(x)$ and $f'(x)$ polynomial. In order to compute the word level of Montgomery products, it is required to calculate the w-length of f0(x) rather than computing the entire polynomial f(x) which is normally known as the length of $k = tw$. It is worth mentioning here that the efficiency of the inversion algorithm is based on observing the polynomial of f0(x),in which its inverse should satisfy $f_0(x)f'(x) = 1\ mod\ x_i$ for $i = 1,2,3 \cdots w$.

On the other hand, Comba [33] proposed accelerating the multiplication by reducing the number of extended references during the time of execution. The proposed idea is based on eliminating the write-back operation just by changing the order of partial product generation/accumulation. In which, each result is to be computed in its entirety and sequence. This operation is to be carried out with least significant word-only as well as values of $A[i]$ and $B[j]$ to be read from memory. Further improvement of comba could help significantly the storage overhead. All polynomials $u(x)$ could be obtained by computing $u(x)b(x)$ of degree less than w (Window Length). However, further detail about Comba algorithm is discussed in Chapter 4.

2.5.7 Group Law

It is very essential to define an algebraic structure for a cryptosystem that is based on elliptic curve E over the filed K. The purpose of the algebraic structure is to explain the arithmetic rules and the relationship of the points on a selected curve as well as defining the identity, zero and inverse elements. However, as we described the Elliptic Curve earlier in (section 2.5) the idea of the elliptic curve is based on having two points in the elliptic curve, and accordingly, we can produce other points. Therefore, in this section, we provide a detailed description of the group laws for the elliptic curve over the prime field and binary field.

Elliptic over Prime field F_p Group Laws:

In general, an elliptic curve E over prime field K with characteristic $K \neq 2,3$ is a set of solutions that usually satisfy the following simplified Weierstrass equation:

$$E/K: y^2 = x^3 + ax + b$$

where $a, b \in K$ and $4a^3 + 27b^2 \neq 0$ combined with point at infinity ∞. In which, the group laws of an elliptic curve over the prime field as shown below[1]:

Identity

$P + \infty = \infty + P = P \; for \; all \; P \in E(K)$

Negatives

If $P = (x, y) \in E(K)$ then $P + Q = \infty$ where the point $Q = (x, -y) \in E(K)$ which is known as negative of P and denoted by $-P$. Note that, $-\infty = \infty$.

Point Addition

Let $P = (x_1, y_1) \in E(K)$ and $Q = (x_2, y_2) \in E(K)$, where $P \neq \pm Q$. Then $P + Q = (y_3, x_3)$, where

$$x_3 = (\frac{y_2 - y_1}{x_2 - x_1})^2 - x_1 - x_2 \text{ and } y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1.$$

Point Doubling

Let $P = (x_1, y_1) \in E(K)$, where $P \neq -P$. Then $2P = (y_3, x_3)$, where

$$x_3 = (\frac{3x_1^2 + a}{2y_1})^2 - 2x_1 \text{ and } y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1.$$

Elliptic over Binary field F_{2^m} Group Laws:

An elliptic curve Eover non-super singular F_{2^m} is the set of solutions to be represented by the following simplified Weierstrass equation[1]:

$$E/F_{2^m}: y^2 + xy = x^3 + ax^2 + b$$

Identity

$P + \infty = \infty + P = P \; for \; all \; P \in E(F_{2^m})$

Negatives

If $P = (x, y) \in E(F_{2^m})$ then $P + Q = \infty$ where the point $Q = (x, x + y) \in E(F_{2^m})$ which is known as negative of P and denoted by $-P$. Note that, $-\infty = \infty$.

Point Addition

Let $P = (x_1, y_1) \in E(F_{2^m})$ and $Q = (x_2, y_2) \in E(F_{2^m})$, where $P \neq \pm Q$. Then $P + Q = (y_3, x_3)$, where

$\lambda_3 = \lambda^2 + \lambda + x_1 + x_2 + a$ and $y_3 = \lambda(x_1 + x_3)x_3 + y_1$

with $\lambda = (y_1 + y_2)/(x_1 + x_2)$.

Point Doubling

Let $P = (x_1, y_1) \in E(F_{2^m})$, where $P \neq -P$. Then $2P = (y_3, x_3)$, where

$$x_3 = \lambda^2 + \lambda + a = x_1^2 + \frac{b}{x_1^2} \text{ and } y_3 = x_1^2 + \lambda x_3 + x_3 \text{ with } \lambda = x_1 + y_1/x_1$$

2.5.8 Point Multiplication Algorithms

In this section, we present different methods related to the computation of [k]P, where P is a point in the elliptic curve and k is an integer. Thus, our aim is primarily focused on describing the methods of scalar points multiplication utilized in this thesis. Therefore, for further understanding and information, we refer the reader to [38]. In fact, point multiplication can be considered as a scalar multiplication operation. Its principle is based on conducting a series of point doubling and point addition operations. In which, the Q = kP is to be generated after performing the full series of point addition and point doubling. It is worth mentioning here that the alternative name of point multiplication is scalar point multiplication.

Elliptic Curve Over $GF(P)$ Scalar Multiplication:

Scalar multiplication is considered dominant of the ECC operation, which consumes about 80% of the time spent to execute the ECC operation [4]. A scalar point multiplication could be implemented using various algorithms include: Double and Addition in binary Algorithm, window method, NAF and wNAF Algorithm, sliding window Algorithm and Montgomery Ladder Algorithm.

a) Double and Addition in Binary Algorithm

The idea of this algorithm is based on interpreting k to binary format, then performing the point addition and point doubling, accordingly. However, to process with such an algorithm, we need to conduct point doubling operation for the $'0'$ bit, whereas point doubling and addition need to be conducted for the $'1'$ bit. For example, if $k = 19 = (10011)_2$, then the

Table 2.3 Double and Addition in Binary Algorithm

1	P	Initializing
0	2P	Doubling
0	4P	Doubling
1	9P	Doubling and Addition
1	19P	Doubling and Addition

following point addition and point doubling will be performed as per Table 2.3.

b) Non adjacent Form (NAF) addition-subtraction

The main purpose of this algorithm is to come up with a binary format in which k is not adjacent nonzero bit close to each other. Thus, the binary form of the previous example could be tackled by changing it below, using the NAF algorithm:

$$k = (10011)_2 = (1010-1)_2$$

Table 2.4 NAF with Addition and Subtraction

1	P	Initializing
0	2P	Doubling
1	4P	Doubling and addition
0	9P	Doubling
-1	19P	Doubling and subtraction

a) Montgomery Ladder Algorithm

The initial principle of proposing this concept was to develop an algorithm that was capable of handling the operations of scalar multiplication in a very efficient way for a specific type of elliptic curves. In accordance with that, such a proposal managed to report better performance from the speed point-of-view, just by introducing the concept of computing (X, Z) coordinates of the presented intermediate points. In fact, such an enhancement made the Montgomery ladder allow more involvement of other algorithms, such as the differential addition algorithm. The differential addition algorithm helps calculating the sum of two points so that their difference is well known. Further explanation about Montgomery ladder algorithm and XYCZ-ADDC algorithms and others are given in Chapter 6.

Elliptic Curve Over $GF(2^m)$ Scalar Multiplication:

In this part, we particularly consider scalar multiplication over the binary curve. In which, an elliptic curve is defined using the below Weierstrass equation:

$$E: y^2 + xy = x^3 + ax^2 + b \ with \ a, b \in F2^m$$

where $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ are two points on a curve$E(F2^m)$. Thus, to compute the point addition, the following should be considered:

$$\begin{cases} {}^{x}3 = \lambda^2 + \lambda + x_1 + x_2 + a, \\ y_3 = (x_1 + x_3)\lambda + x_3 + y_1 \end{cases}$$

Where $\lambda = \begin{cases} \frac{y_1+y_2}{x_1+y_2}\ if\ P_1 \neq P_2 \\ \frac{y_1}{x_1} + x_1\ if\ P_1 = P_2 \end{cases}$ From the above, it is very obvious that point addition and point doubling involve of an inversion

Table 2.5 Number of operations for point addition and point doubling[1]

Coordinate system	General addition	General addition (mixed coordinates)	Doubling
Affine	V+M	---	V+M
Standard projective	13M	12M	7M
Jacobian projective	14M	10M	5M
Lòpez-Dahab	14M	8M	4M

From the above, it is very obvious that point addition and point doubling involve of an inversion operation in $GF(2^m)$, which is considered a costly operation. Therefore, it is recommended to use a projective coordinate instead, as it has the capability to perform the curve operation with a bit more field multiplication operations without field operations.

However, there are different types of coordinates system that include: standard projective coordinate system, Lòpez-Dahab projective coordinate system, Jacobian Projective coordinate system and affine coordinate system. While carefully planning to use any one of these, it is suggested to consider the number of field operations, as summarized in the following table:

However, the standard projective coordinates system $P = (X{:}Y{:}Z)$ is to be corresponded to the affine coordinates $(X/Y, Y/Z)$ that should satisfy the curve equation (1), whereas the Lòpez-Dahab projective coordinates $(X{:}Y{:}Z)$ match to the affine coordinates $(X/Y\ , Y/Z^2)$. Computing the scalar point multiplication for a point $P \in E(GF^m)$ and scalar $k \in N$: where $k.P = \overbrace{P + P + \cdots + P}^{K\ times}$ is to be achieved using different methods.

2.6 Elliptic Curve Domain Parameters & Protocols

Elliptic curve domain parameters play a very important role in ECC protocol implementation. Thus, in the coming subsections, we highlighted these parameters based on ECC standards.

2.6.1 Elliptic Curve Domain Parameters

To ensure a proper implementation of elliptic curve cryptography, domain parameters need to be highly considered beside the curve parameter a and b. The purpose of using Elliptic Curve domain parameters are to validate the primitives of $\mathrm{GF}(2^{\mathrm{m}})$ and $\mathrm{GF}(\mathrm{p})$. More description on Elliptic Curve domain parameters and how they are generated can be obtained from[39].

The main domain parameters for Elliptic Curve over prime GF(P) are **p,a,b,G,n** and **h**. These parameters are defined as below:

1. **a** and **b:** These two parameters are responsible for defining the curve $y^2 \, mod\, p = x^3 + ax + b \, mod\, p$
2. **p**: It is used to define the prime for the finite field.
3. **G**: Is the parameter used to generate the points (X_G, Y_G).
4. **n**: The parameter is used while selecting the scalar for multiplication.
5. **h**: It represents the number of points on an elliptic curve where **h** is a cofactor.

On the other hand, the elliptic curve domain parameter over the binary field $\mathrm{GF}(2^m)$ is nominated by $\boldsymbol{m, f(x), a, b, G, n\ and\ h.}$ Each of these parameters are described below:

1. $\boldsymbol{m}$**:** An integer value used to define finite field of $\mathrm{GF}(2^m)$
2. $\boldsymbol{f(x)}$**:** Represents the irreducible polynomial of the $\mathrm{GF}(2^m)$ degree (m)
3. $\boldsymbol{a, b}$**:** Used to define the $\mathrm{GF}(2^m)$ curve $y^2 + xy = x^3 + ax^2 + b$
4. $\boldsymbol{G}$**:** The role of G is to generate the $(\mathrm{G_x}, \mathrm{G_y})$ points in the elliptic curve
5. $\boldsymbol{n}$**:** Used to represent the order of the elliptic curve.
6. $\boldsymbol{h}$**:** Represents the cofactor and $\mathrm{h} = \neq \mathrm{E}(\mathrm{GF}\left(\frac{2^{\mathrm{m})}}{\mathrm{n}}\right). \neq \mathrm{E}(\mathrm{GF}(2^{\mathrm{m}})$.

2.6.2 Elliptic Curve Protocols

In symmetric cryptography, such as RSA and ECC, the number of bit operations is to be powered to K, in which $(\log \mathrm{k} \log^3 \mathrm{q})$ and log notation without base presents a natural logarithm [40]. However, the number of bit operations reflects the number needed to calculate the coordinates of multiple$\mathrm{K.P}$. Accordingly, it is possible to drive efficient public protocols by adopting the ECC [41].

I. Elliptic Curve Key Pair Generation:

The key generation procedure in ECC is defined as outcomes of multiple additions of one or more points in a finite field $GF(q)$ and point $P \in E/GF(q)$ with order n. Consequently, if a user A intends to generate the elliptic curve pairs, he/she should validate

Algorithm 2.10 Elliptic Curve Key Pair Generation

```
INPUT:  Domain Parameters $T = (P, a, b, G, n, h) or (m, f(x), a, b, G, n, h)$
OUTPUT: Elliptic Curve Key Pair $(d, Q) associated\ with\ T$

    1. Randomly selection for Integer $d$ in the interval$[1, n-1]$.
    2. Compute $Q = dG$
    3. Output $(d, Q)$
```

elliptic curve parameters $T = (p, a, b, G, n, h)$ for elliptic curve over $F_P$or $T = (m, f(x), a, b, Gn, h)$ for elliptic curve over F_{2^m} [24, 39].

II. Elliptic Curve Diffie-Hellman Key Exchange (ECDH)

Historically, a principle of Diffie - Hellman key exchange was initially proposed by [42]. Their proposal was aimed to show the possibilities of communicating selected keys over insecure channels. However, [43] develop a detailed recommendation and specification on implementing the Diffie - Hellman key exchange principle based on Discrete Logarithm Cryptography and Problem. In general, the steps of implementing the key exchange start by the domain verification $T = (q, FR, a, b, P, n, h) or\ (m, f(x), a, b, G, n, h)$ to be conducted by sender and receiver. However, all parties involved in the key generation should maintain its key pairs and its domain parameters. The set of domain parameters could be used in different key generation scheme and could be used for a certain time without changing them. For example, consider that Alice and Bob have to conduct a sort of insecure form of communication, in which they are aware that someone is going to be eavesdropping on any message they pass back and forth. Therefore, Diffie -Hellman allows them to use public and private key pairs to pass messages – in a way such that they need to agree on parameters for some elliptic curve so they can pass a secret back and forth by computing points along the curve based on these public parameters. In which, the elliptic curve discrete logarithm gets involved based on the fact that they need to publicly reveal their points. The ECDH is summarized as shown in Algorithm 2.11.

III. Elliptic Curve Digital Signature Algorithm EDSA (Generation)

The process of ECDSA involves different domain parameters that include (q, FR, a, b, P, n, h), where a signed message $m \in E/GF(q)$ – assuming that both A and B

Algorithm 2.11 Elliptic Curve Digital Signature Algorithm (Generation)

```
INPUT: Domain Parameters $(q, FR, a, b, P, n, h), Private\ key\ K_A\ message\ m$
OUTPUT: Signature for the message $m$ with the pair $(s, r)$
    1. $A$ select a random integer $K_A$in the interval [1,n-1]
    2. Compute$K_A, P = (x_1, y_1)$. Then,
    Let $s = x_1$, where $s$ is supposed to be in the interval $[1, n-1], if\ s = 0$ return to step1
    3. Compute $h(m)$ denotes the hash function SHA-1
    4. Compute $(K_A^{-1})\ Mod\ n$.
    5. Compute $r = K_A^{-1}.(h(m) + K_{A^s})Mod\ n$
       If $r = 0\ return\ to\ step\ 1$
    6. Return $(s, r)$
```

Algorithm 2.12 Elliptic Curve Diffie-Hellman Key exchange(ECDH)

```
INPUT: Domain Parameters T= $(P, a, b, G, n, h) or (m, f(x), a, b, G, n, h)$

OUTPUT: A shared secret Key Z

    1. Compute Elliptic Curve Point $P = (x_p, y_p) = d_u Q_v$

    2. Check if $P \neq O$, output "Invalid" and stop

    3. Return Z
```

have similar authentication parameters plus the public key Q_B [24]. Therefore, to perform ECDSA tasks, a user A is required to use his key pair (Q,$K_{A)}$) [1] as shown in Algorithm 2.12.

IV. Elliptic Curve Digital Signature Algorithm EDSA (Verification)

The EDSA verification process is to be achieved by considering the domain parameters (q, FR, a, b, P, n, h), a message m and proposed signature information. Algorithm 2.13 illustrates the process involving the acceptance or rejection for the entire digital signature assigned with the message. However, this process required a user B to validate a user $A's$ signature (s,r) assigned to message m. Accordingly, user B need to accomplish the rest of the algorithm's steps by using the provided authenticated copy of the $A's$ public key Q.

Algorithm 2.13 Elliptic Curve Digital Signature Algorithm (Verification)

```
INPUT:  Domain Parameters $(q, FR, a, b, P, n, h), Signature\ (s,r), message\ m$
OUTPUT: Signature Acceptance  or Rejection

    1. Verify that $(s,r)$ are integers and the interval [1,n-1] else the signature is
       rejected.
    2. Compute $h(m)$.
    3. Compute $u = r^{-1}$ mod n.
    4. Compute $v_1 = u.h(m) mod\ n \& v_2 Q$ and let $w = x_2 mod\ n.$
    5. Compute $(K_A^{-1})\ Mod\ n.$
    6. Compute $(x_2, y_2) = v_1 P + v_2 Q$ and let $w = x_2\ mod\ n.$
    7. If $w = s$ the signature is verified else rejected
```

V. Elliptic Curve Analogue of ElGamal - Encryption

In the assumption that both users A and B have already communicated their key using public key using Algorithm 2.11, user A could encrypt the required message $m \in E/GF(q)$, where a given point $P \in E/GF(q)$ with n order. Therefore, a message m needs to be depicted as points in E. Particularly, this process is to be carried out by integrating a message m as a point in a curve. However, a full detail of such steps is provided in [40]. Thus, the user A would encrypt the points M just by including it to K_A, Q, where K_A is an integer which should be arbitrarily selected and Q is the public key of the user B. After that, the originated message could be sent by user A to B a ciphered text containing the pair of points (C_1, C_2).

Algorithm 2.14 Elliptic Curve ElGamal Analogue Encryption

```
INPUT:  Domain Parameters $(q, FR, a, b, P, n, h), Public\ key\ Q\ message\ m$
OUTPUT: Cipher text of message $m(C_1, C_2)$

    1. Represent the message $m$ as a point $M$ in a curve $E/GF(q)$
    2. A selects a random integer $K_A$ in the interval [1,n-1].
    3. Compute $C_1 = K_A.P.$
    4. Compute $C_2 = (M + K_A.Q)$
    5. Return $(C_1, C_2)$.
```

VI. Elliptic Curve Analogue of ElGamal - Decryption

Algorithm 2.15 shows the process of restoring the plaintext message, which requires user B to use the same domain parameters shown in Algorithm 2.14. In addition to that, user B needs

Algorithm 2.15 Elliptic Curve ElGamal Analogue Encryption

```
INPUT: Domain Parameters
$(q, FR, a, b, P, n, h), Privatekey\ K_B Cipher\ text\ \ message\ m(C_{1,}C_2)$
OUTPUT: Message $m$
    1. Compute $d_1 = C_2 - K_b . C_1$
    2. Compute $M = C_2 - K_B . C_1$
    3. Extract $m$ from M
    4. Return $m$
```

to use his private key inline with the cipher text $(\mathrm{C}_1, \mathrm{C}_2)$in which, B needs to multiply the first point with his private key and deduce the obtained result from the second point until retrieving the plain text.

2.7 Conclusions

Throughout this chapter, we provided the background details associated with cryptography, such as its history and the differences between symmetric and asymmetric types of cryptography. Accordingly, we introduced fundamental information related to Elliptic Curve Cryptography such as Finite Field algorithms, point addition, point doubling and ECC protocols. However, the main focus of this chapter was to provide preliminary information relevant to work in this thesis.

Chapter 3 Software Design: ECC Implementation on 8-bit & 32-bit Single Core Microcontroller

In this chapter, a detailed description for implementing ECC on an 8-bit microcontroller and 32-bit microcontroller using Relic toolkit is provided. Throughout such implementation, we managed to provide the users of this tool in such constrained devices a best level of obtaining an optimal and efficient performance. Knowing that a relic tool provides a wide range of algorithms related to finite field arithmetic, point addition, point doubling, point multiplication and protocols; in fact, getting them combined during a project compilation process could help to achieve better performance as it has been proofed in this work.

3.1 Introduction

The recent and expected proliferation of wireless sensor networks (WSN) with all its economical and societal benefits across a range of applications spanning healthcare, home, environment, and defense will face serious limitations if security concerns are not addressed. Cryptography plays a very important role in achieving security.

Elliptic Curve Cryptography (ECC) is increasingly becoming the first choice for public key cryptography implementation, as it requires much shorter key sizes compared to the RSA for the same level of security. The implementation of ECC on sensor node platforms remains a challenge due to the resources limitation in these nodes. Therefore, optimal low resource ECC implementations are required with optimization techniques to speed up the ECC operations and to reduce the memory usage without prohibitive complexity.

The Relic-toolkit developed by the scholars in [16] is an attractive platform for providing security in WSN. It has many features in comparison to the other ECC open sources libraries, such as those in [44-46]. And it supports many modern cryptographic functions and protocols, such as ECDSA, ECDH, RSA and ECMQV.

Experimental analysis and evaluation for Elliptic Curve Digital Signature (ECDSA) on both an 8-bit and a 32-bit platform (Arduino mega2560 and Arduino Due) has been carried out and comparative implementation results are given. To our knowledge, no such analysis and results have been reported to date.

The implementation results obtained, show that ECDSA key generation on Arduino Due can be achieved in (90ms) compared to (263ms) on the Arduino Mega for m=163. Furthermore, implementation optimisation (such as multi-precision GF(2m) arithmetic) configurations are shown to enhance the performance of the ECDSA on the Arduino Due to (83 ms). These results will act as a useful benchmark and guidance in selection of the optimization techniques provided by the tool for a given WSN application.

This chapter is organized as follows: Section 3.2 provides the related work of software implementation on microcontroller and ECC background. The third section illustrates the arduino mega2560 and arduino Due architectures. The efficient implementation and optimizations provided by the relic-toolkit are presented in section 3.4. Our proposed optimization using the relic code is discussed in section 3.5. Accordingly, our implementation work is described and results analyses are illustrated in section 3.5. Finally, we conclude this discussion in section 3.6.

3.2 Background

In 1985, both Neal Koblitz and Victor S. Miler independently proposed Elliptic Curve Cryptography which is based on Elliptic Curve theories. Currently, ECC is considered to be one of the main players for implementing security in different applications. Basically, ECC has better features and a better future for cryptography since it has the capability to provide many cryptography schemes, such as key management, digital signature and verification. Beside these services and its powerful security, ECC has more powerful computation with shorter key length sizes compared to other public key cryptography solutions, such as RSA and Diffie-Hellman. ECC could be defined over prime fields and binary fields. However, for the purpose of this work, we consider Elliptic Curve over binary fields. The equation below represents the elliptic curve over binary fields:

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

where $b \neq 0$ and the value of x, y, a and b are polynomials representing $n - bit$ words. Finding points on the curve could be achieved by using generator for polynomials and irreducible polynomial. The rules for points addition in $GF(2^m)$ is different from $GF(p)$. Therefore, if $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ and $Q \neq P$, then can be determined as shown below:

$$\lambda = \frac{y_2 + y_1}{x_2 + x_1}$$

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$$

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1$$

and if $Q = P$ then $R = P + P$ or $R = 2P$ as below:

$$\lambda = x_1 + y_1/x_1$$

$$x_3 = \lambda^2 + \lambda + a$$

$$y_3 = x_2 + (\lambda + 1)x_3$$

On the other hand, the point doubling $2P$ can be found as below:

Let $P_1 = (x_1, y_1) \in E(GF(2^m)$ where $P_1 \neq -P$ and $2P = (x_3, y_3)$ then,

$$x_3 = x_1^2 + {}^{b}\!/_{x_1^2} \;\&\; y_3 = x_1^2 + \lambda x_3 + x_3$$

where $\left[{}^{y_2 + y_1}\!/_{x_2 + x_1}\right] P_1 \neq P_2 \,\&\, \lambda = xx_1 + y_1/x_1P_1 = P_2$

Elliptic Curve Digital Signature (ECDSA) is used for digital signature purposes consisting of three main procedures: key pair generation, signature generation and signature verification. The Elliptic Curve Diffie Hellman (ECDH) protocol is used for exchanging the keys between two parties over an insecure channel. The purpose for having the ECC schemes is to provide a high level of security with smaller key sizes. Therefore, it is important for both parties involved in the communication to have pre-defined and agreed domain parameters for each scheme. The detailed specification can be found in Chapter 2, section 2.6.2.

3.3 The Arduino Mega2560 and Arduino Due Architecture

I. Arduino Mega 2560 Architecture:

This type of microcontroller is based on ATmega2560. This microcontroller is designed to support 54 digital input and output pins. Additionally, a 16 MHz crystal oscillator with a USB connection are accommodated in this microcontroller. In general, it contains everything that will allow the end-user to simply plug it in with his computer using USB cable, or power it using an AC-DC adaptor and accordingly getting started. This type of arduino microcontroller consists of 256 KB flash memory that will allow storing the required code in addition to 8 KB of SRAM and 4 KB for EEPROM purposes [47].

II. Arduino Mega Due Architecture:

In fact, this is the first arduino microcontroller that has been equipped with 32 bit ARM core processor. This type of arduino microcontroller is also designed to support 54 digital input/output ports. In which, 12 of them could be used as PWM outputs. However, compared to the arduino mega2560, this microcontroller has much higher oscillator, which can reach up to 84 MHz. Furthermore, it has been also equipped with USB OTG capable connection and JTAG header as well as reset and erase button. Additional information could be found in [47].

3.4 Efficient Method of Improving Relic toolkit on Arduino Devices

The primary objective for the relic-toolkit is to construct an efficient and configurable cryptographic software capable to implement a certain level of security and algorithms. Therefore, we achieved these objectives through different design principles that we considered during the various stages of implementation.

Security: Security: The relic-toolkit is designed to provide cryptography protocols such as RSA, ECDH, ECDSA, ECSS and ECMQV. In addition to that, relic-toolkits support the implementation of ECC over the prime field and ECC over the binary field. This includes different Elliptic Curve parameters recommended by the Standard for Efficient Cryptography Group (SECG), such as Secp160k1, Secp160r1 and Secp160r2 detailed by [39].

Configurability: The principle of configurability is achieved by allowing the user to select the desired components for the targeted platform during the process of developing the library. Furthermore, the desired performance can be achieved by combining and selecting different types of mathematical optimization provided by the tool.

Portability: The relic-toolkit can be used with different types of the wireless sensor platforms, such as ARM, AVR and MSP. Additionally, the library could be built in different types of operating system such as windows (using MingW), Ubuntu and Mac OS. In this work, we consider importing and testing the relic library in Arduino mega260 (AVR- 8-bit processor) and Arduino Due (ARM-cortex-32 bit processor).

Efficiency: In order to accomplish the desired efficiencies from the tool, we decided to implement the ECC over binary fields based on the potential result reported by the end to end security. We also used an assembly version (shown as K163-asm) file in order to achieve better performance as recommended by [48] wherein a new secure and energy-efficient communication model for the Constrained Application Protocol (CoAP), was developed for smart object networks. This model ensured authenticity over a network of multi-hop topology.

Functionality: This principle is ensured through the practical implementation for different public key cryptography schemes provided by the relic-toolkit such as ECDH and ECDSA.

3.4 Proposed Design

In this section, we aim to provide relevant optimization techniques accomplished with the optimization algorithms available in the tool. The details provided in this regard is limited to the optimization techniques used in this thesis.

Optimization for Multiple Precision Arithmetic:

Comba Algorithm: The Comba algorithm is a technique in which the partial products are ordered and scheduled. The multiple precision is required for big number arithmetic. In the multiple precision arithmetic the computations are carried out on the digits whose precision are constrained by the host system memory availability. It is highly efficient for public key cryptography implementations in resolving memory limitations as well as overcoming overflow issues. The contribution of multiple precision on solving such problems is achieved by increasing the integer representation while using single precision data type [49] shows better performance compared to the school book multiplication method. However, the relic-toolkits allow users to select from different types of multiple precision arithmetic algorithms besides the comba algorithm such as school book multiplication, Karatsuba multiplication and others.

Montgomery-Comb Modular Reduction Algorithm:

A modular reduction is a process of finding the reminder of dividing two products:

$$a \equiv b(mod\ c) \text{ where } b \text{ is restricted with range } 0 \leq b < c^2$$

The implementation of the Montgomery modular reduction algorithm involves fewer single multi precision multiplications in comparison with Barrett Modular reduction, which requires two modified multipliers [86]. Previous software implementation of the Montgomery algorithm reported slower speed. This challenge has been tackled and resolved by the researchers through combing the Montgomery modular reduction and comba algorithms. The combination methodology can be achieved by allowing the comba algorithm to act as a multiplier.

Comba Squaring Algorithm:

Multiple Precision Squaring, which affects the overall implementation performance, is a process of multiplying two equal multiplicands. The software implementation for squaring can be performed using multiplication algorithms or using specialized squaring methods. Using specialized squaring helps to reduce the load operand approximately by half over using multiplication algorithms. Additionally, it helps to enhance the computation performance for the duplicated partial products. Moreover, specialized squaring algorithms contribute to overcome the limitation of baseline multiplication algorithms. These limitations can be summarized into two main points. First, the needs for processing single precision shift inside the nested loop. Second, the challenges of performing the products doubling process inside the inner loop.

The comba squaring algorithm could be used to solve these drawbacks. The concept of comba squaring is to some extent similar to the comba multiplication algorithm with some differences that help to accommodate the single precision shifting and doubling processes. The relic toolkit supports three different multiple precision square algorithms besides the comba squaring, which are the Karatsuba Squaring, the recursive karatsuba and the School book method [49]. In this work, we configure the relic library with Comba squaring to obtain better performance.

Optimization for Elliptic Curve Arithmetic

Point Representations: There are different coordinate systems that can be used to represent the elliptic curve, the most popular being the affine coordinates and projective coordinates. The projective coordinates can be considered an option that can help avoid the costly and expensive multiplication and inversion operations. The results reported by [38] show better performance achievement compared to the affine. The relic-toolkit has been designed to support both, and we selected the projective coordinate to achieve a higher performance.

Point Multiplications: Point multiplication or scalar multiplication is implemented through a series of point addition and point doubling operations. The key has to be obtained after conducting a full cycle of addition and doubling operations. The point multiplication over the binary elliptic curve can be implemented with different algorithms such as left-to-right binary algorithm, halving, right-to-left width-w and others. The relic library consists of six different algorithms, such as the basic binary point multiplication algorithms, Lopez-Dahab point multiplication and right-to-left width-w (T)NAF. Since the sliding windows method is more helpful on speeding up the scalar multiplication, we selected the right-to-left width (T) NAF algorithm for performing the point multiplication. The concept of sliding windows is based on scanning a bit at a time and performing the point doubling for them at the same time [44].

Simultaneous Point Multiplications: Enhancement of the efficiencies and speeding up computation of point multiplication has been extensively considered by many researchers due to its significance in some ECC schemes. For example, the implementation of ECDSA required two types of point multiplication, the first for signature generation which is fixed, the second for signature verification process, which is also fixed but unknown. However, the speed of the signature verification process can be increased by using simultaneous multiple-point multiplication [1]. Different methods have been proposed for simultaneous point multiplications such as Shamir's trick, Joint sparse form and interleaving. With this aspect, the relic-toolkits support all of these methods plus the basic simultaneous point multiplication methods that can be selected during the relic building process

3.5 Implementation Results and Analysis

Point Representations:

We imported the relic-toolkit] into the arduino mega 2560 (8-bit AVR processor) [47] and arduino Due (32-bit ARM processor). Our selection for these platforms is based on the fact that we targeted to implement the ECC schemes on a processor that does not require an operating system support. Furthermore, the 8-bit to 32-bit processor range is a representative range for resource embedded applications. We imported relic-0.3.1 onto the two platform boards, and experimented with the performance of ECDSA and ECDH over binary fields using different NIST curve standard (NIST-K163,NIST-B163). In order to obtain better performance, we examined the presets provided by [46]. The execution timings of the codes were measured using inbuilt millis() function provided by Arduino.h library. Furthermore, we measured the amount of RAM using "MemoryFree.h" library beside the avr-size and arm-none-eabi-size tools.

Experiment Setup:

In order to build the library, we installed the avr-gcc version 4.5.3 compiler and cmake cross-platform version 2.8.7. The recommended presets by [16] shown in Figures 4 and 5 in the Appendix were used for building the library with low memory arm-none-eabi-size tools. optimization algorithms and faster time execution respectively, compared to the original recommended presets. For importing the relic-toolkit in arduino Due, we installed arduino extension plug-in (embedxcode) in Xcode IDE MAC OS X version 10.7.3, and then we imported the relic-toolkits into the XCode IDE.

```
CC=avr-gcc CXX=c++ LINK="-mmcu=atmega2560 -Wl,-gc-sections" COMP="-O2 -
ggdb -Wa,-mmcu=atmega2560 -mmcu=atmega2560 -ffunction-sections -fdata-sections"
cmake -DARCH=AVR -DWORD=8 -DOPSYS=NONE -DSEED=LIBC -DSHLIB=OFF
-DSTBIN=ON -DTIMER=NONE -DWITH="DV;BN;FB;EB;EC;CP;MD" -
DBENCH=20 -DTESTS=20 -DCHECK=off -DVERBS=off -DSTRIP=on -DQUIET=on
-DARITH=easy -DFB_POLYN=163 -
DBN_METHD="COMBA;COMBA;MONTY;SLIDE;STEIN;BASIC" -
DFB_METHD="INTEG;INTEG;QUICK;BASIC;BASIC;BASIC;EXGCD;BASIC;BASI
C" -DFB_PRECO=off -DFB_TRINO=off -DBN_PRECI=160 -
DBN_MAGNI=DOUBLE -DEB_PRECO=on -
DEB_METHD="PROJC;RWNAF;LWNAF;INTER" -DEB_MIXED=on -
DEB_KBLTZ=on -DEB_ORDIN=off -DEB_SUPER=off -DEC_METHD="CHAR2" -
DMD_METHD=SHONE ./CMakeLists.txt
```

Figure 3.1 Recommended Arduino Low area Preset

```
CC=avr-gcc CXX=c++ LINK="-mmcu=atmega2560 -Wl,-gc-sections" COMP="-O2 -
ggdb -Wa,-mmcu=atmega2560 -mmcu=atmega2560 -ffunction-sections -fdata-sections"
cmake -DARCH=AVR -DWORD=8 -DOPSYS=NONE -DSEED=LIBC -DSHLIB=OFF
-DSTBIN=ON -DTIMER=NONE -DWITH="DV;BN;FB;EB;EC;CP;MD" -
DBENCH=20 -DTESTS=20 -DCHECK=off -DVERBS=off -DSTRIP=on -DQUIET=on
-DARITH=avr-asm-163 -DFB_POLYN=163 -
DBN_METHD="COMBA;COMBA;MONTY;SLIDE;STEIN;BASIC" -
DFB_METHD="INTEG;INTEG;QUICK;BASIC;BASIC;BASIC;EXGCD;BASIC;BASI
C" -DFB_PRECO=off -DFB_TRINO=off -DBN_PRECI=160 -DBN_MAGNI=DOUBLE
-DEB_PRECO=on -DEB_METHD="PROJC;RWNAF;LWNAF;INTER" -
DEB_MIXED=on -DEB_KBLTZ=on -DEB_ORDIN=off -DEB_SUPER=off -
DEC_METHD="CHAR2"-DMD_METHD=SHONE ./CMakeLists.txt
```

Figure 3.2 Recommended Arduino High Speed Preset

Due to the importance of time and memory usages, we considered evaluating our ECC implementation based on these two factors. The arduino mega2560 is an 8-bit micro-controller, but it has the capability to manipulate 16X16 bit operations by using two separate registers. From the other perspective, the arduino Due is a 32bit micro-controller and can easily handle 8- and 16-bit operations. We measured the execution time using the inbuilt function millis() provided by the ardunio library. This function returns the timing result in milliseconds using the arduino internal timer #0 or TCNT0. However, the timer runs at 16 MHz in arduino mega2560 and at 84 MHz in arduino DUE. On the other hand, we measured the amount of RAM using arm-none-eabi-size tool for arduino DUE, and we used the avr-size tool for measuring the RAM utilization in arduino mega2560.

ECDSA: In this part, we demonstrate the main obtained results with regards the ECDSA performance. Figure3.3 below shows the time execution for ECDSA key generation on both

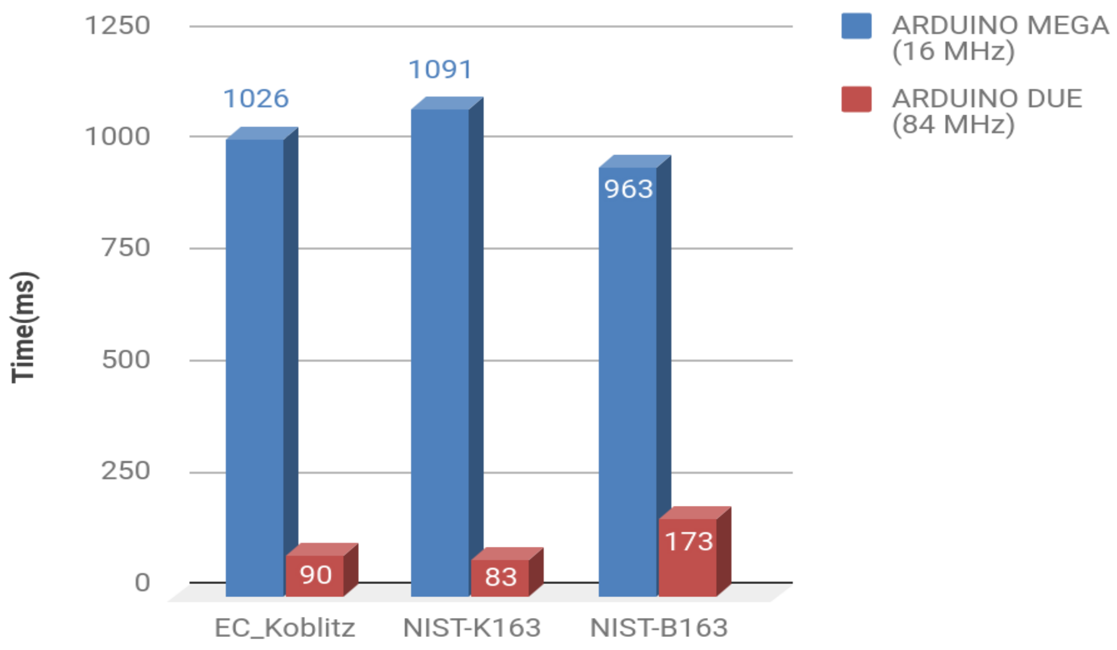


Figure 3.1 Time Execution for EDSA

platforms.

As expected, the arduino mega2560 takes more time to generate the ECDSA keys, as it runs at a much lower clock than the arduino Due The figure 3.4 presents a comparison of the binary field arithmetic with basic and comba algorithms. The binary field arithmetic with BASIC algorithm configuration resulted in an improved performance on the DUE as shown in Figure 3.4.

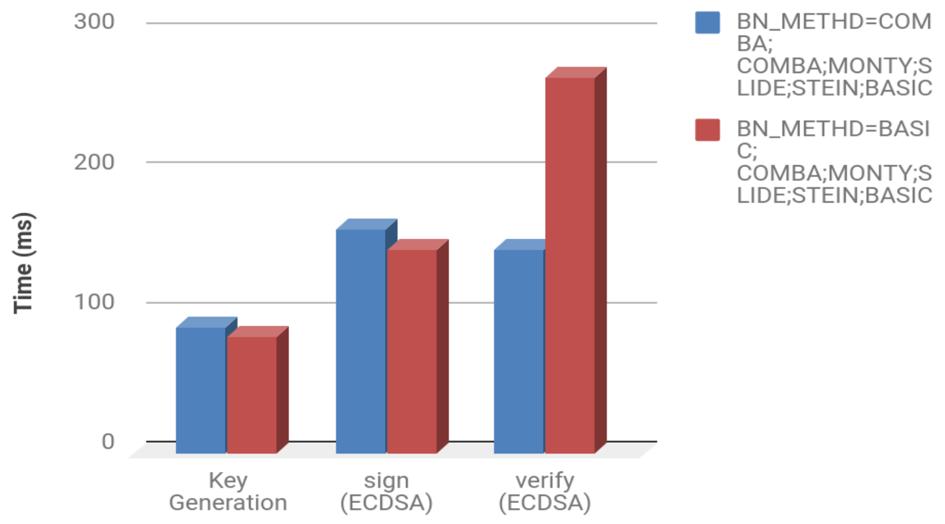


Figure 3.2 Time Execution for EDSA

The performance on the mega2560 was improved using the assembly code provided in the library. This enhancement is represented by the figures which include the time execution improvement and memory usages, respectively. These results show even better performance compared to the results reported by [44] and [45].

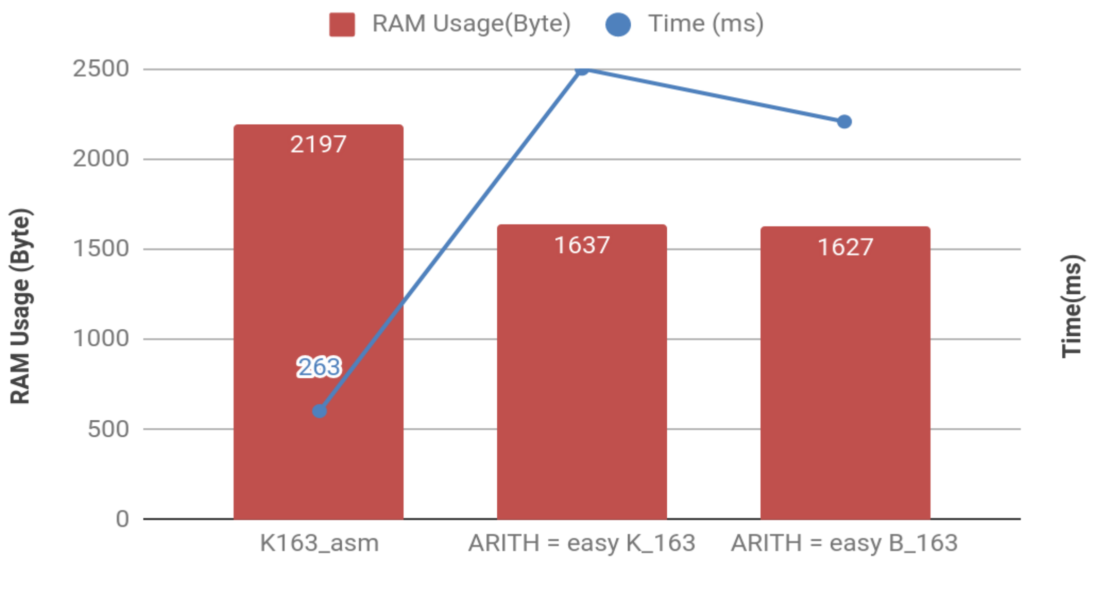


Figure 3.3 Time Execution for EDSA

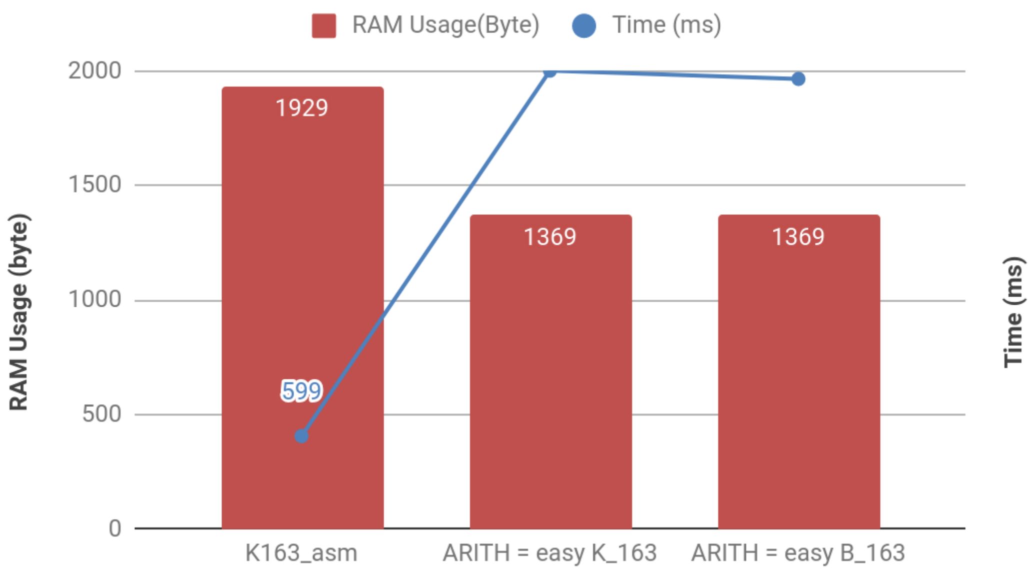


Figure 3.4 Time Execution for EDSA

3.6 Conclusions

In this work, we illustrated the potential of implementing relic-toolkits on sensor node platforms. We also evaluated some of the optimization methods and their effectiveness in the ECDSA implementation performance. The configuration features provided by the relic-toolkit can help enhance the ECC performance, which could be considered as a benchmark and guidance for the developer planning to use the relic in resource constrained processor platforms, such as the ones presented in this thesis.

Chapter 4 Software Design: Efficient Field Arithmetic over $GF(2^{163})$Implementation on A Homogeneous Multicore Microcontroller

Some parts of this work have been published in [50].

Efficient field arithmetic over $GF(2^{163})$ is proposed in this chapter. Thus, our novel proposal here was trying to enhance the performance of Comba algorithm. The reason for such an attempt was to examine the possibility of enhancing its performance using a homogeneous multi core microcontroller. Therefore, we started this chapter by providing and highlighting the importance of multiplication processes in the overall performance of ECC. Then we detailed our proposal of parallelizing the Comba algorithm. After that, we provide the analytical details for the obtained results. In this work, we managed to enhance the Comba Algorithm by about 90%.

4.1 Introduction

The modern technologies of inexpensive constrained devices help to motivate the researchers to use these devices in Wireless Sensor Networks (WSN), such as a hazardous environments, military operations and medical monitoring with high attention of maintaining the necessary security requirement. Recently, Elliptic Curve Cryptography (ECC) proved to be a competitive substitute for standard public key cryptosystems like RSA, DSA and DH. Particularly, it can provide the same level of security provided by RSA with short key size, low processing time and less memory size.

ECC can be implemented based on prime finite field arithmetic $\mathrm{GF(p)}$ or binary finite field arithmetic $\mathrm{GF}(2^{\mathrm{m}})$ where m is prime and its performance highly dependent on the multiplication operation of the finite field arithmetic. Indeed, [51] states that around 80% of the time execution is consumed by the multiplication operation in a software implementation. Therefore, various attempts were conducted to reduce the time execution such as the work done by [52] and [53], where they suggested modified algorithms and provided new multiplication techniques suited to microcontroller platforms that can be used in WSNs. Lately, the work conducted by [54] expressed the benefits of employing multicore embedded platforms in WSNs through energy savings and time execution improvements.

Therefore, this work is an attempt to answer the question of whether it is possible to enhance the software implementation performance of binary finite field multiplication in ECC using homogeneous multicore platforms for resource constrained applications. Even though there are many earlier attempts [51, 53, 55-58] to enhance multi precision multiplication in single core microcontroller. To the best of our knowledge, our work in this paper is the first endeavor aiming at boosting the efficiency of multi precision multiplication using a homogenous multicore microcontroller suitable for low resource environments. We develop a novel parallel software implementation of multi precision multiplication for the comba algorithm suitable for a homogenous multicore implementation. We also propose and deploy a fast algorithm for the reduction operation with word sizes of 8, 16 and 32 bits. Performance is investigated and analyzed on both single and multicore platforms, and the results obtained are presented and compared.

In this chapter, we organized our works as follows. Firstly, we introduced the ECC concept and how multiplication play an importance role in ECC performance in section 4.1. In section 4.2, we discussed the related works that have been conducted to enhance the

Comba multiplication algorithms. Accordingly, we highlighted the XMOS a homogeneous multi core microcontroller in section 4.3. Parallel Comba multiplication on a multicore microcontroller has been discussed in section.4.4. Our implementation results and analysis are detailed in section 4.5. Finally, we conclude our work in section 4.6.

4.2 Related Work

The performance of the polynomial multiplication plays very important role in the overall performance of the ECC. Consequently, having a solid and effective binary polynomial multiplication will result from duplication, squaring and reversal in GF(2 m), and thus can assist in creating a substantial improvement in the entire ECC procedure.

Therefore, many attempts have been made by researchers to enhance the performance of multi-precision multiplication. For instance,[11] suggested a simple architectural improvement using a general-purpose processor core that could assist execute arithmetic operations in GF(2 m) finite binary areas. Their suggestion is based on a recent modification of the MULSC instruction supplied by SPARC V8 Architecture, which was implemented by Lopez and Dahab in the left-to-right comb technique. The authors utilized this technology to describe an increase in the speed of 90 per cent in addition to a remarkable reduction in the use of RAM. It utilizes polynomial bias as well as special polynomials such as trinomials, pentanomial and all one polynomial (AOP) to develop an extensive and careful study of finite field multiplication over GF (2 m). The Montgomery multiplication scheme carries out this multiplication and application of it is also described. It focuses on different arithmetical operation on the elliptic curve cryptography over GF (2m). The parameter performance is also discussed in term of a number of component, latency, space and time complexity.

Michael Hutter and Erich Wenger [53] proposed a new novel multiplication technique to help to increase the performance of multiplication. Their technique is based on the product scanning approach, but it divides the calculation into several rows. In this method, the authors reduced the number of necessary load instructions through caching of operands. The method significantly reduces the number of load instructions required, which is usually one of the most expensive operations on modern processors. I evaluated the new technique on an 8-bit ATmega128 microcontroller and compared the result with existing solutions. The application requires only 2,395 clock cycles for a 160-bit multiplication that exceeds associated job by a factor of 10% to 23%. The amount of load orders required is decreased from 167 (needed to multiply the best-known hybrid) to just 80. Even for larger Integer sizes (required for RSA) and limited register sets, the implementation scales are perfect. It also fully complies with existing multiply-accumulate instructions integrated into most of the processors available. The proposed method was implemented in ATmega128 microcontroller and showed by 23 % compared to the result reported by [51]. The number of load instructions required is usually one of the most expensive operations on modern processors and is reduced by the method. The new technique is evaluated on an 8-bit ATmega128 microcontroller, and the result is compared with existing solutions. There is need of only 2,395 clock cycles for a 160-bit multiplication in the application that exceeds associated job by a factor of 10% to 23%. The amount of load orders required has been reduced from 167 to just 80. The implementation scales are perfect for larger Integer sizes and limited register sets. It also fully complies with existing multiply-accumulate instructions integrated into most of the processors available.

Next, Seo, Hwajeong [58] proposed a novel method nominated as carry-once capable to perform multi-precision multiplication having accumulation of intermediate results. The principal idea of this technique is to optimize the number of addition instructions required for intermediate result update. Through this method, the authors reported better performance of multi-precision multiplication while they implemented 160-bit multiplication over ATmega128.

Z. Liu and J. Großschädl [59] proposed a new software technique for improving the performance of Montgomery modular multiplication on a 8-bit AVR microcontroller. Using assembly language, the authors managed to implement six hybrid Montgomery multiplication algorithms in AVR microcontroller. In fact, the authors take the advantages of the hybrid multiplication and combine it with Montgomery's multiplication to enhance the modular multiplication. Accordingly, they evaluated the performance of the new method for different operands ranging from 160 to 1024 bits.

The work in [5] proposed a new efficient techniques for improving the multiplication, squaring modular reduction and inversion in $GF(2^{163})$ and $GF(2^{233})$ using MICAz Mote microcontroller. In this work, the authors proposed using Karatsuba’s multiplication algorithm to divide the multiplication problem into two sub-problems. These two subproblems are to be manipulated separately. In addition to that, they suggested saving the already shifted results produced in the first phase. This process can help to reload the intermediate result into registers for multi-precision shifting of some of the read memory already released during the first phase.

[60] describes new techniques for parallelizing binary fields in computers equipped with modern vector instruction sets. The authors' detailed methods for implementing field multiplication, squaring and square root, and they present a constant memory lookup-based multiplication mechanism. In this work, the authors implemented the finite field arithmetic as an arithmetic backend of the relic toolkit [16] for testing and benchmarking purposes.

A new record for enhancing the multi-precision multiplication on AVR 8-bit microcontroller has been reported by [56]. In this work, the authors optimized Karatsuba multiplication in AVR 8-bit microcontroller. To help in achieving that, the authors proposed tuning the instruction scheduling in order to minimize the number of live registers that to be used during the Karatsuba multiplication process. Accordingly, the authors managed to obtain new multiplication speed record for multiplying integers between 48 and 256 bits on the ATmega family of microcontrollers.

Speed-up of the arithmetic operation and enhancing its effectiveness in the software implementation of $GF(p)$ is the work proposed by [61]. Their work was mainly focused on increasing the performance of finite field multiplication for 32-bit and 64-bit platform using the Comba algorithm. In this work, the authors suggested implementing carry accumulation by the addition of 32-bit variables in the 64-bit variable accumulator to avoid accounting carry after the addition of variables. However, they proposed to accumulate the carry in the final iteration.

H. Seo, Y. Lee, H. Kim, T. Park, and H. Kim [55] presented further techniques for improving the performance of multi-precision multiplication on an embedded microprocessor. The authors proposed enhancing carry-once method by applying the operand caching methodology and further optimization for multiplication and accumulation (MAC), unbalanced comb and comb-window methods. In this work, the authors managed to optimize the product scanning method by reducing the number of required registers.

However, despite the amount of works that have been conducted to enhance the performance of multi-precision multiplication, we noticed none of the previous works attempted to improve it using the multicore microcontroller. Thus, in the present work, we introduce new methodologies of enhancing the multi-precision multiplication in the multicore microcontroller.

4.3 The Xmos Architecture

xCORE Multicore Microcontroller starter kit is a 32-bit multicore microcontroller capable of providing low latency and timing determinism of the xCORE architecture to different embedded applications. The main advantage of xCORE microcontroller is its capability to execute multiple tasks concurrently as well as the possibilities of conducting the

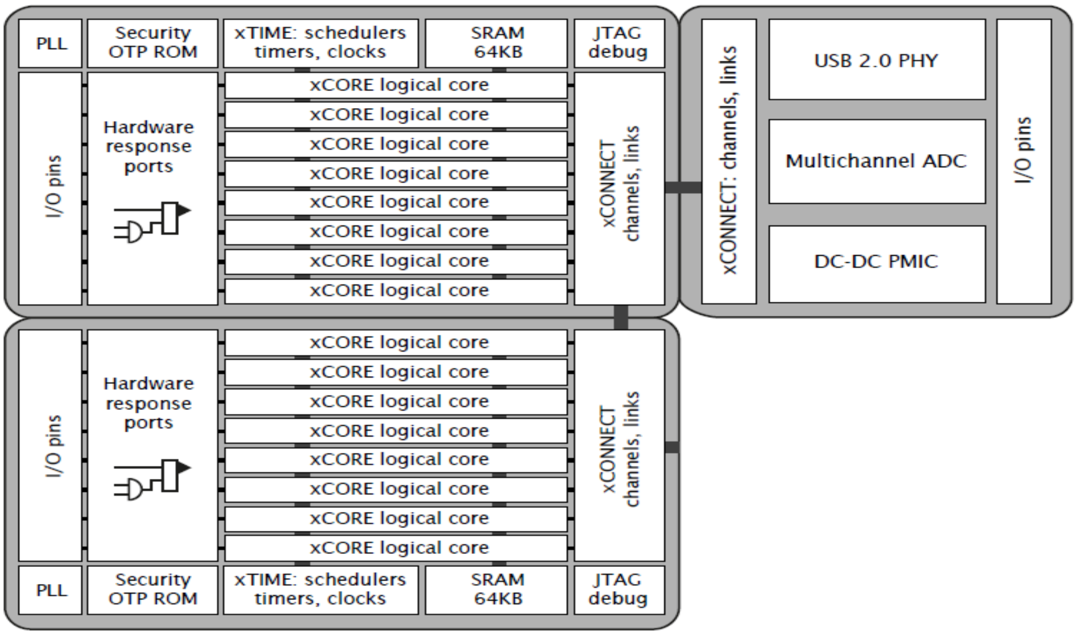


Figure 4.1 XSI -U Series 16 core devices

communication between tasks using a high-speed network

As shown in Figure 4.1, the starting kit xCORE microcontroller is equipped with analog and digital nodes. The digital node consists of xCORE Tile, a switch, and PLL (Phase-Looked-Loop), whereas, the analog node comprises the USB PHY, multi-channel ADC(Analog to Digital Converter), deep sleep memory, an oscillator, a real-time counter and power supply control. To establish the communication between analog and digital node, a necessary link that is capable to switch to the digital node is required.

The system is however programmed using high-level programming language C / C++ and the language XMOS-originated. The XC language is designed to provide extensions to C and to simplify the control over concurrency. Also, it allows the end user to control I/O and timing as well as to use low-level assembler. The xCORE tile is to be considered as a flexible multicore microcontroller component. It consists of the integrated I/O and on-chip memory and multiple logical cores, which can be run simultaneously. In fact, each of the logical cores guaranteed a slice of processing power, can execute computational code and provide a control software and I/O interfaces. The logical cores use channels to exchange data within a tile or across tiles, while the tiles are to be connected using switch network known as xCONNECT. The xCONNECT uses the proprietary of physical layer protocol to add additional resources to a design. Additionally, the I/O pins are determined through intelligent ports, which can help for serializing data, interpret strobe signals, wait for scheduled times or events and make the device ideal for real-time control applications.

Each tile consists of 8 active logical cores, which have a capability to issue the instructions down a shared four-stage pipeline. The instructions generated from active cores are issued using round-robin. However, if up to four logical cores are in use, then each core is allocated with a quarter of the processing cycles. In contrast, activating more than four logical cores results in each core being allocated at least with $1/n$ cycles (for n cores).

Another benefit of using Xmos devices is its capability to work as a scalable architecture, in which the xCORE devices can be connected together allowing the end user to construct one system. Each of xCORE device has an xCONNECT interconnect feature to communicate different tasks that run on the various xCORE tiles on the system.

For further detail about xmos multicore microcontroller, we refer the reader to [14].

4.4 Parallel Comba Multiplication on Multicore Microcontroller

Comba Multiplication is considered as one of the most important multiplication techniques used in public key cryptography computations – be it in modular form in RSA or in finite field form in Elliptic Curve Cryptography, for example. The efficiency of these public key cryptography implementations depends heavily on the efficiency of the implementation of the multiplication operation. Multicore architectures are becoming increasingly important platforms for modern computation. However, cryptography implementations on these platforms is still in its infancy. In this work, we propose a parallel software implementation of the comba multiplication in $GF(2^{163})$ using a homogenous multicore microcontroller.

We obtain performance results and compare these to sequential implementation of comba multiplications with and without modular reduction for different word size 8, 16 and

32 bits on single core microcontrollers. Our obtained results outperform most of the published single core modular multiplication implementations and require much fewer cycles. We achieve more than 85% enhancement of the measured latency in comparison to a single-core implementation.

4.4.1 Finite Field Multiplication

Multi-precision algorithms are important to handle arithmetic operations on general

Algorithm 4.1 Comba's Algorithm over *GF(2^m^)*

```
INPUT: Two $m$ bit polynomials $a(z).b(z) \in GF(2^m)$ and consisting of $t[m/W] - word$
each where $W$ is the word size of the processor.
OUTPUT: $c(z) = a(z).b(z) = (c_{2m-1}, \cdots c_0)$
    1) $s \leftarrow 0$
    2) For $i$ in 0 to $t-1$ do ($i$ denotes column number)
        a. For $j$ in 0 to $i$ do
            i. $S \leftarrow S + (a_j \times b_{i-j})$
           End For;
        b. $c_i \leftarrow S \; mod \; 2^w$ (Partial sum of each column)
        c. $S \leftarrow S/2^w$ (Word is right shifted by $w$ −bits )
           End For;
    3) For $i$ in $t$ to $2t-2$ do
        a. For $j$ in $(i-s+1)$ to$(s-1)$ do
            i. $S \leftarrow S + (a_j \times b_{i-j})$
           End For;
        b. $c_i \leftarrow S \; mod \; 2^w$ (Partial sum of each column)
        c. $S \leftarrow S/2^w$ (Word is right shifted by $w$ −bits)
           End For;
    4) $C_{2d-1} \leftarrow S \; mod 2^w$

    5) Return $C(z)$
```

processors by splitting the operation into smaller blocks. There are many techniques to implement muti-precision multiplication over $GF(2^m)$; these include product scanning [33], hybrid scanning, operand caching, and consecutive operand-caching techniques [11]. Product scanning techniques (known also as Comba) are considered to be the most efficient for large operands. Comba, as illustrated in Algorithm 4.1, is based on two individual outer loops to generate the multiplier operands index and inner loops for generating multiplicand operands index. The multiplicand and multiplier operands are to be produced in column-wise style as explained in Figure 4.2, where t = 4 and the inner loop is iterating 42 or 16 times in total.

Recently, a number of recently research works based on single core microcontroller implementations attempted to improve Comba techniques and to provide a comprehensive explanation for these methodologies on these platforms. In [53], the authors proposed 160-bit multiplication on ATmega128 microcontroller by dividing the product scanning method into individual rows, achieving 23 % fewer clock cycles than [51]. A carry-once method demonstrated in [58] managed to optimize and decrease the number of intermediate product calculation and even led to much better results than [53].

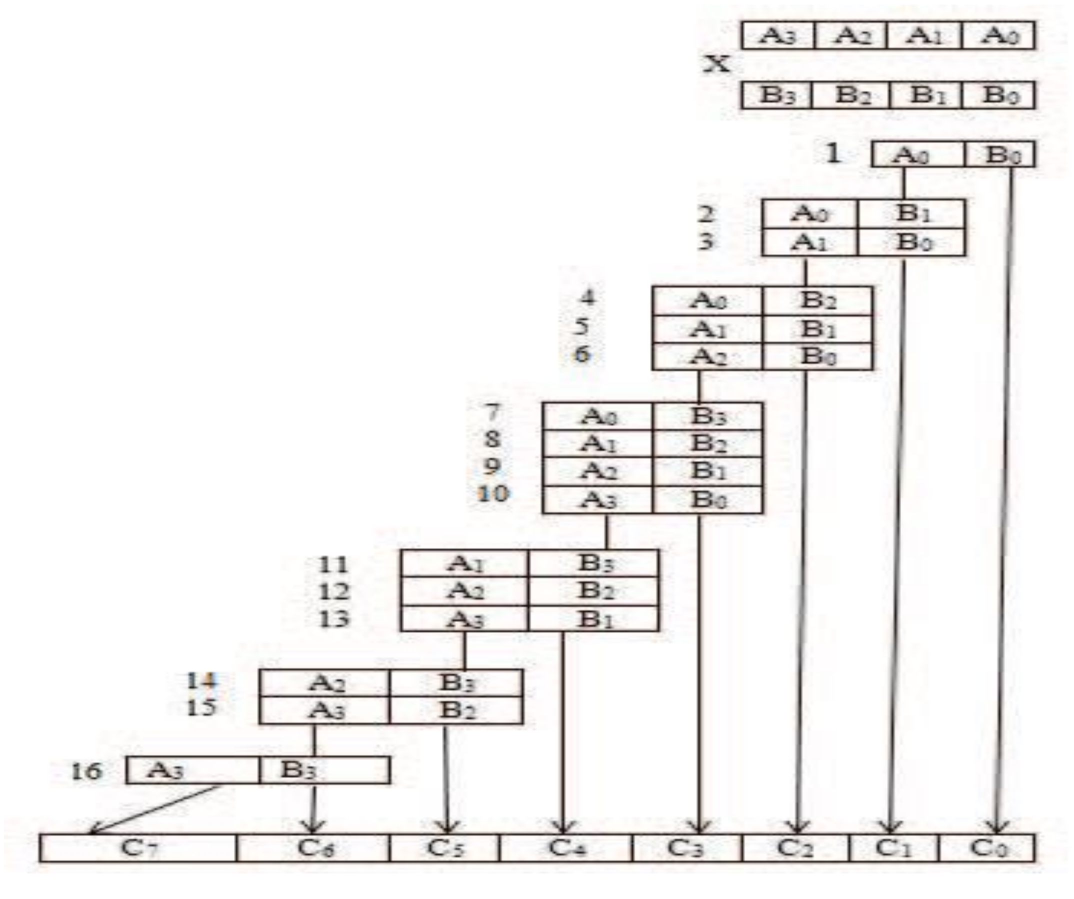


Figure 4.2 Schematic representation $\mathbf{4} \times \mathbf{4}$word multiplication using Comba Algorithm

Recently, Comba algorithms have also been shown to be efficient in enhancing the implementation performance of Fully Homomorphic Encryption Schemes [62] and [63]. In this work, we propose enhancing mutli-precision multiplication using comba on a homogenous multi core microcontroller, which allows to carry out multiple instruction flows concurrently. This could form a basis for further research on implementations on such platforms.

4.4.2 Modular Reduction

A fundamental operation in ECC computations is a modular reduction, which is required in the finite field operations. According to Algorithm 4.2 [64] adopted in this work, it could be observed that the primary operations associated with reduction are XOR and left or right shifts. In order to consider the implementation of Algorithms 1 and 2 on microcontrollers with word sizes of 8, 16 and 32 bits, a necessary adjustment in Algorithm 4.2 was made, as will be explained in the next section.

Algorithm 4.2 Comba's Fast reduction method with $W = 32$ for $GF(2^{163})$ $f(z) = z^{163} + z^7 + z^6 + z^3 + 1$

```
INPUT: A binary polynomial $c(z)$ of degree at most 324
OUTPUT: $c(z) mod\ f(z)$
    1) For $i$ from 10 down to 6 do { Reduce $C[i]z^{324}$ modulo $f(z)$ }
        a. $T \leftarrow C[i]$
        b. $C[i-6] \leftarrow C[i-6] \oplus (T \ll 29)$
        c. $C[i-5] \leftarrow C[i-5] \oplus (T \ll 4) \oplus (T \ll 3) \oplus T \oplus (T \gg 3)$
        d. $C[i-4] \leftarrow C[i-4] \oplus (T \gg 28) \oplus (T \gg 29)$
    2) $T \leftarrow C[5] \gg 3$ { Extract 33-31 of $C[5]$}
    3) $C[0] \leftarrow C[0] \oplus (T \ll 7) \oplus (T \ll 6) \oplus (T \ll 3) \oplus T$
    4) $C[1] \leftarrow C[1] \oplus (T \gg 25) \oplus (T \gg 26)$ {Clear the reduced bits of $C[5]$}
    5) Return $(C[5], C[4], C[3], C[2], C[1], C[0])$
```

4.4.3 Proposed Design

Both functional-parallelism and data-parallelism have been exploited in the design process adopting the Foster design methodology [65]. The Comba algorithm was executed by calling the two parallel tasks at the main loops 2 and 3, as depicted in Algorithm 4.1. For the partitioning step in the sequential Comba algorithm, we divided the two main loop 2 and 3 into two tasks, since there is one common input consisting of two arrays $A(a_{s-1}, \cdots a_1, a_0)$ and $B(b_{s-1}, \cdots, b_1, b_0)$ with word size $w - bit$, where w represents the word size of the processors $(i.e\ w = 16, w = 8\ and\ w = 32)$. However, to overcome the constraints of the internal linkage code due to the carry [5], we used the tasks functions in XC programs that are able to call tasks in parallel on separate logic cores without thinking about the priority and scheduling of the communication between tasks.

4.5 Implementation Results and Analysis

In this work, we used XMOS, a low cost development stratKit which consists of a two-tile XCORE device and eight 32 bit with 500 MIPS xCORE multicore microcontroller [14]. Also, we used Arduino Mega260 (16-bit AVR) processor and Arduino Due (32-bit ARM) processors to compare our parallel multicore approach with a single core approach.

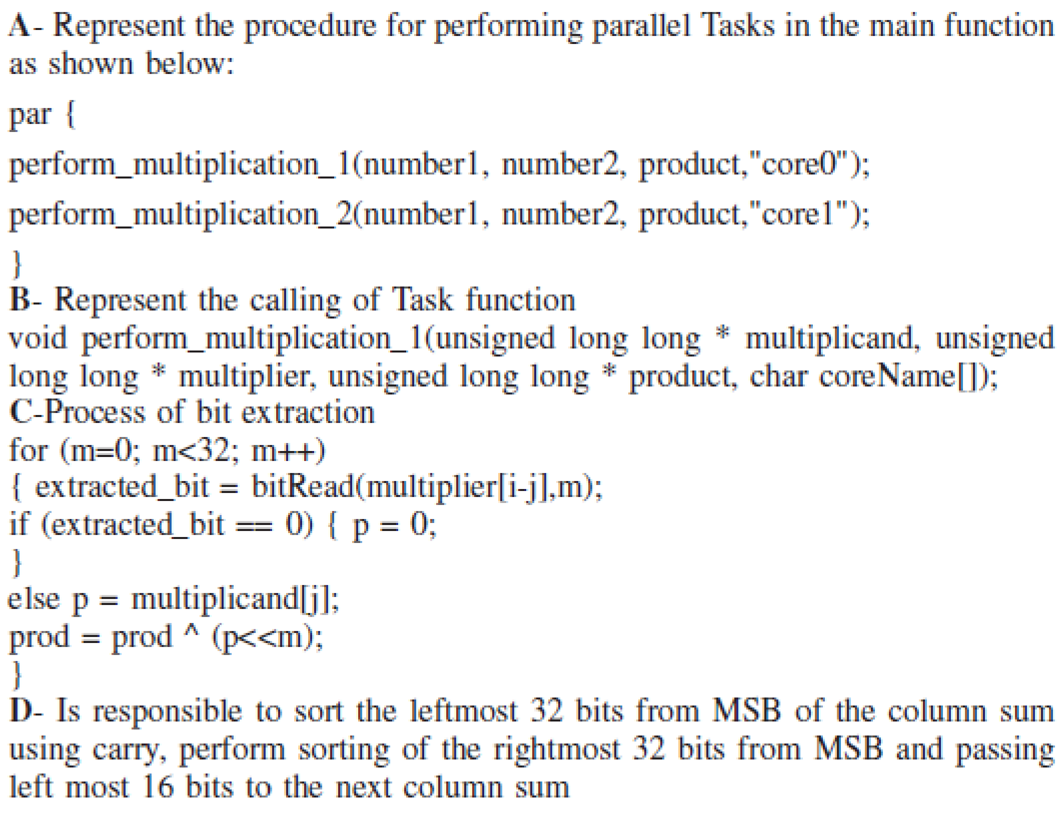
4.5.1 Modified Comba Algorithm - Parallel Multiplication

The implementation of $\mathrm{GF}(2^{163})$ multiplication using the parallel comba algorithm for different word sizes (which include $\mathrm{w} = 16, \mathrm{w} = 8$ and $\mathrm{w} = 32$) is shown in Algorithm 4.3.

Algorithm 4.3 Modified Comba's Algorithm over *GF(2^m^)*

```
INPUT: Two $m$ bit polynomials $a(z).b(z) \in GF(2^m)$ and consisting of $t[^m/_W] - word$
each where $W$ is the word size of the processor.
OUTPUT: $c(z) = a(z).b(z) = (c_{2m-1}, \cdots c_0)$
    1) $s \leftarrow 0$
    2) {Task -1 Function Parallel }
            a. For $i$ in 0 to $t - 1$ do ($i$ denotes column number)
                    i. For $j$ in 0 to $i$ do
                            1. $S \leftarrow S + (a_j \times b_{i-j})$
               End For;
                    ii. $c_i \leftarrow S\ mod\ 2^w$ (Partial sum of each column)
                    iii. $S \leftarrow {}^S/_{2^w}$ (Word is right shifted by $w$ −bits )
    3) {Task -1 Function Parallel }
            a. For $i$ in $t$ to $2t - 2$ do
                    i. For $j$ in $(i - s + 1)$ to$(s - 1)$ do
                            1. $S \leftarrow S + (a_j \times b_{i-j})$
               End For;
                    ii. $c_i \leftarrow S\ mod\ 2^w$ (Partial sum of each column)
                    iii. $S \leftarrow {}^S/_{2^w}$ (Word is right shifted by $w$ −bits)
               End For;
    4) $C_{2d-1} \leftarrow S\ mod 2^w$
    5) Return $C(z)$
```

Accordingly, the multiplication is performed using two outer loop functions and simultaneously called from the main function as shown in Figure 4.3.


A- Represent the procedure for performing parallel Tasks in the main function as shown below:

```
par {
perform_multiplication_1(number1, number2, product,"core0");
perform_multiplication_2(number1, number2, product,"core1");
}
```

B- Represent the calling of Task function

```
void perform_multiplication_1(unsigned long long * multiplicand, unsigned long long * multiplier, unsigned long long * product, char coreName[]);
```

C-Process of bit extraction

```
for (m=0; m<32; m++)
{ extracted_bit = bitRead(multiplier[i-j],m);
if (extracted_bit == 0) { p = 0;
}
else p = multiplicand[j];
prod = prod ^ (p<<m);
}
```

D- Is responsible to sort the leftmost 32 bits from MSB of the column sum using carry, perform sorting of the rightmost 32 bits from MSB and passing left most 16 bits to the next column sum

Figure 4.3 Xtimecompsoer Task-1 Flow Diagram

We applied the 32-bit fast modular multiplication per Algorithm 2, whereas for the 8-bit and 16-bit, we modified algorithm 2, as seen in Algorithm 4.4 and Algorithm 4.5 below, which illustrate fast modular reduction for 8 and 16 bit, respectively. The reduction process started after the multiplication, which is to be based on the 326 bits of the 163 arithmetic multiplication bit result. In this, the 326 is to be divided into $w = 8$ for 8-bit word size, and the calculation of the number of word size is based on $\left({(163 \times 2)}/{8}\right) = 41\, words\ w_0\ to\ w_{40}$. Also, we used the following reduction irreducible polynomial recommended by NIST [64] to execute the reduction process for $GF(2^{163})$:

$$f(z) = z^{163} + z^7 + z^6 + z^3 + 1$$

Algorithm 4.4 Fast reduction Modification using Word size =8

```
INPUT: Binary polynomial $W(z)$ of degree $\leq 324$
OUTPUT: $W(z) mod\ f(z)$ of degree $\leq 163$ where $f(z)$ is irreducible polynomial
    1) $W_{20} \rightarrow W_0 \oplus W_{20} \gg 3 \oplus (W_{20} \gg 3) \ll 3 \oplus (W_{20} \gg 3) \ll 6 \oplus (W_{20} \gg 3) \ll 7 \oplus W_{21} \ll 5$
    2) For $i$ in 1 to 19 (by one ) do
        a. $W_i = W_i \oplus W_{(20+i)} \gg 3 \oplus W_{(20+i)} \oplus W_{(20+i)} \ll 3 \oplus W_{(20+i)} \ll 4 \oplus W_{(21+i)} \ll 5 \oplus W_{(19+i)} \gg 4$
        b. End For;
    3) $W_{20} \rightarrow W_{20} \oplus W_{40} \gg 3 \oplus W_{40} \oplus W_{39} \gg 5 \oplus W_{39} \gg 4$
    4) $W_1 \rightarrow W_1 \oplus W_{40} \oplus W_{40} \gg 2$
    5) $W_0 \rightarrow W_0 \oplus W_{40} \gg 3 \oplus (W_{40} \& 0X7) \oplus (W_{40} \& 0X7) \ll 3 \oplus (W_{40} \& 0X7) \ll 6 \oplus W_{40} \ll 1 \oplus (W_{40} \& 0X7) \ll 4 \oplus W_{39} \gg 7 \oplus (W_{39} \gg) \ll 3 \oplus (W_{39} \gg 7) \ll 6 \oplus (W_{39} \gg 7) \ll 7$
    6) $W_{20} \rightarrow W_{20} \& 0X7$
    7) Return $W_{20}, W_{19}, \cdots, W_1, W_0$
```

For the 8-bit word size, there will be no shifting instruction required at the initial step, and the bits will be copied as it is, and after that, the 3 bits shifting (of bit 163-325) is to be implemented by 6-bit and 7-bit left shift. Following each shift, extra bits moving out of the 325-bit mark due to shifting are replaced back into the starting positions left vacant by shifting.

Then, the extra bit will be serially shifted by 3, 6 and 7 bits. Afterwards, to obtain the 163 reduction in output, all columns starting from bit 0 to bit 163 are to be sequentially added using XOR instruction. Moreover, a similar strategy is used for performing reduction with word size = 16.

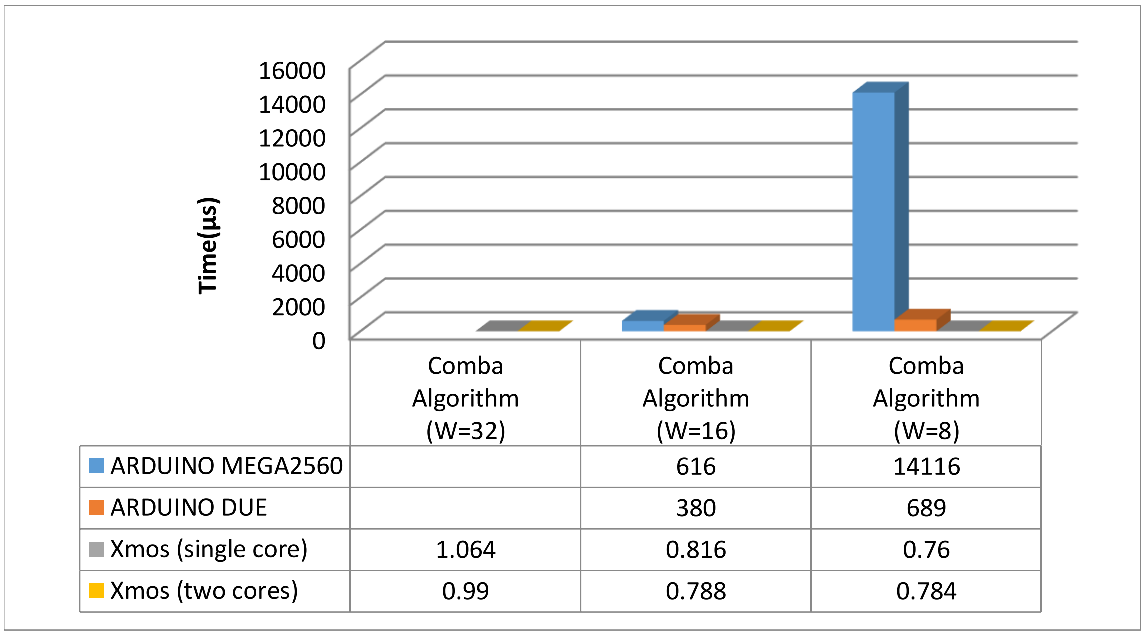
Algorithm 4.5 Fast reduction Modification using Word size =16

```
INPUT: Binary polynomial $W(z)$ of degree $\leq 324$
OUTPUT: $W(z) mod\ f(z)$ of degree $\leq 163$ where $f(z)$ is irreducible polynomial
    8) $W_0 \to W_0 \oplus W_{10} \gg 3 \oplus (W_{10} \gg 3) \ll 3 \oplus (W_{10} \gg 3) \ll 6 \oplus (W_{10} \gg 3) \ll 7 \oplus W_{11} \ll 13$
    9) For $i$ in 1 to 9 (by one ) do
        a. $W_i = W_i \oplus W_{(10+i)} \gg 3 \oplus W_{(10+i)} \oplus W_{(10+i)} \ll 3 \oplus W_{(10+i)} \ll 4 \oplus W_{(11+i)} \ll 4 \oplus W_{(11+i)} \ll 13 \oplus W_{(9+i)} \gg 13 \oplus W_{(9+i)} \gg 12$
        b. End For;
    10) $W_{10} \to W_{10} \oplus W_{20} \gg 3 + W_{20} \oplus W_{19} \gg 13 \oplus W_{19} \gg 12$
    11) $W_0 \to W_0 \oplus W_{20} \gg 3 \oplus (W_{20} \& 0X7) \oplus (W_{20} \& 0X7) \ll 3 \oplus W_{20} \ll 6 \oplus W_{20} \ll 1 \oplus W_{19} \gg 15(W_{20} \& 0X7) \ll 4 \oplus (W_{19} \gg 15) \ll 3 \oplus (W_{19} \gg 15) \ll 6 \oplus (W_{19} \gg 15) \ll 7 \oplus W_{20} \ll 8$
    12) $W_{10} \to W_{10} \& 0X7$
    13) Return $W_{10}, W_9, \cdots, W_1, W_0$
```

For the 8-bit word size, there will be no shifting instruction required at the initial step, and the bits will be copied as it is and after that, the 3 bit shifting (of bit 163-325) is to be implemented by 6-bit and 7-bit left shift. Following each shift, extra bits moving out of the 325-bit mark due to shifting are replaced back into the starting positions left vacant by shifting. Then, the extra bit will be serially shifted by 3, 6 and 7 bits. Afterwards, to obtain the 163 reduced output, all columns starting from bit 0 to bit 163 are to be sequentially added using XOR instruction. Moreover, a similar strategy is used for performing reduction with word size = 16.

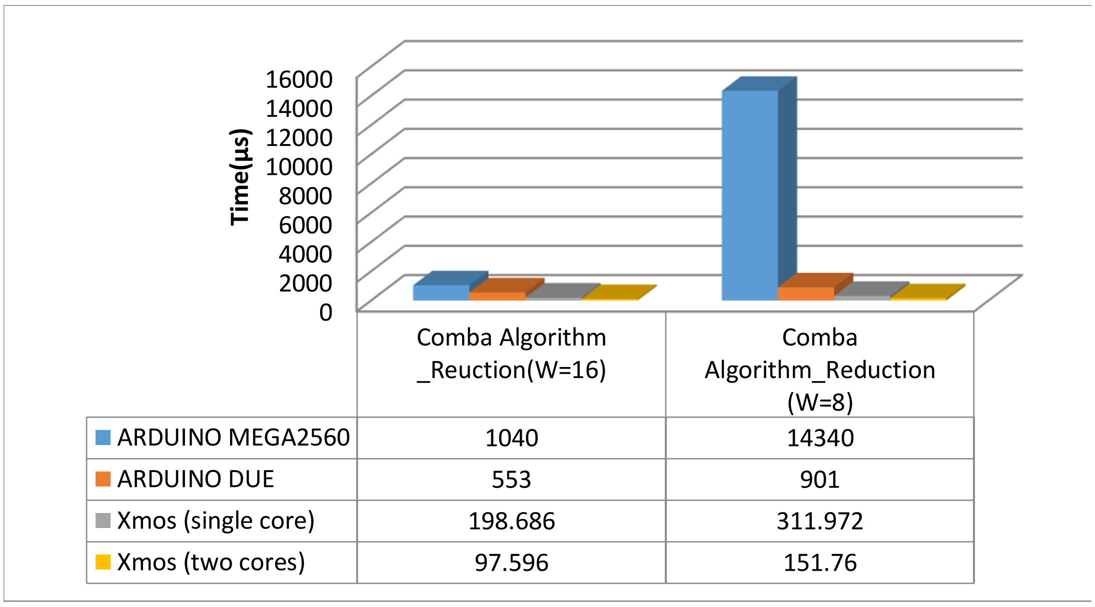
We patterned our design using C and XC programming Language on the xmos startKit. Also, we used xTIMEcomposer development tools for the design and for performing time analysis. Additionally, we implemented the sequential Comba algorithm on different single-core microcontroller platforms for comparison purposes. These include Arduino Mega2560 (AVR 16 bit), Arduino Due (ARM 32 bit) and Xmos (32 bit single core) with different data width (W $= 8, 16$ and 32 bits), as reported in Figure 4.4 and Figure 4.5.

We further provide to summarize and compare the obtained number of cycles for $GF(2^{163})$ using Comba parallel multiplication in single and multicore in comparison with state of art for $GF(2^{163})$ and $GF(160)$ multiplication. It is apparent that different types of algorithms, platforms and finite field types were used in previous state of art works. This makes evaluations more difficult for a fair comparison between our work and other published works. Therefore, we used the number of cycles as a reasonable metric for



	Comba Algorithm (W=32)	Comba Algorithm (W=16)	Comba Algorithm (W=8)
🟦 ARDUINO MEGA2560		616	14116
🟧 ARDUINO DUE		380	689
⬜ Xmos (single core)	1.064	0.816	0.76
🟨 Xmos (two cores)	0.99	0.788	0.784

Figure 4.5 Result Analysis of Implemented Comba Algorithm Without Reduction



	Comba Algorithm _Reuction(W=16)	Comba Algorithm_Reduction (W=8)
🟦 ARDUINO MEGA2560	1040	14340
🟧 ARDUINO DUE	553	901
⬜ Xmos (single core)	198.686	311.972
🟨 Xmos (two cores)	97.596	151.76

Figure 4.4 Result Analysis of Implemented Comba Algorithm with Fast Reduction

comparisons and evaluation. Furthermore, [66] used 21 registers out of 32 to execute the LD Multiplication Algorithm on a single core ATmega128 8-bit processor, whereas our approach of using xmos 32 bits multicore microcontroller used only 12 registers for single core multiplication and 4 registers in each task for performing simultaneous Comba.

Table 4.1 Comparison with State of Art of Comba Implementation

Author	Platform	Algorithm	Word Size (bit)	No of Cycles	Field
Aranha et al[5]	ATMega128L	LD Mult. (new variant)	8	9738	$GF(2^{163})$
Kargl et al[7]	ATMega128L	Comb multiplication with windows4	8	5057	$GF(2^{163})$
Kargl et al[7]	ATMega128L	Comb multiplication	*8*	*2593*	$GF(160)$
This work - Single core without reduction	Xmos StartKit	Sequential Comba Multiplication	8	*95*	$GF(2^{163})$
This work - Single core with reduction	Xmos StartKit	Sequential Comba Multiplication	8	*1140*	$GF(2^{163})$
This work - Single core without reduction	Xmos StartKit	Parallel Comba Multiplication	8	*149*	$GF(2^{163})$
This work - two cores with reduction	Xmos StartKit	Parallel Comba Multiplication	8	*129*	$GF(2^{163})$
Gouve [10]	MPY32	Comba Multiplication	16	*741*	$GF(160)$
Gouve [10]	MSPX	LD Multiplication	16	*3585*	$GF(2^{163})$
This work - Single core without reduction	Xmos StartKit	Sequential Comb multiplication	16	*214*	$GF(2^{163})$
This work- Two cores without reduction	Xmos StartKit	Parallel Comba Multiplication	16	*114*	$GF(2^{163})$
This work Single core with reduction	Xmos StartKit	Sequential Comb multiplication	16	*102*	$GF(2^{163})$
This work - Two cores with reduction	Xmos StartKit	Parallel Comba Multiplication	16	*129*	$GF(2^{163})$
P. Szczechowiak [11]	PXA271	Karatusba Multiplication	32	*13183*	$GF(2^{271})$
L. B. Oliveira[12]	PXA271 wMMX	Lopez-Dahab Algorithm	32	*1411*	$GF(2^{271})$
This work - Single core without reduction	Xmos StartKit	Sequential Comb multiplication	32	*139*	$GF(2^{163})$
This work- Two cores without reduction	Xmos StartKit	Parallel Comba Multiplication	32	*155*	$GF(2^{163})$
This work- Single core with reduction	Xmos StartKit	Sequential Comb multiplication	32	*1402*	$GF(2^{163})$
This work- Two cores with reduction	Xmos StartKit	Parallel Comba Multiplication	32	*155*	$GF(2^{163})$

4.6 Conclusions

In this work, we have shown that homogenous multicore microcontroller platforms are a feasible option to enhance the performance of Comba multiplication over binary finite fields, thereby enhancing the performance of ECC implementations. We have detailed the design of a modified Comba multiplier over the binary finite field $GF(2^{163})$ corresponding to an ECC curve using an Xmos startKit homogenous multicore platform that can be adopted in WSN applications. The design required a modification of both the Comba algorithm and the the fast reduction step to accommodate the reduction process for 8- and 16-bit word sizes. About 90% improvement in cycle performance was achieved compared to the single core implementation. Further work will concentrate on implementing the ECC point multiplication based on the modified Comba multiplier.

Chapter 5 : Software Implementation of Parallelized Elliptic Curve Scalar Point Multiplication over Binary Field

Some parts of this work have been published in [67]

Software implementation of parallelized Elliptic Curve Scalar Point Multiplication over binary field is presented in this chapter. In fact, we start this chapter by providing a much detail on the importance of scalar point multiplication while implementing Elliptic Curve Cryptography. Accordingly, we take the reader to the background of point multiplication and detailing the relationship between them. We also provide our novel proposal for enhancing the ECC point multiplication over $GF(2^m)$*. This is followed by a detail description of how we managed to achieve such proposal using a homogeneous multicore microcontroller known as XMOS. Eventually, we analyzed the performance concern different ECC curves this include:* $GF(2^{163}), GF(2^{233}), GF(2^{283}), GF(2^{409})$ *and* $GF(2^{571})$.

5.1 Introduction

Elliptic curve cryptography (ECC)-based security has potential use in resource constrained applications, such as RFID tags and wireless sensor networks (WSN) and therefore the Internet of Things (IoT). Compared to RSA, ECC requires shorter length keys for the same level of security and is computationally more efficient, and therefore, it has the ability to provide high security with faster processing time and fewer resources. In general, scalar point multiplication (PM) is the main operation in Elliptic Curve Cryptography [1]. The PM can be implemented either over binary extension fields $\mathrm{GF}(2^{\mathrm{m}})$ or over prime fields $\mathrm{GF}(\mathrm{P})$. In the ECC PM, the public key is computed by multiplying a base point on the elliptic curve, P with a private key (integer), K. A Koblitz curve [38] is a special elliptic curve that is resource friendly due to its simplicity, and therefore, it is used in this work; however, a random binary curve can also be implemented with an extra latency overhead. The ECC PM can be implemented in software, hardware, and as a software/hardware co-design. A pure software implementation is attractive on battery-run devices due to its flexibility and low resource requirements.

The crucial problem of software implementations is the latency due to the word-level computations required and frequent memory operations. Thus, different efforts have been conducted by researchers to enhance ECC performance in pure software design by modifying ECC related algorithms as reported by [68] and [69]. General purpose multicore processors are being increasingly adopted as alternative platforms to single core architectures for high-performance domain specific applications, such as ECC. For example, in [11], the authors proposed hardware design for separated hybrid scanning parallelization for Montgomery Multiplication using multicore approach by constructing two, four and eight soft cores on FPGA. The reported results in this work show good speed, large communication latency tolerance and good scalability. [70] proposed fully programmable curve-based crypto processors to accelerate scalar point multiplication of ECC using the GEZEL hardware/software co-design platform. Also, ECC multi-core software implementations on Intel Xeon Quad-Core processors using OpenMP are reported in [71] and [72]. Another example for software multicore implementation is reported by [8] where right to left double and add algorithm is parallelized using two threads through OpenMP library. We observed that almost all of the previous ECC multicore software implementations were implemented on powerful platforms that may not be suited to low-resource WSN type applications using devices with limited resources and lower clock speeds.

Therefore, in this work, we consider multicore ECC implementation as an enabler for deployment of public key protocols in the low-resource end of applications. We advocate the use of a homogeneous, low-power, multicore microcontroller that is suitable for adoption in resource constrained environments, such as (WSN). Key to ECC implementation on multicore platforms is parallelism by avoiding data dependency in the point operations. This requires careful task scheduling and core partitioning.

The contribution of the work presented in this chapter is to demonstrate the possibilities of obtaining better performance for ECC point multiplication by using suitable methodologies on partitioning the tasks between the available processor cores coupled with two novel fundamental algorithmic modifications for performing ECC point multiplication. The first proposed modification is based on performing a vertical parallelization on point doubling and point addition operations. The vertical parallelization approach is based on performing multiple finite field operations that have no data dependency by different parallel logical cores. The second proposed modification is based on modifying the left to right double and add binary point multiplication [1] to remove data dependencies. In this modified algorithm, we initialize point multiplication by scanning the position of the leftmost bit of key with a value of '1'. The scanning can accelerate point multiplication if some of the most significant bits of the key are zeros. In PM, point doubling is then carried out for every bit of key zeros and the point addition operation is performed when $k_i = 1$ where k_i is the i^{th} bit of k. The advantage of binary fields $GF(2^m)$ is that addition and subtraction are simply bitwise xor operations. We also adopt projective coordinate based ECC point multiplication to avoid the expensive field inversion operation.

Here, the proposed multicore ECC point multiplication with a single core version implemented on same Xmos device and on Arduino (Due) for $GF(2^{163})$. The proposed multicore implementation performs 60% better than single core-based implementation. It is also evident that multicore design perfomed better on embedded implementation.

The remainder of this chapter is organized as follows: Section 5.1 gives a mathematical preliminary of ECC. Section 5.3 introduces proposed parallel ECC point multiplication scheme. In Section 5.3, 5.4 and 5.5 we present multi core implementation of the proposed ECC and results. Finally, we end this chapter with some conclusions in section 5.6

5.2 Background

The elliptic curve over binary field (E) is defined as a set of points combined with point of infinity, and O is expressed by the Weierstrass equation:

$$E: y^2 + xy = x^3 + ax^2 + b \qquad (5.1)$$

where $a, b \in GF(2^m)$ and $b \neq 0$. The fundamental operation of the elliptic curve cryptography is scalar point multiplication, which is defined in (2) as follows:

$$Q = k.P \qquad (5.2)$$

where k is an integer, P is a point on the elliptic curve and Q is a new point on the elliptic curve. The new point, Q, is produced by scalar point multiplication, $k.P = P+\ldots+P+P$, where Q is a result of k-1 times point addition of P.

Different point multiplication algorithms are presented in [1] to compute (2). The performance of ECC depends on the point multiplication and its associated coordinates systems. In this work, standard coordinates-based Lopez-Dahab Projective is chosen to avoid the expensive field inversion operation that is involved in the affine coordinates systems. Moreover, the standard coordinates offer less computation than the Jacobian Projective [1] coordinates. Again, in Lopez- Dahab (LD) Projective coordinate, **a** point $[X, Y, Z]$ be point on the elliptic curve corresponding to the affine points $[X/Z, Y, Y/ Z^2]$ where $Z \neq 0$ [1].

5.2.1 Point Doubling

In (2), a group of operations of adding two points *(P, Q)* on curve E is performed for *k-1* times. The point addition can be obtained by point doubling while the value k bit is zero. An advantage of the point doubling is that the computation overhead is lower than that of point addition. The point doubling operations is based on [73], where the point doubling of point $P[X_1, Y_1, Z_1]$ to be given as *2P* $[X_3, Y_3, Z_3]$ and to be calculated per the following steps:

$$Z_3 \leftarrow X^2 Z^2$$

$$X_3 \leftarrow X^4 + bZ_1^4$$

$$Y_3 \leftarrow bZ_1^4 Z_3 + X_3(aZ_3 + Y_1^2 + bZ_1^4)$$

The point doubling based on the LD projective coordinate [1] has 4 finite field multiplication operations, 5 finite field squaring operations and 4 finite field addition operations, as shown in Algorithm 5.1. The algorithm has two temporary variables T_1 and T_2 to save intermediate results of point doubling.

Algorithm 5.1 Point Doubling in LD coordinate system where $a \in \{0, 1\}$

INPUT: $P = (X_1{:}Y_1{:}Z_1)$ *in LD coordinates on* $\frac{E}{K{:}Y^2} + xy = x^3 + ax^2 + b$

OUTPUT:$2P = (X_3{:}Y_3{:}Z_3)$ in LD coordinates

```
1: If P = ∞ then (∞)
2: $T_1 \leftarrow Z_1^2$ { $T_1 \leftarrow Z_1^2$}
3: $T_2 \leftarrow X_1^2$ {$T_1 \leftarrow X_1^2$}
4: $Z_3 \leftarrow T_1.T_2\{Z_3 \leftarrow X_1^2.Z_1^2\}$
5: $X_3 \leftarrow T_2^2\{X_3 \leftarrow X_2^4\}$
6: $T_1 \leftarrow T_1^2\{T_1 \leftarrow Z_1^4\}$
7: $T_2 \leftarrow T_1.b\{T_2 \leftarrow b.Z_1^4\}$
8: $X_3 \leftarrow X_3 + T_2\{X_3 \leftarrow X_1^4 + bZ_1^4\}$
9: $T_1 \leftarrow Y_1^2\{T_1 \leftarrow Y_1^2\}$
10: If $a = 1$ then $T_1 \leftarrow T_1 + Z_3\{T_1 \leftarrow aZ_3 + Y_1^2\}$
11: $T_1 \leftarrow T_1 + T_2\{T_1 \leftarrow aZ_3 + Y_1^2 + bZ_1^4\}$
12: $Y_3 \leftarrow X_3.T_1\{Y_3 \leftarrow X_3(aZ_3 + Y_1^2 + bZ_1^4)\}$
13: $T_1 \leftarrow T_2.Z_3\{bZ_1^4Z_3\}$
14: $Y_3 \leftarrow Y_3 + T_1\{Y_3 \leftarrow bZ_1^4Z_3 + X_3(aZ_3 + Y_1^2 + bZ_1^4)\}$
15: Return $(X_3{:}Y_3{:}Z_3)$
```

5.2.2 Point Addition

The point addition in LD projective coordinate system allows mixing the coordinates for the point addition, where the projective points $P[X_1, Y_1, 1]$ could be added with affine point $Q[x_1, y_1]$ to produce $P + Q[X_3, Y_3, Z_3]$ where $Q \neq \pm P$ using the steps as shown below[73].

$$D \leftarrow B^3(C + aZ_1^2)$$

$$Z_3 \leftarrow C^2$$

$$E \leftarrow A.C$$

$$X_3 \leftarrow A^2 + D + E$$

$$F \leftarrow X_3 + x_2.Z_3$$

$$G \leftarrow (x_2 + y_2).Z_3^2$$

$$Y_3 \leftarrow (E + Z_3).F + G$$

The point addition algorithm has 8 finite field multiplications, 5 squaring and 9 additions as shown in Algorithm 5.2. In addition to that, T_1, T_2, T_3 are temporary variables to be used for the adder operations.

Algorithm 5.2 Point Addition in LD coordinate system where $\boldsymbol{a} \in \{\mathbf{0}, \mathbf{1}\}$

INPUT: $P = (X_1{:}Y_1{:}Z_1)$ *in LD coordinates on* $Q = (x_2, y_2)$ in affine coordinate on
$E/Y : y^2 + xy = x^3 + ax^2 + b$

OUTPUT: $2P = (X_3{:}Y_3{:}Z_3)$ in LD coordinates

```
1: If $P = \infty$ then $(\infty)$
2: If $P = \infty$ then $(x_2{:}y_2{:}1)$
3: $T_1 \leftarrow Z_1X_2\{T_1 \leftarrow X_1Z_1\}$
4: $T_2 \leftarrow Z_1^2\{T_1 \leftarrow Z_1^2\}$
5: $X_3 \leftarrow X_1.T_1\{X_3 \leftarrow B = X_1.Z_1 + X_1\}$
6: $T_1 \leftarrow Z_1.X_3\{T_1 \leftarrow C = Z_1B\}$
7: $T_3 \leftarrow T_2.y_2\ \{T_3 \leftarrow Y_2Z_1^2\}$
8: $Y_3 \leftarrow T_1 + T_3\ \{Y_3 \leftarrow A = Y_2.Z_1^2 + Y_1\}$
9: If $X_3 = 0$ then (a)if
   $Y_3 = 0$ use Point doubling Algorithm to compute $(X_3{:}Y_3{:}Z_3) = 2(x_2{:}y_2{:}1)$ and return $(X_3{:}Y_3{:}Z_3)$
10: $Z_3 \leftarrow T_1^2\{Z_3 \leftarrow C^2\}$
11: $T_3 \leftarrow T_1.Y_3\{Z_3 \leftarrow E = AC\}$
12: (b) If a =1 then $T_1 \leftarrow T_1 + T_2\ \{T_1 \leftarrow C + aZ_1^2\}$
13: $T_1 \leftarrow T_2.Z_3\ \{T_1 \leftarrow bZ_1^4Z_3\}$
14: $X_3 \leftarrow T_2.T_1\ \{X_3 \leftarrow D = B^2(C + aZ_1^2)\}$
15: $T_2 \leftarrow Y_3^2\{T_2 \leftarrow A^2\}$
16: $X_3 \leftarrow X_3 + T_2\ \{A^2 + D\}$
17: $(X_3 \leftarrow X_3 + T_3)\{X_3 \leftarrow A^2 + D + E\}$
18: $(T_2 \leftarrow x_2.Z_3)\{T_2 \leftarrow X_2Z_3\}$
19: $T_2 \leftarrow T_2 + X_3\{T_2 \leftarrow F = X_3 + X_2Z_3\}$
20: $T_1 \leftarrow\ \{T_1 \leftarrow Z_3^2\}$
21: $T_3 \leftarrow T_3 + Z_3\{T_3 \leftarrow E + Z_3\}$
22: $Y_3 \leftarrow T_3.T_2\{Y_3 \leftarrow (E + Z_3)F\}$
23: $T_2 \leftarrow x_2 + y_2\{T_2 \leftarrow X_2 + Y_2\}$
24: $T_3 \leftarrow T_1.T_2\{T_3 \leftarrow G = (X_2 + Y_2)Z_3^2\}$
25: $Y_3 \leftarrow Y_3 + T_3\{Y_2 \leftarrow (E + Z_3)F + G\}$
26: Return $(X_3{:}Y_3{:}Z_3)$
```

5.2.3 Scalar Point Multiplication

The scalar point multiplication is to be defined as a series of point doubling and addition operations. Again, the result of the LD point doubling and point addition algorithms is the $Q[X,Y,Z]$ will be projective coordinates that can be easily converted into affine coordinates $Q[x_2,y_2]$ by using the steps below:

$$x_2 \leftarrow X/Z \text{ and } y_2 \leftarrow Y/Z^2$$

For the projective to affine coordinate conversion, a single field inversion is used. In this work, we consider implementing left-to-right binary method for the point multiplication as shown in Algorithm 5.3 [1] that is suitable for an initial operation.

Algorithm 5.3 Left-to-right binary method for point multiplication

```
INPUT: $k = (k_{t-1}, \cdots k_1, k_0)_2, P \in E(F_q)$

OUTPUT:$kP$

1) $Q \leftarrow \infty$
2) For $i$ from $t-1$down to do
      a. $Q \leftarrow 2Q$
      b. if $k_i = 1\ then\ Q \leftarrow Q + P$
3) Return $(Q)$
```

5.2.4 Binary Field Arithmetic

Binary fields are attractive for ECC-based public key cryptography [15, 29]. The point operations involve finite field multiplication, finite field squaring, finite field addition and finite field inversion over $GF(2^m)$ [1]. The finite field inversion is the costliest operation but can efficiently be performed by using multiplicative inversion. Thus, field multiplication is the most crucial and complex arithmetic operation.

5.2.5 Finite Field Multiplication

The efficiency of ECC is highly dominated by the efficiency of the field multiplication operation [51]. The field multiplication is performed by multiplying two elements $a(x).b(x) \in GF(2^m)$ yielding a binary elements of degree $(2m-1)$ followed by decreasing the product modulo an irreducible polynomial $F(x) \in GF(2^m)$ [1]. There are various approaches to implementing multiplication over $GF(2^m)$ using a word-level computational environment (i.e., embedded processor); these include product scanning [33], hybrid scanning, operand cashing, and consecutive operand-cashing techniques [11].

In this work, we adopt the product scanning algorithm (i.e. Comba algorithm) for its reported efficiencies [74]. The algorithm runs using two individual nested loops as outer loop and inner loop. The outer loop handles the index of the multiplier and the inner loop is responsible for generating the index of the multiplicand. The amount of time the inner loop iterates relies on the amount of words required for a given field (m). For example, if there are $''\text{t}''$ words, the number of iteration will be $'\text{t}'$. In each inner loop iteration, there is one $GF(2^m)$ multiplication, one *xor* and two load operations for collecting each column products. The column products are 2w size. Thus, a second store operation is essential at the outer loop to hold the result of the partial product.

5.2.6 Finite Field Squaring

Squaring over binary fields is a linear operation. The square operation can be implemented by manipulating the simple bits of the original polynomial *a(z)*. To speed up the process of the squaring operation, a look-up table are used with a size of 512 bytes. The look up table is based on squaring needing pre-computing of 8-bit polynomials input into 16-bit squared results. The disadvantage of the look-up table method is that it requires large memory that may increase with the increase of field size. In this work, we consider a linear polynomial squaring [1].

The main idea of the linear algorithm is to accomplish the squaring by inserting zeros between every corresponding bit of *a(z)* from bit position *"1"*. In this process, the odd positions are to be filled up with zeros and even positions loaded with bits of the input polynomial. After the square operation, the output is *2m-1* bit that is required to be reduced to m bit by using a reduction operation.

5.2.7 Finite Field Addition

The addition of two elements is simply calculated using a bit wise XOR operation. In this work, the field addition is a word level XOR operation [75].

5.2.8 Modular Reduction

Each of the resulting field multiplication and field squaring operation is *2m-1* bits without reduction. We need to reduce the result to m bit. In this chapter, we consider NIST fast reduction polynomials [1]. Thus, a modified 8-bit fast reduction algorithm we proposed in [74] is implemented over $GF(2^{163})$ with the irreducible polynomial. Additionally, the rest of $GF(2^m)$ were adopted and implemented as described in [1].

$$f(z) = Z^{163} + Z^7 + Z^6 + Z^3 + 1 \tag{3}$$

$$f(z) = Z^{233} + Z^{74} + 1 \; (with\, w = 32)$$

$$f(z) = Z^{283} + Z^{12} + Z^7 + Z^5 + 1 \; (with\, w = 32)$$

$$f(z) = Z^{409} + Z^{87} + 1 \; (with\, w = 32)$$

$$f(z) = Z^{571} + Z^{10} + Z^5 + Z^2 + 1 \; (with\, w = 32)$$

As illustrated in Figure 5.1, the result of multiplication or square operation is 325 bits. We need to reduce the 325 bits to 163 bits (i.e. 162 to 0). In the fast reduction method, the 163 th to 325 th bits are added with 0th to 162th bit with shifting. For the irreducible polynomial in (3) over $GF(2^m)$, the bitwise addition is performed with: no-shifting, 3-bit shifting, 6-bit shifting and finally, 7-bit shifting. Moreover, the 3-bit shifting, 6-bit shifting and 7-bit shifting shifts

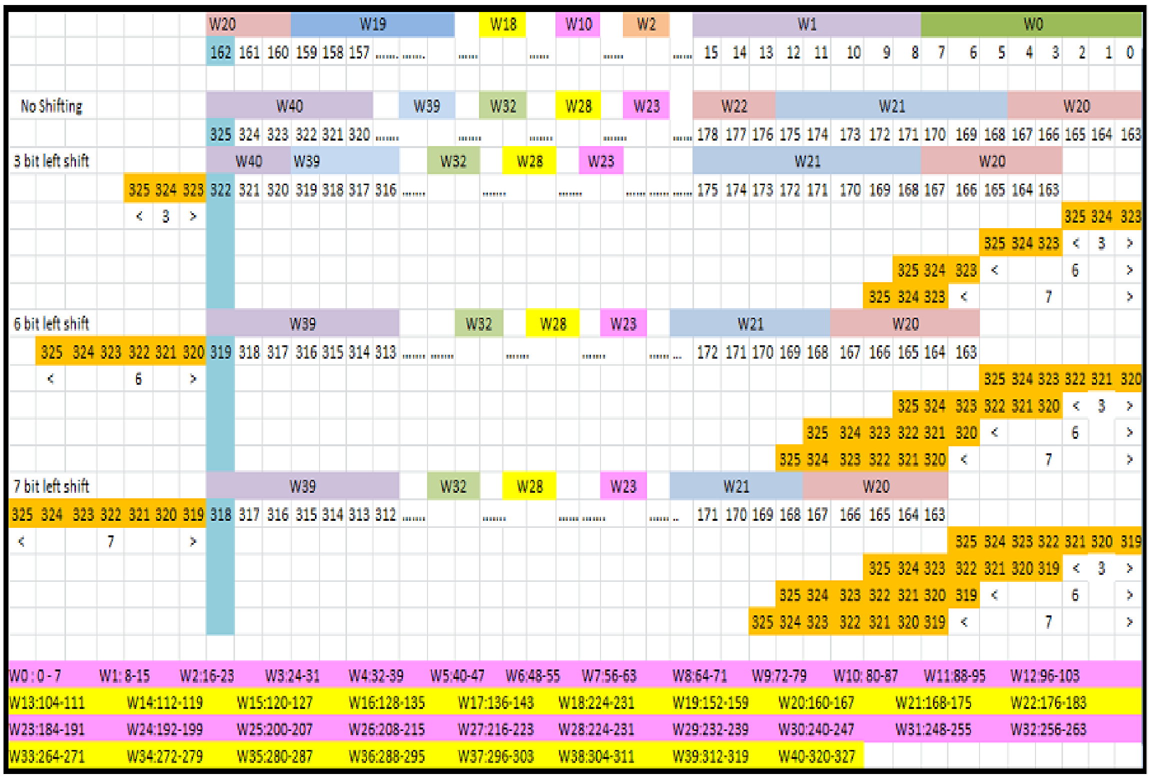


Figure 5.1 Shifting Operation in Fast Reduction Process with word size=8

extra bits over the 162^{th} bit as shown in the Figure 5.1. For example, three extra bits overflow due to the 3-bit shifting, and therefore, it is required to add the extra bits with the bits from the rightmost side bit (0^{th} bit) in a shifted fashion as per the order of the irreducible polynomial. Thus, in 3-bit shifting, the extra bits (323^{th}, 324^{th} and 325^{th}) are added with no-shift, 3-bit shift, 6-bit shift and finally 7-bit shift from the rightmost bit (0^{th} bit). The same approach applies for the 6-bit shifting and 7-bit shifting cases, as shown in Figure 5.1.

5.3 Proposed Concurrent ECC Point Multiplication

There are data dependencies in the LD point operation algorithms that can prohibit achieving parallel field operations [8]. We extract potential field operations from the LD algorithm that can be performed in parallel by avoiding data dependency. We present a new vertical parallelism mechanism for both point doubling and point addition separately by avoiding data dependency. The parallel operation utilizes separate cores or a group of cores to operate several instructions concurrently.

5.3.1 Parallel Lopez-Dahab Point Doubling

Algorithm 5.4 Modified Point Doubling in LD coordinate system where a *ϵ {0,1}*

```
INPUT: $P = (X_1 : Y_1 : Z_1)$ $in\ LD\ coordinates\ on \frac{E}{K:Y^2} + xy = x^3 + ax^2 + b$
OUTPUT:$2P = (X_3 : Y_3 : Z_3)$ in LD coordinates

1: If $P = \infty$ then $(\infty)$
2: $T_1 \leftarrow Z_1^2\ \{T_1 \leftarrow Z_1^2\}$
3: $T_2 \leftarrow X_1^2\ \{T_1 \leftarrow X_1^2\}$
4: $Z_3 \leftarrow T_1.T_2\{Z_3 \leftarrow X_1^2.Z_1^2\}||X_3 \leftarrow T_2^2\{X_3 \leftarrow X_2^4\}||T_1 \leftarrow T_1^2\{T_1 \leftarrow Z_1^4\}||T_2 \leftarrow T_1.b\{T_2 \leftarrow b.Z_1^4\}$[Parallel Operation]
5: $X_3 \leftarrow X_3 + T_2\{X_3 \leftarrow X_1^4 + bZ_1^4\}$
6:$T_1 \leftarrow Y_1^2\{T_1 \leftarrow Y_1^2\}$
7:$If\ a = 1$ then $T_1 \leftarrow T_1 + Z_3\{T_1 \leftarrow aZ_3 + Y_1^2\}$
8:$T_1 \leftarrow T_1 + T_2\{T_1 \leftarrow aZ_3 + Y_1^2 + bZ_1^4\}||Y_3 \leftarrow X_3.T_1\{Y_3 \leftarrow X_3(aZ_3 + Y_1^2 + bZ_1^4)\}||T_1 \leftarrow T_2.Z_3\{bZ_1^4Z_3\}$[Parallel Operation]
9: $Y_3 \leftarrow Y_3 + T_1\{Y_3 \leftarrow bZ_1^4Z_3 + X_3(aZ_3 + Y_1^2 + bZ_1^4)\}$
10: Return $(X_3 : Y_3 : Z_3)$
```

The point doubling algorithm is modified to perform parallel operations. A Read-After-Write (RAW) dependency is investigated to extract possible parallel operations. In addition, we accomplish several parallel operations in the point doubling that are compatible with our target platform. For example, we parallelized two field operations as shown in Algorithm 5.4: $(Z_3 \leftarrow T_1T_2)$ with $(X_3 \leftarrow T_2^2)$, since there is no dependency.

Similarly, three fields operations are performed concurrently, as shown in step 8. However, step 9 $(Y_3 \leftarrow Y_3 + T_1)$ cannot be parallelized since it depends on the output of previous step$(Y_3 \leftarrow X_3.T_1)$. With a careful rescheduling in the point doubling operations algorithm proposed by [1], we were able to minimize the total steps down to 9 as shown in the modified Algorithm 5.4.

5.3.2 Parallel Lopez-Dahab Point Addition

The point addition implementation in Algorithm 5.2 was modified by removing dependencies to enable vertical parallelism.

Algorithm 5.5 Modified Point Addition in LD coordinate system where a ϵ {0,1}

INPUT:$p = (X_1 : Y_1 : Z_1) in\ LD\ coordinates\ on\ Q = (x_2, y_2)$in affine coordinates $\frac{E}{K:Y^2} + xy = x^3 + ax^2 + b$

OUTPUT:$2P = (X_3 : Y_3 : Z_3)$ in LD coordinates

```
1: If $P = \infty$ then $(\infty)$
2: If $P = \infty$ then $(x_2, y_2 : 1)$
3: $T_1 \leftarrow Z_1 X_2 \{T_1 \leftarrow X_1 Z_1\}$
4: $T_2 \leftarrow Z_1^2 \{T_1 \leftarrow Z_1^2\}$
5: $X_3 \leftarrow X_1.T_1 \{X_3 \leftarrow B = X_1.Z_1 + X_1\}$
6:$T_1 \leftarrow Z_1.X_3 \{T_1 \leftarrow C = Z_1 B\}$
7: $T_3 \leftarrow T_2.y_2\ \{T_3 \leftarrow Y_2 Z_1^2\}$
8: $Y_3 \leftarrow T_1 + T_3\ \{Y_3 \leftarrow A = Y_2.Z_1^2 + Y_1\}$
9: If $X_3 = 0\ then\ (a)$if $Y_3 = 0$ use Point doubling Algorithm to compute $(X_3 : Y_{3:} Z_3) = 2(x_2 : y_2 : 1)$and return $(X_3 : Y_3 : Z_3)$
10: $Z_3 \leftarrow T_1^2 \{Z_3 \leftarrow C^2\} || T_3 \leftarrow T_1.Y_3 \{Z_3 \leftarrow E = AC\}$[ParallelOperation}
11: (b) If a =1 then $T_1 \leftarrow T_1 + T_2\ \{T_1 \leftarrow C + aZ_1^2\}$
12: $T_1 \leftarrow T_2.Z_3\ \{T_1 \leftarrow bZ_1^4 Z_3\}$
13: $X_3 \leftarrow T_2.T_1\ \{X_3 \leftarrow D = B^2(C + aZ_1^2)\}$
14: $T_2 \leftarrow Y_3^2 \{T_2 \leftarrow A^2\}$
15: $X_3 \leftarrow X_3 + T_2\ \{A^2 + D\}$
16: Interleaving Parallel[$(X_3 \leftarrow X_3 + T_3)\{X_3 \leftarrow A^2 + D + E\} || (T_2 \leftarrow x_2.Z_3)\{T_2 \leftarrow X_2 Z_3\} || (T_2 \leftarrow T_2 + X_3 \{T_2 \leftarrow F = X_3 + X_2 Z_3\} || T_1 \leftarrow \{T_1 \leftarrow Z_3^2\}$[Parallel inside Parallel operation]
17: $T_3 \leftarrow T_3 + Z_3 \{T_3 \leftarrow E + Z_3\} || Y_3 \leftarrow T_3.T_2 \{Y_3 \leftarrow (E + Z_3)F\}$[Parallel Oeration]
18: $T_2 \leftarrow x_2 + y_2 \{T_2 \leftarrow X_2 + Y_2\}$
19: $T_3 \leftarrow T_1.T_2 \{T_3 \leftarrow G = (X_2 + Y_2)Z_3^2\} || Y_3 \leftarrow Y_3 + T_3 \{Y_2 \leftarrow (E + Z_3)F + G\}$[ Parallel Operation]
20: Return $(X_3 : Y_3 : Z_3)$
```

n Algorithm 5.2, there are 26 steps, including the conditional step that is triggering the point doubling operation, $Y_1 = 0$. In the modified algorithm, Algorithm 5.5, we managed to minimize the number of steps to 20. Furthermore, in this work, we advocate a new approach of parallelization, namely interleaving parallelization as shown, for example, in the modified algorithm for step 16.

5.3.2 Proposed Left to Right Double and Add Scalar Point Multiplication

In order to obtain better time complexity performance, we modified the left to right algorithm[1]. The enhancement is achieved by performing the initial scanning most significant bits *(MSB*) of k in order to track down the first none zero bit from *MSB*. If the non-zero first bit is found, then *P's* coordinates will be filled in Q to start loop operation is shown in Algorithm 3. The start position of the loop is the position of the first non-zero bit that is

Algorithm 5.6 Modified Left-to-right binary method for point multiplication

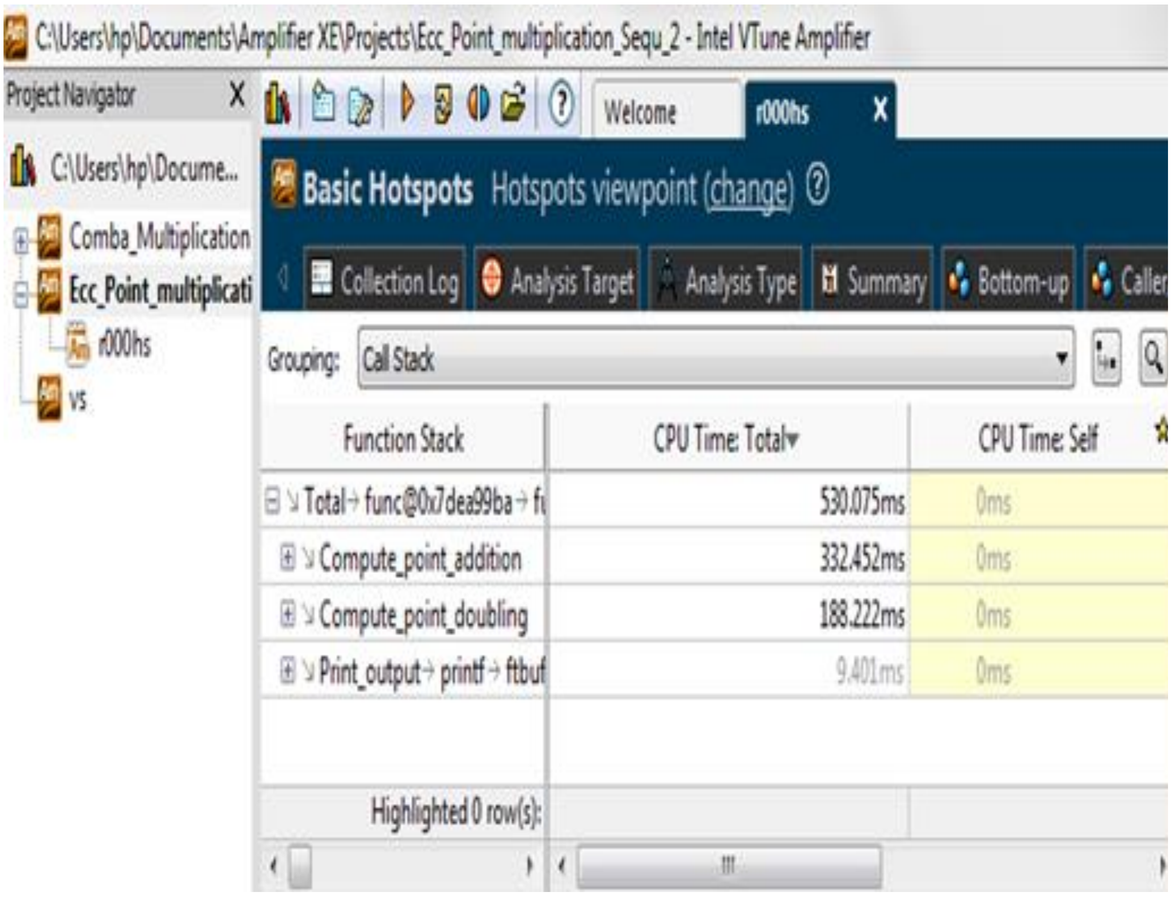
```
INPUT: Base Point $P = (P_x, P_y)$ and scalar $k = (k_{i-1}, \cdots, k_1, k_0)_2, P \in E(F_q)$
OUTPUT: Point $Q$on the elliptic curve such that
Q=kP=$(Q_x, Q_y, Q_z)$

    1: $Q \leftarrow \infty$
    2: For i From t-1down to do
        a) if $K_i =' 0'$then
            1. break
            2. end if
        b) else if $K_i =' 1'$then
            1. load Q = $P$
            2. store i in pos
            3. end if
        c) end for
    3: For i in pos-1 to 0 do
        a) Perform_point_doubling: $Q \leftarrow 2Q$
        b) if $K_i =' 1'$then
            1. Perform_point_addition: $Q \leftarrow P + Q$
            2. end if
        c) end for
4: Return (Q)
```

listed in variable *"pos"* as shown in the algorithm. The loop operation then continues from *"pos-1*" to 0th bit of k. The scanning process reduces latency for the case of *k* input with zeros in the *MSBs.* In [1], the time complexity for point addition *(A)* and point doubling *(D)* are *mD* and *(m/2)A* respectively. In the proposed Algorithm 3, the loop operation of the main algorithm starts only after finding of the first from the MSBs. Thus, the number of point addition operations will be *(m/2-1)* and point-doubling operations would be (m-(m-1-pos)) where *(m-1-pos)* represents the number at the *MSB*. Therefore, the expected running time for the scalar point multiplication could be represented by *(m/2-1).A+(m-(m-1-pos)-1)D,* which is potentially lower than original algorithm in [1].

5.4 Implementation Details

The proposed ECC scalar point multiplication based on the modified algorithms is implemented kit using xtimecomposer IDE. The Xmos Kit consists of a two tiles XCORE device and eight 32-bit logic cores that can support up to 500 MIPS xCORE multi-core microcontroller and 100MHz processor speed [14]. A sequential implementation was first carried out to help identify any performance bottlenecks in the execution of the point operations, as shown in Figure 5.2.



C:\Users\hp\Documents\Amplifier XE\Projects\Ecc_Point_multiplication_Sequ_2 - Intel VTune Amplifier

Project Navigator

- C:\Users\hp\Docume...
  - Comba_Multiplication
  - Ecc_Point_multiplicati
    - r000hs
  - VS

Welcome | r000hs

Basic Hotspots Hotspots viewpoint (change)

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | Caller

Grouping: Call Stack

Function Stack	CPU Time: Total	CPU Time: Self
Total→ func@0x7dea99ba→ f	530.075ms	0ms
Compute_point_addition	332.452ms	0ms
Compute_point_doubling	188.222ms	0ms
Print_output→ printf→ ftbuf	9.401ms	0ms

Highlighted 0 row(s):

Figure 5.2 Sequential ECC Scalar Point Multiplication Intel Vtune Analysis [3]

In addition to that, we analyzed our sequential implementation on two different platforms including Arduino Due (ARM processor) and Xmos platform as shown in Figure 5. 3. In fact, these two figures show that around 63% the scalar point multiplication is taken by point addition operations.

5.5 Performance Analysis

We analyzed our sequential implementation on two different platforms, including Arduino Due (ARM processor) and Xmos platform, as shown in Fig 5.3. In fact, these two figures show that around 63% of the scalar point multiplication is taken by point addition operations. After a successful implementation, the proposed vertical parallelization managed to enhance the performance of ECC scalar point multiplication up to 49% and around 44% in point addition operations, as illustrated in Fig. 4. In Table 5.1, we show the number of logic cores used, execution time and number of clock cycles for some steps that we parallelized in Algorithms 4 and 5 to conduct point doubling and point addition. These cores are automatically assigned by the tools as we are using the parallel instruction, “par statement” outside the main function. To achieve, a further parallel operation, we assigned “par statement” to some of the arithmetic functions belonging to the main point doubling and point addition functions individually.

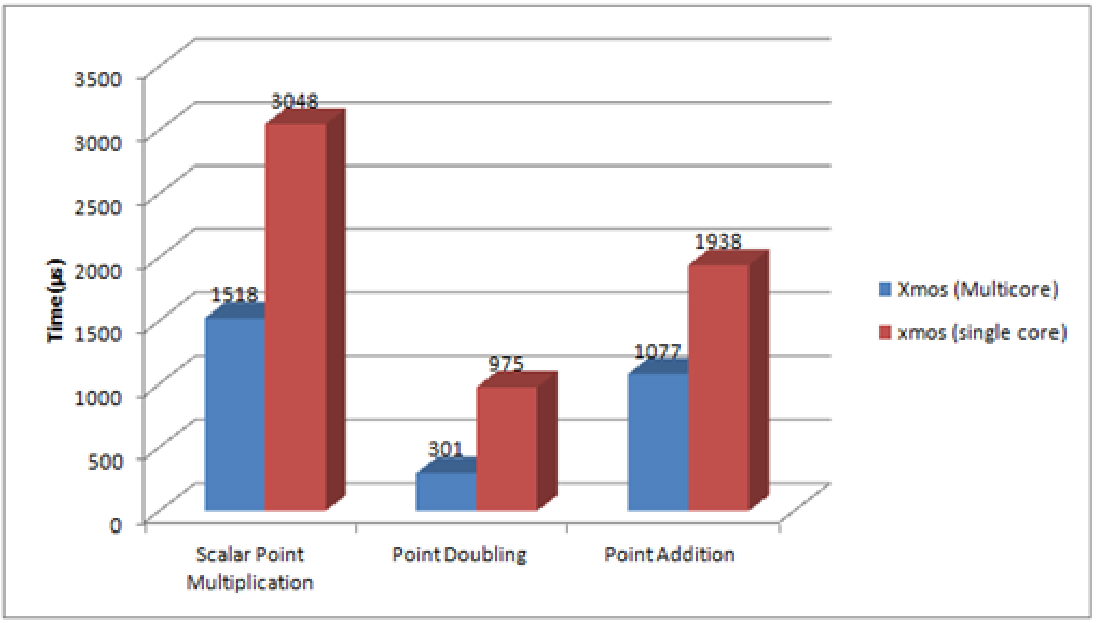


Figure 5.3 Parallel ECC Scalar Point Multiplication xmos single core

The optimal utilization of the core is a key part of the optimization of ECC point multiplication, as shown in Table 5.1. In this work, we managed to utilize all of the existing 8 cores provided. Notably, the performance of each binary finite field arithmetic operation that is implemented on a particular core of the Xmos can be obtained. Again, the parallel operation consumes a latency that is equivalent to a multiplication latency, whereas the other operations (field squaring and field addition) are operating on the fly. Furthermore, the overall number of processors for performing the scalar point multiplication over projective coordinates is tested for *k=3{"00…011"},* where a load *Q* operation for the *MSB* of *k, 1* followed by point doubling and point addition for the last bit of *k, 1.*

To quantify, a k input of the best case (k=3), average case (as same as the complexity in [8]) and worst case (k=0x"07ff..f') inputs were investigated. Table 5.2 illustrates the time complexity of the scalar point multiplication for different inputs.

Table 5.1 Time Complexity Table

k	***Operations involved***	***No of Clock Cycles***	***Time Execution (μs)***
$k = 0$	Infinity	69093	64
$k = 1$	Loading	70270	65
$k = 2$	Loading+Doubling	374095	369
$k = 3$	Doubling+Addition	1523×10^3	1518
$k = 0x07 \cdots 0xFF$ (Worst Case)	163 Doubling +163 Addition	$24750 * 10^3$	247501

Table 5.2 Operation Details and Performance

Operation	***Number logic Cores used***	***Operation***	***Time Execution (μs)***	***No of Clock Cycles***
Point Doubling for Step Number 4	4 logic cores (core 0 To core 3)	Multiplication	5	318
		Squaring	5.648	353
		Addition	1.536	96
Point Addition Step No 10	One Core used (Core 0)	Multiplication	5	318
		Squaring	5.648	353

As already stated, to our knowledge, our work is the first attempt to implement ECC scalar point multiplication on a homogeneous multicore microcontroller. For comparisons, we tried to put context more than provide like for like comparisons as the platforms are not comparable in terms of resources available, as shown in Table 5.3.

Table 5.3 Comparison With State of Arts

Author	***platf orm***	***Algorithm***	***Time Execution (μs)***	***No of Clock Cycles***	***Operation Field***
C. Negre[6]	Intel Core i7	Parallel (Double, Halve)-and-add with NAF4	29	97×10^3	$GF(2^{233})$
C. Negre[6]	Qualcom m SnapDrag on	Parallel (Double, Halve)-and-add with NAF4	1060	1591×10^3	$GF(2^{233})$
J.M[8]	Intel Core i7	Halve-Double and add	-------	130696	$GF(2^{233})$
This Work	Xmos Start Kit	left to right double and add scalar point	51.32	1523×10^3	$GF(2^{163})$
This Work	Xmos Start Kit	left to right double and add scalar point	18.590	8887×10^3	$GF(2^{233})$
This Work	Xmos Start Kit	left to right double and add scalar point	77.489	1059×10^3	$GF(2^{283})$
This Work	Xmos Start Kit	left to right double and add scalar point	109.835	2111×10^3	$GF(2^{409})$
This Work	Xmos Start Kit	left to right double and add scalar point	151.778	3956×10^3	$GF(2^{571})$

5.6 Conclusions

In this chapter, we proposed new modified algorithms that overcome data dependencies in ECC computations and hence enable parallel implementation of ECC on multi-core platforms efficiently. A pure software implementation for ECC scalar point multiplication over $GF(2^m)$ using the Xmos multi-core microcontroller was implemented using these algorithms, which confirmed the feasibility and improvements of adopting parallelism in ECC implementations. It is advocated that homogeneous multicore platforms can be useful for resources constrained applications where strong security is required. Potentially, our parallelization approach could be adopted to improve cryptography operations and to open up the potential of having strong public key cryptography in software with high performance and flexibility on a range of multicore microcontroller platforms. Hence, future work will investigate deploying these algorithms on alternative multicore platforms.

Chapter 6 Software Design: Fast Parallel ECC Point Multiplication over Prime Fields

This chapter presents fast parallel ECC Point Multiplication over prime Field $GF(P)$. *In this work, we attempted to enhance the performance of ECC point Multiplication over prime field* $GF(128), GF(192), GF(256) and\ GF(384)$ *by using a homogenous multicore microcontroller. Our aim is based on the fact of necessity of providing highly-secure communication on resource constrained devices, which can accordingly use in different applications such RFID, Wireless Sensor Network, and IoT. This chapter provides an overall description of ECC over prime field with some mathematical descriptions of Modular Multiplication, Modular reduction, and point multiplication. Accordingly, we detailed our proposal to enhance the ECC point multiplication. This is followed by a detailed description of our implementation and a final presentation of the enhancement as shown in the result analysis*

6.1 Introduction

There is an expectation that the emergence of multicore processing would enable a new generation of sensor nodes that suits the anticipated growth of information-rich applications using Wireless Sensor Nodes (WSNs) [76]. Specifically, the lower power homogeneous multicore microcontrollers can be very attractive for sensing the necessary data and carrying out the required computations across the multicores concurrently. The security remains a major concern, similar to that of single core sensor nodes. Hence there is a need to consider the safety of the devices at the time of implementation has developed as an alternative to RSA in the applications of public key-based security due to its enhanced performance of security per bit. However, the complex ECC computations based on either binary $GF(2^m)$ or prime $GF(P)$ fields make their implementation on small, low-power devices challenging.

Although [77] concluded that ECC over binary fields is faster than ECC over prime fields in hardware implementation, the timing result reported by [1] for the ECC software implementation using Intel Pentium Processors shows the opposite. In fact, [1] reported faster performance when using for NIST recommended prime fields compared to the binary fields curves. A pure ECC software implementation over prime fields optimized for high performance would be attractive for implementation on sensor node processors. Different efforts have been made by researchers to enhance the ECC performance in these devices like the works reported by [78-80].

Parallel ECC implementations over prime fields have been reported for both hardware and software platforms [8, 81, 82]. For example, [8, 81, 82] managed to improve the ECC performance over prime field for 256-bit and 160-bit in GF(P) using Read after Write (RAW) strategy implemented in Modular Arithmetic Logic Unit (MALU) using hardware design. Similarly, in [8, 81, 82] the authors proposed software parallel approach for enhancing the scalar point multiplication over the prime field (p = 2255 −19) in an Intel Core 2 workstation.

We noticed that ECC over prime field parallel software was implemented on powerful platforms. These implementations may not suit to low-resource, WSN-type applications using devices with limited resources and lower clock cycles.

The contribution of this work is mainly to further explore the feasibility and potential for parallelizing improve ECC scalar point multiplication over prime field for four SECG curves Spec128r1, Spec192r1, Spec256 and Spec384r1 on a homogeneous multicore microcontroller(xmos) [15] and [14]. To our knowledge, this is the first ECC over prime implementation reported on such platform.

The proposed implementation is based on three novel fundamental algorithmic modifications. Firstly, we present first time parallel Comba Algorithm proposed by [17] and [18] on multicore microcontroller. Secondly, we proposed a novel parallel approaches for parallelizing Jacobian point doubling proposed by [83], and we managed to reduce it computational steps to 15 steps. Also, we presented a new parallel approach for parallelizing point addition algorithms proposed by [83]. We used Read After Write (RAW) investigation and data dependencies check to extract possible parallel operations. Finally, we analyze the timing performance of our implemented techniques on the xmos start kit platform using its xTimecomposer IDE. The obtained results show 85% improvement compare to single core implementation.

The remainder of this chapter is organized as follows. Section 6.3 details the Elliptic Curve Cryptography design and implementation. The implementation results and analysis are devoted in sections 6.4 and 6.5. Finally, the chapter is concluded in 6.5.

6.2 Mathematical Background

Mathematically, the elliptic curve over prime field consists of an integer P over finite field F_P and the elements $a, b \in F_p$ are to be defined by the equation below:

$$E: y^2 \equiv x^3 + ax + b \ (mod\ p) \qquad (6.1)$$

Recently, ECC has been standardized by different standard bodies such as ANSI (American National Standard), NIST (National Institute of Standards and Technology) and others. In this work, we specifically used the following recommended elliptic domain parameters over F_P for secp128r1, secp192, secp256r1 and secp384r1, as specified by [15].

The domain parameters for these curves consists of six main parameters nominated as sextuple and presented as below:

$$T = (p, a, b, G, n, h)$$

where $a, b \in F_P$ and P represents an integer of modulus to specify the finite field, and for the purpose of our work, we consider the following recommended modulus:

$P_{128} = 2^{128} - 2^{97} - 1, P_{192} = 2^{192} - 2^{64} - 1, P_{256} = 2^{224}(2^{32} - 1) + 2^{192} + 2^{96} - 1, P_{384} = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1.$

The purpose of parameters G, n, h are to define the base point, define the order of G and define the cofactor, respectively. These four curves are following the definition specified by equation (1). In fact, these curves have the same basic arithmetic operations which work for all of them. The only differences between them are the modulus and size of the numbers change. Therefore, in the coming subsections. we provide more detail for the arithmetic prime finite field operations involved in our work.

In principle, the construction of Elliptic Curve is mainly depending on the selection of point representation, point doubling, point addition and point multiplication. Having these operations implemented will allow to create a trapdoor for implementing different protocols such Digital Signature Algorithm (DSA) and Diffie-Hellman based encryption. However, there are different ECC standard bodies recommended to implement ECC underlying finite field, a Galois Field *(GF)* prime field or binary fields $GF(2^m)$ [15].

6.2.1 Modular Multiplication

Algorithm 6.1 Comba Multiplication Technique

INPUT: $A = (a_{s-1}, \cdots, a_1, a_{0,}) \; and \; B = (b_{s-1}, \cdots, b_1, b_{0,})$.

OUTPUT: Product $= A.B \; (P_{2s-1}, \cdots, P_1, P_0)$

```
1: $(t,u,v) \leftarrow 0$
2: for i from 0 by 1 to s do
3:    for j from 0 by 1 to i do
4:        $(t,u,v) \leftarrow (t,u,v) + a_j \times b_{i-j}$
5:    end for
6:    $P_i \leftarrow v$
7:    $v \leftarrow u, \; u \leftarrow t, t \leftarrow 0$
8: end for
9: for i from  s by 1 to 2s-1 do
10:    for j from i+1--s by 1 to s do
11:        $(t,u,v) \leftarrow (t,u,v) + a_j \times b_{i-j}$
12:    end for
13:    $P_i \leftarrow v$
14:    $v \leftarrow u, \; u \leftarrow t, \; t \leftarrow 0$
14: end for
15: $P_{2s-1} \leftarrow v$
```

In addition to the replacement of the multiplication algorithm, we also propose parallelizing the Comba algorithm which is elaborated in section 6.3.1.

In this work, we considered the FIPS 186-2 standard that is used to provide different moduli illustrated below that can help creating fast reduction algorithms especially on word size 32. We also provide an example of these algorithms (Algorithm 6.2). Therefore, for further information and algorithms, we refer the reader to [1] section 2.2.6:

$$P_{192} = 2^{192} - 2^{64} - 1$$

$$P_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$$

$$P_{384} = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$$

Algorithm 6.2 Fast Reduction modulo$P_{256} = 2^2 + 2^{224} + 2^{192} + 2^{96} - 1$

```
INPUT: An integer $c = (c_{15}, \cdots, c_2, c_1, c_0)$ in base $2^{32}$ with $0 \leq c < P_{256}^2$ .
OUTPUT: $c\ modP_{256}$

    1) $Define\ 256 - bit\ integers$

        $s_1 = (c_8, c_7 c_6, c_4, c_3 c_2, c_1, c_0),$
        $s_2 = (c_{15}, c_{14}, c_{13}, c_{12}, c_{11}, 0,0,0),$
        $s_3 = (0, c_{15}, c_{14}, c_{13}, c_{12}, 0,0,0),$
        $s_4 = (c_{15}, c_{14}, 0,0,0, c_{10}, c_9, c_8),$
        $s_5 = (c_8, c_{13}, c_{15}, c_{14}, c_{13}, c_{11}, c_{10}, c_9),$
        $s_6 = (c_{10}, c_8, , 0,0,0, c_{13}, c_{12}, c_{11}),$
        $s_7 = (c_{11}, c_9, 0,0,0, c_{15}, c_{14}, c_{13}, c_{12}),$
        $s_8 = (c_{12}, 0, , c_{10}, c_9, c_8, c_8, c_{15}, c_{14}, c_{13}),$
        $s_9 = (c_{13}, 0, c_{11}, c_{10}, 0, c_{15}, c_{14}),$

    2) Return $(s_1 + 2s_2 + 2s_3 + s_4 + s_5 - s_6 - s_7 - s_8 - s_9)\ mod\ P_{256}$
```

6.3 Proposed Design ECC point Multiplication over Prime Fields

The point multiplication is to be accomplished by adding a point to itself for a certain number of times and denoted by *kP*, where k is a scalar number of times that we intend to add *P* to itself. For example, *3P* could be literally represented as *P+P+P*, which specifically consist of point double and addition. The scalar point multiplication dominates the execution time for Elliptic Curve Cryptography scheme. Because of this, several algorithms have been proposed to help enhance scalar point computation, such as [81, 84] and [83]. [19] proposed a Montgomery Ladder with (X, Y)-only co-Z addition algorithm suggested by [83] to conduct scalar point multiplication. The algorithm contains three main computations that include: (n-1) XYCZ-ADDC algorithm, (n-1) XYCZ-ADD computations, point doubling algorithm and Final inversion of Z operations.

6.3.1 Proposed Elliptic Curve Point Representation

Basically, (x,y) points are to be represented by the coordinate referred to as Affine Coordinates (A). However, it is very common practice that projective coordinates are used in replacing the Affine Coordinates. This is because Affine Coordinates over prime field is costly due to the operations of field inversion which are required during the Elliptic Curve Scalar Point Multiplication (ECSPM) computations.

There are several types of projective coordinates that can help avoid the inversion operations; these include Standard Projective coordinates, Jacobian Projective Coordinates and Chudnovsky coordinates. Hence, we used nano-ecc [19], which is an open-source library, and the Jacobian Coordinates is considered in this work. The selection of this type is based on the good results reported by [1] and [65], [69],[85].

6.3.2 Proposed Parallel Comba Multiplication over Prime Fields

Finite field multiplication is an important operation for every ECC system. [19] proposed to use the Comba algorithm designed by [85]. However, we noticed this algorithm could be optimized using the Comba algorithm, as shown in Algorithm 6.1, and which was proposed by [17] and [18]. Comba algorithm mainly consists of two outer loops and two simple inner loops to perform a bulk of computations.

Algorithm 6.3 Modified Comba Multiplication Technique

```
INPUT: $A = (a_{s-1}, \cdots, a_1, a_{0,})$ and $B = (b_{s-1}, \cdots, b_1, b_{0,})$.
OUTPUT: Product $= A.B\ (P_{2s-1}, \cdots, P_1, P_0)$

1: $(t, u, v) \leftarrow 0$
2: par{
3: for i from 0 by 1 to s do
4:     for j from 0 by 1 to i do
5:         $(t, u, v) \leftarrow (t, u, v) + a_j \times b_{i-j}$
6:     end for
7:     $P_i \leftarrow v$
8:     $v \leftarrow u,\ u \leftarrow t, t \leftarrow 0$
10:   end for
11:   end par}
12: par{
13:  for i from  s by 1 to 2s-1 do
14:      for j from i+1--s by 1 to s do
15:          $(t, u, v) \leftarrow (t, u, v) + a_j \times b_{i-j}$
16:      end for
17:      $P_i \leftarrow v$
18:      $v \leftarrow u,\ u \leftarrow t,\ t \leftarrow 0$
19:  end for
20: end par}
21: $P_{2s-1} \leftarrow v$
```

During each iteration, the inner loops are responsible for performing a Multiplication and Accumulate operations, in which $2w$-bit words is multiplied, and in accordance to that, $2w$-bit product is to be added to the cumulative sum. However, the output of sum operation is most likely to be longer than $2w$ bit, which will require three w-bit registers to store them. In fact, as shown Algorithm 6.1, cumulative sum process is presented by *(t,u,v)* which are used to present the integer value $t.2^{2w} + u.2^{w} + v$. Meanwhile, the operations carried out in line 7 and 14 illustrated in Algorithm 6.1 is used to represent w-bit right-shift of the cumulative sum *(t,u,v)*.

6.3.3 Proposed Parallel Jacobian Point Doubling

The point doubling formula used in this library is based on the modified point doubling formula described by [83]. As it can be seen from the following formula, the cost of point doubling operations has been reduced from 4M+6S+8A for the general Jacobian case to 4M+4S+9A [1]:

$$X_3 = B^2 - 2A, Y_3 = B(A - X_3) - Y_1^4, Z_3 = Y_1 Z_1 \qquad (6.2)$$

where $A = \frac{3(X_1+Z_1^2)(X_1-Z_1^2)}{2}$ $and\ B = Y_1 Z_1$ with formula (6.2).

The above algorithm can be calculated using 6 field registers when $a = -3$. In this library, a modified Jacobin point doubling (Algorithm 14) developed in [83] has been selected. However, we propose a modification to speed up the point doubling computation process.

Our proposed modification is based on performing parallel operations for finite field operations that are to be executed within the point doubling algorithm. This has been achieved after conducting Read After Write (RAW) investigation in the algorithm to extract possible parallel operations. Accordingly, we managed to conduct several parallel operations in point doubling that are compatible with our target platform. For example, we parallelized step 3 ($T_4 \leftarrow T_4^2$) with step 4 ($T_2 \leftarrow T_2 . T_3$) since there is no data dependency. Using this methodology and in line with careful rescheduling, we managed to reduce the point doubling

Algorithm 6.4 Modified Jacobian doubling ($\mathbf{a = -3}$)

```
INPUT: $\boldsymbol{P} \equiv (X_1, Y_1, Z_1)$
OUTPUT: : $\mathbf{2}\boldsymbol{P}(X_3, Y_3, Z_3)$

    $T_1 = X_1, T_2 = Y_1, T_3 = Z_1)$
1: $T_4 \leftarrow T_2^2$                                   $[Y_1^2]$
2: $T_5 \leftarrow T_1 . T_4$                               $[X_1 Y_1^2 = A]$
3: $T_4 \leftarrow T_4^2$  $[Y_1^4]$|| $T_2 \leftarrow T_2 . T_3$ $[Y_1 Z_1 = Z_3]$||
 $T_3 \leftarrow T_3^2$ $[Z_1^2]$ [ Parallel Operations]
4:$T_1 \leftarrow T_1 + T_3$ $[X_1 + Z_1^2]$ || $T_3 \leftarrow T_3 + T_3$ $[2Z_1^2]$  [Parallel Operation]
5: $T_3 \leftarrow T_1 - T_3$                               $[X_1 - Z_1^2]$
6: $T_1 \leftarrow T_1 . T_3$                               $[X_1^2 - Z_1^4]$
7: $T_3 \leftarrow T_1 + T_1$                               $[2(X_1^2 - Z_1^4)]$
8: $T_1 \leftarrow T_1 + T_3$                               $[3(X_1^2 - Z_1^4)]$
9:$T_1 \leftarrow T_1 / 2$                                  $[\frac{3}{2(X_1^2 - Z_1^4)} = B]$
10: $T_3 \leftarrow T_1^2$                                  $[B^2]$
11: $T_3 \leftarrow T_3 - T_5$                              $[B^2$-A]
12: $T_3 \leftarrow T_3 - T_1$                              $[B^2 - 2A = X_3]$
13: $T_5 \leftarrow T_5 - T_3$                              $[A - X_3]$
14: $T_1 \leftarrow T_1 . T_5$                              $[B(A - X_3)]$
15: $T_1 \leftarrow T_1 - T_4$                              $[A - X_3) - Y_1^4 = Y_3]$

Return $(T_3, T_1, T_2)$
```

steps to 15 steps compared to 18 steps reported by [83]. The following algorithm provided a low-level description for the new parallel Modified Jacobian Algorithm depicted in [83].

6.3.4 Proposed Parallel co-Z addition point doubling addition

[19] has taken into account the advantages of using co-Z addition proposed by [83]. The co-Z addition works by sharing the same Z-coordinate for the two input points. Let $P = (X_1, Y_1, Z)$ and $Q = (X_3, Y_3, Z_3)$ nominated as co-Z addition of P and Q (with $(P \neq Q)$ is defined as $P + Q = (X_3, Y_3, Z_3)$, where:

$$X_3 = D - (B + C), Y_3 = (Y_2 - Y_1)(B - X_3) - E \; and \; Z_3 = Z(X_2 - X_1) \tag{6.3}$$

with $A = (X_2 - X_1)^2, B = X_1A,\ C = X_2A, D = (Y_2 - Y_1)^2\ and\ E = Y_1(C - B)$ (6.4)

This mechanism yields very efficient co-Z point addition with a cost of 5M+2S+7A point addition. In this work, we proposed parallel co-Z point addition by removing dependencies to help speed up the point addition process. Therefore, we reduced the steps required for trigging the co-Z point addition to 7 steps compared to the 13 steps reported by [83], as shown in Algorithm 6.4.

Algorithm 6.5 Modified (X, Y)- only co-Z addition with update XYCZ -

```
INPUT: $(X_1, Y_1)$ and $(X_2, Y_2) s.t\ P \equiv (X_1: Y_1: Z)\ and\ Q \equiv (X_2: Y_2: Z)$for some $Z \in F_q, P, Q \in E(F_q)$
OUTPUT: : $(X_3, Y_3)$ and $(X'_2, Y'_2) s.t\ P \equiv (X'_1: Y'_1: Z_3)\ and\ P + Q \equiv (X_3: Y_3: Z_3)$for some $Z_3 \in F_q$

   $(T_1 = X_1, T_2 = Y_1, T_3 = X_2, T_4 = Y_2)$
1: $T_5 \leftarrow T_3 - T_1$                    $[X_2 - X_1]$
2: $T_5 \leftarrow T_5^2$                        $[(X_1 - X_1)^2 = A]$
3: $T_1 \leftarrow T_1.T_5$ $[X_1A = B]$|| $T_3 \leftarrow T_3.T_5$ $[X_2A = C]$[Parallel
Operations]
4:$T_4 \leftarrow T_4 - T_2$                     $[Y_2 - Y_1]$
5: $T_5 \leftarrow T_4^2$                        $[(Y_2 - Y_1)^2 = D]$
6: $T_5 \leftarrow T_5 - T_1$                    $[D - B]$
7: $T_5 \leftarrow T_5 - T_3$  $[X_3]$ || $T_3 \leftarrow T_3 - T_1$  $[C - B]$
   $T_2 \leftarrow T_2.T_3$  $[Y_1(C - B)]$ || $T_3 \leftarrow T_1 - T_5$    $[B - X_3]$
   $T_4 \leftarrow T_4.T_3$ $[(Y_2 - Y_1)(B - X_3)]$|| $T_4 \leftarrow T_4 - T_2$  $[Y_3]$
[Parallel Operations]

Return $((T_5, T_4)(T_1, T_2))$
```

In addition to the proposed parallelization shown in Algorithm 6.4, we advocate a new way for parallelizing a sequence of multiple of parallel operations within one of the outer parallel loop to help speed the co-Z addition. Also, [83] proved the possibilities to obtain the conjugate $\mathrm{P - Q}$ coordinate with little cost, since it is sharing the same Z-coordinates with $\mathrm{P + Q}$. In which, $\mathrm{P - Q = (X_3', Y_3', Z_3)}$ where:

$$X_3' = F - (B + C), Y_3' = (Y_1 + Y_2)(X_3 - B) - E\ and\ Z_3 = Z(X_2 - X_1) \qquad (6.5)$$

with $\mathrm{F = (Y_1 + Y_3)^2}$ and $\mathrm{(A, B, C, D, E)}$ as defined in (4). In [83], the authors proposed co-Z conjugate addition algorithm with 19 steps with cost of 5M+3+16A by involving 6 field registers in this operation. However, in this work, the proposed approach requires just 15 steps as shown in the Algorithm 6.4.

6.4 Implementation Details

In this work, we implemented an efficient parallel ECC on an xmos start kit development board. This development board consists of 8 logical cores CPU allowing parallel execution design using XC programming language and C codes. We used xTIMEcompser development tool to execute our implementation and to perform timing analysis.

6.5 Result Analysis

In this work, we first implemented a sequential implementation for ECC in xmos device for the four prime curves, and performance results are illustrated in Figure 6.1.

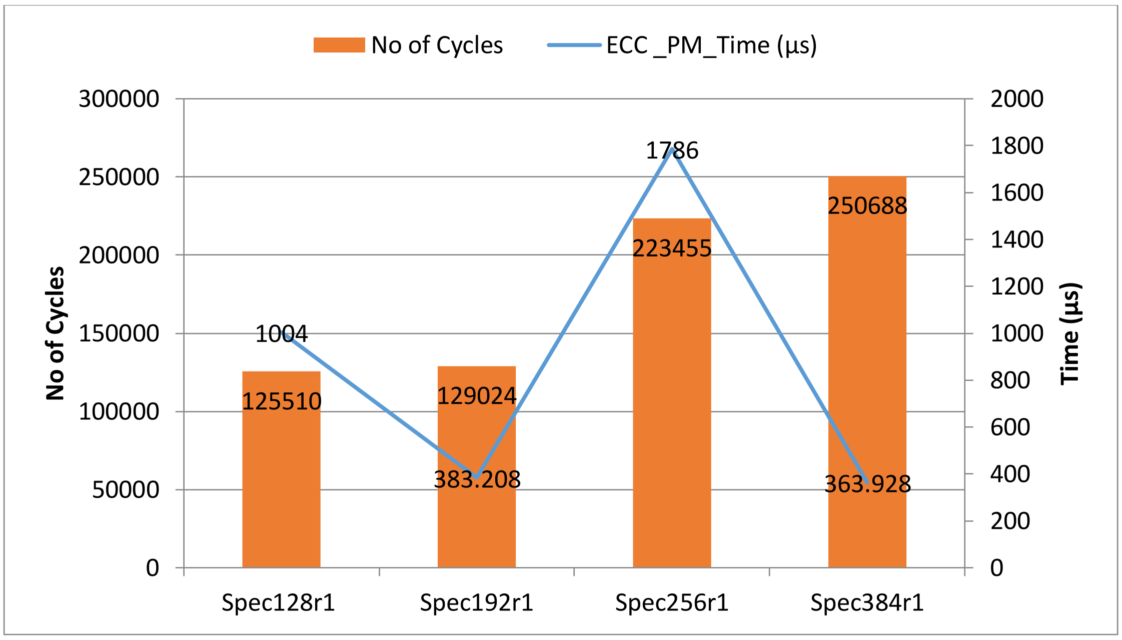


Figure 6.1 Sequential Single Core ECC Scalar Point Multiplication in xmos

Then, we implemented our parallelization in Comba algorithm, point doubling, and point Algorithms, as shown in Figure 6.2.

As it can be seen from these figures, we managed to reduce the time and number clock cycles for computing ECC scalar point multiplication by 85% for some curves. In this work, we utilized 6 logic cores from the 8 logic cores provided by the xmos start kit.

To our knowledge, our work is the first attempt to implement ECC scalar point multiplication over prime field *GF(P)* on a homogenous multicore microcontroller. We reported the latest contribution concerned with software parallel implementation. In this table, we tried to put context more than like-for-like comparison, as the platforms are not comparable with respect to resource availability. To sum up, our implementation shows the feasibility of using parallel approach in a homogenous multicore microcontroller to improve ECC performance in constrained devices. This is an attractive option in case of low-resource applications that contribute to towards enabling the strong public key cryptography schemes and protocols to be implemented faster in smaller devices.

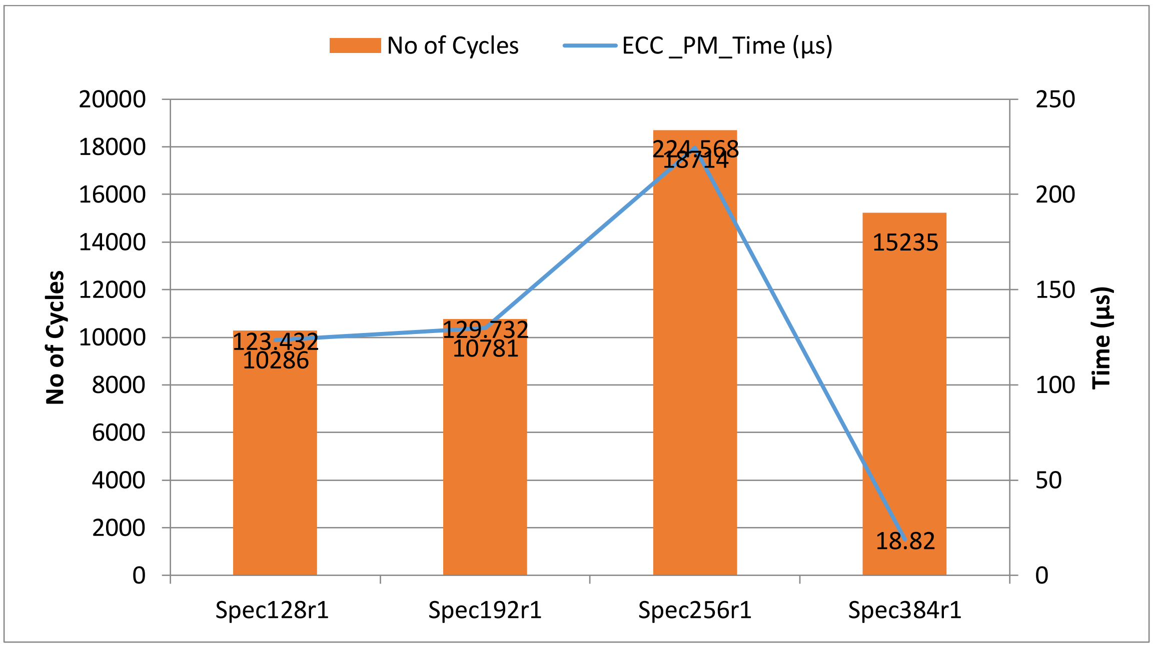


Figure 6.2 Parallel Multicore ECC Scalar Point Multiplication in xmos

Table 6.1 Comparison With State-Of-Art

Author	Platform	Time Execution (s)	Algorithm	Clock Cycles	Operation Field GF(P)
[4]	IMX6Quad- 1.00 GHz.	0.84 µs	1 Thread- Best seq.	--------	128
		0.47 µs			256
		0.61 µs			384
		2.21 µs	2 Threads - Montgomery Bipartite		128
		0.85 µs			256
		0.97 µs			384
		1.01 µs	3 Threads - Montgomery 2-ary Multi. v2		128
		1.17 µs			256
		1.10 µs			384
		1.16 µs	4 Threads Montgomery 4-ary Multi. v2		128
		1.23 µs			256
		1.32 µs			384
	Intel Core I7	1.22 µs	1 Thread- Best seq.	--------	128
		1.84 µs			256
		2.78 µs			384
		2.95 µs	2 Threads - Montgomery Bipartite		128
		3.52 µs			256
		3.86 µs			384
		3.10 µs	3 Threads - Montgomery 2-ary Multi. v2		128
		3.62 µs			256
		2.08 µs			384
		3.52 µs	4 Threads Montgomery 4-ary Multi. v2		128
		2.01 µs			256
		2.22 µs			384
[8]	Intel Core i7r-2600 Sandy Bridge 3.4GHz,	--------------	NAF D&A	207000	WCurve25519
			NAF D&A	176000	JQCurve25519
[9]	AMD Athlon 64x2, 2.4GHz Dual Core Processor	3.203 ms	Karatsuba	———	32
		7.187 ms			64
		13.203 ms			96
		18.203 ms			128
		28.188 ms			160
		47.203 ms			192
		3.187 ms	Montgomery		32
		6.219 ms			64
		12.219 ms			96
		16.219 ms			128
		26.234 ms			160
		46.234 ms			192
This work	**Xmos strat kit – 6 logical cores**	**123.432µs**	**Montgomery Ladder**	**10286**	**128**
		129.732 µs		**10781**	**192**
		224.568 µs		**18714**	**256**
		18.82		**15235**	**384**

6.6 Conclusion

In this chapter, we addressed the practical feasibility of parallel software implementation of ECC scalar point multiplication on a homogeneous multicore microcontroller. In particular, we proposed an efficient ECC scalar point multiplication implementation which can be hosted on the xmos start kit so that it scales to target different ECC standard curves underlying prime field GF(P) recommended by SECG [16]. To maximize the performance of ECC point multiplication on multicore microcontroller, three novel modified parallelization have been proposed. These include: the parallelization of finite field multiplication Comba algorithm, point doubling and point addition algorithms. This implementation runs the whole ECC point multiplication in only 123µs for Spec128r1, 129µs for Spec192r1, 224µs for Spec256 and 18.82µs for Spec384r1. Potentially, our proposed methods were able to boost cryptography operations and provide the potential of having strong public key cryptography in parallel software implementation with high performance and flexibility using a multicore microcontroller platform.

Chapter 7 : Conclusions and Future Research Work

This chapter provides an overview of the research and the new concepts offered in this thesis. In addition to that, a number of the suggested future works and open up ideas for further researches are discussed here.

7.1 Summary and Conclusions

In this research, we have specifically concentred on providing a cutting-edge contribution for enhancing the Elliptic Curve Cryptography performance on constrained devices. Our proposed solutions could be used in different applications, such as WSN, WSBN and IoT. Knowing that these applications are a microcontroller-based technology and providing communication, security is one of the major challenges. This is because of the microcontroller limitations, such as speed of the microcontroller built-in processor, the power consumption and the number of input/output ports. Thus, trying to encapsulate a powerful a symmetric cryptography scheme such as ECC is one of the concerns that required higher attention and research. Therefore, in this work, we managed to provide solutions where a powerful software implementation of ECC is implemented in microcontrollers suitable for such mentioned applications.

Throughout the third chapter of this thesis, we proposed enhancing the performance of ECC in arduino 8-bit and 16-bit microcontrollers. In this work, we imported a very well-known modern cryptographic supporting C code relic-toolkit. A relic toolkit managed to provide high efficiency and flexibility of modern cryptography. Also, the toolkit could support different types of cryptographies protocols such EDSA, ECDH, ECMQV and ECSS. Additionally, it supports a wide range of configurable structure algorithms, which could be configured during the relic setup. In particular, in this work, relic-0.3.1 is imported onto the two platform boards, and we experimented with the performance of ECDSA and ECDH over binary fields using different NIST curves standard (NIST-K163,NIST-B163). Accordingly, we enhanced the performance of ECDSA by providing the best combination of algorithms in the relic presets and succeed to report better performance in arduino DUE compared to arduino mega2560.

In the fourth chapter, an effort has been taken to enhance multiplication operation of $GF(2^{163})$. Two novelties have been demonstrated here. Firstly, we managed to parallelize Comba algorithm using a Homogenous multi core microcontroller (XMOS). This, in fact, results in much better performance compared to a single core Comba multiplication in software implementation in a microcontroller.

A second novel claim is that we modified a fast modular reduction 32-bit algorithm to support 8-bit in $GF(2^{163})$ and get it integrated with Comba algorithm. In this, we were able to report better performance of Comba algorithm for both implementations of parallel Comba without reduction and Comba multiplication with reduction.

In Chapter 5, we illustrate the capability of a homogenous multicore microcontroller (XMOS) to improve overall performance of scalar point multiplication for ECC implementation over $GF(2^m)$. Therefore, we hereby summarized the novelties related to this work:

A new modified parallelized Lopez-Dahab point doubling been proposed here. Such a novel idea helps us to obtain high performance for overall ECC scalar point multiplication. Our method shows that we were able to reduce the number of steps from 14 steps of point doubling operations algorithm to only 9 steps.

The second novel contribution concerns on modifying the Lopez-Dahab point addition algorithm and accordingly the parallelizing some of its steps. We also managed to reduce the algorithm steps from 26 steps to 20 steps using parallelization mechanisms. Furthermore, we also introduce the concept of parallelizing inside a parallelized round.

Thirdly, we enhanced the overall performance of Left to Right Double and Add Scalar Point Multiplication algorithm. This was achieved by designing the algorithm that should work initially by performing the initial scanning most significant bits *(MSB*) of k in order to track down the first none zero bit from *MSB*. If the non-zero first bit is found, then *P's* coordinates will be filled in Q to start loop operation.

From all over all that, in this work, we able to report much higher performance in ECC point multiplication over binary fields for all related standardized curves, including: $GF(2^{163}), GF(2^{233}), GF(2^{283}), GF(2^{409}), GF(2^{571})$.

In addition to all that is mentioned above, we tried to tackle improving the ECC over constrained devices from different perspectives. Thus, in Chapter 6, we managed to obtained better performance of ECC point multiplication over the prime field $GF(P)$. We also here claimed three more novelties that could be summarized as below:

The first claimed novelty is parallelizing the Comba Algorithm for $GF(P)$ gets the benefits from the previous novelty of parallelizing Comba Algorithm in $GF(2^{163})$. In this, we managed to parallelize the multiplication arithmetic of ECC over $GF(P)$.

Secondly, we proposed Parallel Jacobian Point Doubling over $GF(P)$. Implementing such a solution helps us to reduce the steps involved in a sequential manner from 18 steps to 15 steps. Such enhancement positively impacted the overall performance or ECC point multiplication over $GF(P)$.

The third novelty has been achieved by parallelizing co-Z addition point addition. In this approach, we proposed a parallel implementation for this algorithm and managed to reduce the steps to 7 steps compared to 13 steps proposed by the original algorithm.

In this thesis, we considered how to improve the efficiency of software implementation for Elliptic Curve Cryptography on Microcontroller platform. Therefore, for a purpose of our research we considered implementation mainly on three types of Microcontrollers, which include: Arduino Mega2560 with 8-bit microcontroller, Arduino DUE with 32-bit ARM processor Microcontroller and XMOS start Kit. We also confirm the scalability of implementing different types of curves per SEC1 standard, in which different SEC1 curves have been implemented for both $GF(2^m)$ and $GF(P)$ and bench mark results have been presented.

In fact, several attempts have been made to improve the performance of ECC on software implementation methodology. Besides that, there are also different efforts to boost efficiencies using assembly programming language or by modifying the existing related algorithms. However, none of these attempts have tried to improve the ECC performance using a homogenous multicore microcontroller.

Therefore. to our knowledge proposed solutions herein for boosting the ECC performance in a homogenous device is the first ever proposal of enhancing the ECC performance in such type of microcontroller at the time of writing this thesis. It is also worth mentioning that our proposed solution could be easily acquired to be smoothly integrated with any application using XMOS microcontroller or arduino microcontroller. Through this, ECC is to be used as a security service provider in these platforms.

7.1 Future Research Works

Our aim in this thesis was to come up with solutions that can help with implementing ECC on a constrained microcontroller. Although we had shown a significant improvement of the ECC performance, there are still some other concerns that could be considered for future work or that remain as open problems as provided below:

Besides the speed enhancement the ECC performance that we have achieved in this work, power consumption is one of the counter measures part that need to taken into consideration. Considering that, our implementation is proposed for a microcontroller to be used in applications (WSN, IoT) that are highly depending on low-power resources like batteries. Introducing such ECC algorithms alongside the process and communication related to the data sensing and others inside these microcontroller should be highly evaluated and addressed.

One of challenges that needs to considered while implementing cryptography for any system is studying the strength of a security system from any cryptanalysis mechanisms and algorithms. In fact, lightweight applications, such as the devices we used in our work, need to evaluated against any side-channel attacks. Using strong cryptography in such applications does not mean that these devices are strong enough to defend from any side-channel attack. The side-channel attack could be implemented in a passive manner, where time and power are simultaneously used during the attack. One possible way to defend such an attack is to use resistance algorithms, but such a solution requires large memory and double execution time. Therefore, a deep investigation on how to prevent the ECC on XMOS and arduino devices from side-channel attacks is one of the issues that could further investigated.

Another potential future work is parallelizing lattice-based cryptography on a homogeneous multicore microcontroller, which is an area of research that needs to be tackled. One advantage of lattice-based cryptography in any system include its ability to defend from a quantum attack. So, getting such cryptography in a constrained well adds a lot of security benefit to these devices and applications involved.

End-to-end encryption for parallelizing ECC using XMOS devices is one of the open problems we suggested here. Therefore, we recommend implementing our proposed solutions to check the effectiveness in a real-time network environment. This environment might be WSN or IoT.

Examining our proposed solution with different platforms, such as parallax, is one of open problems, since we have examined our solutions with only one type of a homogenous multicore microcontroller. This work can help to understand how other platforms will be able to react and cope with such solutions.

To conclude, the existing growth of technology shows a very high demand on Wireless Sensor Network and IoT. Such an application requires a powerful security system that could prevent them from different types of attacks. However, until recently, people used to hesitate to implement cryptography in these constrained devices due to the microcontroller limitation. This lead to widely impression that implementing cryptography in such devices is infeasible, since a symmetric cryptography is excessively massive and cannot be accommodated in such devices. This could additionally contribute towards scaling down the entire efficiency of a microcontroller.

[1] D. Hankerson, S. Vanstone, and A. J. Menezes, *Guide to elliptic curve cryptography*. Springer, 2004.

[2] The Basics of ECC [Online]. Available: https://www.certicom.com/index.php/the-basics-of-ecc

[3] (2016). *IntelÂ® Parallel Studio XE 2016: High Performance for HPC Applications and Big Data Analytics | IntelÂ® Developer Zone*. Available: https://software.intel.com/en-us/blogs/Intel-Parallel-Studio-XE-2016

[4] Y. R. Venturini and U. Sorocaba, "Performance analysis of parallel modular multiplication algorithms for ECC in mobile devices," *Revista de Sistemas de Informaçao da FSMA,* no. 13, pp. 57-67, 2014.

[5] D. F. Aranha, R. Dahab, J. López, and L. B. Oliveira, "Efficient implementation of elliptic curve cryptography in wireless sensors," *Advances in Mathematics of Communications,* vol. 4, no. 2, pp. 169-187, 2010.

[6] C. Negre and J.-M. Robert, "New parallel approaches for scalar multiplication in elliptic curve over fields of small characteristic," *Computers, IEEE Transactions on,* vol. 64, no. 10, pp. 2875-2890, 2015.

[7] A. Kargl, S. Pyka, and H. Seuschek, "Fast Arithmetic on ATmega128 for Elliptic Curve Cryptography," *IACR Cryptology ePrint Archive,* vol. 2008, p. 442, 2008.

[8] J.-M. Robert, "Software Implementation of Parallelized ECSM over Binary and Prime Fields," 2014.

[9] U. S. Kanniah and A. Samsudin, "Multi-threading elliptic curve cryptosystems," in *Telecommunications and Malaysia International Conference on Communications, 2007. ICT-MICC 2007. IEEE International Conference on*, 2007, pp. 134-139: IEEE.

[10] C. P. Gouvêa, L. B. Oliveira, and J. López, "Efficient software implementation of public-key cryptography on sensor networks using the MSP430X microcontroller," *Journal of Cryptographic Engineering,* vol. 2, no. 1, pp. 19-29, 2012.

[11] S. Tillich and J. Großschädl, "A simple architectural enhancement for fast and flexible elliptic curve cryptography over binary finite fields GF (2 m)," in *Advances in Computer Systems Architecture*: Springer, 2004, pp. 282-295.

[12] L. B. Oliveira *et al.*, "TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks," *Computer Communications,* vol. 34, no. 3, pp. 485-493, 2011.

[13] C. Alcaraz, J. Lopez, R. Roman, and H.-H. Chen, "Selecting key management schemes for WSN applications," *Computers & Security,* vol. 31, no. 8, pp. 956-966, 2012.

[14] (2015). *What is startKIT? | XMOS*. Available: http://www.xmos.com/startKit/what

[15] (2014). *sec1_final.pdf (application/pdf Object)*. Available: http://www.secg.org/collateral/sec1_final.pdf

[16] D. F. A. a. C. P. L. Gouv. *RELIC is an Efficient LIbrary for Cryptography*. Available: http://code.google.com/p/relic-toolkit/

[17] A. Szekely and S. Tillich, "Algorithm exploration for long integer modular arithmetic on a SPARC V8 processor with cryptography extensions," in *null*, 2005, pp. 187-194: IEEE.

[18] R. Brumnik, V. Kovtun, and A. Okhrimenko, "Techniques for performance improvement of integer multiplication in cryptographic applications."

[19] (2016). *iSECPartners/nano-ecc*. Available: https://github.com/iSECPartners/nano-ecc

[20] B. A. Forouzan, *Cryptography And Network Security (Sie)*. Tata McGraw-Hill Education, 2011.

[21] (2016). *http://www.billthelizard.com/2009/05/brief-history-of-cryptography.html*.

[22] D. Kahn, *The codebreakers*. Weidenfeld and Nicolson, 1974.

[23] F. Cohen, "A short history of cryptography," *Fred Cohen & Associates,* 2001.

[24] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 2010.

[25] S. William and W. Stallings, *Cryptography and Network Security, 4/E*. Pearson Education India, 2006.

[26] C. Paar and J. Pelzl, *Understanding cryptography: a textbook for students and practitioners*. Springer, 2010.

[27] S. S. Kumar, "Elliptic curve cryptography for constrained devices," Ruhr University Bochum, 2006.

[28] R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications*. Cambridge university press, 1994.

[29] I. Branovic, R. Giorgi, and E. Martinelli, "Instruction Set Extensions for Elliptic Curve Cryptography over Binary Finite Fields."

[30] ECC Holds Key to Next-Gen Cryptography [Online]. Available: http://www.design-reuse.com/articles/7409/ecc-holds-key-to-next-gen-cryptography.html

[31] F. Brechenmacher, "A history of galois fields," 2012.

[32] E. W. Weisstein. (2014). *Finite Field -- from Wolfram MathWorld* [Text]. Available: http://mathworld.wolfram.com/FiniteField.html

[33] P. G. Comba, "Exponentiation cryptosystems on the IBM PC," *IBM systems journal,* vol. 29, no. 4, pp. 526-538, 1990.

[34] D. Knuth, "The Art of Computer Programming. Seminumerical Algorithms, vol. 2, 1981," *Distributed Sensor Networks International Journal of Mechanical Engineering Advances in*.

[35] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," in *Soviet physics doklady*, 1963, vol. 7, p. 595.

[36] C. Koc and T. Acar, "Montgomery Multiplication in GF(2k)," *An International Journal,* vol. 14, no. 1, pp. 57-69.

[37] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation,* vol. 44, no. 170, pp. 519-521, 1985.

[38] D. Hankerson, J. L. Hernandez, and A. Menezes, "Software implementation of elliptic curve cryptography over binary fields," in *Cryptographic Hardware and Embedded Systems—CHES 2000*, 2000, pp. 1-24: Springer.

[39] S. Certicom, "SEC 2: Recommended elliptic curve domain parameters," *Proceeding of Standards for Efficient Cryptography, Version,* vol. 1, 2000.

[40] N. Koblitz, *A course in number theory and cryptography*. Springer Science & Business Media, 1994.

[41] N. Koblitz, A. Menezes, and S. Vanstone, "The state of elliptic curve cryptography," *Designs, codes and cryptography,* vol. 19, no. 2-3, pp. 173-193, 2000.

[42] R. C. Merkle, "Secure communications over insecure channels," *Communications of the ACM,* vol. 21, no. 4, pp. 294-299, 1978.

[43] E. Barker, L. Chen, A. Roginsky, and M. Smid, "Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography," *NIST special publication,* vol. 800, p. 56A, 2013.

[44] A. Liu and P. Ning, "TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks," in *Information Processing in Sensor Networks, 2008. IPSN'08. International Conference on*, 2008, pp. 245-256: IEEE.

[45] S. C. Seo, H. Dong-Guk, H. C. Kim, and H. Seokhie, "TinyECCK: Efficient Elliptic Curve Cryptography Implementation over< I> GF</I>(< I> 2</I>< I>< SUP> m</SUP></I>) on 8-Bit Micaz Mote," *IEICE transactions on information and systems,* vol. 91, no. 5, pp. 1338-1347, 2008.

[46] (2014). *AVRCryptoLib*. Available: http://www.emsign.nl/

[47] (2014). *Arduino - HomePage*. Available: http://www.arduino.cc/

[48] M. Sethi, J. Arkko, and A. Keranen, "End-to-end security for sleepy smart object networks," in *Local Computer Networks Workshops (LCN Workshops), 2012 IEEE 37th Conference on*, 2012, pp. 964-972.

[49] T. S. Denis, *BigNum Math: Implementing Cryptographic Multiple Precision Arithmetic*. Syngress Publishing, 2006.

[50] M. Albahri and M. Benaissa, "Parallel comba multiplication in GF (2163) using homogenous multicore microcontroller," in *Electronics, Circuits, and Systems (ICECS), 2015 IEEE International Conference on*, 2015, pp. 641-644: IEEE.

[51] N. Gura, A. Patel, A. Wander, H. Eberle, and S. C. Shantz, "Comparing elliptic curve cryptography and RSA on 8-bit CPUs," in *Cryptographic Hardware and Embedded Systems-CHES 2004*: Springer, 2004, pp. 119-132.

[52] R. Brumnik, V. Kovtun, A. Okhrimenko, and S. Kavun, "Techniques for Performance Improvement of Integer Multiplication in Cryptographic Applications," *Mathematical Problems in Engineering,* vol. 2014, 2014.

[53] M. Hutter and E. Wenger, "Fast multi-precision multiplication for public-key cryptography on embedded microprocessors," in *Cryptographic Hardware and Embedded Systems–CHES 2011*: Springer, 2011, pp. 459-474.

[54] A. Munir, A. Gordon-Ross, and S. Ranka, "Multi-core embedded wireless sensor networks: Architecture and applications," *Parallel and Distributed Systems, IEEE Transactions on,* vol. 25, no. 6, pp. 1553-1562, 2014.

[55] H. Seo, Y. Lee, H. Kim, T. Park, and H. Kim, "Binary and prime field multiplication for public key cryptography on embedded microprocessors," *Security and Communication Networks,* vol. 7, no. 4, pp. 774-787, 2014.

[56] M. Hutter and P. Schwabe, "Multiprecision multiplication on AVR revisited," 2014.

[57] H. Seo and H. Kim, "Implementation of Multi-Precision Multiplication over Sensor Networks with Efficient Instructions," *Journal of Information and Communication Convergence Engineering,* vol. 11, no. 1, pp. 12-16, 2013.

[58] H. Seo and H. Kim, "Optimized multi-precision multiplication for public-key cryptography on embedded microprocessors," *International Journal of Computer and Communication Engineering,* vol. 2, no. 3, pp. 255-259, 2013.

[59] Z. Liu and J. Großschädl, "New speed records for montgomery modular multiplication on 8-bit AVR microcontrollers," in *Progress in Cryptology–AFRICACRYPT 2014*: Springer, 2014, pp. 215-234.

[60] D. F. Aranha, J. López, and D. Hankerson, "Efficient software implementation of binary field arithmetic using vector instruction sets," in *Progress in Cryptology–LATINCRYPT 2010*: Springer, 2010, pp. 144-161.

[61] V. Kovtun and A. Okhrimenko, "Approaches for the performance increasing of software implementation of integer multiplication in prime fields," *IACR Cryptology ePrint Archive,* vol. 2012, p. 170, 2012.

[62] C. Moore, M. O'Neill, N. Hanley, and E. O'Sullivan, "Accelerating integer-based fully homomorphic encryption using Comba multiplication," in *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*, 2014, pp. 1-6: IEEE.

[63] C. Moore, M. O'Neill, E. O'Sullivan, Y. Doroz, and B. Sunar, "Practical homomorphic encryption: A survey," in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, 2014, pp. 2792-2795: IEEE.

[64] (2014). *sec2_final.pdf (application/pdf Object)*. Available: http://www.secg.org/collateral/sec2_final.pdf

[65] M. J. Quinn, *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill Education Group, 2003.

[66] D. F. Aranha, R. Dahab, J. López, and L. B. Oliveira, "Efficient implementation of elliptic curve cryptography in wireless sensors," *Adv. in Math. of Comm.,* vol. 4, no. 2, pp. 169-187, 2010.

[67] M. Albahri, M. Benaissa, and Z. U. A. Khan, "Parallel Implementation of ECC Point Multiplication on a Homogeneous Multi-Core Microcontroller," in *Mobile Ad-Hoc and Sensor Networks (MSN), 2016 12th International Conference on*, 2016, pp. 386-389: IEEE.

[68] J. Taverne, A. Faz-Hernández, D. F. Aranha, F. Rodríguez-Henríquez, D. Hankerson, and J. López, "Software implementation of binary elliptic curves: impact of the carry-less multiplier on scalar multiplication," in *Cryptographic Hardware and Embedded Systems–CHES 2011*: Springer, 2011, pp. 108-123.

[69] J. López and R. Dahab, "Fast multiplication on elliptic curves over GF (2m) without precomputation," in *Cryptographic Hardware and Embedded Systems*, 1999, pp. 316-327: Springer.

[70] F. Rodrıguez-Henrıquez, N. A. Saqib, and A. Dıaz-Pérez, "A fast parallel implementation of elliptic curve point multiplication over GF (2 m)," *Microprocessors and Microsystems,* vol. 28, no. 5, pp. 329-339, 2004.

[71] J. V. Tembhurne and S. R. Sathe, "Performance evaluation of long integer multiplication using OpenMP and MPI on shared memory architecture," in *Contemporary Computing (IC3), 2014 Seventh International Conference on*, 2014, pp. 283-288: IEEE.

[72] M. Purnaprajna, C. Puttmann, and M. Porrmann, "Power aware reconfigurable multiprocessor for elliptic curve cryptography," in *Design, Automation and Test in Europe, 2008. DATE'08*, 2008, pp. 1462-1467: IEEE.

[73] J. López and R. Dahab, "Improved algorithms for elliptic curve arithmetic in GF (2n)," in *Selected areas in cryptography*, 1998, pp. 201-212: Springer.

[74] M. Albahri and M. Benaissa, "Parallel comba multiplication in GF (2163) using homogenous multicore microcontroller," in *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 2015, pp. 641-644: IEEE.

[75] J. Großschädl and E. Savaş, "Instruction set extensions for fast arithmetic in finite fields GF (p) and GF (2 m)," in *Cryptographic Hardware and Embedded Systems-CHES 2004*: Springer, 2004, pp. 133-147.

[76] A. Munir, A. Gordon-Ross, and S. Ranka, "Multi-core Embedded Wireless Sensor Networks: Architecture and Applications," 2013.

[77] E. Wenger and M. Hutter, "Exploring the design space of prime field vs. binary field ECC-hardware implementations," in *Information Security Technology for Applications*: Springer, 2011, pp. 256-271.

[78] T. VanAmeron and W. Skiba, "Implementing efficient 384-bit NIST Elliptic Curve over prime fields on an ARM946E," in *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pp. 1-7: IEEE.

[79] P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, and R. Dahab, "NanoECC: Testing the limits of elliptic curve cryptography in sensor networks," in *Wireless sensor networks*: Springer, 2008, pp. 305-320.

[80] Z. Liu, J. Großschädl, and D. S. Wong, "Low-weight primes for lightweight elliptic curve cryptography on 8-bit AVR processors," in *Information Security and Cryptology*, 2013, pp. 217-235: Springer.

[81] K. Sakiyama, E. De Mulder, B. Preneel, and I. Verbauwhede, "A parallel processing hardware architecture for elliptic curve cryptosystems," in *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, 2006, vol. 3, pp. III-III: IEEE.

[82] S.-C. Chung, J.-W. Lee, H.-C. Chang, and C.-Y. Lee, "A high-performance elliptic curve cryptographic processor over GF (p) with SPA resistance," in *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, 2012, pp. 1456-1459: IEEE.

[83] M. Rivain, "Fast and Regular Algorithms for Scalar Multiplication over Elliptic Curves," *IACR Cryptology ePrint Archive,* vol. 2011, p. 338, 2011.

[84] P. Longa and A. Miri, "Fast and flexible elliptic curve point arithmetic over prime fields," *Computers, IEEE Transactions on,* vol. 57, no. 3, pp. 289-302, 2008.

[85] P. Longa and C. Gebotys, "Efficient techniques for high-speed elliptic curve cryptography," in *Cryptographic hardware and embedded systems, CHES 2010*: Springer, 2010, pp. 80-94.