Performance Optimization of Public Key Cryptography on Embedded Platforms

Krishna Chaitanya Pabbuleti

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Patrick R. Schaumont, Chair
Cameron D. Patterson
Michael S. Hsiao

29 April, 2014
Blacksburg, Virginia

Keywords: Elliptic Curve Cryptography, Modular Arithmetic, SIMD, Hash-based Signatures, MSP430, Wireless Sensor Node

Performance Optimization of Public Key Cryptography on Embedded Platforms

Krishna Chaitanya Pabbuleti

(ABSTRACT)

Embedded systems are so ubiquitous that they account for almost 90% of all the computing devices. They range from very small scale devices with an 8-bit microcontroller and few kilobytes of RAM to large-scale devices featuring PC-like performance with full-blown 32-bit or 64-bit processors, special-purpose acceleration hardware and several gigabytes of RAM. Each of these classes of embedded systems have unique set of challenges in terms of hardware utilization, performance and power consumption. As network connectivity becomes a standard feature in these devices, security becomes an important concern. Public Key Cryptography is an indispensable tool to implement various security features necessary on these embedded platforms. In this thesis, we provide optimized PKC solutions on platforms belonging to two extreme classes of the embedded system spectrum.

First, we target high-end embedded platforms Qualcomm Snapdragon and Intel Atom. Each of these platforms features a dual-core processor, a GPU and a gigabyte of RAM. We use the SIMD coprocessor built into these processors to accelerate the modular arithmetic which accounts for the majority of execution time in Elliptic Curve Cryptography. We exploit the structure of NIST primes to perform the reduction step as we perform the multiplication. Our implementation runs over two times faster than OpenSSL implementations on the respective platforms.

The second platform we targeted is an energy-harvested wireless sensor node which has a 16-bit MSP430 microcontroller and a low power RF interface. The system derives its power from a solar panel and is constrained in terms of available energy and computational power. We analyze the computation and communication energy requirements for different

signature schemes, each with a different trade-off between computation and communication. We investigate the Elliptic Curve Digital Signature Algorithm (ECDSA), the Lamport-Diffie one-time hash-based signature scheme (LD-OTS) and the Winternitz one-time hash-based signature scheme (W-OTS). We demonstrate that there's a trade-off between energy needs, security level and algorithm selection. However, when we consider the energy needs for the overall system, we show that all schemes are within one order of magnitude from each another.

∼ To my Parents, for all the love and support ∼

Acknowledgments

First of all, I would like to thank my research advisor, Dr. Patrick Schaumont, for all the guidance and support during my research. It was an amazing experience being a part of his research group and interacting with him. I would like to thank Dr. Cameron Patterson and Dr. Michael S. Hsiao for serving on my thesis committee.

I would like to express my sincere gratitude to my parents for their priceless love and support all through my life. Without their support, I would not be what I am today.

I thank my labmate, collaborator and good friend Deepak Mane who was always there to support me.

I would like to thank my friends Harsha, Hareesh, Skanda, Krushi and Narendra for always being there to talk to me. I thank my friends Sriram, Dhiraj, Sundeep, Sumedha, Sowmya, Nikhil and Krishna for all the weekend cooking sessions and dinner trips. Special thanks to Sarvesh and Vireshwar for helping out with Matlab and Latex while writing papers and thesis.

This work was supported in part through a NIST grant 60NANB10D004 and NSF grants 0855095 and 1314598.

Krishna Pabbuleti

April 2014

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Public key cryptography is an important building block for various security features in current embedded systems. Optimizing cryptography enables higher levels of equivalent security to be implemented on the same system and also leads to higher availability of the system for other tasks. In this thesis, we provide optimized PKC solutions on two different kinds of platforms, one is a highend platform with PC-like performance and other is a resource-constrained low-end platform.

1.1 SIMD Acceleration of Modular Arithmetic

1.1.1 Introduction

Handheld computing is an emerging market with increasingly complex and powerful processors to provide wide range of services to the end user. Current processors targeted at the handheld market contain several coprocessors and accelerators which can perform specialized functions, such as signal processing and video acceleration, without burdening the

main processor. Information security is a common requirement for those platforms as well. Public key cryptographic algorithms like RSA, DSA and elliptic curve cryptography (ECC) are needed to support key exchange and signature protocols. ECC in particular is popular in embedded context because of the smaller key sizes compared to other public key cryptographic systems. For example, at 128-bit equivalent security, we need 3072-bit public key in RSA, whereas ECC requires only a 256-bit public key.

Our work focuses on efficient implementation of modular arithmetic which forms the basis for ECC, by exploiting the capabilities of SIMD coprocessors in Intel Atom and Qualcomm Snapdragon processors. High-performance implementations are important for handheld computing platforms as well. Indeed faster cryptography enables higher levels of equivalent security, or higher availability of handheld computing resources for other tasks. Furthermore, new applications such as e-cash and privacy-friendly attributes make extensive use of public-key primitives.

1.1.2 Motivation

We propose efficient implementation techniques for modular multiplication using advanced architectural features (and SIMD in particular) on modern SoCs. Yan and others showed that DSP processors are efficient in accelerating modular arithmetic [1, 2]. Morozov later demonstrated how embedded DSP cores in SoC could be used to accelerate modular arithmetic [3]. Similarly, the SSE extension on x86 processors enables efficient big number multiplication [4]. Bernstein et al showed that significant performance gain can be achieved by using NEON SIMD extension [5]. However, his demonstration used a specialized curve called `curve25519`. In this work, we target NIST recommended curves over prime fields. NIST primes are special primes which are of the form $2^m \pm 2^n - \quad ..... \quad - 1$. Reduction is very

easy in these fields because of the structure of the prime number. The five 5 NIST primes (P192, P224, P256, P384, P512) are used for standard curves called `nistp192`, `nistp224`, ..., which require modular operations using 192-bit, 224-bit, ... prime field numbers. Recent work by Kasper demonstrated portable (non-SIMD), 64-bit optimized versions of `nistp224` [6]; our work explores the opportunities offered by SIMD processor extensions.

1.1.3 Contribution

We implement high performance modular arithmetic on two different hardware platforms supporting SIMD instructions and we compare their performance. We evaluate our implementation on Qualcomm Scorpion and on Intel Atom processor. The Scorpion processor is one of the seven processors integrated in Qualcomm's Snapdragon platform; we will refer to the more common term Snapdragon from now on. We evaluate the performance of `nistp192` and `nistp224` elliptic curves on both the platforms. The same concept could be extended to other standard curves as well.

The major contributions of this work are:

- Implementation of efficient modular multiplication on SIMD architecture for NIST primes.
- Apples-to-apples performance comparison of vectorized modular arithmetic on contemporary embedded platforms, including cycle count performance and analysis of the instruction set.

1.2 Energy Budget Analysis for Signature Schemes

1.2.1 Introduction

Wireless Sensor Nodes (WSN) systems have been extensively researched over the past decade and a half. These small, resource constrained devices monitor their surroundings and they provide a real-time, distributed view on a physical process. The Internet-of-Things, which may turn every WSN into an Internet host, is an important opportunity for this class of devices. This trend is accelerated by new technologies such as IPv6, ultra-low-power microprocessors, and novel MEMs sensors. Because all these devices may become interconnected, security will be an important factor to their eventual success. This contribution looks at a specific type of WSN, one which is in capability just above a passive RFID. We consider a wireless sensor node which harvests energy from its surroundings [7]. Energy harvesting considerably simplifies the installation and maintenance of such devices. Without battery replacement or wiring requirements, they can be installed in physically challenging or inaccessible environments - and their lifetime appears to become infinite. The downside of energy harvesting is that it severely limits the energy budget available for WSN operation [8]. For example, vibration-based [9] or piezo-electric based harvesters [10] deliver a few microwatt up to a milliwatt; solar-based harvesters deliver a few tens to hundreds of milliwatt [11].

1.2.2 Motivation

The implementation of PKC in constrained environments remains a challenging problem, and there is an extensive body of work on efficient implementation of ECC-based [12–14] and hash-based [15] signatures. Several researchers have analyzed the energy cost of public-key cryptography in detail [16–18]. A similar effort was made recently for symmetric-key

cryptography [19]. Our primary objective is to understand the energy design-space of PKC, covering computation as well as communication overhead. This was previously investigated by de Meulenaer for ECDSA [20], and by Wander for ECDSA and RSA [21]. Our efforts differ from this previous work in three aspects: (a) we present actual measurement data rather than estimates, (b) we investigate the impact of security level, and (c) we present results for hash-based signatures. Finally, we note that the importance of the energy design space of PKC was also raised by Struik at DIAC 2013 [22].

Previous work on obtaining the energy cost of ECC concluded the following [16, 17]: the strategy that minimizes the energy cost per signature is one that runs the microcontroller as fast as possible. This minimizes the loss through static energy dissipation. Hence, in an energy-harvested sensor node, it is best to hold off on activities until the energy store has sufficient energy to support at least one complete iteration of the signature protocol.

1.2.3 Contribution

This effort brings the following contributions to the challenging design space of energy-harvested WSN.

1. We measure the energy-performance characteristics of three identification protocols, each using a different public-key algorithm. The three PKC signing systems include the Elliptic Curve Digital Signature Algorithm (ECDSA), the Lamport Diffie One Time Signature Algorithm (LD-OTS), and the Winternitz One Time Signature Algorithm (W-OTS). The first is based on Elliptic Curve Cryptography, while the latter two are based on hash functions. For each type of signature, we measure both the computation energy as well as the communication energy.

2. We perform an in-depth analysis of the energy needs for each PKC, isolating the energy

required for computations from the energy required for communications. These three PKC have very different characteristics. ECSDA is computationally expensive, but it has relatively short signatures (e.g. 512 bit for 128-bit security level). Hence, for a protocol based on ECDSA, the dominating energy factor is attributed to computations. LD-OTS is computationally much simpler, but it carries very long signatures (eg. 16 Kbytes for 128-bit security level). In a protocol based on LD-OTS, the dominating energy factor is attributed to communications. Finally, W-OTS enables a trade-off between communication cost and computation cost: by increasing the amount of hashes, a shorter signature can be obtained (eg. 532 bytes for 128-bit security level). Therefore, we conclude that it's important to perform a comprehensive analysis when assessing the energy needs from a WSN that implements a PKC based protocol. Both the computational load, as well as the communication load, are important factors.

1.3 Organization

The rest of the thesis is organized to cover details of these two solutions. Chapters 2 and 3 discuss the high performance implementation of modular arithmetic on SIMD platforms. Chapters 4 and 5 discuss the details about energy budget analysis for various signature schemes for constrained platforms.

The individual chapters are structured as follows: Chapter 2 introduces preliminary knowledge that is needed for the first solution, which include Elliptic Curve Cryptography, NIST Primes and SIMD Platforms. Chapter 3 presents the implementation details of our algorithm on two SIMD platforms namely, ARM neon and Intel SSE2 and also provide an analysis of our results on these platforms.

Chapter 4 presents various hardware components used for building the wireless sensor node

and various signature schemes we have analysed namely, Elliptic Curve Digital Signature Algorithm, the Lamport-Diffie one-time hash-based signature scheme and the Winternitz one-time hash-based signature scheme. Chapter 5 provides operation modes of the system and an analysis of the energy consumption data for these schemes.

Finally, Chapter 6 summarizes the contributions of our work and identifies potential future targets.

1.4 Related Articles

Our work is described in the following papers:

- Krishna Chaitanya Pabbuleti, Deepak Hanamant Mane, Avinash Desai, Curt Albert, Patrick Schaumont, "'SIMD acceleration of modular arithmetic on contemporary embedded platforms"', HPEC'13.
- Krishna Chaitanya Pabbuleti, Deepak Hanamant Mane, Patrick Schaumont, "'Energy Budget Analysis for Signature Schemes"', Submitted to RFIDSec'14, Under review.

Chapter 2

SIMD Acceleration of Modular Arithmetic: Background

In this chapter, we provide a brief overview of Elliptic Curve Cryptography, NIST Primes and SIMD platforms.

2.1 Elliptic Curve Cryptography

ECC was independently proposed by Neal Koblitz and Victor S. Miller and is based on point algebra of elliptic curves over a finite field. The security of ECC is based on the Elliptic Curve Discrete Logarithm Problem which states that given two points P and Q on the curve such that $Q = k.P$, it is very hard to find k. This operation is the basis for secure protocols for signing and key exchange: the private key k is a very long integer chosen at random and the public key is derived by scalar multiplication of point P with k. The scalar multiplication thus forms the basis of ECC. All points on an elliptic curve form a group. One can either add two different points, or one can double a point (add it to itself). Using point adding

and point doubling, a very basic implementation of $k.P$ is shown in Algorithm 1. Each point operation (adding or doubling) requires multiple modular-arithmetic operations in the underlying field. For example, for a Weierstrass curve with Jacobian coordinates (a common choice for NIST curves), one may find 7 modular multiplications and 3 squaring operations for each point doubling, and 12 modular multiplications and 2 modular squarings for point adding [23]. It's easy to see that computing $k.P$ is dominated by modular multiplications. If we assume a 192-bit random k, then for each point multiplication $k.P$, we may expect to see about 190 point-double and 85 point-add operations, leading to 1362 modular multiplications and 740 modular squarings on operands of size 192 bits.

Algorithm 2.1 Right-to-left binary method for point multiplication [23]

INPUT: $k = (k_{t-1}, ..., k_1, k_0)_2, \quad P \in E(F_q)$.
OUTPUT: kP.
1. $Q \leftarrow \infty$.
2. For i from 0 to t-1 **do**
  2.1 If $k_i = 1$ then $Q \leftarrow Q + P$.
  2.2 $P \leftarrow 2P$.
3. Return(Q).

2.1.1 Modular Arithmetic

Modular arithmetic is the most fundamental operation in Elliptic Curve Cryptography. Majority of execution time of ECC is spent in performing modular operations on underlying fields. Modular arithmetic is a system of arithmetic for integers, in which numbers wrap around after reaching a certain value, called modulus. So, the integers in this system are always smaller than the chosen modulus and are finite in number. If the chosen modulus is a prime number p, the resulting set of p integers smaller than p form a finite prime field. Any operation between two field elements should result in another field element. If the result is greater than the modulus, then a modulo operation is performed to get an integer smaller

than the modulus and is a field element. This operation is called reduction. For example, let us consider a field with modulus $p = 13$. So the set of integers $\{0, 1, 2, ...12\}$ form a prime field modulo 13. If we add two field elements 10 and 11, the result is 21, which is greater than 13 and not a field element. So, we divide the result by 13 and take the remainder. The result is 8, which is less than 13 and a field element. So, $(10+11) \mod 13 = 8$. A reduction operation involves a division with the prime number which is computationally very expensive. Special techniques like montgomery multiplication can perform reduction by only using multiplication operations. In this work, we achieve reduction without division by exploiting the special structure of NIST primes.

2.1.2 NIST Primes

In public key cryptographic algorithms like RSA, prime numbers are chosen randomly during the run-time. But, in ECC, unlike RSA, prime number is fixed and known beforehand. This enables implementation of some prime-number specific optimizations. There are certain prime numbers standardized by organizations like National Institute of Standards and Technology (NIST) and Standards for Efficient Cryptography Group (SECG). The current work focuses on NIST primes. NIST defines five prime numbers with sizes ranging from 192 bits to 521 bits. The prime numbers are named with a prefix 'P' followed by the size of the prime number in bits. The five prime numbers defined by NIST are P192, P224, P256, p384 and P521. More details on NIST primes can be found in table 2.1. NIST primes have a special structure of the form $2^m \pm 2^n - \quad ..... \quad - 1$. Because of this special structure, reduction is very easy in these fields. For example, let us consider the smallest NIST prime $P192 = 2^{192} - 2^{64} - 1$. Because of the structure of the prime number, any number bigger than P192 can be reduced as sum of two smaller numbers using the relation $2^{192} \equiv 2^{64} + 1$. The division step required for reduction essentially becomes an addition. Listing 2.1 shows

an example how reduction is simplified in NIST prime field P192. Similar relations can be obtained for all the NIST primes as shown in table 2.1.

In Prime field P192, (2^{192}) mod P192 $= (2^{192} - 2^{64} - 1 + 2^{64} + 1)$ mod P192
$= ((2^{192} - 2^{64} - 1)$ mod P192$) + ((2^{64} + 1)$ mod P192$)$
$= (2^{64} + 1)$ mod P192

When we add two 192−bit numbers, there will be an overflow into bit 192.
The carry bit is equal to 2^{192} which can be replaced with $2^{64} + 1$, using the above relation.
Same logic can be applied when multiplying two 192−bit numbers resulting in a 384−bit number.
The upper 192 bits in the result is multiplied with $2^{64} + 1$ and added to lower 192 bits.
The process is repeated till there is no overflow.

Listing 2.1: Reduction in P192

Table 2.1: NIST Primes

Name	Prime Number	Size	Reduction Relation
P192	$2^{192} - 2^{64} - 1$	192	$2^{192} \equiv 2^{64} + 1$
P224	$2^{224} - 2^{96} + 1$	224	$2^{224} \equiv 2^{96} - 1$
P256	$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$	256	$2^{256} \equiv 2^{224} - 2^{192} - 2^{96} + 1$
P384	$2^{384} - 2^{218} - 2^{96} + 2^{32} - 1$	384	$2^{384} \equiv 2^{218} + 2^{96} - 2^{32} + 1$
P521	$2^{521} - 1$	521	$2^{521} \equiv 1$

2.2 SIMD Platforms

We compare the performance of modular arithmetic on two different hardware platforms namely, Qualcomm Snapdragon APQ8060 and Intel Atom N2800. Snapdragon is based on modified ARM core with a SIMD extension called Neon. Intel Atom is based on x86 architecture with a SIMD extension called SSE2.

2.2.1 ARM Neon

The Snapdragon features a dual CPU (Scorpion) architecture, running up to 1.7 GHz each, two VeNum (NEON) 128-bit SIMD multimedia coprocessors and combined 512 KByte L2

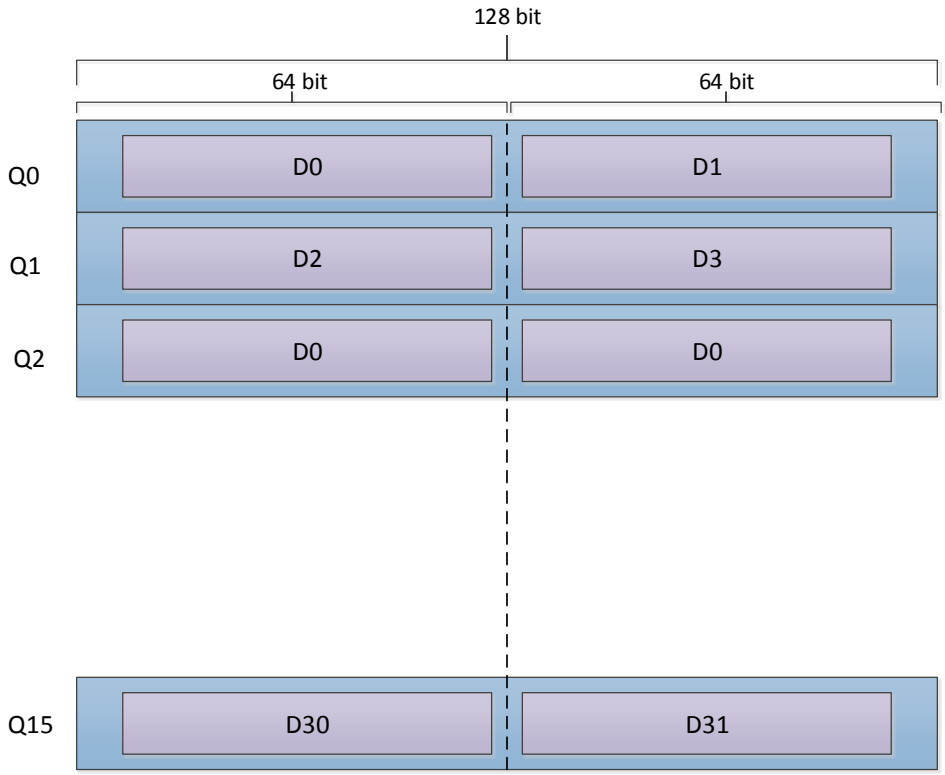

Figure 2.1: Registers in NEON

cache. The Scorpion processor is designed in-house but has many architectural similarities with the ARM Cortex-A8 and ARM Cortex-A9 CPUs. Functionally, Scorpion is an intermediary step between Cortex-A8 and Cortex-A9, supporting some but not all of Cortex-A9's out-of-order instruction execution capabilities. Scorpion-based Snapdragon SOCs implement Cortex-A8 and A9-compliant floating-point and NEON SIMD engines. We specifically target NEON coprocessor for our SIMD optimizations.

The NEON architecture has sixteen 128-bit vector registers, `q0` through `q15` as shown in Figure 2.1. It also includes thirty-two 64-bit vector registers, `d0` through `d31`, but these registers share physical space with the 128-bit vector registers: `q0` is the concatenation of `d0` and `d1`, `q1` is the concatenation of `d2` and `d3`, and so on. The basic ARM architecture has only sixteen 32-bit registers, `r0` through `r15`. Register `r13` is the stack pointer and register

`r15` is the program counter, leaving only fourteen 32-bit registers for general use. An obvious benefit of NEON for cryptography is that it provides much more space in registers, reducing the number of loads and stores from memory. The 128-bit arithmetic unit can perform four 32-bit operations in each cycle. The Cortex-A8 NEON microarchitecture has one 128-bit arithmetic unit and one 128-bit NEON load/store unit that runs in parallel with the NEON arithmetic unit.

2.2.2 Intel SSE2

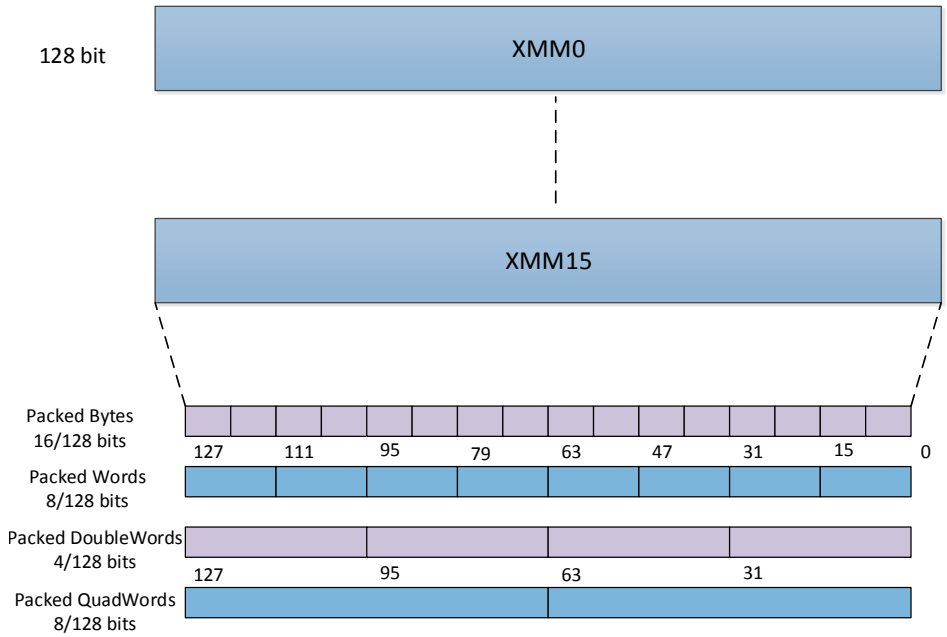

Figure 2.2: Fundamental 128-Bit packed SIMD data types in SSE2

The second processor we use is the Intel Atom N2800 Processor. The Atom platform aims at the power performance of a RISC architecture while still maintaining the x86 instruction set. The Intel Atom N2800 (formerly known as Cedar Trail) contains a dual core 1.86 GHz processor with support for the SSE2 instruction set. The SSE2 instruction set adds sixteen 128-bit SIMD registers to the general purpose x86-64 register set.

The SSE2 instruction set adds support for SIMD instructions through the addition of sixteen 128-bit registers, `XMM0` through `XMM15`. Eight of these were introduced with the original SSE instruction set and eight more were added in the Intel 64 architecture. In the Intel architecture, unlike MIPS, a word consists of 16 bits. Figure 2.2 shows the SIMD data types in SSE2. More information on SSE2 SIMD coprocessor can be found at [24].

Chapter 3

SIMD Acceleration of Modular Arithmetic: Implementation

In this chapter, we provide details on implementing modular arithmetic using SIMD coprocessors available on the respective platforms. We provide a comparison of our results with standard libraries like GMP and OpenSSL.

3.1 Modular Multiplication in P192

The smallest prime among NIST primes is $P192 = 2^{192} - 2^{64}$ - 1. Any number larger than this prime can be reduced by using the relation $2^{192} = 2^{64} + 1$ (mod P192). We describe two methods to perform modular multiplication in this field. The first method is a variant of schoolbook multiplication, where each of the partial products is reduced as they are computed. The second method involves reducing one of the multiplicands before computing each of the partial product rows.

3.1.1 Schoolbook Multiplication with Intermediate Reduction

First, the number is represented in a polynomial form in radix 32 as below:

$f = f_0.2^0 + f_1.2^{32} + f_2.2^{64} + f_3.2^{96} + f_4.2^{128} ... + f_5.2^{160}$

$g = g_0.2^0 + g_1.2^{32} + g_2.2^{64} + g_3.2^{96} + g_4.2^{128} ... + g_5.2^{160}$

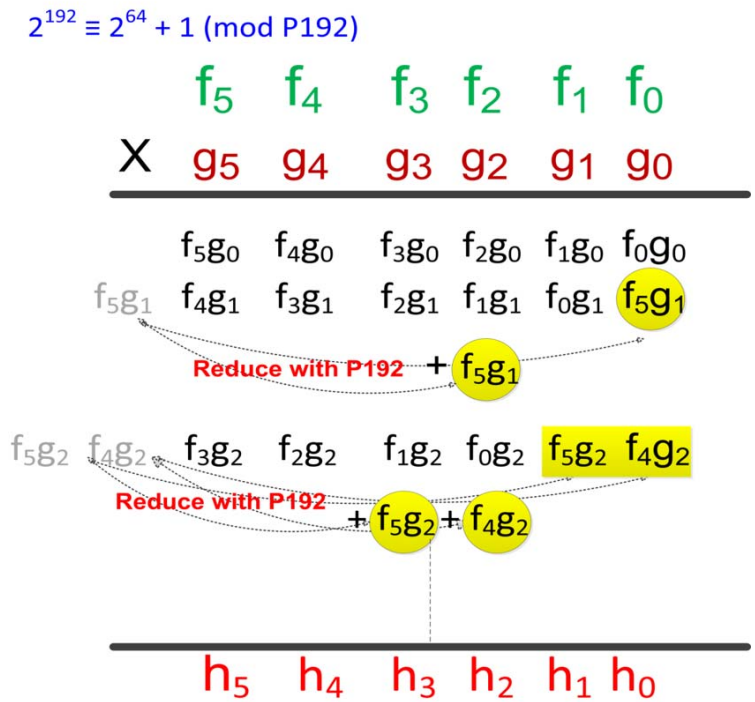

Figure 3.1: Modular Multiplication with Intermediate Reduction

The basis for this approach is schoolbook multiplication as shown in Figure 3.1. Each row of the partial products is reduced as they are calculated. The partial products whose coefficients are greater than 2^{192} are added back into 2^{64} and 2^0. This is possible because of the relation $2^{192} = 2^{64} + 1$ in P192 prime field. This gives us expressions for terms in polynomial form

$$h_0 = f_0g_0 + f_1g_5 + f_2g_4 + f_3g_3 + f_4g_2 + f_5g_1 + f_5g_5$$
$$h_1 = f_0g_1 + f_1g_0 + f_2g_5 + f_3g_4 + f_4g_3 + f_5g_2$$
$$h_2 = f_0g_2 + f_1g_1 + f_1g_5 + f_2g_0 + f_2g_4 + f_3g_3 + f_3g_5 + f_4g_2 + f_4g_4 + f_5g_1 + f_5g_3 + f_5g_5$$
$$h_3 = f_0g_3 + f_1g_2 + f_2g_1 + f_2g_5 + f_3g_0 + f_3g_4 + f_4g_3 + f_4g_5 + f_5g_2 + f_5g_4$$
$$h_4 = f_0g_4 + f_1g_3 + f_2g_2 + f_3g_1 + f_3g_5 + f_4g_0 + f_4g_4 + f_5g_3 + f_5g_5$$
$$h_5 = f_0g_5 + f_1g_4 + f_2g_3 + f_3g_2 + f_4g_1 + f_4g_5 + f_5g_0 + f_5g_4$$

Listing 3.1: Expressions for terms in polynomial form of the product

of the product shown in Listing 3.1. f and g represent the multiplicands and h represents the reduced product.

It can be seen in the expressions of Listing 3.1 that we are adding multiple 64-bit values, which may result in overflow. In order to avoid this overflow, operands are represented using a redundant representation. These redundant representations have some slack at the MSB side, so that the carry does not overflow. For example, we have chosen radix-24 to represent a number. So, the partial products are of size 48 bits. Thus there are 16 bits of slack to accommodate the overflow. Each number is represented in polynomial form in radix-24 as in Listing 3.2.

$$f = f_0.2^0 + f_1.2^{24} + f_2.2^{48} + f_3.2^{72} + f_4.2^{96} + f_5.2^{120} + f_6.2^{144} + f_7.2^{168}$$
$$g = g_0.2^0 + g_1.2^{24} + g_2.2^{48} + g_3.2^{72} + g_4.2^{96} + g_5.2^{120} + g_6.2^{144} + g_7.2^{168}$$

Listing 3.2: Polynomial Form

We now multiply f and g term by term in the same fashion as we do above using schoolbook multiplication, and reduce the partial products. The powers greater than 2^{192} are added back into 2^{64} and 2^0, reduced and arranged in terms of their power as shown in Listing 3.3.

It can be observed that there are powers which are not present as terms in the original polynomial. For example, 2^{16} and 2^{64} are not present in the original polynomial, but are present in the result. So, these have to be split and added into the nearest powers. For example, the lower 8 bits of 2^{16} are shifted 16 places and added into 2^0 and the upper 16

$2^0 \quad : f_0g_0 + f_1g_7 + f_2g_6 + f_3g_5 + f_4g_4 + f_5g_3 + f_6g_2 + f_7g_1$
$2^{16} : f_7g_7$
$2^{24} : f_0g_1 + f_1g_0 + f_2g_7 + f_3g_6 + f_4g_5 + f_5g_4 + f_6g_3 + f_7g_2$
$2^{48} : f_0g_2 + f_1g_1 + f_2g_0 + f_3g_7 + f_4g_6 + f_5g_5 + f_6g_4 + f_7g_3$
$2^{64} : f_1g_7 + f_2g_6 + f_3g_5 + f_4g_4 + f_5g_3 + f_6g_2 + f_7g_1$
$2^{72} : f_0g_3 + f_1g_2 + f_2g_1 + f_3g_0 + f_4g_7 + f_5g_6 + f_6g_5 + f_7g_4$
.
.

Listing 3.3: Reduction of Partial Products

bits are added in 2^{24}. After eliminating all the non-aligned powers, carries are propagated and final reduced result is produced. The partial products f_0g_0, f_0g_1, ..., f_7g_7 are computed on SIMD coprocessor, two at a time. The accumulation, shifting and carry propagation are all carried on the main processor. SIMD coprocessor is very efficient in performing multiplications, but performs poorly for other operations like shifting and carry propagation. So, partial products are computed on SIMD coprocessor, whereas the subsequent operations are carried out on main processor.

3.1.2 Multiplicand Reduction

Instead of the shifting and reducing the partial products, one of the multiplicands is reduced after calculating one row of partial product. Mane discusses an FPGA implementation of modular multiplication in which one of the operands is reduced as the multiplication progresses [25]. We have adapted this technique on SIMD platforms by reducing the number of reduction stages. We have also eliminated intermediate carry propagation by using redundant representation of operands. The operands f and g are represented in redundant representation in radix-24 in the same way as discussed in the previous method.

First row of partial products are obtained by multiplying f with g_0. Now, f is shifted left by 24 bits, the 24 bits beyond 2^{192} are added to 2^0 and 2^{64}, which produces reduced multiplicand.

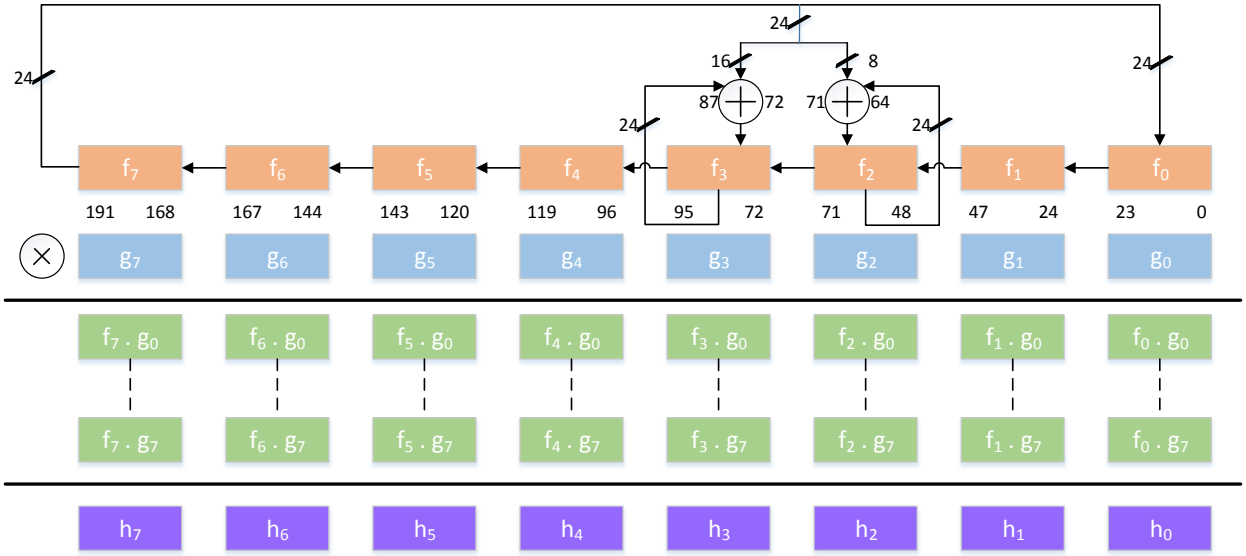

Figure 3.2: P192 Implementation

This shifting and adding back is based on the relation $2^{192} = 2^{64} + 1$ in P192 prime field, which produces the reduced result. It should be noted that, because of the redundant representation, lower 8 bits of 2^{64} are in f_2 and remaining 16 bits are present in f_3. So, lower 8 bits of f_7 are added in upper 8 bits of f_2 and upper 16 bits of f_7 are added in lower 16 bits of f_3. We do not need to worry about carry because of the slack present due to redundant representation. Now, the second row of partial products is calculated by multiplying reduced f with g_1 and added to the previous partial products. Similarly, all partial product rows are computed by shifting f and reducing it, followed by a multiplication with g. In the end, carries are propagated and any trailing bits beyond 2^{192} are again reduced by adding them to 2^0 and 2^{64}.

3.2 Modular Multiplication in P224

The next bigger prime in NIST primes is $P224 = 2^{224} - 2^{96} + 1$. Any number larger than this prime can be reduced by using the relation $2^{224} = 2^{96} - 1$ (mod P192). Results of P192 indicate that the multiplicand reduction method is significantly faster than the schoolbook multiplication with intermediate reduction method. Therefore, we have implemented only multiplicand reduction method for P224.

3.2.1 Multiplicand Reduction

In the same way as above, one of the multiplicands is shifted and reduced after calculating one row of partial product. The numbers f and g are represented in redundant representation in radix-28 as in Listing 3.4

$$f = f0.2^0 + f1.2^{28} + f2.2^{56} + f3.2^{84} + f4.2^{112} + f5.2^{140} + f6.2^{168} + f7.2^{196}$$
$$g = g0.2^0 + g1.2^{28} + g2.2^{56} + g3.2^{84} + g4.2^{112} + g5.2^{140} + g6.2^{168} + g7.2^{196}$$

Listing 3.4: Redundant representation in Radix-28

First row of partial products are obtained by multiplying f with g_0. Now, f is shifted left by 28 bits, the 28 bits beyond 2^{224} are added to 2^{96} and subtracted from 2^0, which produces the reduced result. In the same way as above, because of the redundant representation, the lower 16 bits of 2^{96} are in f_3 and the remaining 12 bits are present in f_4. The lower 16 bits of f_7 are added in upper 16 bits of f_3 and the upper 12 bits of f_7 are added in lower 12 bits of f_4. When f is shifted left, f_0 is zero and we need to subtract f_7 from f_0. We need to get borrow from neighbor, but the neighbor is not guaranteed to be a non-zero number. The borrow is taken from f_7 which is being added to f_3 and f_4. Because we have taken the borrow from non-neighbor, we have to add 0xfffffff to the members between f_3 and f_0, i.e., f_1 and f_2. This is best illustrated in the Figure 3.3.

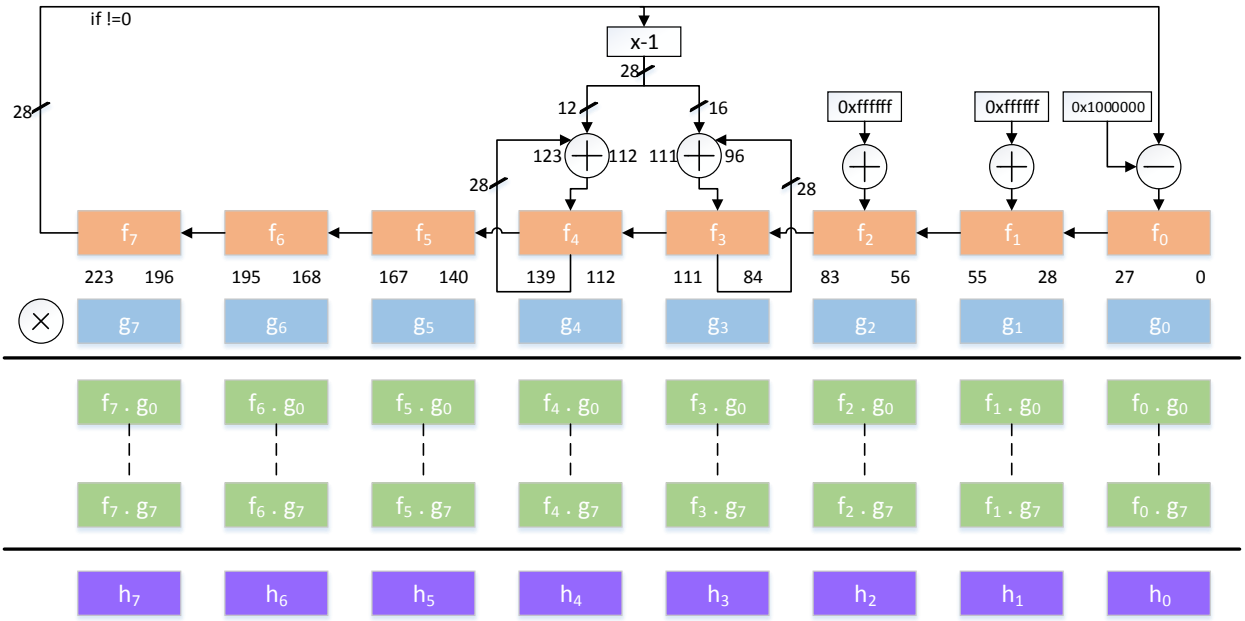

Figure 3.3: P224 Implementation

3.3 Platform Specific Optimizations

This section explains specific implementation details for NEON and SSE2 vector coprocessors.

3.3.1 ARM Neon

NEON can do two multiplications or two multiply and accumulate operations at a time. Moreover, we can take advantage of multiply and accumulate instruction. As shown in Figures 3.2 and 3.3, we first multiply f_0 and f_1 with g_0 in one cycle and obtain partial products f_0g_0 and f_1g_0. In the next cycle, we multiply f_2 and f_3 with g_0 and obtain f_2g_0 and f_3g_0 and so on. After multiplying all the limbs of f with g_0, f is shifted left by 24-bits and reduced. It should be noted that this reduction is performed on main processor because it is more efficient in handling these operations compared to Neon coprocessor. In the next

step, f_0 and f_1 are multiplied with g_1 and the results are added to previously calculated f_0g_0 and f_1g_0, using multiply accumulate instruction. This gives additions for free and also saves registers, as there is no need to save the intermediate partial products. In the end, the partial products are moved to main processor, carry propagation is done and final product is calculated.

3.3.2 Intel SSE2

In SSE2, the modular multiplication is implemented as follows. We use `PMULUDQ` to perform two multiplication operations simultaneously. `PMULUDQ` can perform multiplication operation on two of the four 32-bit values read from memory. We can use `PSHUFD` to rearrange the 32-bit values to allow us to use all of the data read in from memory before writing it back or reading in new data. `PADDQ` is used to accumulate the 64-bit partial products produced in the multiplicand reduction algorithm. `PSHUFD` instruction is used to order the result so that it can be properly written back to memory, 128 bits at a time. The downside of the SSE2 instructions is that they overwrite one of the operands. When you call an instruction on two operands, it calculates the result and stores it in one of the operand registers overwriting the operand. In order to preserve the operand, we have store it in a different register beforehand which adds overhead. Moreover, SSE2 doesn't have multiply and accumulate instruction, so the intermediate results have to be saved and added to the partial product, adding further overhead.

3.4 Results

Table 3.1 and figue 3.4 show the performance for the two proposed approaches on the Qualcomm Snapdragon and Intel Atom for prime field P192. The code is compiled using gcc version 4.4.5 on ARM and gcc version 4.6.3 on Intel Atom. As our modular multiplication is coded in assembly, we expect that the difference in compiler versions has minimal impact. Clock cycles are measured by reading counter registers from Performance Monitoring Unit inside CP15 coprocessor of ARM. On Intel, clock cycles are measured using `RDTSC` instruction. Schoolbook multiplication is significantly slower than the multiplicand reduction method, because the former method involves several shifts and adds which adds a lot of overhead. Atom takes more cycles than ARM in all the cases. Similar trend can be noticed in the numbers obtained from eBACS benchmark in Table 3.3 [26]. Vectorising the multiplications increases the performance by almost 30% in all the cases. Both our methods are faster than the modular multiplication using GMP multiprecision library.

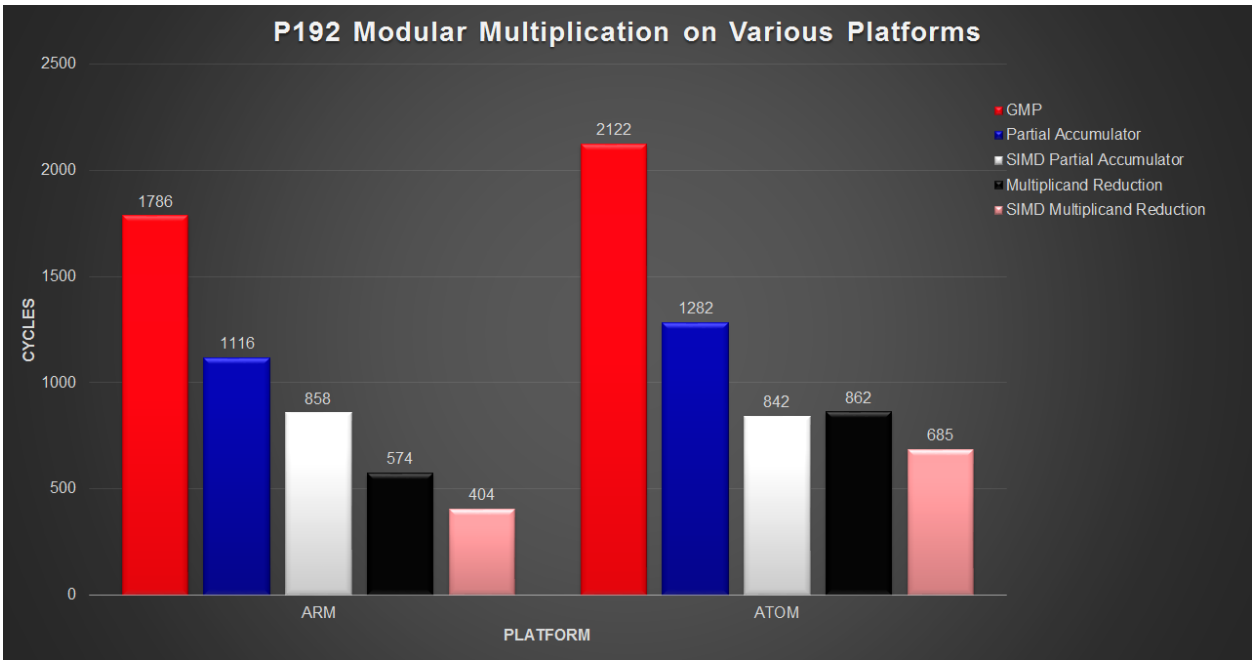

Figure 3.4: Cycles for modular multiplication for P192

Table 3.1: Cycles for modular multiplication for P192

Algorithm	Snapdragon	Snapdragon +NEON	Atom	Atom +SSE2
Intermediate Reduction Method	1116	858	1282	842
Multiplicand Reduction Method	574	404	864	685
GMP	1786	-	2122	-

Table 3.2 and figure 3.5 show clock cycles comparison for multiplicand reduction method on the ARM and Atom for prime field P224. We did not implement schoolbook multiplication for P224 because results from P192 show that multiplicand reduction is already faster. Vectorising the modular multiplication gives a 40% improvement in the performance on ARM whereas gives only 22% improvement on Intel Atom. This may be attributed to overheads present in SSE2 like non-availability of multiply-accumulate instruction and result replacing one of the operands.

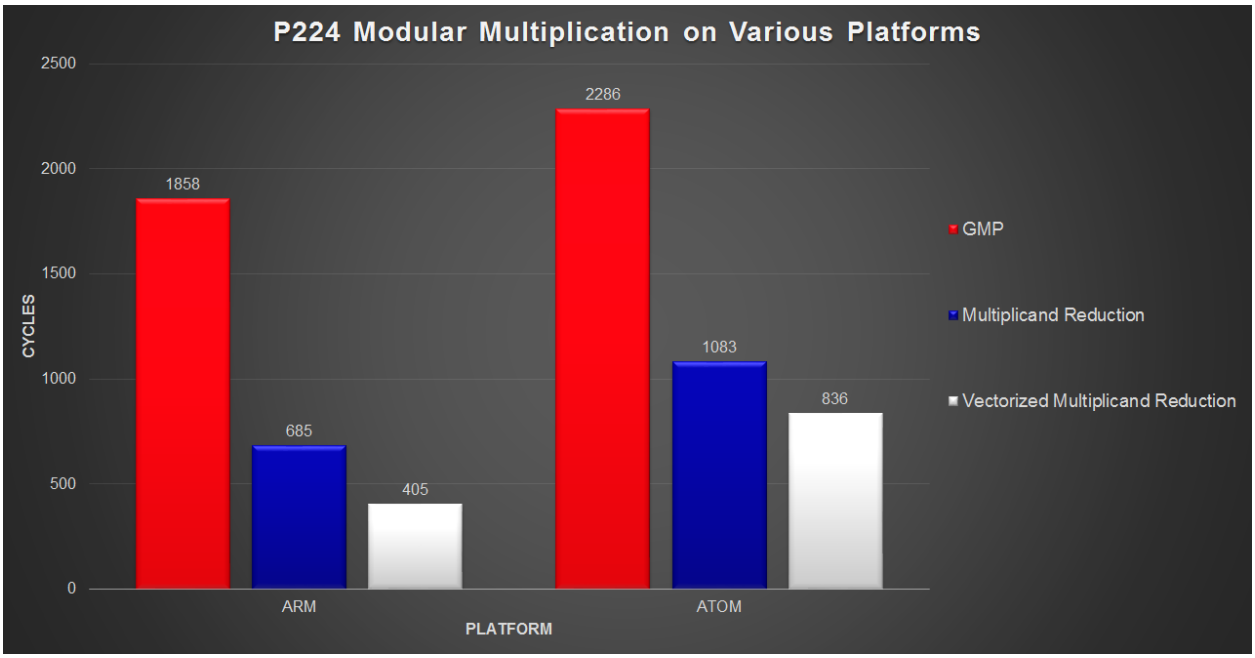

Figure 3.5: Cycles for modular multiplication for P224

Table 3.3 shows the comparison of point multiplication on various embedded platforms against point multiplication using our modular multiplication techniques. The point multi-

Table 3.2: Cycles for modular multiplication for P224

Algorithm	Snapdragon	Snapdragon +NEON	Atom	Atom +SSE2
Multiplicand Reduction Method	685	405	1083	836
GMP	1858	-	2286	-

Table 3.3: Scalar Multiplication Performance for various Embedded Platforms

Platform	Curve and Implementation	Cycles
TI OMAP 3530(ARM A8 + C64x + DSP)[3]	secp160r1	1,054K
TI OMAP 3530(ARM A8 + C64x + DSP)[3]	secp224r1	2,175K
Qualcomm Snapdragon	Ed25519 [26]	3,295K
Intel N280	Ed25519 [26]	5,774K
Qualcomm Snapdragon	P192 ebacs_OpenSSL	3,143K
Qualcomm Snapdragon	P224 ebacs_OpenSSL	3,996K
Intel N280	P192 ebacs_OpenSSL	4,973K
Intel N280	P224 ebacs_OpenSSL	6,349K
Qualcomm Snapdragon with NEON	Curve25519 [5]	511K
Qualcomm Snapdragon	P192 with Multiplicand Reduction	**1,591K**
Qualcomm Snapdragon with NEON	P192 with Multiplicand Reduction	**1,243K**
Intel N280	P192 with Multiplicand Reduction	**2,390K**
Intel N280 with SSE2	P192 with Multiplicand Reduction	**2,115K**

plication is performed in projective coordinate system in all the implementations given in the table. Our implementation is faster than the other generic implementations of comparable security level. However, specialized implementations like `curve25519` on Snapdragon by Bernstein[5] performs better than our implementation. As expected, Atom takes more number of cycles than Snapdragon because the modular multiplication on Atom takes more number of cycles. A similar trend is found in the cycle counts obtained from the eBACS benchmark suite, where Atom takes 1.5 to 1.7 times more number of cycles than Snapdragon.

Figure 3.6 shows the data from Table IV in a manner to appreciate the merit of SIMD vectorization. The figure shows the number of Point Multiplications completed per million cycles of the target architecture, for different curve sizes and different architectures. The vectorized implementations (SSE2 on Atom, NEON on Snapdragon, DSP64x on OMAP) are shown as solid symbols; the default ones are shown as open symbols.

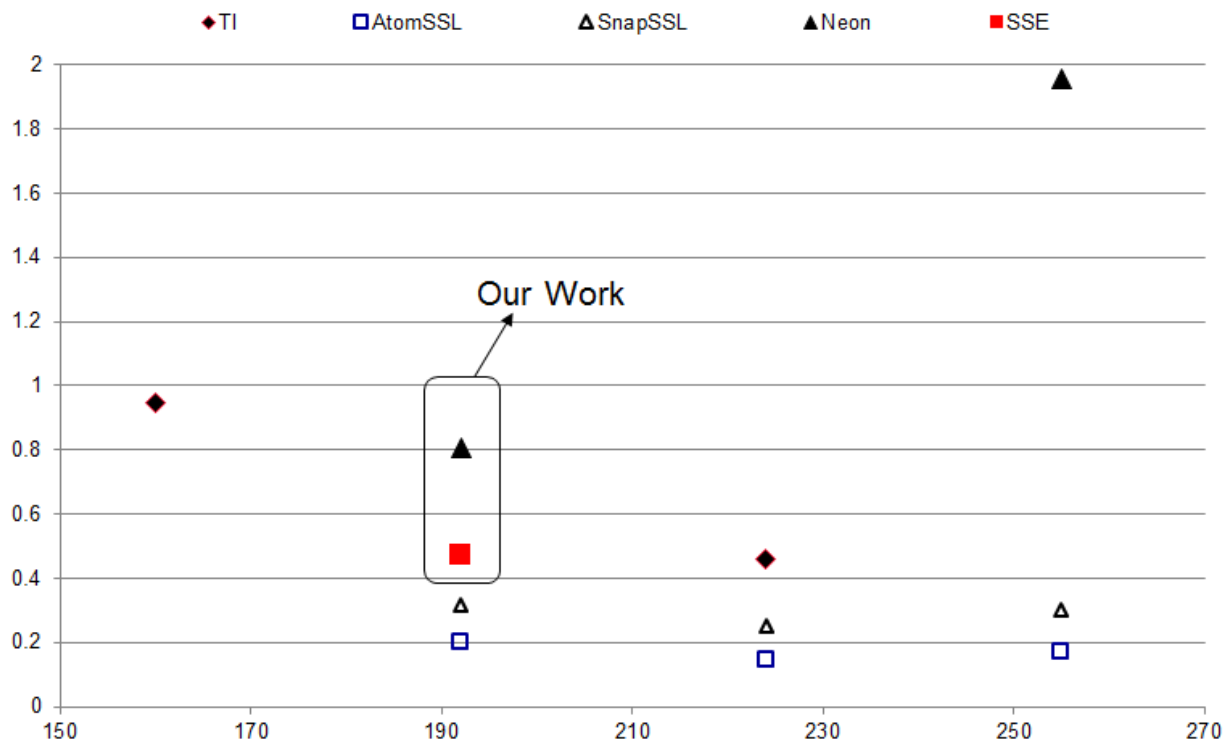

Figure 3.6: Number of Point Mult per million Cycles

Chapter 4

Energy Budget Analysis for Signature Schemes: Background

This chapter provides a brief overview of the hardware components we have used to build the wireless sensor node. It also provides an overview of signature protocols we have analyzed on this system.

4.1 Hardware Platform

This section explains the system architecture of our system and a brief description of different hardware blocks. Figure 4.1 shows the system architecture of the wireless sensor node. It consists of two major blocks, namely self powered wireless sensor node (WSN) and a central control unit. WSN contains three main components namely MSP430 microcontroller, RF front-end and energy harvesting module. WSN node runs different signature schemes and sends the signature over the RF interface to the central control unit for the verification. The integrated energy harvester provides power to the entire node circuitry. The Central Control

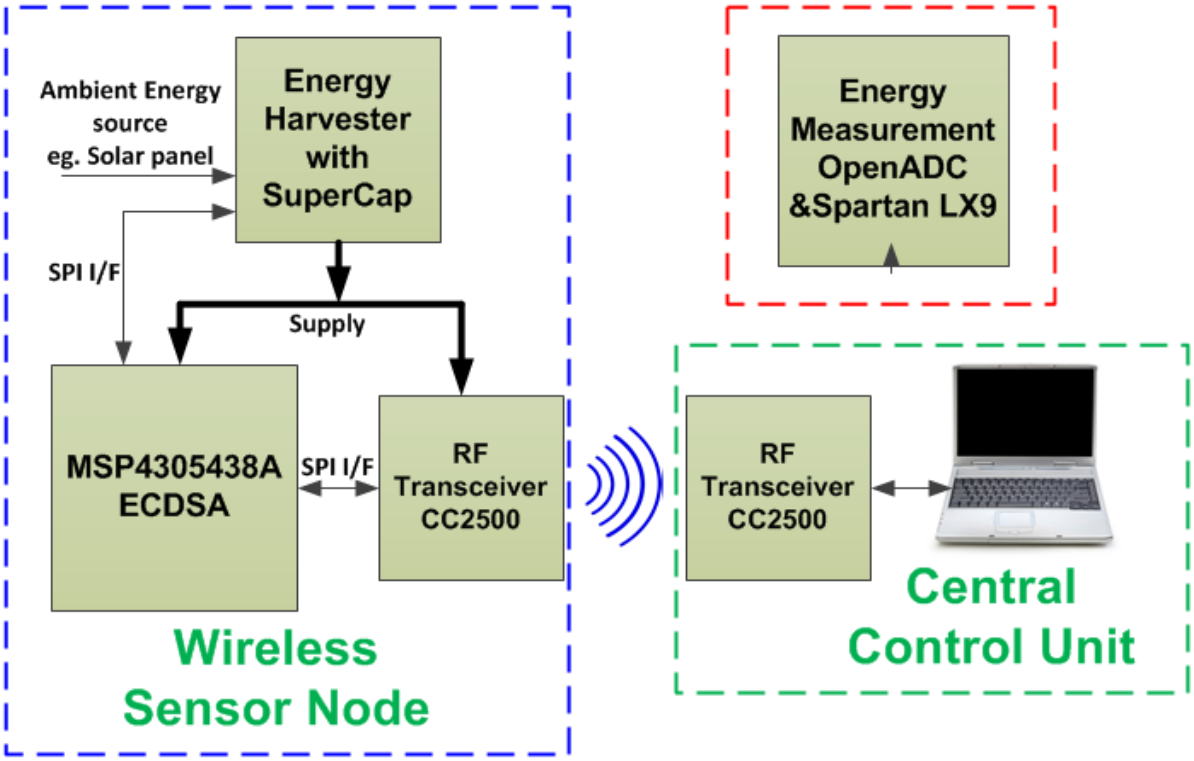

Figure 4.1: Wireless Sensor Node Block Diagram

Unit performs the signature verification. We measure the energy consumption of different hardware blocks in the system using precise energy measurement setup [17].

- Microcontroller - The main computing block of our system is MSP430F5438A, an ultra-low power microcontroller from Texas Instruments. It features a 16-bit CPU, 256KB flash, 16KB SRAM, up to 25 MHz CPU clock, a 32-bit hardware multiplier and is optimized for low power, resource- constrained systems. It supports five low power modes with different levels of power consumption which can be configured using a programmable power management unit [27]. It supports 4-wire SPI interface which allows easy integration different hardware modules into the system.

- RF Interface - CC2500 is a 2.4 GHz low power RF transceiver from Texas Instruments

which supports various programmable modulation schemes and power levels [28]. It has an SPI interface for configuration and data transfer. It supports two low power modes, namely Power down mode and Wake-On-Radio mode. In power down mode, all the chip peripherals including radio frontend and digital circuitry are off and the power consumption is lowest at 2μA. In Wake-On-Radio mode, the RF receiver wakes up periodically, checks for any available packets and goes back to sleep if no packet is available. In power down mode, only sensor node can initiate a transfer, whereas in Wake-On-Radio mode, server or node can initiate the communication.

- Energy Harvester - The sensor node contains an integrated energy harvester which can scavenge energy from ambient sources and store it in a low leakage supercapacitor. Energy management and power output regulation is handled by a custom IC from Anagear, AN1010 [29]. AN1010 also has an SPI interface which can be used by the microcontroller for configuration and status checking.

- Central Control Unit - The Central Control Unit is a PC with an RF transceiver board connected to a USB port. The control unit receives the packets from different sensor nodes and performs signature verification.

- Energy Measurement Infrastructure - The energy measurement infrastructure consists of OpenADC [30, 31] attached to a Spartan FPGA [32]. This system measures the energy consumed for generation and transmission of various signatures.

4.2 Signature Schemes

In this section, we describe the identification protocols running on the energy-harvesting WSN. We are comparing three different methods of implementing PKC signatures, while

the top-level protocol remains identical in each case. The next few subsections describe the top-level protocol, and they briefly review our implementation for each of the PKC algorithms.

4.2.1 Two-pass Unilateral Authentication

The ISO/IEC 9798-3 standard describes a mechanism for a two-pass authentication protocol using signatures. It is based on the following steps:

$$Server \rightarrow WSN : N_S \tag{4.1}$$

$$WSN \rightarrow Server : N_S, N_W, ID_S, Sig_W(N_W, N_S, ID_S) \tag{4.2}$$

In this protocol, N_S and N_W are nonces generated by the server and WSN, respectively, ID_S is a public server ID, and $Sig_W()$ is a signature scheme executed by the WSN. The nonces guarantee freshness, while the server ID prevents man-in-the-middle attacks. An alternate one-pass protocol is possible provided that the Server and WSN maintain a synchronized counter or timer. In that case, the ISO/IEC 9798-3 standard describes the following case:

$$WSN \rightarrow Server : T_W, ID_S, Sig_W(T_W, ID_S) \tag{4.3}$$

In this protocol, T_W is a timestamp, assumed to be available on the WSN as well as the server. We can implement both protocols in our setup; see Section 5.1 for a description of

the operational modes on the WSN. The signature $Sig_W()$ can be implemented in any of the three different algorithms as described in the following subsections.

4.2.2 ECDSA

ECDSA is a well known signature mechanism based on elliptic curve cryptography [33]. We have implemented ECDSA using two different prime-field curves, `secp160r1` and `nistp256`, which have a security level of 80 bit and 128 bit respectively. In ECDSA, signing costs one point multiplication, while verification costs two. Our code is written using the RELIC library [34], and with the following parameters. The scalar multiplication is implemented using a left-to-right window-3 NAF multiplication, and with Jacobian Projective Coordinates. The field operations are basic Comba multiplication and squaring, with Montgomery reduction. SHA-1 is used for hashing and as a pseudo-random generator.

4.2.3 Lamport-Diffie One-Time Signature Scheme

The second signature algorithm uses hash functions. It was first proposed by Lamport and Diffie as a one-time-signature scheme (LD-OTS): this implies that a single key pair can be used for exactly one signature. The LD-OTS scheme works as follows [35]. For a security level b, the signer generates a secret key of $4b$ random strings, each $2b$ bits long. The $4b$ random strings of the secret key can be thought of as two arrays of $2b$ random strings: $x(0,0),..,x(0,2b-1)$ and $x(1,0),..,x(1,2b-1)$.

At 128-bit security, the signer will create a 16 KByte secret key. The public key is obtained by computing the digest of each of the $4b$ strings: $y(0,0) = H(x(0,0))$, .., $y(0,2b-1) = H(x(0,2b-1))$ and $y(1,0) = H(x(1,0))$, .., $y(1,2b-1) = H(x(1,2b-1))$. Each digest is $2b$ bits long; at 128-bit security level, we use SHA256. To sign a message m, the signer

computes a digest over the salted message $H(m, r)$, and breaks this digest into $2b$ bits: $D(0)$ .. $D(2b-1)$. The signature is now formed by selecting a subset of the random strings from the secret key: $x(D(0), 0)$ .. $x(D(2b-1), 2b-1)$. A signature thus is 8Kbyte plus the length of the salt r. To verify the signature, the verifier computes the hash of each string in the signature: $H(x(D(0), 0))$ .. $H(x(D(2b-1), 2b-1))$. The verifier also computes the digest of the salted message $H(m, r)$, splits the digest into bits $v(0)$ .. $v(2b-1)$, and finally checks if $y(v(0), 0) = H(x(D(0), 0))$, .., $y(v(2b-1), 2b-1) = H(x(D(2b-1), 2b-1))$. Generating the key costs $4b$ hash operations, verifying a signature costs $2b$ hash operations.

The LD-OTS scheme is simple, easy to compute, but it has a large signature and key pair. Furthermore, the key can only be used a single time. This last drawback can be eliminated by *chaining*: at each signing, a new key pair is generated, the new public key is signed, and appended to the signature. This triples the length of the message (from 8Kbyte to 24 Kbyte), and it requires the verifier to check all signatures in sequence. In our experimental setup, we have implemented chaining in order to obtain a fair comparison with ECDSA. We refer to this scheme as LD-OTS-C (with the C indicating chaining). Merkle has proposed improvements to chaining using a hash-tree, but we have not implemented these.

In our implementation of LD-OTS-C, we paid attention to the memory usage. Indeed, a secret key of 128-bit equivalent security requires 16KByte, which is well over the capabilities of most microcontrollers, and which is at the limit of the MSP430F5438 device we have used. To address this problem, we generate the LD-OTS secret key on the fly from a 128-bit secret seed and AES in counter mode. This drastically reduces memory usage at the overhead of recomputing the secret key (and the public key) during signing. We have implemented two security levels for LD-OTS: one at 80-bit security level using SHA1 as a hash function, and a second one at 128-bit security level using SHA256 as a hash function.

4.2.4 Winternitz One-Time Signature Scheme

Another improvement to the LD-OTS scheme was made by Winternitz [35]. In the following description, we summarize the idea but have omitted some details for brevity: refer to the literature for a formal definition.

The fundamental insight from Winternitz was to construct signatures from hash chains rather than isolated hash digests. A hash chain is a sequence of digests computed as $x_1 = H(x_0)$, $x_2 = H(x_1)$, and so forth. In the Winternitz One Time Signature (W-OTS) scheme, the secret key is located at the start of the hash chain, and the public key is located at the end. A hash chain of length l can sign a bit field of length $log_2(l)$ bits. To see how, consider this field of $log_2(l)$ bits as an index v into the chain and return element $l-v$ from the digest chain as the signature for this field. Verification is now done by continuing the hash chain for v more steps; the final element found should correspond to the public key. To sign messages longer than a field of $log_2(l)$ bits, one can define multiple hash chains. For a message of fixed length (say, 256 bits), there is a trade-off between the depth of the hash chains and the number of chains (or field length). For example, a 256 bit message can be signed using 256 chains of length 2, or 128 chains of length 4, or 64 chains of length 8, or 32 chains of length 16. The length of the hash chains, in turn, determines the computational overhead of signing and verification.

Since the public key only includes the endpoints of the chains, there is a considerable reduction in signature length possible by using fewer, but longer, chains. In our experiments, we have experimented with chain lengths of length 2, 4, 8, and 16 and a security level of 256 bit. These require a public key size of 4Kbytes, 2Kbyte, 1Kbyte and 512 byte respectively. We settled on a chain length of 4, which resulted in signature sizes that are halfway between ECDSA-128 and LD-OTS-128.

Table 4.1: Security Parameters, Algorithms, Key Sizes, and Operation Counts

Scheme	**Alg**	**Public** (byte)	**Secret** (byte)	**Sig** (w chain) (byte)	**Sign** (Ops)	**Verify** (Ops)
ECDSA-80	secp160r1	20	20	40	1 Pt Mul	2 Pt Mul
ECDSA-128	nistp256	32	32	64	1 Pt Mul	2 Pt Mul
LD-OTS-C-80	SHA1	3200	3200	9600	320 SHA1	160 SHA1
LD-OTS-C-128	SHA256	8192	8192	24576	512 SHA2	256 SHA2
W-OTS-C-256*	SHA256	2128	2128	4256	256 SHA2	256 SHA2

* for hash chains of length 4. Average operation counts.

Rohde *et al* describe an implementation of W-OTS and a Merkle hash tree, on constrained devices: they confirm the implementation on AVR is feasible, but they do not show energy consumption [15]. To get around the one-time nature of W-OTS, we apply chaining in a similar fashion as for LD-OTS, and we use W-OTS-C in our experiments. Table 4.1 shows the five schemes that we have evaluated in our experiments: two security levels for each of ECDSA and LD-OTS-C, and one security level of W-OTS-C.

Chapter 5

Energy Budget Analysis for Signature Schemes: Implementation

This chapter provides details of various operational modes of our system and a detailed analysis of the results for various signature schemes.

5.1 Operational modes

The way various nodes communicate with each other in the internet of things may vary widely depending on the application. In our experiments, we implement following modes of operation gated by amount of available energy and analyze the performance of WSN.

1. **Asynchronous TX/RX Mode:** The server initiates a communication and sends a request to WSN; the node replies back with appropriate data. The WSN waits for a service request from the server.

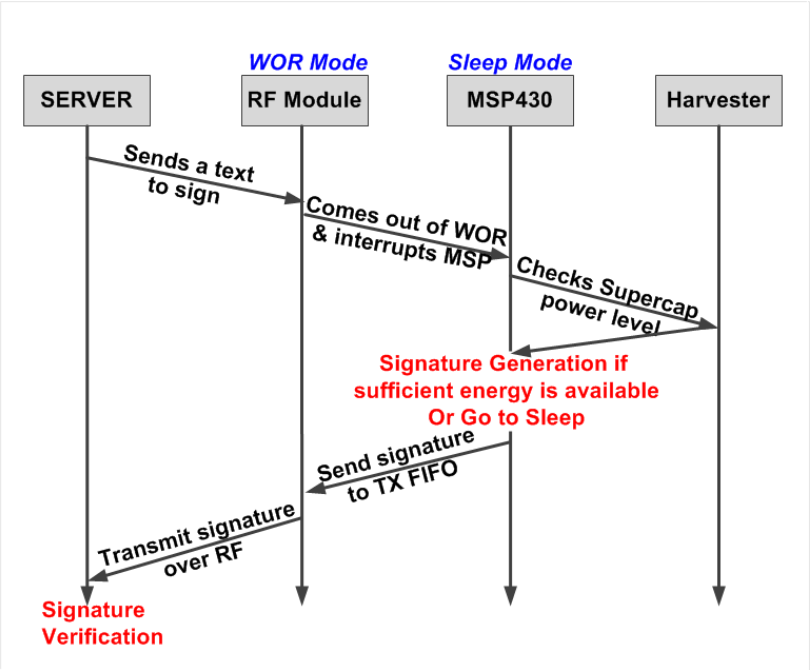

Figure 5.1: Asynchronous Receive/Transmit Mode

2. **Periodic TX Mode:** In a second mode of operation, the WSN starts the protocol by sending the sensed data to the server. The server then authenticates the node and takes further action.

5.1.1 Asynchronous Mode

Figure 5.1 explains asynchronous mode of operation. The server initiates a communication and sensor node responds to the request. In this mode of operation, CC2500 is in Wake-On-Radio mode and MSP430 is in sleep mode. When the server sends a service request, CC2500 comes out of Wake-On-Radio mode and interrupts the MSP430. The MSP430 wakes up from sleep mode and reads the server's request from CC2500 receiver buffer. MSP430, then reads the amount of energy available with the energy harvester. If the amount of energy available is sufficient for one request, MSP430 goes ahead with the signature generation. It then sends

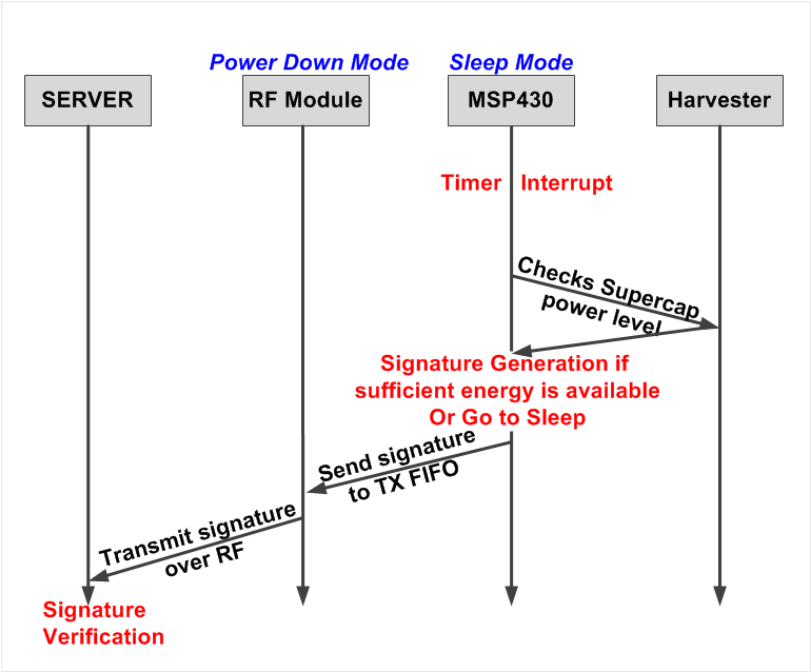

Figure 5.2: Periodic Transmit mode

the signature to CC2500, which in turn transmits it over the RF interface. MSP430 goes back to the sleep mode and CC2500 goes back to WOR mode waiting for next request from the server. The server, on receiving the signature, performs the signature verification. If the amount of energy is insufficient for one signature, the node simply goes back to sleep and waits for next request from the server.

5.1.2 Periodic Transmit Mode

In this mode, the WSN wakes up periodically based on the application duty cycle and transmits a signature to the server. Initially both CC2500 and MSP430 stay in sleep mode. MSP430 wakes up from sleep mode on timer interrupt and reads the amount of energy available in the supercapacitor. If the amount of energy available is sufficient for one signature, MSP430 generates a signature. It then wakes up the CC2500 and sends the signature to its

Table 5.1: Code size (Bytes) of the implementations on the MSP430F5438A

	ECDSA-80	ECDSA-128	LD-OTS-C-80	LD-OTS-C-128	W-OTS-C-256
Flash	35,058	38,174	21,210	23,234	17,088
RAM	2,046	2,365	3,634	8,654	406

TX FIFO, which in turn transmits it over the RF interface. MSP430 and CC2500 both go back to the sleep mode waiting for the next timer interrupt. The server, on receiving the signature, performs the signature verification. If the amount of energy is not sufficient for one signature, the node simply goes back to sleep and waits for next timer interrupt. Figure 5.2 explains this operation mode in detail.

5.2 Results

In this section, we present experimental results of our energy measurements and compare the performance of three signature schemes. We measure the energy needed for computation of one signature on MSP430 and transmitting it over RF. We also measure the signature generation and transmission time to find the throughput of the system, i.e., number of authentications performed per second. We use the `gcc 4.6.3` cross-compiler for MSP430 family of microcontrollers. Table 5.1 shows the footprint of different signature protocol implementations.

Figure 5.3 compares the energy consumption of different signature schemes at 10MHz. ECDSA is based on point multiplication over finite fields which is computationally expensive. Hence, computational energy for ECDSA is higher and it scales up cubically as we increase the required security level from 80 to 128. However, ECDSA has smaller signature lengths which results in less energy for communication. On the other hand, LD-OTS and W-OTS

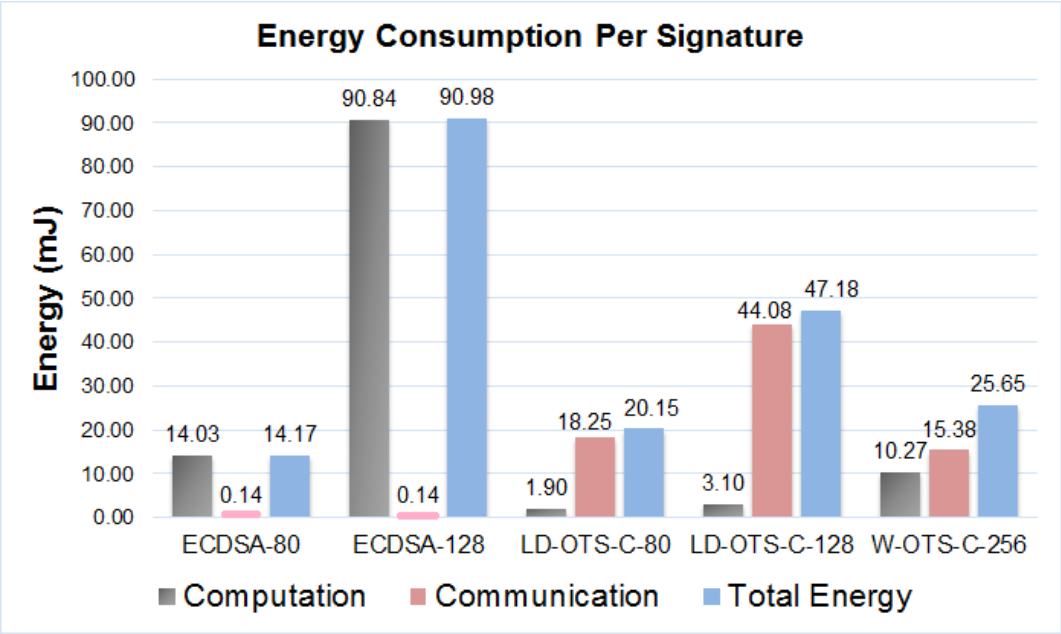

Figure 5.3: Comparison of ECDSA and hash based signatures at 10MHz, 2.7V

based signatures use hash functions to generate key and signature and are relatively easier to compute, but have longer signatures which result in higher energy for communication. Our results show that for a security level of 80, LD-OTS based signatures need almost 8 times less energy for computation and for a security level of 128, LD-OTS is needs 30 times lesser energy than ECDSA. However, these hash based signatures require 100 to 300 times more energy for transmitting a signature than that of ECDSA scheme because of longer signature lengths. This shows that ECDSA based signatures are computationally constrained whereas LD-OTS schemes are communication constrained. W-OTS based signatures are optimal in terms of computational requirements and signature size and show a tradeoff between computation and communication overheads.

Figure 5.4 shows total energy per signature measured at different operating frequencies of the microcontroller. Signature schemes with security level of 128 bits need more energy than those with security level of 80 bits because of the increased computational complexity and longer signature size. It is shown that ECDSA is computationally expensive whereas

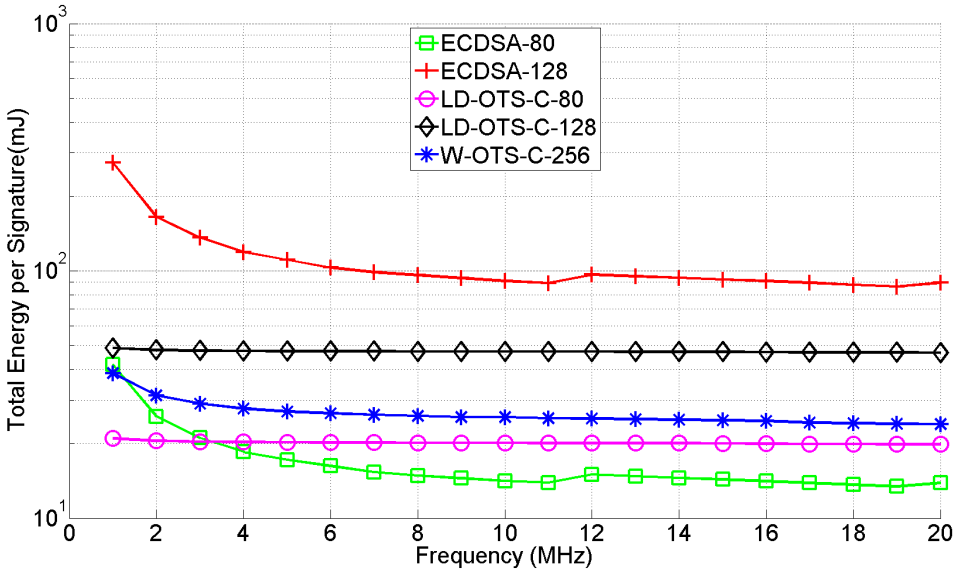

Figure 5.4: Total Energy Consumption per Signature

Hash-based signatures LD-OTS and W-OTS are expensive in terms of communication. As the frequency increases, the time needed for computation decreases and the energy needed for computation increases. But, the communication energy remains same irrespective of frequency of computation. In case of ECC, because the computation energy is the major contributor for the total energy, the total energy decreases as the frequency of operation increases. But, in case of LD-OTS, communication energy is the major contributor to the total energy, which does not change with the frequency. So, the total energy vs frequency is almost a horizontal line for LD-OTS. In case of W-OTS scheme, computation and communication energy contribute almost equally to the total energy. So, the total energy comes down as the frequency increases, but not as much as ECC. Figure 5.4 shows a bump at 12 MHz caused by reprogramming of the power management system of MSP430. Internal core voltage needs to be increased in order to operate MSP430 at higher frequencies [36].

Figure 5.5 shows total energy per signature and corresponding throughput of the system at

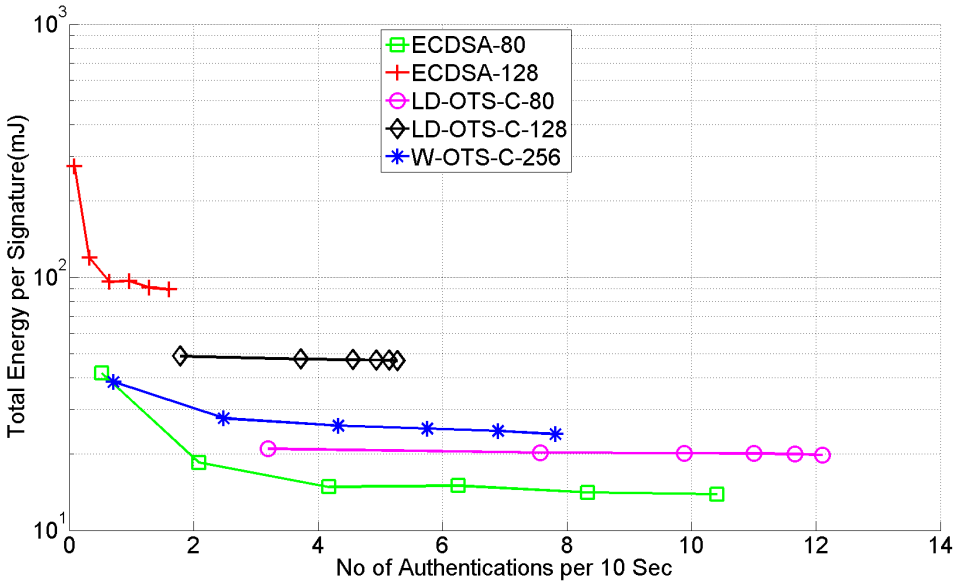

Figure 5.5: Total Energy Vs Throughput of a system

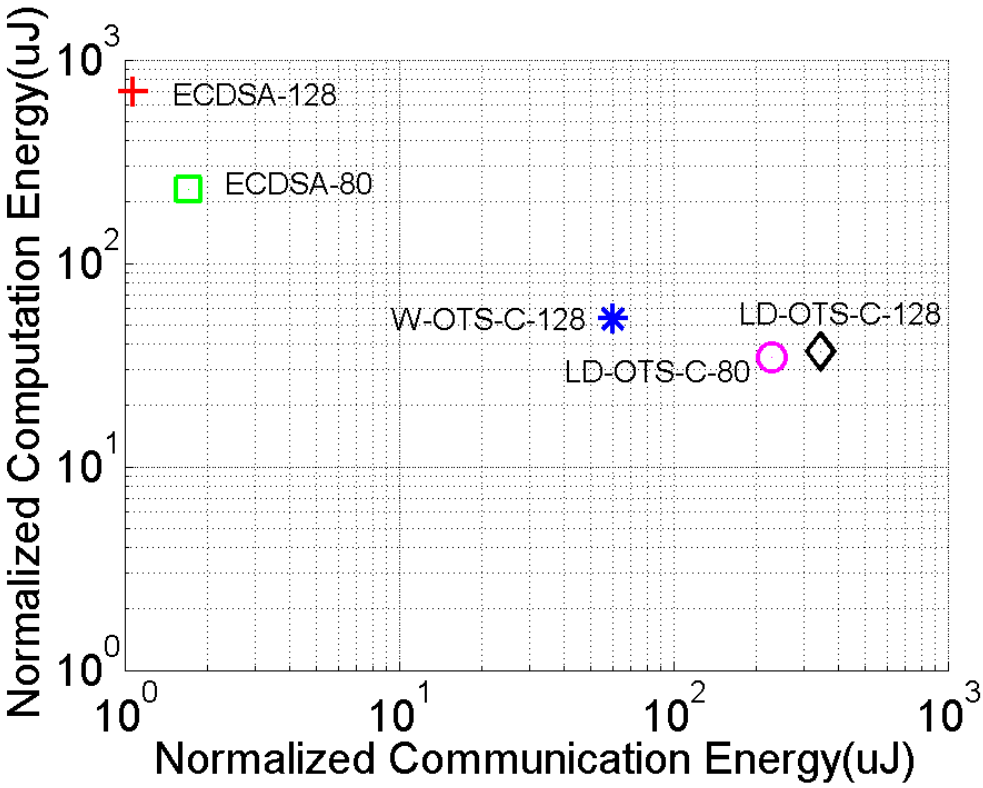

Figure 5.6: Normalized computational energy Vs normalized communication energy for normalized throghput of 1 signature per 5 sec

different operating frequencies. This chart can be used to choose a signature algorithm given the amount of energy available and required throughput.

Figure 5.6 shows the normalized energy for computation and computation at a throughput of one signature per five seconds. The energy is normalized for one bit of equivalent security level offered by the scheme. The graph shows that the impact of the security level is less than then impact of the algorithm selected. We also note that the sum of computation and communication energy, in all algorithms, falls within one order of magnitude of variation.

Our results show that there is a possibility to further reduce the energy budget if hash based signatures can be reduced in size without compromising the security level. W-OTS based signature is one example of such implementation.

Chapter 6

Conclusion

In this thesis, we analyzed and optimized public key cryptographic algorithms on two different kinds of platforms. First, we targeted high-end embedded platforms like Qualcomm Snapdragon and Intel Atom. We used SIMD coprocessors available on these platforms to accelerate the modular arithmetic which forms the base for elliptic curve cryptography. We presented a generalized implementation for modular multiplication over NIST primes on SIMD platforms. Results show that our implementation is over two times faster than OpenSSL implementations on the respective platforms. Our results show that accelerating basic cryptographic operations on mobile platforms can have large impact on the performance of secure mobile applications. As a future work, it would be interesting to measure and compare power consumptions of these platforms with and without SIMD optimizations.

In the second part, we look at a low-end energy constrained platform, a self-powered wireless sensor node. We analyzed and compared the energy requirements for computation and communication for different signature schemes- ECDSA and Hash based signatures. Our experimental results showed the impact of algorithm complexity and signature length on total energy consumption. We found that ECDSA is computationally intensive whereas hash

based signatures are communication dominant. A signature scheme with faster computation and shorter signature size would result in minimum energy consumption.

Bibliography

[1] H. Yan, Z. J. Shi, and Y. Fei, "Efficient implementation of elliptic curve cryptography on DSP for underwater sensor networks," in *Proceedings of the 7th Workshop on Optimizations for DSP and Embedded Systems (ODES-7)*, 2009.

[2] C. Tergino, "Efficient binary field multiplication on a VLIW DSP," Master's thesis, Virginia Tech, 2009.

[3] S. Morozov, C. Tergino, and P. Schaumont, "System integration of elliptic curve cryptography on an omap platform," in *Proceedings of the 9th IEEE Symposium on Application Specific Processors SASP'11*, pp. 52–57, 2011.

[4] Intel, "Using Streaming SIMD Extensions (SSE2) to perform Big Multiplications,"

[5] D. J. Bernstein and P. Schwabe, "NEON crypto," in *Proceedings of the 14th international conference on Cryptographic Hardware and Embedded Systems CHES'12*, pp. 320–339, 2012.

[6] E. Kasper, "Fast Elliptic Curve Cryptography in OpenSSL," in *Financial Cryptography and Data Security: FC 2011 Workshops, RLCPS and WECSR*, 2011.

[7] J. A. Paradiso and T. Starner, "Energy scavenging for mobile and wireless electronics.," *IEEE Pervasive Computing*, vol. 4, no. 1, pp. 18–27, 2005.

[8] P. Mitcheson, E. Yeatman, G. Rao, A. Holmes, and T. Green, "Energy harvesting from human and machine motion for wireless electronic devices," *Proceedings of the IEEE*, vol. 96, pp. 1457–1486, Sept 2008.

[9] E. Lai, A. Redfern, and P. K. Wright, "Vibration powered battery-assisted passive rfid tag," in *EUC Workshops*, pp. 1058–1068, 2005.

[10] C. D. H. N. Kong, T, "A self-powered power management circuit for energy harvested by a piezoelectric cantilever," in *Applied Power Electronics Conference and Exposition (APEC), 2010 Twenty-Fifth Annual IEEE*, APEC2010, 2010.

[11] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava, "Design considerations for solar energy harvesting wireless embedded systems," in *Proceedings of the*

4th International Symposium on Information Processing in Sensor Networks, IPSN '05, (Piscataway, NJ, USA), IEEE Press, 2005.

[12] E. Wenger and M. Werner, "Evaluating 16-bit processors for elliptic curve cryptography," in *CARDIS*, pp. 166–181, 2011.

[13] C. Pendl, M. Pelnar, and M. Hutter, "Elliptic curve cryptography on the wisp uhf rfid tag," in *RFIDSec*, pp. 32–47, 2011.

[14] E. Wenger, M. Feldhofer, and N. Felber, "Low-resource hardware design of an elliptic curve processor for contactless devices.," in *WISA* (Y. Chung and M. Yung, eds.), vol. 6513 of *Lecture Notes in Computer Science*, pp. 92–106, Springer, 2010.

[15] S. Rohde, T. Eisenbarth, E. Dahmen, J. Buchmann, and C. Paar, "Fast hash-based signatures on constrained devices," in *CARDIS*, pp. 104–117, 2008.

[16] A. Liu and P. Ning, "TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks," in *Proceedings of the 7th international conference on Information processing in sensor networks*, IPSN '08, (Washington, DC, USA), pp. 245–256, IEEE Computer Society, 2008.

[17] D. H. Mane and P. Schaumont, "Energy-architecture tuning for ecc-based rfid tags," in *RFIDSec*, pp. 147–160, 2013.

[18] V. Cervenka, D. Komosny, L. Malina, and L. Mraz, "Energy efficient public key cryptography in wireless sensor networks," in *Innovations and Advances in Computer, Information, Systems Sciences, and Engineering* (K. Elleithy and T. Sobh, eds.), vol. 152 of *Lecture Notes in Electrical Engineering*, pp. 497–509, Springer New York, 2013.

[19] L. Batina, A. Das, B. Ege, E. B. Kavun, N. Mentens, C. Paar, I. Verbauwhede, and T. Yalçin, "Dietary recommendations for lightweight block ciphers: Power, energy and area analysis of recently developed architectures," in *RFIDSec*, pp. 103–112, 2013.

[20] G. de Meulenaer, F. Gosset, O.-X. Standaert, and O. Pereira, "On the energy cost of communication and cryptography in wireless sensor networks," in *Networking and Communications, 2008. WIMOB '08. IEEE International Conference on Wireless and Mobile Computing,*, pp. 580–585, Oct 2008.

[21] A. Wander, N. Gura, H. Eberle, V. Gupta, and S. Shantz, "Energy analysis of public-key cryptography for wireless sensor networks," in *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, pp. 324–328, March 2005.

[22] R.Struik, "Aead ciphers for highly constrained networks," in *DIAC*, 2013. http://2013.diac.cr.yp.to/slides/struik.pdf.

[23] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[24] D. Garcia, "Great Ideas in Computer Architecture SIMD II," Oct 2012. http://www-inst.eecs.berkeley.edu/~cs61c/fa12/lectures/18LecFa12DLPII.pdf.

[25] S. Mane, L. Judge, and P. Schaumont, "An Integrated Prime-Field ECDLP Hardware Accelerator with High-Performance Modular Arithmetic Units," in *International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pp. 198–203, 2011.

[26] D. J. Bernstein and T. Lange, "eBACS: ECRYPT Benchmarking of Cryptographic Systems."

[27] "Texas Instruments MSP430F5438A Mixed Signal Microcontroller." http://www.ti.com/lit/ds/symlink/msp430f5438a.pdf.

[28] "Texas Instruments Low Power 2.4GHz RF Transceiver." http://www.ti.com/lit/ds/swrs040c/swrs040c.pdf.

[29] "Anagear Power Management." http://www.anagear.com/content/ANG1010.

[30] C. O'Flynn, "OPENADC," 2012. http://newae.com/tiki-index.php?page=OpenADC.

[31] C. O'Flynn, "Power analysis for cheapskates," 2012. https://media.blackhat.com/ad-12/O%27Flynn/bh-ad-12-for-cheapskates-o%27flynn-WP.pdf.

[32] "Xilinx Spartan-6 FPGA LX9 MicroBoard." http://www.em.avnet.com/en-us/design/drc/Pages/Xilinx-Spartan-6-FPGA-LX9-MicroBoard.aspx.

[33] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003.

[34] L. B. Oliveira, D. F. Aranha, C. P. L. Gouvêa, M. Scott, D. F. Câmara, J. López, and R. Dahab, "TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks," *Computer Communications*, vol. 34, no. 3, pp. 485–493, 2011.

[35] C. Dods, N. Smart, and M. Stam, "Hash based digital signature schemes," in *Cryptography and Coding* (N. Smart, ed.), vol. 3796 of *Lecture Notes in Computer Science*, pp. 96–115, Springer Berlin Heidelberg, 2005.

[36] "Texas Instruments MSP430x5xx and MSP430x6xx Family User's Guide 2013." http://www.ti.com/lit/ug/slau208m/slau208m.pdf.