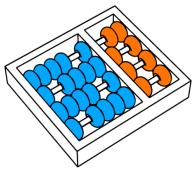
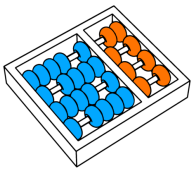



Conrado Porto Lopes Gouvêa

# "Software Implementation of Cryptography for Wireless Sensors and Mobile Processors"

# "*Implementação em Software de Criptografia para Sensores sem Fio e Processadores Móveis*"

**CAMPINAS**
**2013**




**University of Campinas**
**Institute of Computing**

***Universidade Estadual de Campinas***
***Instituto de Computação***

**Conrado Porto Lopes Gouvêa**


# "Software Implementation of Cryptography for Wireless Sensors and Mobile Processors"

Supervisor:
*Orientador(a):* **Prof. Dr. Julio César López Hernández**

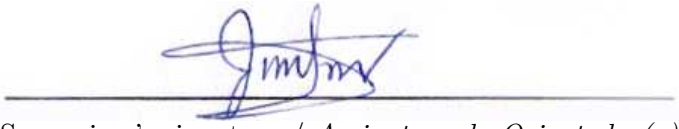
# *"Implementação em Software de Criptografia para Sensores sem Fio e Processadores Móveis"*

PhD Thesis presented to the Post Graduate Program of the Institute of Computing of the University of Campinas to obtain a PhD degree in Computer Science.

*Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Doutor em Ciência da Computação.*

This volume corresponds to the final version of the Thesis defended by Conrado Porto Lopes Gouvêa, under the supervision of Prof. Dr. Julio César López Hernández.

*Este exemplar corresponde à versão final da Tese defendida por Conrado Porto Lopes Gouvêa, sob orientação de Prof. Dr. Julio César López Hernández.*



Supervisor's signature / *Assinatura do Orientador(a)*

CAMPINAS
2013

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Maria Fabiana Bezerra Muller - CRB 8/6162

G745s

Gouvêa, Conrado Porto Lopes, 1984-
Software implementation of cryptography for wireless sensors and mobile processors / Conrado Porto Lopes Gouvêa. – Campinas, SP : [s.n.], 2013.

Orientador: Julio César López Hernández.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Criptografia. 2. Curvas elípticas. 3. Emparelhamentos bilineares. 4. Aritmética de computador. I. López Hernández, Julio César,1961-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

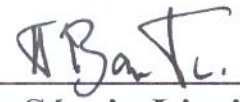
Informações para Biblioteca Digital

**Título em outro idioma:** Implementação em software de criptografia para sensores sem fio e processadores móveis
**Palavras-chave em inglês:**
Cryptography
Elliptic curves
Bilinear pairings
Computer arithmetic
**Área de concentração:** Ciência da Computação
**Titulação:** Doutor em Ciência da Computação
**Banca examinadora:**
Julio César López Hernández [Orientador]
Josep Maria Miret Biosca
Paulo Sérgio Licciardi Messeder Barreto
Marco Aurélio Amaral Henriques
Ricardo Dahab
**Data de defesa:** 08-11-2013
**Programa de Pós-Graduação:** Ciência da Computação

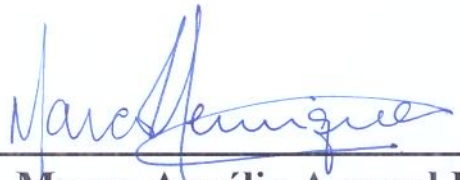
# TERMO DE APROVAÇÃO

Tese Defendida e Aprovada em 08 de novembro de 2013, pela Banca examinadora composta pelos Professores Doutores:

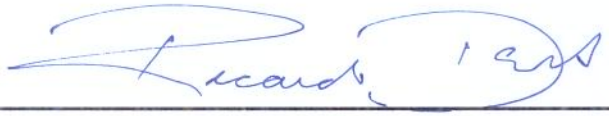
**Prof. Dr. Josep Maria Miret Biosca**
**UNIVERSITAT DE LLEIDA**



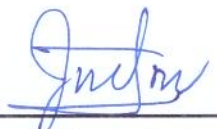
**Prof. Dr. Paulo Sérgio Licciardi Messeder Barreto**
**PSC / USP**



**Prof. Dr. Marco Aurélio Amaral Henriques**
**FEEC / UNICAMP**



**Prof. Dr. Ricardo Dahab**
**IC / UNICAMP**



**Prof. Dr. Julio César López Hernández**
**IC / UNICAMP**

Institute of Computing /*Instituto de Computação*
University of Campinas /*Universidade Estadual de Campinas*

# Software Implementation of Cryptography for Wireless Sensors and Mobile Processors

**Conrado Porto Lopes Gouvêa**[1]

November 08, 2013

**Examiner Board/*Banca Examinadora*:**

- Prof. Dr. Julio César López Hernández (Supervisor/*Orientador*)
- Prof. Dr. Josep Maria Miret Biosca
  Department of Mathematics - University of Lleida
- Prof. Dr. Paulo Sérgio Licciardi Messeder Barreto
  Department of Computer and Digital Systems Engineering - University of São Paulo
- Prof. Dr. Marco Aurélio Amaral Henriques
  School of Electrical and Computer Engineering - University of Campinas
- Prof. Dr. Ricardo Dahab
  Institute of Computing - University of Campinas
- Prof. Dr. Anderson Clayton Alves Nascimento
  Electrical Engineering Department - University of Brasília (Substitute/*Suplente*)
- Dr. Roberto Alves Gallo Filho
  Institute of Computing - University of Campinas (Substitute/*Suplente*)
- Prof. Dr. Paulo Lício de Geus
  Institute of Computing - University of Campinas (Substitute/*Suplente*)

[1]Financial support: FAPESP scholarship (process 2010/15340-3) 2010–2013. The author also acknowledges the financial support given to this work, under the project "Security Technologies for Mobile Environments — TSAM", granted by the Fund for Technological Development of Telecommunications (FUNTTEL) of the Brazilian Ministry of Communications, through Agreement Nr. 01.11.0028.00 with the Financier of Studies and Projects — FINEP / MCTI.

# Abstract

The efficient and secure implementation of cryptographic schemes is an important aspect of practical cryptography. In this work, we focus on the software implementation of relevant algorithms in elliptic curve cryptography (ECC), pairing-based cryptography (PBC) and in authenticated encryption (AE). Two modern computational platforms were targeted: the MSP430 microcontroller, often used in wireless sensor networks, and the ARM processor, widely employed in mobile devices such as smartphones and tablets which are increasingly becoming ubiquitous. Techniques for improving the software performance by taking advantage of instruction sets, peripherals and algorithmic enhancements are described. The secure implementation, which aims at thwarting common side-channel attacks, is also studied and new techniques are provided for improving its efficiency on ARM processors. These results contribute to the building of efficient and secure cryptographic systems on wireless sensors and mobile processors.

# Resumo

A implementação eficiente e segura de esquemas criptográficos é um aspecto importante da criptografia aplicada. Neste trabalho, foca-se na implementação em software de algoritmos relevantes da criptografia de curvas elípticas (CCE), criptografia baseada em emparelhamentos (CBE), e de cifração autenticada (CA). Duas plataformas computacionais modernas foram utilizadas: o microcontrolador MSP430, bastante utilizado em redes de sensores sem fio, e o processador ARM, amplamente empregado por dispositivos móveis como smartphones e tablets que estão se tornando cada vez mais populares. Técnicas para a melhoria de desempenho em software utilizando conjuntos de instruções, periféricos e melhorias algorítmicas são descritas. A implementação segura, cujo objetivo é prevenir certos ataques de canais secundários, também é estudada e novas técnicas são providas para reduzir seu impacto na velocidade em processadores ARM. Tais resultados contribuem para a construção eficiente e segura de sistemas criptográficos em sensores sem fio e processadores móveis.

# Acknowledgements

À minha família: Ubirajara, Solange (in memoriam) e Anahy, por todo o apoio, carinho e incentivo.

À Daniela, pelo amor, e paciência quando eu insistia que "ia dar tudo errado".

Ao Florivaldo e Shirley, pela continuada acolhida em Campinas.

Ao meu orientador e professor, Julio López, pelos importantes conselhos, sugestões e idéias.

Ao professor Ricardo Dahab, Danilo Câmara, Diego Aranha, Leonardo Oliveira, Roberto Gallo e demais integrantes do Laboratório de Criptografia Aplicada pela colaboração e apoio.

Aos professores Orlando Soares e Carlos Maziero, da PUCPR, por me incentivarem a seguir carreira acadêmica.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e à Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), pelo apoio financeiro.

*"One person's clear mental image is another person's intimidation."*
William Thurston

# Contents

# List of Acronyms

**AD** Associated Data

**AES** Advanced Encryption Standard

**AE** Authenticated Encryption

**BLS** Boneh-Lynn-Shacham (short signature algorithm)

**CCM** Counter with CBC-MAC

**CTR** Counter (block cipher chaining mode)

**DES** Data Encryption Standard

**DLP** Discrete Logarithm Problem

**DSA** Digital Signature Algorithm

**DSP** Digital Signal Processor

**ECC** Elliptic Curve Cryptography

**ECDHP** Elliptic Curve Diffie-Hellman Problem

**ECDH** Elliptic Curve Diffie-Hellman (key agreement algorithm)

**ECDLP** Elliptic Curve Discrete Logarithm Problem

**ECDSA** Elliptic Curve Digital Signature Algorithm

**ECMQV** Elliptic Curve Menezes-Qu-Vanstone (key agreement algorithm)

**ECSS** Elliptic Curve Schnorr Signature

**FIPS** Federal Information Processing Standards

**GCC** GNU Compiler Collection

**GCM** Galois/Counter Mode

**GLV** Gallant-Lambert-Vanstone (point multiplication technique)

**HMAC** Hash-Based Message Authentication Code

**HTTPS** Hypertext Transfer Protocol Secure

**IBC** Identity-Based Cryptography

**IPI** Internet Performance Index

**IPSec** Internet Protocol Security

**KGC** Key Generation Center

**KNV** Karatsuba/NEON/VMULL (binary multiplier)

**LD** López-Dahab (binary multiplication, point coordinates)

**MAC** Multiply and Accumulate (also Message Authentication Code)

**NAF** Non-Adjacent Form

**NIST** National Institute of Standards and Technology

**OCB3** Offset Codebook

**PBC** Pairing-Based Cryptography

**PKC** Public Key Cryptography

**PKI** Public Key Infrastructure

**RAM** Random Access Memory

**RFID** Radio-Frequency Identification

**RISC** Reduced Instruction Set Computing

**ROM** Read-Only Memory

**RSA** Rivest-Shamir-Adleman (signature and encryption algorithms)

**SCA** Side Channel Attack

**SCR** Side Channel Resistance

**SGCM** Sophie Germain Counter Mode

**SIMD** Single Instruction Multiple Data

**SOK** Sakai-Ohgishi-Kasahara (key agreement algorithm)

**SSH** Secure Shell (network protocol)

**TLS** Transport Layer Security (network protocol)

**WSN** Wireless Sensor Network

**ZSS** Zhang-Safavi-Naini-Susilo (short signature algorithm)

# List of Tables

# List of Figures

# List of Algorithms

# Chapter 1

# Introduction

Modern cryptography is mainly built on two areas of study: symmetric and asymmetric cryptography. Symmetric systems allow participants who share the same key to communicate with confidentiality and authentication. Confidentiality means that only parties who know the key used to encrypt a message will be able to read it; authentication means that the recipient of a message can detect if it was really generated by a party who holds the key used to authenticate it. Even though symmetric systems offer a high degree of security and efficiency, they are difficult to use in practice since the participants need to agree on which key to use in a secure manner, even though they can be far apart and only have access to insecure channels of communication that can be eavesdropped or manipulated. Asymmetric cryptography attempts to solve this issue by splitting the key into two parts: the private key, which is kept secret by its owner, and the public key, which can be sent to any other participant without security risks. This allows secure communication with any party, as long as their public key is known.

Symmetric cryptography itself is, for the most part, composed by two types of schemes: ciphers and message authentication codes (MACs). The former enable confidentiality and the latter enable authentication. Ciphers can be classified as block ciphers (which split the message into fixed-size blocks) and stream ciphers (which can process each bit of the message separately). The first widespread block cipher was Data Encryption Standard (DES), which was published in 1977. Amazingly, even today it can only be broken by brute force, and is no longer being used due to its small key size (56 bits) and block size (64 bits). The Advanced Encryption Standard (AES), published in 1998, is now the widespread block cipher, supporting key sizes of 128, 192 and 256 bits. Stream ciphers are more employed in restricted hardware since they are often more efficient; one well known stream cipher is RC4, which suffers from some weaknesses such as biased output. Two popular MACs are CBC-MAC and HMAC, the first is built upon a block cipher and the second upon a hash function. Cipher and MACs are often used together;

however, since there are many pitfalls when employing both type of schemes together (for example, different keys must be used for each one), a new type of scheme surfaced: authenticated encryption (AE), which provides both confidentiality and authentication in a single scheme.

The great idea of asymmetric cryptography (also known as Public Key Cryptography (PKC)) was born with the work of Diffie and Hellman [43] in 1976, which proposed a public key agreement scheme where two parties can compute a shared key through a channel which can be eavesdropped by adversaries; this shared key can then be used with symmetric cryptography to carry out the communication between parties. However, it was the work of Rivest, Shamir and Adleman [134] in 1977 which enabled the widespread adoption of public key cryptography with the RSA public key encryption and signature schemes. It is important to also mention the work of Ralph Merkle, who first proposed a public key scheme in 1974 which was rejected both by his professor and by the journal Communications of the ACM; and the work of James H. Ellis, Clifford Cocks, and Malcolm Williamson at the Government Communications Headquarters (GCHQ) in the United Kingdom, which developed asymmetric schemes in 1973 — a fact which was declassified only in 1997. In addition to key agreement, other important asymmetric schemes are encryption (which provides confidentiality) and digital signatures (which provides not only authentication but also non-repudiation, since a signed message can be traced to only one person: the holder of the private key used for signing).

Public key cryptography, however, has its flaws. A critical requirement of PKC is to have assurance on who is the owner of the public key being used in an asymmetric scheme. Without this assurance, it is possible to carry out a man-in-the-middle attack where an adversary can intercept the communication between two parties and replace the public key being sent with his own. This problem can be solved with a Public Key Infrastructure (PKI), where certification authorities verify the ownership of public keys and generate signed certificates binding the publics keys to the identity of their holders. This leaves the problem of authenticating the public keys of the authorities themselves; this is accomplished by a certification chain leading to a set of root authorities which the participants of the scheme must trust.

This is the approach used for secure communication in the Internet; however, it is not adequate in many scenarios where the complexity of a PKI can be too costly. In this context, Identity-Based Cryptography (IBC) was born with the work of Shamir [144] in 1985, where the public key is replaced with the identity of the participant (e.g. its ID number, email, etc), thus making it implicitly authenticated. The downside of this approach is that the private keys must be generated by a Key Generation Center (KGC), which therefore is able to impersonate any participant of the system. Nevertheless, IBC can be useful in scenarios where there is trust in a central authority, e.g. inside a company

or in wireless sensor networks. Concrete IBC schemes were only developed much later, with three independent works [28, 38, 139].

Asymmetric cryptography is known for its number theoretic grounds. RSA uses the arithmetic of numbers modulo the product of two large primes, and relies on the difficulty of factoring these products. However, the existence of specialized algorithms for factoring with sub-exponential complexity forces the use of relatively larger numbers. For example, 3072-bit numbers are required in the 128-bit level of security, which means that an attacker must carry computations in the order of $2^{128}$ steps in order to break a RSA system employing 3072-bit numbers. Thus, an alternate construction of asymmetric cryptography was proposed independently by Neal Koblitz [86] and Victor Miller [111]: the Elliptic Curve Cryptography (ECC). It can be much faster than RSA since, except in special cases, only exponential attacks are known for breaking ECC. For example, for the 128-bit level of security, numbers with only 256 bits are required. Since then, ECC gained more popularity, specially in environments where performance is critical. The most efficient identity-based systems also employ ECC, but only as the basis for its main building block: the bilinear pairing. This is a powerful mathematical tool which enabled the creation of not only IBC, but other interesting schemes such as tripartite one-round key agreement and short signatures, which can be half the size of usual ECC-based digital signatures. The efficient computation of pairings was based on the work of Miller [112] in 1986; schemes using them compose the Pairing-Based Cryptography (PBC).

## 1.1 Objectives

The implementation of cryptographic schemes is a important subject for mainly two reasons: efficiency and security.

Ideally, cryptography should be transparent, with low impact on the performance of computer systems. Since security is intangible and often not appreciated by users (unless when broken), it is often sacrificed in the name of performance. For example, laptops should have their disk encrypted in order to prevent access to sensitive data if stolen (which often happens). However, if disk encryption reduced the performance of the system too greatly, most users would refrain to use it. This issue becomes more important with the advent of mobile devices, which have reduced processing power and, maybe more importantly, limited battery charge. Another example of system where efficiency is crucial are those with heavy usage, such as popular webmail services. To give a concrete example, Google Mail switched to encrypted connections by default in 2010 [140], claiming that the increase in CPU usage was only 1% [92]. However, they use RC4 as the default cipher, which was known for having weaknesses which led to a concrete attack against Transport Layer Security (TLS) in 2013 [4]. The reason for using RC4 in the first place was probably

due to its high efficiency compared to AES, showing how a lack of performance can lead to sacrifices in security. Another interesting observation is that many HTTPS servers (including Google's) use RSA with 1024-bit modulus, which is now often considered too small and is being phased out by NIST [16], who disallows its use after 2013. Again, the reason for using 1024-bit is probably to improve performance.

Secure implementations are also extremely important. Of course, a incorrect implementation may lead to breaks in security, but even a correct implementation may be attacked by using information from side-channels. Side Channel Attacks (SCAs) take advantage from information such as the execution time, power consumption and cache usage from cryptographic schemes in order to obtain secret information such as plaintext or even keys. Fortunately, it is possible to avoid many types of these attacks with careful implementation of the schemes. It is worth mentioning that these attacks are more easily carried out in mobile devices due to their small size, which facilitates physical access by attackers.

Based on these observations, the objective of this thesis is to propose and describe methods for implementing existing cryptographic schemes in software with efficiency and security, in two concrete platforms. The first platform is the MSP430 microcontroller, a cheap processor with extremely low power consumption which is often used in Wireless Sensor Networks (WSNs). For the MSP430, only efficiency was considered, since in the scenario of WSNs it is often considered acceptable that the attacker may compromise part of the nodes. The second platform is the ARM processor, which is widespread in mobile devices. These devices are becoming ubiquitous and often hold a lot of sensitive information from its owner, indicating the need for cryptographic protection. Due to their limited power supply with batteries, the performance of cryptographic schemes becomes relevant for them. Secure implementation is now more important, since it is easy for an attacker to obtain physical access to the devices and any compromise can be critical.

Three types of schemes were implemented: digital signatures, key agreement and authenticated encryption. Together, they can meet most of the cryptographic needs of many systems. For the asymmetric schemes, we employ ECC due to its high efficiency and PBC due to its interesting applications, including IBC. For symmetric cryptography, AE was chosen since it is becoming increasingly popular due to repeated failures in composing ciphers and authentication codes correctly, as illustrated by the recent attack against TLS [5].

## 1.2 Methodology

An efficient implementation usually starts with a reference implementation, whose correctness can be determined with a test suite. In our case, the RELIC toolkit [7] was

used. This implementation can then be profiled (using tools such as `gprof`) in order to determine which parts of the source take the most time in execution and thus can lead to more gains when optimized. This step can by sped up with previous knowledge of critical paths; for example, it is widely known that finite field arithmetic takes the most part of the execution time in ECC schemes.

In order to obtain better performance, the characteristics of the underlying platform (instructions and their timings, peripherals, etc.) must be studied in order to improve the performance of the critical paths. It is also possible to improve the algorithm themselves with mathematical tricks, manipulating formulas and so on.

Finally, each improvement must be benchmarked in order to evaluate its effectiveness. This benchmark can also be done with profiling tools; however, these tools usually interfere with the timings reported and can lead to misleading results. For this reason, it is also important to measure the speed of algorithms using non-interfering techniques, such as simply measuring the time taken by an algorithm to execute (usually inside a loop with many iterations, to improve the precision). These measurements can be done either with regular timers (which use e.g. nanoseconds) or cycle-based timers, which are often more accurate.

A secure implementation starts with determining which data must be kept secure (usually keys, but sometimes other values) and which parts of the algorithms deal with this data. The implementation must avoid paths dependent on this secret data, for example, a loop whose number iteration is the number of bits of the data; and it must also avoid branching on secret data and using secret data as indices into arrays. More details are given later in this document.

Our software implementation follows the usual approach of an efficient implementation: using the C language for high level code, and assembly code for specific critical functions in the underlying arithmetic.

The implementation of ECC and PBC was built in a modular manner, as illustrated by Figure 1.1; each abstraction level relies on the lower level. The base level consists of the performance-critical functions mostly implemented in assembly. The AE implementation is simpler, with a high-level interface and some specific functions in assembly.

On the MSP430, the MSPsim simulator was used [50] in order to obtain timings and memory usage. Since it is a cycle accurate simulator it provides the same exact timings from real hardware. Nevertheless, a development board was also used to check the accuracy of the timings. On the ARM, timings were obtained using the `clock_gettime` Linux function in three development boards (with ARM Cortex A8, A9 and A15 processors).

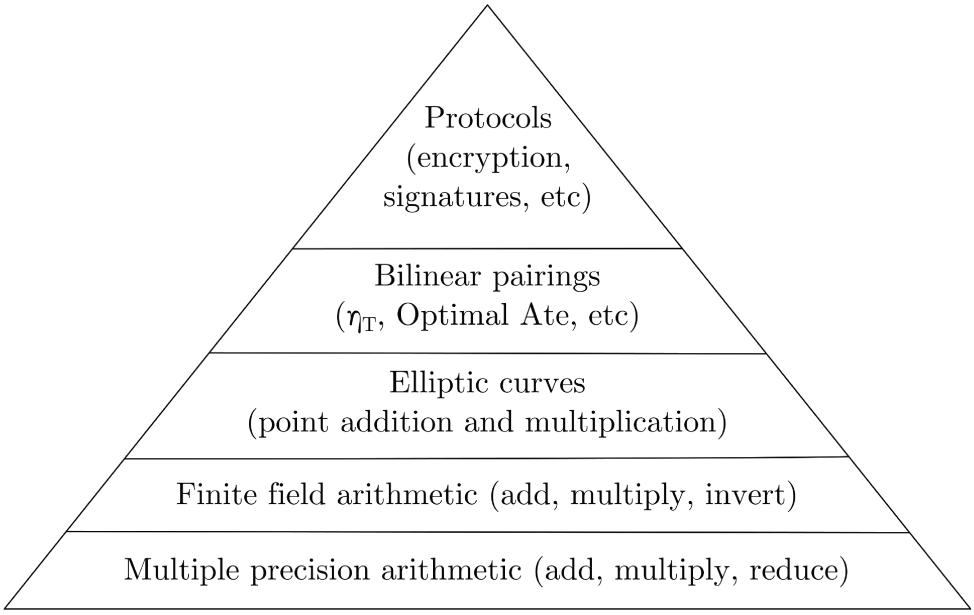



Figure 1.1: Structure of the software implementation

## 1.3 Architectures

### 1.3.1 MSP430

The MSP430 is a family of 16-bit microcontrollers which share the same architecture but have different clock, RAM, flash and peripherals. Due to its low power consumption, it is widely used in application such as wireless sensor networks, and is present in many wireless sensors such as the Tmote Sky and TelosB. Clock speeds range from 4 to 25 MHz; RAM sizes range from 125 bytes to 66 KB; and flash sizes range from 512 bytes to 512 KB.

The MSP430 architecture is fairly simple, with a instruction set where operands can be in registers or in memory. Most notably, it provides 1-bit shifts only, and no multiply nor divide instructions. There are 12 general purpose registers. Instruction timings are completely predictable, mainly involving one cycle to fetch a word from memory (which applies to instructions, extension words, or operands) and two cycles to write a word into memory. Many devices in the family have a hardware multiplier, which supports $16 \times 16$-bit multiply and multiply-and-accumulate. A few devices with integrated radio transceivers also feature an AES accelerator, which is able to encrypt and decrypt using 128-bit keys.

Since the architecture was limited to a 16-bit address space, which totals 64 KB, it was then extended into the MSP430X architecture which provides 20-bit registers, increasing the address space to 1 MB. Some new instructions were added to handle 20-bit values, and additional support for up to 4-bit shifts was added.

Three memory models where defined for these new devices. The small memory model uses the old MSP430 architecture and its memory is limited to the 16-bit address space. The medium memory model uses 20-bit pointers to functions but 16-bit pointers to data, allowing code to occupy the entire 20-bit address space while limiting data to the 16-bit address space. This is the simplest choice for taking advantage of the MSP430X, since code space is often more critical than data space in cryptographic implementations; additionally, porting MSP430 assembly code for this model is simpler since it only requires changing the function call and return instructions. For this reason, the medium model was used in this work. The large memory model uses 20-bit pointers for both instruction and data, while limiting the size of data objects to 32 KB. It suffers from lower performance, since every memory read and write is doubled (20-bit data is stored in two 16-bit words) and it hinders the porting of MSP430 assembly code. The huge memory model is similar to the large model, but lifts the restriction on the size of data objects.

### 1.3.2 ARM

The ARM is a 32-bit architecture which is widely employed in mobile processors due to its low power consumption compared to similar processors. It is actually just an architecture specification, which is implemented by a multitude of vendors. The architecture has a series of versions, the most current being the ARMv7. Recent mobile devices feature the ARM Cortex series, which are processor core designs implementing the ARMv7 architecture. Again, these designs are also implemented by a multitude of vendors.

The ARM architecture features a RISC instruction set where operands must be explicitly loaded from and written to memory. It provides an integrated shifter which allows one of the operands of most instructions to be shifted without the need for a separate instruction. It provides 13 general purpose 32-bit registers. The ARMv7 also specifies an optional Single Instruction Multiple Data (SIMD) extension branded as the NEON engine. It features several instructions operating on 64-bit registers and, in some cases, 128-bit registers. There are 32 64-bit NEON registers, which can also be viewed as 16 128-bit registers.

Since ARM is a complex architecture compared to MSP430, including wide pipelines and caches, it is hard to estimate the cycles taken by a program and for this reason all measurements were taken on actual hardware. Still, the manual of the Cortex A8 and A9 processors list the cycles taken by each instruction, including in which cycle of the pipeline the operands are read and written. This implies that pipeline stalls depend not only on the order of instructions, but also on which instructions are being processed. This information is useful when designing assembly algorithms. On the Cortex A15, however, the presence of out-of-order execution and dual-issue makes it hard to estimate the number

of cycle taken and the mentioned list is no longer provided by the vendor.

## 1.4 Mathematical Foundation

### 1.4.1 Elliptic Curves

Elliptic curves are a mathematical construction which appeared in the study of elliptic integrals involved in the computation of the arc length of an ellipse. Their first application in cryptography was shown by Hendrik W. Lenstra [95] on a method for factoring integers; afterward, they were used to build cryptosystems by Victor S. Miller [111] and Neal Koblitz [86], independently. They are now the basis of many cryptographic protocols and are also the foundation of bilinear pairings.

An elliptic curve $E$ over a field $K$, denoted $E/K$, is defined by the equation

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1.1}$$

where $a_1, a_2, a_3, a_4, a_6 \in K$. Equation 1.1 is also known as the generalized Weierstrass equation. In cryptography, only finite fields are used; in particular fields with prime order (denoted $\mathbb{F}_p$, where $p$ is a large prime, and referred to as prime fields) and fields whose order is a power of two (denote $\mathbb{F}_{2^m}$, where $m$ is prime, and referred to as binary fields). A generic field is written as $\mathbb{F}_q$. If the elliptic curve is defined over $\mathbb{F}_p$ then it can be transformed into the simplified equation

$$y^2 = x^3 + ax + b \tag{1.2}$$

with $a, b \in \mathbb{F}_p$. If the elliptic curve is defined over $\mathbb{F}_{2^m}$ and $a_1 \neq 0$ then it can be transformed into the simplified equation

$$y^2 + xy = x^3 + ax^2 + b \tag{1.3}$$

with $a, b \in \mathbb{F}_{2^m}$. A point in a elliptic curve is a pair $P = (x, y)$ which satisfies the curve equation.

The striking aspect of elliptic curves is that it is possible to construct a group with their set of points. The group operation, named "point addition", can be defined geometrically. The sum of two points $P, Q \in E/K$ is shown in Figure 1.2, and can be defined as follows. Draw a line through the points $P$ and $Q$. This line will intercept the curve in a third point $T$. Now, draw a vertical line through $T$; the line will intercept the curve in another point $R$, which is defined as the sum of $P$ and $Q$. If the points $P$ and $Q$ are the same, as illustrated in Figure 1.3, then the first line is drawn tangent to the point; this is also called "point doubling" since $P + P = 2P$. If the first line drawn is vertical, then the result is

defined as the "point at infinity", written as $\infty$. Note that the figures show curves over the reals, even though finite fields are used in cryptography. However, the exact same group operation applies to them, despite losing the geometric visualization.

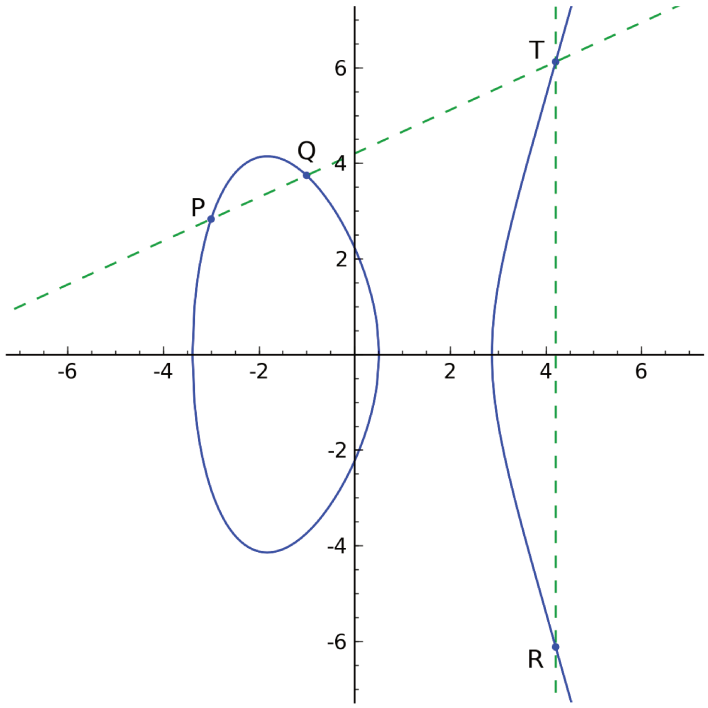



Figure 1.2: Elliptic curve over the reals given by $y^2 = x^3 - 10x + 5$ (solid line) showing the sum of points $P + Q = R$.

Let $E/K$ be an elliptic curve over the field $K$. The set

$$E(K) = \{(x, y) \in K \times K : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{\infty\}$$

forms a group together with the $+$ operation defined above, where the point at infinity is the identity element. A multiplication of a point $P$ by an scalar $k$ is defined as the sum of $k$ terms equal to $P$ and is written as $kP$. The order of an elliptic curve is defined as the number of elements in $E(K)$ and is denoted by $\#E(K)$. The order of a point $P$ is the smaller positive integer $k$ such that $kP = \infty$.

The Frobenius trace of an elliptic curve $E/\mathbb{F}_q$ is defined as $t = q + 1 - n$, where $n = \#E(\mathbb{F}_q)$. The well known Hasse theorem states that $|t| \leq 2\sqrt{q}$. This implies that $q$ and the order of the curve have approximately the same size in bits. Cryptography usually employs a prime order subgroup of $E(\mathbb{F}_q)$ generated by a point $G$, the generator of the curve. It is interesting to use the largest prime order subgroup possible, since it offers the most efficiency: using a smaller subgroup would require the same $\mathbb{F}_q$ arithmetic

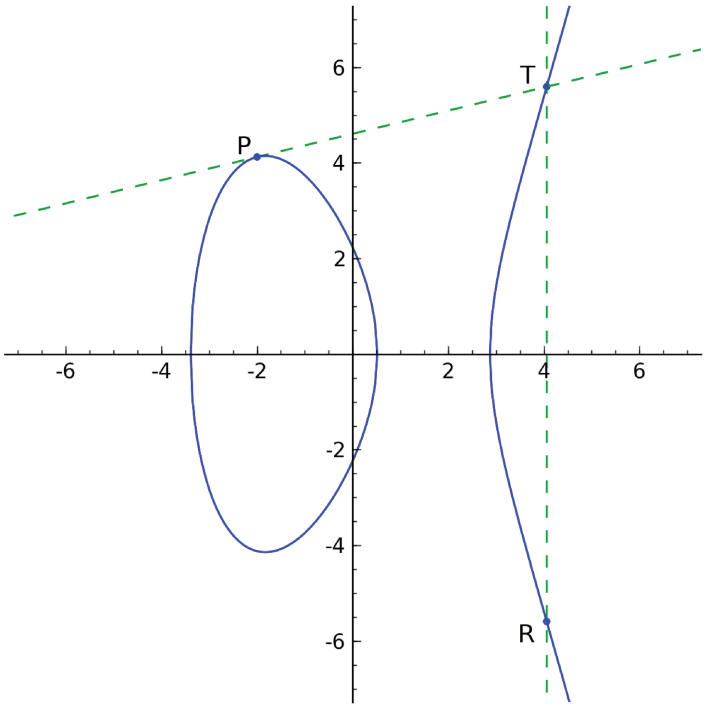



Figure 1.3: Elliptic curve over the reals given by $y^2 = x^3 - 10x + 5$ (solid line) showing the point doubling $P + P = R$.

but would offer a smaller level of security due to the smaller size. Thus, the curve order can be written as $n = rc$ where $r$ is the largest prime divisor of $n$ and $c$ is an integer. This way, $r$ is the order of the subgroup of $E(\mathbb{F}_q)$ actually used in cryptographic protocols; $c$ is named the curve cofactor.

The security of elliptic curve cryptography relies on the difficulty of computing $k$ given the points $P, Q$ such that $Q = kP$. This is known as the Elliptic Curve Discrete Logarithm Problem (ECDLP), and specialized algorithms for breaking it have not been found (except in special cases, e.g. using pairings as later described); currently we must rely on generic discrete logarithm algorithms which work on any group and have exponential complexity, such as Pollard's rho [118]. Many protocols rely on slightly modified problems, such as the Elliptic Curve Diffie-Hellman Problem (ECDHP) (given $kP$ and $\ell P$, compute $k\ell P$), which in theory could be easier to solve than the ECDLP. However, in most cases the easiest known way to solve them is to solve the ECDLP.

### 1.4.2 Bilinear Pairings

The first well known application of bilinear pairings were in a proof of a special case of the Riemann hypothesis by André Weil [156] in 1941. In 1986, Victor S. Miller described

an algorithm [112] to compute the Weil pairing in polynomial time. His intention was to try to reduce the discrete logarithm problem over the multiplicative subgroup of finite fields (on which relies the security of the Diffie-Hellman and ElGamal cryptosystems) to the ECDLP. This would show that the security of ECC would be at least as good as the classical asymmetrical cryptosystems. However, he managed to prove the inverse reduction, from ECDLP to the regular discrete logarithm problem, using the Weil pairing which can be computed in polynomial time in a specific class of elliptic curves. This was precisely the MOV attack [110] published in 1993 which demonstrated that these special curves could be broken more easily by using the Weil pairing. Later, it was observed that pairings could be used constructively by carefully selecting curves where the cost of breaking it through ECDLP was the same as breaking it through the Weil pairing. The first such application was the tripartite one-round key-exchange by Joux [78], which was soon followed by the birth of IBC protocols using pairings by Sakai, Ohgishi and Kasahara [139] and by Boneh and Franklin [28].

A bilinear pairing is abstractly defined as follows. Given three groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ with prime order $r$, with $\mathbb{G}_1$ and $\mathbb{G}_2$ written additively and $\mathbb{G}_T$ multiplicatively, a bilinear pairing is defined as a map $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties:

1. (bilinearity) for all $R$, $S \in \mathbb{G}_1$ and $T \in \mathbb{G}_2$ we have that

$$e(R+S,T) = e(R,T)e(S,T)$$

   and for all $R \in \mathbb{G}_1$ and $S$, $T \in \mathbb{G}_2$ we have that

$$e(R,S+T) = e(R,S)e(R,T)\,.$$

   An important consequence is that for every $a, b \in \mathbb{Z}$, $R \in \mathbb{G}_1$ and $S \in \mathbb{G}_2$ we have that $e(aR,bS) = e(R,S)^{ab}$;

2. (non-degeneracy) $e(R,S) = 1$ for all $S \in \mathbb{G}_2$ if and only if $R = 0$, analogously $e(R,S) = 1$ for all $R \in \mathbb{G}_1$ if and only if $S = 0$;

3. (computability) the map $e$ is efficiently computable.

When $\mathbb{G}_1 = \mathbb{G}_2$ we have a symmetric pairing, which must be constructed with supersingular elliptic curves. When $\mathbb{G}_1 \neq \mathbb{G}_2$, the pairing is called asymmetric.

### Rational Functions and Divisors

In order to give a concrete pairing construction, some definitions will be required. For a positive integer $r$, the *$r$-torsion points* of $E(\mathbb{F}_q)$ are the set $\{P \in E(\mathbb{F}_q) \mid rP = \infty\}$,

denoted by $E(\mathbb{F}_q)[r]$. In a field $\mathbb{F}_q$, the *r-th roots of unity* are the set $\{x \in \mathbb{F}_q \mid x^r = 1\}$. For a elliptic curve group $E(\mathbb{F}_q)$ of order $n$, given a prime $r$ which divides $n$ but does not divide $q-1$, the *embedding degree* of $E(\mathbb{F}_q)$ in relation to $r$ is the smallest positive integer $k$ such that $r$ divides $q^k - 1$. If $k$ is the embedding degree then $E(\mathbb{F}_{q^k})[r]$ has order $r^2$ and the $r$-th roots of unity in $\mathbb{F}_{q^k}$ have $r$ elements.

A *rational function* $f(x, y)$ in $E(\mathbb{F}_q)$ is a function $f(x, y) = g(x, y)/h(x, y)$ where $g(x, y)$ and $h(x, y)$ are polynomial functions with coefficients in $\mathbb{F}_q$ and where the domain of $f(x, y)$ is restricted to points $(x, y)$ in $E(\mathbb{F}_q)$. For a point $P = (x_P, y_P)$, it is possible to write $f(x_P, y_P)$ as $f(P)$. The *zeros* of a rational function $f(x, y) = g(x, y)/h(x, y)$ are all points $P$ such that $f(P) = 0$ and $g(P) \neq 0$, while the *poles* of the same rational function are all points $P$ such that $f(P) \neq 0$ and $g(P) = 0$.

A *divisor* $D$ on $E(\mathbb{F}_q)$ is a formal sum $D = a_1\langle P_1\rangle + \ldots + a_n\langle P_n\rangle$ where $a_i$ are integers and $P_i \in E(\mathbb{F}_q)$. The name "formal sum" implies that this is just a way to associate integers with points, and that the point multiplications and additions should not be carried out. Divisors can be added and subtracted similarly to polynomials. The *divisor of a rational function* $f(x, y)$ on $E(\mathbb{F}_q)$, denoted by $\mathrm{div}(f)$, is a divisor $D$ on $E(\mathbb{F}_q)$, written as above, where $a_i$ is the multiplicity of $P_i$ if it is a zero of $f(x, y)$ or the negative of the multiplicity of $P_i$ if it is a pole of $f(x, y)$. It is true that $a_1P_1 + \ldots + a_nP_n = \infty$ and $a_1 + \ldots + a_n = 0$ for all divisors of rational functions on elliptic curves.

In pairings, line functions play an important role. The line function for a vertical line through $P = (x_P, y_P) \in E(\mathbb{F}_q)$ is given by $v_P(x, y) = x - x_P$; its divisor is $\mathrm{div}(v_P) = \langle P\rangle + \langle -P\rangle - 2\langle\infty\rangle$. The line function for line tangent to $P$ is given by $t_P(x, y) = y - y_P - \lambda(x - x_P)$ where $\lambda = (3x_p^2 + a)/(2y_P)$; its divisor is $\mathrm{div}(t_P) = 2\langle P\rangle + \langle -2P\rangle - 3\langle\infty\rangle$. The line function for a non-vertical, non-tangent line through $P = (x_P, y_P), Q = (x_Q, y_Q) \in E(\mathbb{F}_q)$ is given by $\ell_{P,Q}(x, y) = y - y_P - \lambda(x - x_P)$ where $\lambda = (y_Q - y_P)/(x_Q - x_P)$; its divisor is $\mathrm{div}(\ell_{P,Q}) = \langle P\rangle + \langle Q\rangle + \langle -(P + Q)\rangle - 3\langle\infty\rangle$.

### Miller Function and Miller's Algorithm

A Miller function is the cornerstone of the pairing computation, and is defined as the rational function $f_{i,P}(Q)$ on $E(\mathbb{F}_{q^k})$ whose divisor is $\mathrm{div}(f_{i,P}) = i\langle P\rangle - \langle iP\rangle - \langle (i-1)\infty\rangle$, where $i$ is a positive integer and $P \in E(\mathbb{F}_q)$. (A function can be completely defined by its divisor, up to multiplication by a constant.) Miller's observation was that it is possible

to compute $f_{i,P}(Q)$ in polynomial time by taking advantage of the following recursion:

$$f_{i,P}(Q) = \begin{cases} 1 & \text{if } i = 1; \\ f_{i/2,P}(Q)^2 \cdot \dfrac{t_{(i/2)P}(Q)}{v_{iP}(Q)} & \text{if } i \text{ even}; \\ f_{i-1,P}(Q) \cdot \dfrac{\ell_{(i-1)P,P}(Q)}{v_{iP}(Q)} & \text{if } i \text{ odd}. \end{cases}$$

which can be written iteratively as described in Algorithm 1.1 (Miller's Algorithm). Note that Miller's algorithm implicitly computes $rP$ in the $Z$ variable.

**Algorithm 1.1** Miller's Algorithm

**Input:** $P \in E(\mathbb{F}_{q^k})[r]$, $Q \in E(\mathbb{F}_{q^k})$, $r \in \mathbb{N}$
**Output:** $f_{r,P}(Q) \in \mathbb{F}_{q^k}$

```
1: (r_{n-1}, ..., r_0)_2 ← r
2: x ← 1
3: Z ← P
4: for i = n − 2 to 0 do
5:     x ← x^2 · t_Z(Q)/v_{2Z}(Q)
6:     Z ← 2Z
7:     if r_i = 1 then
8:         x ← x · ℓ_{Z,P}(Q)/v_{Z+P}(Q)
9:         Z ← Z + P
   return x
```

### Tate Pairing

While the Weil pairing [156] was the first, the Tate pairing [153] quickly took over for being more efficient. In its modern version with contributions by Lichtenbaum [96] and Barreto [19], the Tate pairing $\tau_r \colon E(\mathbb{F}_q)[r] \times E(\mathbb{F}_{q^k})[r] \to \mathbb{F}_{q^k}^*$ is defined by $\tau_r(P, Q) = f_{r,P}(Q)^{(q^k-1)/r}$ for a prime $r$ that divides the order of $E(\mathbb{F}_q)$.

### Ate Pairing

The Ate pairing [72] halves the loop size in the Miller algorithm, and therefore can be faster. It uses the Frobenius map $\pi_q \colon E(\mathbb{F}_{q^e}) \mapsto E(\mathbb{F}_{q^e})$ which is defined as $\pi_q(Q) = (x_Q^q, y_Q^q)$ for any positive integer $e$.

The Ate pairing $\text{ate} \colon \{P \in E(\mathbb{F}_{q^k})[r] \colon \pi_q(P) = qP\} \times E(\mathbb{F}_q)[r]$ is defined by $\text{ate}(Q, P) = f_{T,Q}(P)^{(q^k-1)/r}$ where $T = t - 1$, $t$ is the Frobenius trace of $E(\mathbb{F}_q)$, and $r$ divides the order of $E(\mathbb{F}_q)$.

### BN Curves

Bilinear pairings require specific elliptic curves where the embedding degree $k$ is small, but not too small. This is because they can be attacked in two fronts. First, an adversary can try to attack the elliptic curve group $E(\mathbb{F}_q)$; the best known attacks for this (with some exceptions) are generic algorithms for computing discrete logarithms with exponential complexity such as Pollard's Rho [130]. Alternatively, the adversary can use the pairing itself to break the discrete logarithm in the curve by mapping it to the multiplicative subgroup of the finite field $\mathbb{F}^*_{q^k}$; the best known attack for this are sub-exponential algorithms such as the index calculus [1]. For this reason, $k$ must be large enough such that the cost of breaking $\mathbb{F}_{q^k}$ is at least the same as the cost of breaking $E(\mathbb{F}_q)$. However, if it is too big, then the pairing becomes too costly to compute.

To give a concrete example, at the 128-bit level of security, an elliptic curve with a subgroup with 256-bit prime order is required; while a field with 3072-bit size is required. Therefore, the ideal $k$ in this situation is $k = 12$.

For randomly chosen curves, $k$ usually has the same size as their order, which is too large to offer efficient pairings. For this reason, special families of curves were developed which have a fixed embedding degree. One of the most known families, which is extensively used in this work, are the BN curves [20].

For a given integer $z$, it is possible to attempt to create a BN curve $E/\mathbb{F}_p\colon y^2 = x^3 + b$ such that $p = p(z) = 36z^4 + 36z^3 + 24z^2 + 6z + 1$, its order is given by $n(z) = 36z^4 + 36z^3 + 18z^2 + 6z + 1$ and its Frobenius trace $t(z) = 6z^2 + 1$. Not every $z$ value successfully creates a BN curve, but they are numerous nevertheless.

### Eta Pairing

Another option of curves suitable for pairings are supersingular curves, which have $k \leq 6$. They are particularly interesting over binary fields, since in this case it is the only known way to achieve a small embedding degree. A binary curve is supersingular if it has the form $y^2 + cy = x^3 + ax + b$; its embedding degree is $k = 4$. Since $k$ is relatively small, it is necessary to use a larger base field in order to make $\mathbb{F}_{q^k}$ large enough. To give a concrete example, in the 80-bit level security, $q = 2^{353}$ is required. (This does not take into account recent attacks discovered while this work was being prepared, which will be described later.)

The eta pairing [17] (from which Ate was derived) applies to supersingular algebraic varieties, which includes supersingular elliptic curves, and it also halves the Miller loop. In the case of binary supersingular curves, the eta pairing $\eta_T\colon E(\mathbb{F}_{2^m})[r] \times E(\mathbb{F}_{2^m})[r]$ is defined by $\eta_T(Q, P) = f_{T,Q}(P)^{(2^{mk}-1)/r}$ where $T = 2^{(m+1)/2} \pm 1$.

**Optimal Ate pairing**

In 2008, the R-ate pairing [94] was created, which could further reduce the size of the Miller loop; in the case of BN curves, to 1/4th of the original size. The Optimal Ate pairing [154] was then created, being slightly faster, and its author conjectured that these pairing were optimal in the sense that no further reduction to the Miller loop could be obtained. This was later proven in [71].

The Optimal Ate pairing oate: $\{P \in E(\mathbb{F}_{q^k})[r] \colon \pi_q(P) = qP\} \times E(\mathbb{F}_q)[r]$ is defined in BN curves by $\text{oate}_r(Q, P) = (f \cdot \ell_{Q_3,-Q_2}(P) \cdot \ell_{-Q_2+Q_3,Q_1}(P) \cdot \ell_{Q_1-Q_2+Q_3,(6z+2)Q}(P))^{(p^k-1)/r}$ where $Q_i = \pi_p^i(Q)$, $z$ is the BN curve parameter, and $r$ divides the order of $E(\mathbb{F}_q)$.

## 1.5 Algorithms

The cornerstone of ECC is the point multiplication algorithm: given a point $P \in E(\mathbb{F}_q)$ and an integer $k$ such that $0 < k < \#E(\mathbb{F}_q)$, to compute $Q = kP$. The basic algorithm for point multiplication is the binary algorithm, based on the following recursion:

$$kP = \begin{cases} \infty & \text{if } k = 0, \\ 2(k/2)P & \text{if } k \text{ even}, \\ (k-1)P + P & \text{if } k \text{ odd}. \end{cases}$$

It can be converted in the iterative algorithm listed in Algorithm 1.2. It initializes an accumulator with $\infty$ and it loops through the bits of the binary expansion of $k$, doubling the accumulator and adding $P$ if the bit is 1. Note that, in average, it requires $t$ doublings and $t/2$ additions where $t$ is the bit-length of $k$.

**Algorithm 1.2** Binary point multiplication algorithm

**Input:** $k \in \mathbb{N}$, $P \in E(\mathbb{F}_q)$
**Output:** $kP$
1: $(k_{t-1}, \ldots, k_1, k_0)_2 \leftarrow k$
2: $Q \leftarrow \infty$
3: **for** $i = t - 1$ **to** 0 **do**
4: $\quad Q \leftarrow 2Q$
5: $\quad$ **if** $k_i = 1$ **then**
6: $\quad\quad Q \leftarrow Q + P$
7: **return** $Q$

There are many variations of the algorithm which speed it up. A common approach is to use a different recoding for $k$, such as the Non-Adjacent Form (NAF) [147] which writes

$k$ in base 2 but with coefficients in $\{-1, 0, 1\}$. The NAF-based multiplication requires $t$ doublings and $t/3$ additions, which is bit faster than the binary algorithm. There are also windowing methods, which either look at a window of $w$ bits of $k$ in each iteration or recode $k$ with coefficients inside a larger range (e.g. in $\{-3, -1, 0, 1, 3\}$); they require the precomputation of a table storing multiples of the base point $P$. The $w$NAF [147] combines both approaches, and requires $t$ doublings and $t/(w+1)$ additions.

Some schemes require the multiplication of a point which is known in advance. In this case, windowing methods can employ larger precomputed tables. A very efficient algorithm for the fixed point multiplication is the Comb method [98]. For $t$-bit scalars and window size $w$, let $d = \lceil t/w \rceil$. The Comb method is based on the observation that $kP$ can be written as

$$\begin{aligned} kP &= k_0P + k_1 2P + k_2 2^2 P + \ldots + k_{t-1} 2^{t-1} P \\ &= k_0P + k_d 2^d P + k_{2d} 2^{2d} P + \ldots + k_{(w-1)d} 2^{(w-1)d} P \\ &+ 2(k_1P + k_{d+1} 2^d P + k_{2d+1} 2^{2d} P + \ldots + k_{(w-1)d+1} 2^{(w-1)d} P) \\ &+ 2^2(k_2P + k_{d+2} 2^d P + k_{2d+2} 2^{2d} P + \ldots + k_{(w-1)d+2} 2^{(w-1)d} P \\ &+ \ldots \\ &+ 2^{d-1}(k_{d-1}P + k_{d+(d-1)} 2^d P + k_{2d+(d-1)} 2^{2d} P + \ldots + k_{(w-1)d+(d-1)} 2^{(w-1)d} P)\,. \end{aligned}$$

Let $T_a = a_0P + a_1 2^d P + a_2 2^{2d} P + \ldots + a_{w-1} 2^{(w-1)d} P$; then, by precomputing $T_a$ for every $w$-bit integer $a$ it is possible to compute the above sum line by line, bottom up. In each step, $w$ bits of $k$ are processed, leading to around $d$ additions and $d$ doublings. The method is detailed in Algorithm 1.3. The precomputed table needs to hold $2^w$ points.

**Algorithm 1.3** Comb method for fixed point multiplication

**Input:** $k \in \mathbb{N}$, $P \in E(\mathbb{F}_q)$, $w \in \mathbb{N}$, $T_a$ for every $w$-bit integer $a$
**Output:** $kP$

1: $(k_{t-1}, \ldots, k_1, k_0)_2 \leftarrow k$
2: $d \leftarrow \lceil t/w \rceil$
3: $Q \leftarrow \infty$
4: **for** $i = d - 1$ **to** $0$ **do**
5: $\quad Q \leftarrow 2Q$
6: $\quad a \leftarrow (k_{(w-1)d+i}, \ldots, k_{2d+i}, k_{d+i}, k_i)_2$
7: $\quad Q \leftarrow Q + T_a$
8: **return** $Q$

There are two improvements for the Comb algorithm. It is possible to run it over a encoding of $k$ with coefficients in $\{-1, 1\}$ [69, 97]; in this case, it is possible to halve the table size for a given $k$ since $T_{-a}$ for a given $a$ can be computed on the fly as $-T_a$. It is

also possible to employ multiple precomputed tables [97], where the additional tables are the same as the original multiplied by powers of 2. For example, using two tables, the second table is the first multiplied by $2^{t/2}$. In this case, two separate bit windows are read from the scalar $k$ in each step and added to the accumulator, sharing the point doubling. Therefore, with $n$ tables, the number of doublings is divided by $n$.

Side-channel attacks are a critical concern in point multiplication algorithms, since in many protocols the scalar $k$ is the private key, or a secret number which can lead to the private key. Naive implementations of the binary algorithm have been shown to be very susceptible to these attacks, since it is possible to determine if the bit processed in each step is 0 or 1 by measuring energy consumption. Windowing algorithms are also at risk: using a bit window from the scalar as an index in a lookup table can leak the bit window, since the lookup affects the processor cache [127].

The usual approach to avoid side-channel attacks in point multiplication is to make sure that in each step the same amount of computation is carried out. This can be achieved with special algorithms which obey this rule, such as the Montgomery ladder [118]. This algorithm keeps in memory a pair of points whose difference is the base point $P$ being multiplied. Let $k_{[u]}$ denote the leftmost $u$ bits of $k$; then $kP = k_{[t]}P$ can be computed with the following recursion:

$$(k_{[u]}P,\, k_{[u]}P + P) = \begin{cases} (\infty,\, P) & \text{if } u = 0, \\ (2k_{[u-1]}P,\, k_{[u-1]}P + (k_{[u-1]}P + P)) & \text{if } k_{[u]} \text{ even}, \\ (k_{[u-1]}P + (k_{[u-1]}P + P),\, 2(k_{[u-1]}P + P)) & \text{if } k_{[u]} \text{ odd}. \end{cases}$$

Note that, in the $u$-th step, the points $k_{[u-1]}P$ and $k_{[u-1]}P + P$ are available from the previous step. This implies that each step requires exactly a point doubling and a point addition, regardless of the bit being processed. The Montgomery ladder is listed in iterative form in Algorithm 1.4. If the elliptic curve is binary and non-supersingular, then it is possible to improve the speed of the ladder since the $x$ coordinate of the sum of a pair of points whose difference is a known point can be easily computed from only the $x$ coordinates of the pair [103].

It is also possible to modify existing point multiplication algorithms to add resistance to side-channel attacks. Lookup tables must be either avoided, or protected by reading all elements from the table sequentially in each index lookup, selecting the correct value with arithmetic operations as described in [91].

**Algorithm 1.4** Montgomery ladder point multiplication algorithm

**Input:** $k \in \mathbb{N}$, $P \in E(\mathbb{F}_q)$
**Output:** $kP$
1: $(k_{t-1}, \ldots, k_1, k_0)_2 \leftarrow k$
2: $Q_0 \leftarrow P$
3: $Q_1 \leftarrow 2P$
4: **for** $i = t - 2$ **to** $0$ **do**
5: **if** $k_i = 0$ **then**
6: $Q_1 \leftarrow Q_0 + Q_1$
7: $Q_0 \leftarrow 2Q_0$
8: **else**
9: $Q_0 \leftarrow Q_0 + Q_1$
10: $Q_1 \leftarrow 2Q_1$
11: **return** $Q_0$

## 1.6 Schemes and Protocols

### 1.6.1 Authenticated Encryption

An authenticated encryption scheme is composed of two algorithms: authenticated encryption and decryption-verification (of integrity). The authenticated encryption algorithm is denoted by the function $\mathcal{E}_K(N, M, A)$ that returns $(C, T)$, where $K \in \{0,1\}^k$ is the $k$-bit key, $N \in \{0,1\}^n$ is the $n$-bit nonce, $M \in \{0,1\}^*$ is the message, $A \in \{0,1\}^*$ is the associated data, $C \in \{0,1\}^*$ is the ciphertext and $T \in \{0,1\}^t$ is the authentication tag.

The *nonce* is a non-secret value that must be unique for each message (for a certain key); its purpose is to prevent the same plaintext being always encrypted to the same ciphertext. The impact of repeating a nonce depends on the scheme, but it can be catastrophic — a recent target of research are schemes where the impact of such misusage is limited. Some schemes support variable-length nonces, though in this work they are assumed to have fixed size. The *associated data (AD)* is authenticated by the algorithm, but not encrypted; this can be useful if some header must be sent in plaintext along with the encrypted message, for example, in an internet packet. It can also be useful to add a counter or timestamp to prevent replay attacks. The authentication tag is a key-dependent digest of the message; if an attacker alters the ciphertext in transit, the authentication tag will enable the recipient to detect the tampering. Some schemes also support variable-length tags; again, in this work they are assumed to have fixed size.

The decryption-verification algorithm is denoted by the function $\mathcal{D}_K(N, C, A, T)$ that returns $(M, V)$ where $K, N, C, A, T, M$ are as above and $V$ is a boolean value indicating

if the given tag is valid (i.e. if the decrypted message and associated data are authentic).

There are two authenticated encryption schemes standardized by NIST: CCM [157] and GCM [109]. CCM combines the CTR mode of encryption with the CBC-MAC authentication code. GCM uses binary field arithmetic in order to provide authentication, and CTR mode for encryption. Adoption was slow, but is increasing: both schemes were included in version 1.2 of the TLS protocol which is widely used by web browsers; they are regarded as the safest alternative considering recent attacks against other cipher suites used in TLS (e.g. the CRIME attack [135] against CBC mode as employed in TLS and the attack against RC4 [4] as employed in TLS). There are many other authenticated encryption schemes such as OCB3, Hummingbird-2, MASHA, and EAX. Currently, a competition (CAESAR [33]) is underway in order to select a new standard authenticated encryption scheme, analogous to the AES and SHA-3 competitions.

### 1.6.2 Elliptic Curve Cryptography

Elliptic Curve Cryptography requires participants to use the same elliptic curve in order to successfully carry out protocols. For this reason, a set of curves have been standardized; the most well known are the NIST curves [121], which are known by P-192, P-224, P-256, P-384, and P-521 (prime curves over finite fields with the specified bit-length); B-163, B-233, B-283, B-409, and B-571 (binary curves) and K-163, K-233, K-283, K-409, and K-571 (binary Koblitz curves). There exists a P-160 curve which has been deprecated, but it can be useful in some scenarios.

Two popular types of public key ECC schemes are digital signature and key agreement schemes. The first allows the holder of a key pair to sign a message using the private key such that any other party holding her public key can verify that the message received was exactly the same message signed (integrity), and that only the holder of the correspondent private key could sign it (non-repudiation). A key agreement scheme allows two parties to computed a shared secret over a insecure channel, holding only the public key of the other party. This shared secret can then be used to establish a secure channel using symmetric key cryptography, including authenticated encryption.

A well known ECC digital signature scheme is the Elliptic Curve Digital Signature Algorithm (ECDSA) [121]. It is composed of three algorithms: key generation, signing and verification. Key generation and signing both require a fixed point multiplication, while a simultaneous point multiplication is used in signature verification. A delicate part of ECDSA is the handling of the value $k$ in the signature generation. It must be randomly generated for each message being signed; otherwise an attacker can trivially compute the private key of the signer from two signatures generated with the same $k$ (which happened with the PlayStation 3 videogame console [32]). Additionally, the value $k$ must not leak;

for example, the timing information leaked by the OpenSSL ECDSA signature generation enabled a remote private key recovery attack [31].

The ECDSA requires the computation of the inverse of $k$ modulo the curve order $n$. This is an expensive operation, specially with side-channel resistance. The Elliptic Curve Schnorr Signature (ECSS) [141] is an alternative to ECDSA which does not require this inversion, and otherwise is much similar to ECDSA while providing signatures 25% smaller. It did not have much adoption probably due to its patents; however, they have recently expired.

The Elliptic Curve Diffie-Hellman (ECDH) is a key agreement protocol where the key generation uses a fixed point multiplication and key agreement requires a random point multiplication. The protocol can be extended in order to provide forward secrecy, which means that if an attacker is able to compromise the private key of one party than he will not be able to compute any key agreed in the past by this party. This basically requires using ephemeral key pairs and signing the public ephemeral key before sending it to the other party; this is the approach used, for example, in the TLS protocol. A more efficient alternative is to use an specialized key agreement protocol with forward secrecy such as Elliptic Curve Menezes-Qu-Vanstone (ECMQV), which again did not have a large adoption due to patents.

### 1.6.3 Pairing-Based Cryptography

For a $t$-bit level of security, the ECDSA and ECSS usually provide $4t$-bit and $3t$-bit signatures respectively, which are smaller than RSA signatures (whose size is the same as the RSA modulus size) for the same level of security. However, in some applications even shorter signatures are useful; for example, when the signature must be printed in a small receipt. This is exactly what is provided by pairing-based short signatures, which have $2t$ bits. The Boneh-Lynn-Shacham (BLS) [30] uses a fixed point multiplication in $\mathbb{G}_2$ for the key generation, a random point multiplication in $\mathbb{G}_1$ for the signing and two pairing computations for verification. The Zhang-Safavi-Naini-Susilo (ZSS) [158] (also independently invented by Boneh and Boyen [27]) also uses a fixed point multiplication in $\mathbb{G}_2$ for the key generation, a fixed point multiplication in $\mathbb{G}_1$ for signing and both fixed multiplication in $\mathbb{G}_2$ and a pairing computation for the verification.

The Sakai-Ohgishi-Kasahara (SOK) [139] is a non-interactive identity-based key agreement scheme. Being identity-based, each party only need to know each other's identities in order to compute a shared key. Being non-interactive, they do not need to communicate in order to compute this shared key, which is adequate in scenarios where communication is expensive. Each SOK key agreement uses a single pairing computation.

## 1.7 Contributions

The first line of the research presented in this work consisted in the efficient software implementation of ECC and PBC for the MSP430 microcontroller. The following works were published:

- Leonardo B. Oliveira, Diego F. Aranha, Conrado P. L. Gouvêa, Michael Scott, Danilo F. Câmara, Julio López, and Ricardo Dahab. TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks. *Computer Communications*, 34(3):485–493, 2011;

- Conrado P. L. Gouvêa and Julio López. Implementação em software de criptografia assimétrica para redes de sensores com o microcontrolador MSP430. In *Anais do X Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 419–432, 2010;

- Leonardo B. Oliveira, Aman Kansal, Conrado P. L. Gouvêa, Diego F. Aranha, Julio López, Bodhi Priyantha, Michel Goraczko, and Feng Zhao. Secure-TWS: Authenticating node to multi-user communication in shared sensor networks. *The Computer Journal*, 55(4):384–396, 2012;

- Conrado P. L. Gouvêa, Leonardo B. Oliveira, and Julio López. Efficient software implementation of public-key cryptography on sensor networks using the MSP430X microcontroller. *Journal of Cryptographic Engineering*, 2(1):19–29, 2012.

In the first paper, it was presented an efficient implementation of the SOK protocol, at the 70-bit level of security, for the AVR, MSP430 and ARM processors. The non-interactive nature of the protocol can save energy since it reduces the amount of communication needed. In this work, the author was responsible for the MSP430 implementation, where a 26% speedup was obtained compared to the state of the art.

The second paper presents efficient implementations of both ECC and PBC with the ECDSA and SOK protocols, over both prime and binary fields, at the 80- and 128-bit levels of security. The pairing computation obtained was 21–28% faster compared to the state of the art, a result achieved with improvements in the finite field arithmetic.

The third paper presents a comparison of short (BLS, ZSS) and regular (Schnorr, ECDSA) signature schemes, over both prime and binary fields, at the 80-bit level of security, for the AVR and MSP430 microcontrollers. It was concluded that in this particular scenario short signatures consume more energy than regular ones, since the energy saved in the transmission is smaller than the increased cost of computation. In this work, the author was responsible for the MSP430 implementation.

The fourth paper is an expanded version of the second (which was presented in a local symposium). It also studied the new MSP430X family of microcontrollers, featuring a 32-bit hardware multiplier. It described the implementation of Schnorr, ECDSA and ZSS signatures schemes and of the SOK and ECMQV key agreement schemes, over both prime and binary fields, at the 80- and 128-bit levels of security. It was shown that the new multiplier resulted in 20-30% faster protocol timings.

The second line of research focused on the efficient software implementation of authenticated encryption for the MSP430X. The following works were published.

- Conrado P. L. Gouvêa and Julio López. High speed implementation of authenticated encryption for the MSP430X microcontroller. In *ECRYPT Workshop on Lightweight Cryptography*, 2011;

- Conrado P. L. Gouvêa and Julio López. High speed implementation of authenticated encryption for the MSP430X microcontroller. In *LATINCRYPT 2012*, volume 7533 of *Lecture Notes in Computer Science*, pages 288–304. Springer Berlin / Heidelberg, 2012.

The second work is an expanded version of the first (which was presented in a peer-reviewed workshop without proceedings). It describes an implementation and comparison of six authenticated encryption schemes: Counter with CBC-MAC (CCM), Galois/Counter Mode (GCM), Sophie Germain Counter Mode (SGCM), Offset Codebook (OCB3), Hummingbird-2 and MASHA. The AES accelerator and 32-bit multiplier were studied in order to improve the performance of the schemes, and techniques for improving their speed were presented. The work also presents an efficient implementation of the AES block cipher specially tailored for 16-bit platforms.

The final line of research has focused on the ARM processor, more specifically on the ARM Cortex series. The following work was published.

- Danilo Câmara, Conrado P. L. Gouvêa, Julio López, and Ricardo Dahab. Fast software polynomial multiplication on ARM processors using the NEON engine. In *Security Engineering and Intelligence Informatics*, volume 8128 of *Lecture Notes in Computer Science*, pages 137–154. Springer Berlin / Heidelberg, 2013.

The main contribution of this work is a novel software binary polynomial multiplier using a specific instruction from the NEON engine, a single-instruction multiple-data extension present in the ARMv7 architecture. This multiplier is used in an efficient binary field implementation, whose use is demonstrated in ECC and authenticated encryption with the GCM scheme. The paper also employs state-of-the-art algorithms with side-channel resistance, describing how to implement them efficiently using NEON. While

it was not possible to break speed records held by non-standard prime curves in the literature, there was significant improvement over other binary curve implementations (70% speedup) and also in the GCM scheme.

**RELIC.**

The software implementation produced in this work is available online [1]. Most of it was integrated (and the rest is being) to the RELIC toolkit [7], a cryptographic open source library maintained by Diego Aranha and the author. This allows the reproduction and immediate application of the results obtained.

## 1.8 Recent Attacks

Cryptography is a ever evolving area of research with the peculiarity that an algorithm or an entire subarea can be rendered useless overnight by advances in cryptanalysis. This section covers attacks which are relevant to the schemes and algorithms implemented in this work.

Related-key attacks for the Hummingbird-2 authenticated cipher were published by Chai and Gong [35], Zhang et al. [159] (which contained errors and was later retracted), and finally by one of its authors, Saarinen [138]. A related key attack requires the collection of plaintext / ciphertext pairs which were encrypted by a set of keys which are somehow related; for example, with a short Hamming distance between then. The relevance of this kind of attack is still widely debated since keys should be chosen randomly, which would stop the attack. For this reason, their impact on this work is not great.

Saarinen has also identified a class of weak keys in the GCM authenticated encryption scheme [137]. When used, these keys leave the possibility of an attacker swapping certain blocks in an encrypted message such as the tag is still considered as authentic, which is considered a forgery. The probability of this attack working is $n/2^{128}$, where $n$ is the number of blocks in the message; this is smaller than the expected security of $1/2^{128}$ but may be considered small if $n$ is not too large. It also worth mentioning that the original proof of security of GCM [109] conceded a $n/2^{128}$ probability for attacks; thus the attack described does not reduce the assumed security of GCM.

Chatterjee et al. [36] pointed out that any authenticated encryption schemes whose only restriction on the nonce is that it must be not repeated under the same key are subject to an attack in the multi-user scenario, as follows. In the first phase, the attacker collects authenticated tags for a fixed nonce-message pair from $n$ different users; in the second phase, she computes the authentication tag of the same pair for any $w$ distinct

[1] `http://conradoplg.cryptoland.net/software/`

keys. If there is a collision between a tag in the first phase and a tag from the second (generated with a key $k$), then there is a good chance that the key used to generate the first tag is $k$. The attacker, now in possession of the user key, can make arbitrary forgeries (or decryptions) for this user. If the key and the tag both have $t$ bits then this attack has 50% chance of working if $nw = 2^t$, and has complexity $O(w)$. For instance, the attack requires $O(2^{108})$ time for $n = 2^{20}$ users and 128-bit tags and keys, which is smaller than the expected $O(2^{128})$. This attack applies to most AE schemes, including CCM, GCM, SGCM and OCB3. However, it is difficult to carry out in practice, since it requires a large number of users and does not reduce the security margin enough for a concrete break. The main objective of [36] was to point out that schemes which were supposedly proven secure could still be subject to attacks, even if theoretical. Additionally, the attack can be entirely avoided by using random nonces or incrementing nonces starting from a random value.

The most striking result comes from a series of works published independently by Antoine Joux [80] and Göloğlu et al. [58] breaking the discrete logarithm in some large binary fields such as $\mathbb{F}_{2^{14\times127}}$, $\mathbb{F}_{2^{27\times73}}$ and $\mathbb{F}_{2^{28\times113}}$, in a relatively small number of CPU hours, using specialized function field sieve algorithms. This is directly relevant to the security of pairing-based cryptography over binary curves $E(\mathbb{F}_{2^m})$, since their security also rely on the Discrete Logarithm Problem (DLP) over $\mathbb{F}_{2^{4\times m}}$. Intuitively speaking, these attacks are more dangerous to PBC when the largest prime factor of the exponent is larger; in the mentioned attacks, these factors are 73, 113 and 127, while PBC uses 271 (believed to provide 70 bits of security) and 353 (80 bits). Since there was a gap between those, it was not clear how big was the impact of these breaks. However, a few weeks later Joux announced a discrete logarithm computation in $\mathbb{F}_{2^{24\times257}}$ [79], which is much closer to the fields used in PBC. With this, it must be concluded that PBC over binary elliptic curves is insecure. Unfortunately, this renders part of this work obsolete, including the whole paper [124] and parts of other papers using the $\eta_T$ pairing. However, there is no reason to believe that prime PBC is threatened, neither ECC over binary curves.

## 1.9 Organization

This thesis contains a collection of the most important papers published by the author during his PhD, in non-chronological order. Papers were selected considering the results achieved and the participation of the author. In Chapter 2 [63], the work with ECC and PBC on the MSP430 is described, followed by the implementation of authenticated encryption also for the MSP430 in Chapter 3 [59]. Chapter 4 [37] presents the work with ECC and authenticated encryption for the ARM processor. Chapter 5 [125] presents the comparison of signatures and short signatures in wireless sensors; we remark that its

MSP430 timings were later improved in [59], but the reason we include it is to show a more concrete application of the algorithms studied in this work and to highlight a work with high degree of collaboration. Chapter 6 presents concluding remarks.

## 1.10 Corrections

Due to university regulations and copyright issues, the papers in this thesis were included without modifications. However, they contain some issues that were discovered or pointed out after publication. For this reason, this sections discusses those issues.

In most chapters, a distinction is made between "standard" and "non-standard" elliptic curves. The criteria used for this classification is to consider whether the curve is included in well-known standards such as from NIST and SECG; for this reason, a more accurate name would be "standardized" and "non-standardized" curves. I believe this criteria is important since many applications require interoperability with existing implementations, and using standardized curves can ease this task.

In Chapter 2, is is mentioned that changing from MSP430 to MSP430X improves timings by up to 15%, but does not explicitly explain why. The reason for this speed up is that, if an operand is written to but not read from, then the instruction takes one less cycle to execute in the MSP430X.

In Chapter 4, the ARM processor is described as being "low power". Clearly, this remark sounds odd when the microcontroller MSP430 is being covered in the other sections, since the MSP430 has a much lower power consumption compared to ARM. However, ARM is considered more energy efficient compared to similar 32-bit processors such as Intel x86.

Again in Chapter 4, some algorithms are described as taking "constant time". This is not entirely correct since the time taken by the algorithms also depends on the parameter size. What was meant is that the time taken by the algorithm is constant for a given elliptic curve or finite field.

# Chapter 2

# Efficient Software Implementation of Public-Key Cryptography on Sensor Networks Using the MSP430X Microcontroller

Conrado P. L. Gouvêa, Leonardo B. Oliveira and Julio López

## Abstract

In this work we describe a software implementation of Elliptic Curve Cryptography (ECC) and Pairing-Based Cryptography (PBC) for the MSP430 microcontroller family, which is used in wireless sensors. Digital signature, short signature and key distribution protocols were implemented at the 80- and 128-bit levels of security, over both binary and prime fields. The timing results of our software implementation show an improvement of about 25–30% in the pairing computation over previous implementations. We also provide results for the MSP430X extension of the original family, which has new instructions. In particular, using the new 32-bit hardware multiplier available in some MSP430X models, we have achieved a further improvement of about 45% in the prime field multiplication and 20–30% in protocol timings. The combination of fast algorithms and improved hardware allows us to show that even the 128-bit level of security can be considered feasible for this platform.

Conrado P. L. Gouvêa, Leonardo B. Oliveira, and Julio López. Efficient software implementation of public-key cryptography on sensor networks using the MSP430X microcontroller. *Journal of Cryptographic Engineering*, 2(1):19–29, 2012, with kind permission from Springer Science and Business Media.

## 2.1 Introduction

Wireless Sensor Networks (WSNs) are ad hoc networks consisted mainly of small sensor nodes with limited resources and one or more base stations, which connect the sensor nodes to the rest of the world. They are used for monitoring environments and provide fine-grained sensing to users.

Security in WSNs is of paramount importance because their applications range from battlefield reconnaissance and emergency rescue operations to surveillance and environmental protection. Besides, the fact that they monitor the environment raises innumerous privacy concerns that can be only mitigated through the use of security.

In this work, we describe a high speed software implementation of both Elliptic Curve Cryptography (ECC) and Pairing-Based Cryptography (PBC) for wireless sensors using the MSP430 family of microcontrollers (used in popular sensor nodes platforms such as the TelosB and the Tmote Sky). We show how one can combine algorithmic strategies and platform technology to efficient implement in software the Elliptic Curve Digital Signature Algorithm (ECDSA), Elliptic Curve Schnorr Signature (ECSS), and Zhang-Safavi-Naini-Susilo (ZSS) signature schemes as well as the Sakai-Ohgishi-Kasahara (SOK) and Elliptic Curve Menezes-Qu-Vanstone (ECMQV) key agreement protocols at both 80-bit and 128-bit security levels over both prime and binary fields.

Our main contributions are (i) to show how to implement these protocols efficiently on the MSP430; (ii) to our knowledge, present the fastest times published so far for the platform; (iii) assess the impact on performance of the MSP430X, the most recent representative of the MSP430 family; (iv) show how to take advantage of the new 32-bit hardware multiplier available in some MSP430X models; (v) and show the feasibility of the 128-bit level of security on modern MSP430 devices. Our results indicate improvements that ranges from 25% to 30% over previous timings.

The remainder of this work is organized as follows. In Section 2.2, we present the characteristics of the MSP430 family. In Section 2.3 we discuss the cryptographic protocols we have implemented. In Sections 2.4 and 2.5 we describe our implementation techniques and optimizations along with the results obtained. Finally, we discuss related work in Section 2.6 and conclude in Section 2.7.

## 2.2 The MSP430 Family

The microcontrollers from the MSP430 family have many characteristics in common, such as being 16-bit, having the same instruction set and 12 general-purpose registers. The clock frequency and ROM/RAM sizes varies for each member. Table 2.1 presents the features of some relevant microcontrollers, including the MSP430F1611 used in the

Table 2.1: Features of relevant MSP devices

| Microcontroller | MSPX | Clock (MHz) | ROM (KB) | RAM (KB) | MPY32 |
|---|---|---|---|---|---|
| MSP430F1611 | No | 8 | 48 | 10 | No |
| MSP430F2417 | Yes | 16 | 92 | 8 | No |
| CC430F6137 | Yes | 20 | 32 | 4 | Yes |
| MSP430F5529 | Yes | 25 | 128 | 8 | Yes |

Tmote Sky and TelosB sensors, and the MSP430F2417 used in the TinyNode 184.

The MSP430 family instruction set has addition, subtraction and one-bit only shifts. A swap byte instruction is available, which can be used for cheaper 8-bit shifts. Integer multiplication is carried out with a hardware multiplier — a memory-mapped peripheral that is present in all mentioned models. The cost of using this hardware is simply the cost of writing the operands and reading the result to/from a certain memory address. There is no division instruction. Operands can be referenced by four addressing modes: register direct, indexed, register indirect and indirect with auto-increment. Instructions can use immediate constants that are codified in offset words adjacent to the instruction. The number of cycles that an instruction takes to execute can be computed easily, with a few exceptions. It takes one cycle to fetch the instruction and one cycle to read each offset word, if any. Add one cycle for each in-memory source (read) and two cycles for a in-memory destination (write).

The MSP430 architecture was later expanded into the backward-compatible MSP430X architecture. These microcontrollers (also referred to as MSPX) are able to address up to 1 MB of memory with 20-bit pointers. New instructions are available, such as pushing and popping multiple registers with only one instruction and up to 4-bit shifts with one instruction (which still take the same number of cycles than using separate instructions). Its instructions timings are also different; the most important distinction is that moving data to memory takes one less cycle to execute. Some new MSP430X models feature a new 32-bit hardware multiplier (referred to as MPY32) whose usage is similar to the old 16-bit multiplier and which can be used to greatly improve the performance of cryptographic operations as will be described later.

## 2.3 Cryptographic Protocols

An elliptic curve $E$ over a field $K$ is defined by the equation $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$, with $a_i \in K$. A point in an elliptic curve is a pair $(x, y) \in K \times K$ that satisfies

Table 2.2: Elliptic curves used in this work

| Curve | Security (bits) | Field | Random | Used in |
|---|---|---|---|---|
| secp160r1 [34] | 80 | Prime | Yes | ECC |
| secp160k1 [34] | 80 | Prime | No | ECC |
| K-163 [121] | 80 | Binary | No | ECC |
| BN-158 [125] | 80 | Prime | No | PBC |
| SS-353 [125] | 80 | Binary | No | PBC |
| P-256 [121] | 128 | Prime | Yes | ECC |
| secp256k1 [34] | 128 | Prime | No | ECC |
| K-283 [121] | 128 | Binary | No | ECC |
| BN-254 [123] | 128 | Prime | No | PBC |

the curve equation. It is possible to define a point addition operation such that the set of points in the curve — together with an identity element called the point at infinity — forms a group. This group is denoted $E(K)$. Given a point $P = (x_P, y_P)$ in $E(K)$ and an integer $k$, the point multiplication operation $kP$ can be defined as the sum of $k$ copies of the point $P$, using the point addition operation of the elliptic curve. Table 2.2 lists the curves used in this work. A random curve is a curve which was generated verifiably at random.

Given three groups $\mathbb{G}_1$, $\mathbb{G}_2$ e $\mathbb{G}_T$ of prime order $r$, with $\mathbb{G}_1$ e $\mathbb{G}_2$ additive and $\mathbb{G}_T$ multiplicative, a bilinear pairing is defined as a map $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the bilinearity property, that is, for all $a, b \in \mathbb{Z}$, $R \in \mathbb{G}_1$ and $S \in \mathbb{G}_2$ we have that $e(aR, bS) = e(R, S)^{ab}$. In PBC, usually, $\mathbb{G}_1$ and $\mathbb{G}_2$ are subgroups of the group of points in certain elliptic curves over finite fields and $\mathbb{G}_T$ is the multiplicative subgroup of a finite field. If $\mathbb{G}_1 = \mathbb{G}_2$, the pairing is called symmetric, otherwise, it is called asymmetric. In this work, the pairing over the binary field is symmetric and the pairing over the prime field is asymmetric.

### 2.3.1 ECDSA

We have implemented the Elliptic Curve Digital Signature Algorithm (ECDSA) [121] since it is extremely popular and standardized. It is composed of three operations: key generation, signature and verification. The key generation requires a point multiplication of a fixed point $G$ that is known by all participants. The signature also requires a fixed point multiplication while the signature requires the computation of $kG + \ell Q$ where $G$ is fixed and $Q$ is the point corresponding to the public key of the signer. For more details we refer to the standard [121].

For implementing ECDSA we have chosen the curves secp160r1 and secp160k1 (over

160-bit prime fields) from the SECG standard [34]; P-256 and secp256k1 (over 256-bit prime fields) from the NIST and SECG standards [34, 121]; K-163 (over the binary field $\mathbb{F}_{2^{163}}$) and K-283 (over $\mathbb{F}_{2^{283}}$) from NIST [121]. Those curves allow a fast modular reduction with only additions, shifts and xors (and a few multiplications in some cases) due to the special form of their prime moduli and reduction polynomials.

### 2.3.2 ECSS

We have also implemented the Elliptic Curve Schnorr Signature (ECSS) [141] which is very similar to ECDSA but does not require an inversion modulo the group order, thus being faster. The ECSS has been patented, but its patents seem to have expired recently. The same curves used in ECDSA were employed for ECSS.

### 2.3.3 ZSS Short Signature Scheme

The Zhang-Safavi-Naini-Susilo (ZSS) scheme [158] is a short signature scheme based on pairings that allows signatures with half the size of ECDSA's. Its signature generation requires a fixed point multiplication, while the verification requires one pairing computation and one fixed point multiplication.

We remark that using supersingular binary curves in order to implement pairing-based short signatures defeats the short signature property, as was pointed out in [30]. Since these curves have a small embedding degree, they require larger fields and therefore provide larger signatures. For example, at the 80-bit level of security, a 353-bit field is required which results in a 353-bit signature, in contrast to the 320-bit signature achieved by ECDSA. For this reason, only prime fields were used for this protocol. Two BN curves [20] over prime fields with 158 bits (BN-158) and 254 bits (BN-254) were chosen, using the Optimal Ate pairing [154].

### 2.3.4 ECMQV Authenticated Key Agreement Protocol

The Elliptic Curve Menezes-Qu-Vanstone (ECMQV) [93] is a family of authenticated key agreement protocols; we have implemented the two-pass variant. This protocol is composed of two operations: key generation and key agreement. The key generation is the same as ECDSA's and requires a fixed point multiplication. The key agreement itself has two stages: first, the parties exchange public keys; then, a ephemeral key pair is generated (again with a fixed point multiplication) and the ephemeral public key is sent to the other party, while the ephemeral public key of the other party is received. The agreed key is then computed using two random point multiplications. The curves used for ECMQV were the same curves used for ECDSA.

### 2.3.5 SOK Non-Interactive Authenticated Key Agreement Protocol

Public key authentication is the main problem of ECC, since it requires an infrastructure that can be expensive for sensors. The Identity-Based Cryptography (IBC) provides an alternative to classic asymmetric cryptography where public keys are the identities of the users (such as an email address) and thus they are implicitly authenticated. Its downside is the requirement for a Key Generation Center (KGC) which generates the private keys of the users and therefore can impersonate them. However, this center is acceptable in the wireless sensor network scenario since the sensors trust the KGC as they are all maintained by the same organization. The most popular method to instantiate IBC is based on pairings.

One of the best IBC schemes for wireless sensors networks is the Sakai-Ohgishi-Kasahara (SOK) non-interactive authenticated key agreement protocol [139]. This protocol enables two parties to combine a mutual key without any communication, knowing only each other's identities; afterwards, communication can be carried out with symmetric encryption. For this reason, it is well suited for sensor networks where communication is often costly. In SOK, a key agreement requires a single pairing computation. If the pairing is asymmetric, its implementation is slightly more complex. While [48] suggests computing two pairings to solve the asymmetric case, it can be done with only one pairing computation [60].

Loosely speaking, the security of SOK depends on the difficulty of solving the discrete logarithm problem in the pairing groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$. Since $\mathbb{G}_1$ and $\mathbb{G}_2$ are elliptic curves, the underlying fields must have at least 160 and 256 bits at the 80- and 128-bit levels of security, respectively. The finite field from which $\mathbb{G}_T$ is a subgroup, however, must have a larger size than $\mathbb{G}_1$ and $\mathbb{G}_2$ due to the existence of sub-exponential attacks to its discrete logarithm problem. For prime fields, the standard [120] recommends orders with 1024 and 3072 bits at the 80- and 128-bit levels of security, respectively. For binary fields, due to Coppersmith's attack [40], a more conservative approach is required: at least approximately 1412 and 4036 bits at the 80- and 128-bit levels, respectively.

In order to implement SOK the two aforementioned BN curves were used (BN-158 and BN-254) along with a supersingular curve over the binary field $\mathbb{F}_{2^{353}}$ (SS-353) [125]. Note that the curve over $\mathbb{F}_{2^{271}}$ used in works such as [124, 149] offers only 70 bits of security due to [40], while the curve over $\mathbb{F}_{2^{353}}$ offers 80 bits of security. A binary curve offering 128 bits of security was not implemented since it would required a huge base field (more than 1000 bits) and thus would result in very poor performance.

## 2.4 Efficient Software Implementation

Our implementation was written in the C language, with critical finite field operations written in assembly. The IAR Embedded Workbench 5.40 was used for development and execution, using both its simulator and boards equipped with a MSP430F2417 and a CC430F6137.

### 2.4.1 Prime Field Arithmetic

The prime field arithmetic is composed of addition, subtraction, multiplication, squaring and inversion modulo a prime $p$. The most important are multiplication and squaring, since ECC and PBC protocols spend around 70% of their computation in those operations. The multiplication in a prime field has two steps: the integer multiplication of the two $n$-word operands into a $2n$-word intermediate result, and the reduction modulo $p$ in order to obtain the $n$-word result.

The integer multiplication is carried out with the Comba algorithm [39], a column-oriented version of the usual multiplication algorithm. There is also a hybrid algorithm that combines both techniques [143]. However, Comba offers a better performance in this platform since its fundamental multiply-and-accumulate step is exactly what is provided by the Multiply and Accumulate (MAC) operation of the MSP430 hardware multiplier [60]. The Karatsuba [83] algorithm can also be used in order to reduce the size of the multiplier required. We have determined that using Karatsuba leads to a better performance in the 256-bit field, and thus was used in this case.

The modular reduction of the intermediate result is computed with the Montgomery algorithm [117], since it does not requires division, which is not available in the MSP430. It has the same structure as the Comba multiplication, but with the first operand being the prime modulus and the second operand generated on the fly during the algorithm. Therefore, we can use the same MAC optimization [60].

When using the 32-bit multiplier, we employ the same Comba method, now with 32-bit digits and still taking advantage of the MAC operation. In order to fully use the power of the multiplier, it is important to enable the MPYDLYWRTEN bit which allows writing the operands of the next MAC operation while the previous is still being carried out.

In the PBC context, a very effective optimization for modular reduction was found, which was briefly mentioned in [125] and is further detailed here. When using BN curves, the prime modulus is given by the polynomial $p(x) = 36x^4 + 36x^3 + 24x^2 + 6x + 1$ where $x$ is the parameter that defines the curve. Consider the curves generated by the values $x_1 = 2^{38} + 2^5 + 2^4 + 1$ (suitable at the 80-bit level of security) and $x_2 = -2^{62} - 2^{55} - 1$ (suggested by [123], suitable at the 128-bit level of security). The prime moduli in those

cases are

$$p(x_1) = \texttt{0x2400 0000 6ED0 0000 7FE9}$$
$$\texttt{C000 419F EC80 0CA0 35C7},$$
$$p(x_2) = \texttt{0x2523 6482 4000 0001 BA34 4D80 0000 0008}$$
$$\texttt{6121 0000 0000 0013 A700 0000 0000 0013},$$

respectively, in base 16. Notice that $p(x_1)$ has two 16-bit digits with a zero value, while $p(x_2)$ has five digits with zero value and one digit is the number one. Since Montgomery reduction is analogous to a multiplication, the steps with multiplications by zero can be discarded and the steps with multiplication by one can be simplified. For example, when using $p(x_2)$, it was required $16^2 = 256$ multiply-and-accumulate steps during the Montgomery reduction. Ignoring the steps with multiplications by zero, $256 - 16 \cdot 5 = 176$ steps are now required, a 31% reduction. We remark that the generic optimization technique that takes advantage of sparse primes has been used before [65, 85], but to the best of our knowledge this work and [125] provide the first application of the technique in the context of PBC. The focus of the curve suggested by [123] was probably in the fact that a sparse parameter leads to a faster Miller loop, but it also speeds up the field arithmetic as shown.

When using the 32-bit multiplier, it is more difficult to apply this technique. In $p(x_2)$, the three zeros in odd-indexed 16-bit digits allow the replacement of some 32x32-bit multiplications by 32x16-bit multiplications, which is a little faster. However, we were unable to take advantage of the zeros in even-indexed digits.

### 2.4.2 Binary Field Arithmetic

The binary field arithmetic is composed by the addition, subtraction, multiplication, squaring, inversion and square root operations over the field $\mathbb{F}_{2^m}$. Addition in this field becomes a simple xor of the operands; multiplication becomes more expensive due to the absence of a hardware multiplier and therefore must be implemented very efficiently; and squaring can be computed much faster with precomputed tables.

The binary field multiplication is also composed of two steps: polynomial multiplication of the operands followed by a polynomial reduction. The polynomial multiplication is computed with the López-Dahab (LD) algorithm [104], which appears to be the most efficient in this scenario [9, 149]. Optionally, the LD algorithm can be combined with the Karatsuba algorithm; we have employed it in the 283- and 353-bit fields. The square of a binary field element can be computed very quickly with the help of precomputed tables stored in ROM, as described in e.g. [9].

The polynomial reduction after polynomial multiplication and squaring can be computed efficiently due to the special form of the reduction polynomial in the chosen curves. It can be computed with only shifts and xors, as described in e.g. [70]. For the curve over $\mathbb{F}_{2^{353}}$ used in the $\eta_T$ pairing, we have used the trinomial $z^{353} + z^{95} + 1$ for the field $\mathbb{F}_{2^{353}}$ since reduction, in this case, can be carried out with 1- and 2-bit shifts only [125].

The square root of a binary field element $a(z)$ is the element $c(z)$ such that $c(z)^2 = a(z)$. In this work, it is only used on the curve $\mathbb{F}_{2^{353}}$, during the computation of the $\eta_T$ pairing. As described in [53], it can be computed using lookup tables and a multiplication by $\sqrt{z}$. This multiplication can be computed efficiently in the given curve, using 1-bit shifts only, by noting that $\sqrt{z} \equiv z^{177} + z^{48}$ in $\mathbb{F}_{2^{353}}$.

Inversion in the binary field can be computed using the extended Euclidean algorithm. In order to optimize it we have used the same approach as [9], using dedicated functions for shifting one to eight bits which are timing consuming during inversion.

### 2.4.3 Point Arithmetic

The main operations in ECC are the the random point multiplication ($kP$), fixed point multiplication ($kG$, where $G$ is known in advance) and the simultaneous multiplication of a fixed and a random point ($kG + \ell P$), all of which are based on point addition and duplication. Using the names from the Explicit-Formulas Database[1], the prime curves use Jacobian coordinates and the `madd-2007-bl` mixed addition [25] formulas; the `dbl-2001-b` doubling [22] formulas were used for the secp160r1 and P-256 curves while the `dbl-2009-l` doubling [11] formulas were used for the secp160k1, secp256k1, BN-158 and BN-254 curves (except for the pairing computation in the latter two curves). For the K-163 and K-283 curves, the `madd-2005-dl` mixed addition formula [2] was employed; point doubling is not needed for Koblitz curves.

For fixed point multiplication, the Comb [98] method was selected in the prime case and the $w$TNAF method in the binary case [147]. These methods take advantage of precomputation tables containing multiples of the fixed point $G$ in order to speed up the computation. For simultaneous multiplication, the $w$NAF- and $w$TNAF-based interleaving method [114] were chosen for the prime and binary cases, respectively. This method uses a precomputation table for the fixed point and a smaller table, computed on the fly, for the random point. For all methods mentioned, we have used precomputed tables with 8 points and on-the-fly tables with 4 points.

On the secp160k1 and secp256k1 curves, which are used for ECDSA and ECSS, and on the BN curves used for ZSS, the Gallant-Lambert-Vanstone (GLV) method [56] enables a faster point multiplication. These curves all have the form $y^2 = x^3 + b$ over $\mathbb{F}_p$ with $p \equiv 1$

[1] `http://www.hyperelliptic.org/EFD/`

(mod 6) and have a map $\phi\colon E(\mathbb{F}_p) \mapsto E(\mathbb{F}_p)$ defined by $(x, y) \mapsto (\beta x, y)$ where $\beta \in \mathbb{F}_p$ is an element of order 3. Let $\lambda$ be an integer satisfying $\lambda^2 + \lambda \equiv -1 \pmod{n}$. Then $\phi(P) = \lambda P$. It is possible to take advantage of this fact in order to improve the speed of point multiplication $kP$, as follows. Write $k$ as $k = k_0 + \lambda k_1$, then compute $k_0 P + k_1 \phi(P)$. If $k_0$ and $k_1$ have half the bit length of $k$, then it is possible to use the interleaved point multiplication [114] which is faster than computing $kP$ directly, since the number of point duplications will be cut in half.

The critical operation in the signing algorithm of ZSS, ECDSA, and ECSS is the fixed point multiplication, where the Comb method is often faster than $w$NAF. However, mixing Comb and GLV is not straightforward. It can be done, as mentioned in [125], by considering the two-table version of the Comb method. In this version, two tables are precomputed with multiples of $P$ and $2^{t/2}P$, where $t$ is the bit length of $k$. Write $k = k_0 + 2^{t/2}k_1$; now, $kP$ can be computed taking advantage of the two tables. When the GLV method is available, we can decompose $k$ as usual ($k = k_0 + \lambda k_1$). However, the second table, now with multiples of $\phi(P)$, does not need to be precomputed since we can just apply $\phi$ to the elements of the first table when necessary. Therefore, we can use half of the space of the usual Comb multiplication while achieving the same performance, or we can use the same space (doubling the size of the first table) while cutting the number of point duplications in half — which we have chosen.

For GLV to be effective, the decomposition of $k$ must be computed efficiently. This can be done using some precomputation, as hinted in [70] and mentioned in [125]. We describe the approach in [125] in a more detailed manner as follows. In order to decompose $k$ as $k_0$ and $k_1$ with the appropriate bit lengths, we precompute the vectors $\vec{v_1}, \vec{v_2} \in \mathbb{Z} \times \mathbb{Z}$ as described in [56]. Now, we must solve the equation $(k, 0) = \beta_1 \vec{v_1} + \beta_2 \vec{v_2}$ for $\beta_1$ and $\beta_2$, then round the result to the nearest integers as $b_1 = \lfloor \beta_1 \rceil$ and $b_2 = \lfloor \beta_2 \rceil$. Compute $\vec{v} = b_1 \vec{v_1} + b_2 \vec{v_2}$ and finally let $(k_0, k_1) = (k, 0) - \vec{v}$.

The critical operation when decomposing $k$ is to solve the given equation. Write $\vec{v_1} = (v_{10}, v_{11})$ and $\vec{v_2} = (v_{20}, v_{21})$. Let $d = v_{10}v_{21} - v_{11}v_{20}$. It is then possible to compute $\beta_1, \beta_2$ as $\beta_1 = (kv_{21})/d$, $\beta_2 = -(kv_{11})/d$. The multiple precision division by $d$ is the most expensive part, but it can be avoided as follows. Let $t = \lfloor k \rfloor + 2$ and compute $g_1 = \lfloor (2^t v_{21})/d \rceil$ and $g_2 = \lfloor (2^t v_{11})/d \rceil$. Note that $g_1$ and $g_2$ are integer constants which can be precomputed. We can now compute both $\beta_1' = \lfloor (kg_1)/2^t \rfloor$, $\beta_2' = -\lfloor (kg_2)/2^t \rfloor$, where the floored division by $2^t$ is a simple right shift of $t$ bits. The last bit discarded in this right shift must be stored, and if it is 1, then $b_1 = \beta_1' + 1$, otherwise $b_1 = \beta_1$. The same applies to $\beta_2'$. The decomposition of $k$ now follows as described.

### 2.4.4 Pairing Computation Algorithms

The standard algorithm for computing pairings is the Miller algorithm [112]. The first pairings used were the Weil and Tate, but the fastest ones are called optimal pairings [154]. Among them, the Optimal Ate pairing was chosen [154] for the prime case and the $\eta_T$ pairing [17] for the binary case. A pairing computation is divided in two parts: the Miller loop, which is analogous to a point multiplication with some additional computation; and the final exponentiation, which maps the result of the Miller loop — a coset element — to a canonical representation of the coset.

In this work, BN curves [20] were used in the prime case. They are a family of pairing-friendly elliptic curves of the form $y^3 = x^3 + b$, with embedding degree 12. When using a BN curve with the Optimal Ate pairing, the pairing groups become $\mathbb{G}_1 = E(\mathbb{F}_p)$, $\mathbb{G}_2 = \{Q \in E(\mathbb{F}_{p^{12}})\colon pQ = (x_Q^p, y_Q^p), rQ = \infty\}$ and $\mathbb{G}_T = \{x \in \mathbb{F}_{p^{12}}^*\colon x^r = 1\}$. The $\mathbb{G}_2$ group can be represented by the preimage of the twisting isomorphism $\phi\colon E' \to E$, where $E'$ is the sextic twist of $E$. Since this group is defined over $\mathbb{F}_{p^2}$, it has smaller elements and its operations are more efficient. The Miller loop therefore requires point arithmetic in $E'(\mathbb{F}_{p^2})$. Points are represented with projective coordinates using the formulas from [8]. The loop also requires arithmetic in $\mathbb{F}_{p^2}$ and $\mathbb{F}_{p^{12}}$. The latter is built as an extension tower $\mathbb{F}_{p^2} \to \mathbb{F}_{p^6} \to \mathbb{F}_{p^{12}}$. In $\mathbb{F}_{p^2}$ we employ the lazy reduction technique [155]. This allows computing $ab + cd$, for example, with two multiplications and a single reduction, instead of the usual two multiplications and two reductions — this is accomplished by multiplying $ab$ and $cd$, adding the double precision results, and reducing the result. It is possible to employ this technique for the entire $\mathbb{F}_{p^{12}}$ as described in [8], but it can be costly in terms of memory, as will be examined later.

Still in the prime case using BN curves, the final exponentiation requires $\mathbb{F}_{p^{12}}$ arithmetic. We follow the approach from [142], with the field squaring optimized as described in [64]. It is also possible to employ the compressed squaring technique from [82] as detailed and improved in [8] but again it can be costly in terms of memory. Hashing to $\mathbb{G}_2$ in the asymmetric case is another important operation since it requires a expensive point multiplication by the cofactor of $E'(\mathbb{F}_{p^2})$, which has the same size of $p$. However, this multiplication can be greatly sped up using the technique from [54].

For the binary case, a supersingular curve over a 353-bit field was used. Here, the pairing groups become $\mathbb{G}_1 = \mathbb{G}_2 = E(\mathbb{F}_{2^{353}})$ and $G_T = F_{2^{4\times 353}}$. The $\eta_T$ pairing computation follows its original description [17].

Table 2.3: Timings in cycles for prime field arithmetic

| | 80-bit | | | 128-bit | | |
|---|---|---|---|---|---|---|
| Algorithm | MSP | MSPX | MPY32 | MSP | MSPX | MPY32 |
| Comba mult. | 1,565 | 1,299 | 741 | 3,563 | 2,981 | 1,620 |
| Comba squaring | 1,350 | 1,056 | 630 | 2,946 | 2,435 | 1,369 |
| *Reduction* | | | | | | |
| Montgomery, in [60] | 1,785 | | | 3,989 | | |
| Montgomery | 1,659 | | | 3,600 | | |
| Montgomery, sparse | 1,413 | 1,174 | 853 | 2,670 | 2,232 | 1,695 |

## 2.5 Results

Using the MSPsim and the IAR simulators, the number of cycles taken by each operation was measured. In some cases we give the timing in seconds; these are derived from the number of cycles assuming a 8 million hertz clock. The exact clock depends on the MSP430 model being used and in the voltage fed to the microcontroller.

The MSP430F6137 microcontroller used for testing the 32-bit multiplier features only 4 KB of RAM and 32 KB of ROM, making it difficult to fit our code in the available space. In this case, we have used the IAR simulator to run the programs for the MSP430F5529, which has 128 KB of ROM and the same 32-bit multiplier.

### 2.5.1 Prime Field Timings

Table 2.3 gives the timings of the prime field operations. First, we note that there is a small gain in comparison to [60], which presents the fastest timings so far for the MSP430 to the best of our knowledge. Particularly, in the modular reduction, we have obtained an 8% improvement by noting that is not necessary to read the prime modulus from the memory — since it is fixed, we can embed its digits in the instructions as immediate constants. However, the most expressive savings come from the sparse modulus optimization — 14% in the 160-bit field and 26% in the 256-bit field.

The MSP430X improves the timings by roughly 15%, while the MPY32 results in multiplications and squarings that are around 40% faster and reductions that are around 30% faster. The limited improvement for reduction is due to not being able to take full advantage of the sparse primes.

Table 2.4: Timings in cycles for binary field arithmetic

| | 80-bit, ECC | | 128-bit, ECC | | 80-bit, PBC | |
|---|---|---|---|---|---|---|
| Algorithm | MSP | MSPX | MSP | MSPX | MSP | MSPX |
| LD mult. | 3,907 | 3,585 | 8,655 | 8,166 | 13,868 | 12,902 |
| Squaring | 249 | 199 | 389 | 325 | 489 | 415 |
| Square root | | | | | 934 | 852 |
| Fast reduction | 448 | 397 | 888 | 835 | 494 | 426 |

### 2.5.2 Binary Field Timings

Timings for binary field arithmetic are listed in Table 2.4. Previous works have used the 70-bit level for PBC [124, 149], which prevents comparisons. The reduction in $\mathbb{F}_{2^{283}}$ requires many shifts due to standardized reduction polynomial used in the K-283 curve; this explains why it is slower than the reduction in the larger field $\mathbb{F}_{2^{353}}$. The MSP430X improves the timings around 5–15%, less than in the prime case probably due to the smaller number of memory writes in the LD multiplication.

### 2.5.3 Protocol Timings

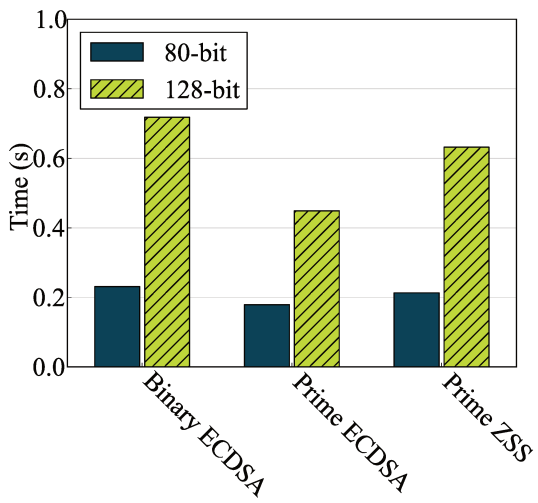
Table 2.5 presents the timings of signature algorithms. The fastest timings are plotted for comparison in Fig. 2.1a and 2.1b. For ECDSA, it can be seen that the binary curves offer a better performance than the random prime curves, but the special prime curves are even faster. Using MSP430X leads to a approximately 15% improvement in the prime case and around 5–10% in the binary case. Using MPY32 gives a further 20–30% improvement. Timings for ECSS were omitted for brevity but they are roughly 10% faster than ECDSA for signing and 5% faster for verifying.

Even though ZSS provides signatures with half the bit length of ECDSA, it appears that it does not give any substantial advantage in this scenario — the energy cost of the increased signing time defeats any energy saving with the decreased communication cost due to the smaller signature; as already shown in [125]. Nevertheless, it can be useful in scenarios where communication is very expensive, such as underwater sensor networks [55]. Also, note that verification in ZSS is costly since it requires one pairing computation; but this may not be a issue if only signing is required by the sensors and verification is done by more powerful devices.

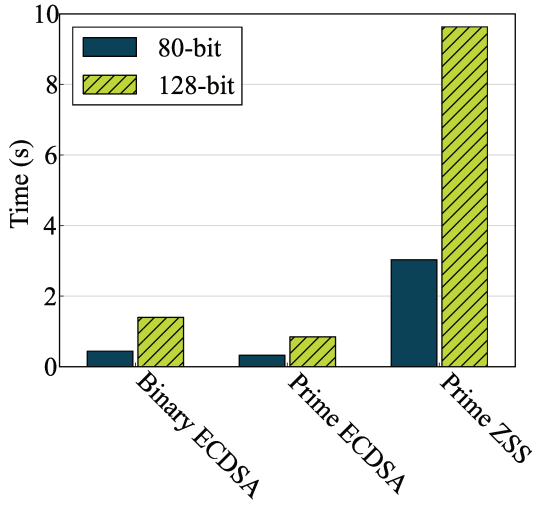
Timings for pairing computation and key agreement protocols are given in Table 2.6 and the fastest version of each plotted for comparison in Fig. 2.2. We observe again a

Table 2.5: Timings in seconds for signature protocols

| | | Sign | | | Verify | | |
|---|---|---|---|---|---|---|---|
| Protocol | Curve | MSP | MSPX | MPY32 | MSP | MSPX | MPY32 |
| *80-bit* | | | | | | | |
| ECDSA | secp160r1 | 0.315 | 0.266 | 0.218 | 0.625 | 0.549 | 0.449 |
| | secp160k1 | 0.254 | 0.214 | 0.179 | 0.450 | 0.387 | 0.327 |
| | K-163 | 0.256 | 0.231 | | 0.481 | 0.443 | |
| ZSS | BN | 0.356 | 0.284 | 0.213 | 5.908 | 4.077 | 3.032 |
| *128-bit* | | | | | | | |
| ECDSA | P-256 | 0.924 | 0.807 | 0.557 | 1.882 | 1.671 | 1.156 |
| | secp256k1 | 0.719 | 0.618 | 0.449 | 1.275 | 1.099 | 0.847 |
| | K-283 | 0.772 | 0.718 | | 1.474 | 1.397 | |
| ZSS | BN | 1.056 | 0.890 | 0.632 | 16.611 | 14.027 | 9.631 |




(a) Signature timings for MSP430X with MPY32 at 8 MHz




(b) Verification timings for MSP430X with MPY32 at 8 MHz

Figure 2.1: ECDSA and ZSS timings

Table 2.6: Timings in seconds for pairing computation and key agreement protocols

| Algorithm | Curve | MSP | MSPX | MPY32 |
|---|---|---|---|---|
| *80-bit* | | | | |
| $\eta_T$ | | 2.586 | 2.395 | |
| Optimal Ate | | 3.791 | 3.202 | 2.470 |
| SOK | SS-353 | 2.668 | 2.493 | |
| SOK ($\mathbb{G}_1$) | BN-158 | 3.875 | 3.285 | 2.533 |
| SOK ($\mathbb{G}_2$) | BN-158 | 4.554 | 3.861 | 2.982 |
| ECMQV | K-163 | 0.719 | 0.658 | |
| | secp160r1 | 0.982 | 0.838 | 0.671 |
| | secp160k1 | 0.730 | 0.626 | 0.513 |
| *128-bit* | | | | |
| Optimal Ate | | 9.930 | 8.461 | 5.967 |
| SOK ($\mathbb{G}_1$) | BN-254 | 10.202 | 8.693 | 6.134 |
| SOK ($\mathbb{G}_2$) | BN-254 | 11.766 | 10.036 | 7.121 |
| ECMQV | K-283 | 2.291 | 2.150 | |
| | secp256r1 | 2.842 | 2.437 | 1.742 |
| | secp256k1 | 2.065 | 1.776 | 1.298 |

near 15% improvement with MSP430X and a further 25–30% gain with MPY32. Using MPY32, the prime curve almost reaches the performance of the binary curve at the 80-bit level of security. The reason for the binary curve being so competitive — despite the fact that there is no hardware accelerator involved in its computation — is that, in both Koblitz curves and in the $\eta_T$ pairing, point doublings are replaced by the Frobenius endomorphism which is simply the extremely fast squaring of the point coordinates. The ECMQV protocol is around 75–80% faster than SOK, but of course it requires a great deal of communication (to receive the certificate with the other party's public key, to send our certificate to the other party, to receive the other party's ephemeral public key, and to send our ephemeral public key to the other party) which makes it much less attractive in this scenario. It is interesting to note that the SOK with hash in $\mathbb{G}_2$ is more expensive than in $\mathbb{G}_1$ in the asymmetric case due to the requirement of multiplying the hashed point by the curve cofactor.

All timings reported so far have been scaled to a 8 MHz clock in order to allow comparisons with previous works. However, there are MSP430X microcontrollers with up to 25 MHz maximum clock. For this reason, to give a better picture of what is achievable

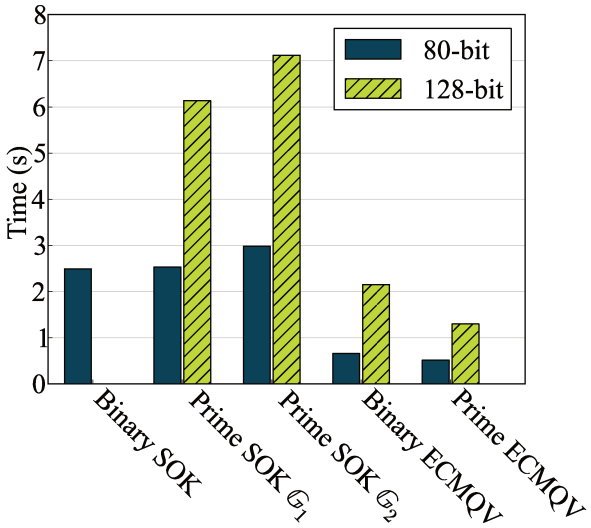



Figure 2.2: SOK and ECMQV key agreement timings for MSP430X with MPY32 at 8 MHz

with current technology, we report the timings using the 25 MHz clock in Table 2.7 for the best choices of the implemented protocols: ECDSA, ZSS, and ECMQV over GLV-enabled prime curves; SOK over binary curve at the 80-bit level and over BN curve at the 128-bit level. We remark that it is possible to sign in 0.2 s at the 128-bit level of security, and to compute a SOK key agreement in less than one second at the 80-bit level and under 2.5 s at the 128-bit level.

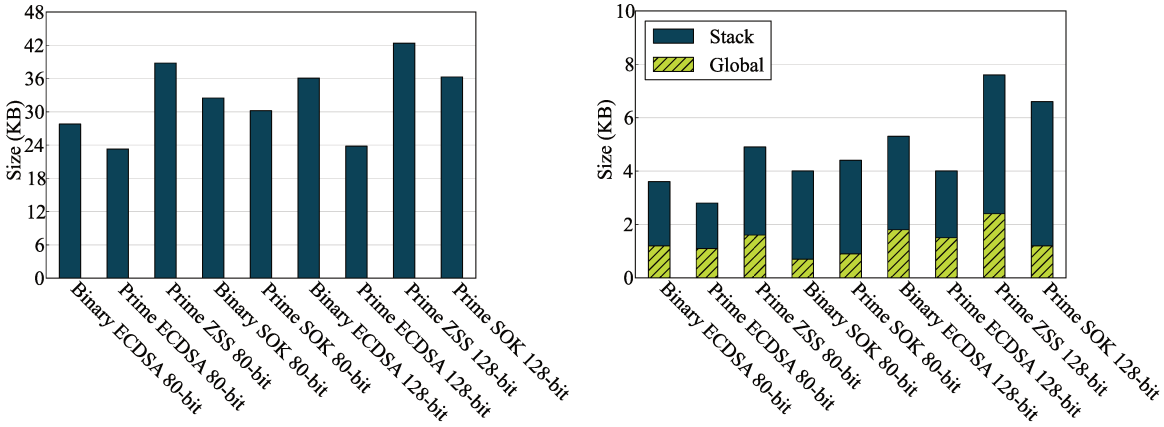
### 2.5.4 ROM, RAM and Energy

Figure 2.3a shows the ROM usage of the programs, while Fig. 2.3b shows the global RAM and the maximal stack RAM usage (ECSS and ECMQV are roughly the same as ECDSA). The ROM usage falls into the 23–42 KB range, with ECC protocols taking less space. The RAM usage falls into the 2–8 KB range. The ROM usage, while high, may be acceptable in devices such as the MSP430F5529 that have a 128 KB flash. The RAM usage can also be considered acceptable by noting that most part of it is allocated from the stack and freed after computation. More RAM may be freed by moving the precomputed tables to ROM as a trade off.

Programs using prime fields are smaller due to the hardware multiplier which allows computation with a smaller number of instructions. The MSP430X and MPY32 also reduce the ROM usage by up to 800 bytes and 1.6 KB, respectively. Energy consumption seems to be not critical in this scenario by considering that with a 500 mAh battery it is possible to compute the most expensive operation (SOK key agreement in $\mathbb{G}_2$) roughly

Table 2.7: Timings in seconds for best schemes at 25 MHz

| | 80-bit | | 128-bit | |
|---|---|---|---|---|
| Algorithm | MSPX | MPY32 | MSPX | MPY32 |
| ECDSA sign | 0.068 | 0.057 | 0.198 | 0.144 |
| ECDSA ver. | 0.124 | 0.105 | 0.352 | 0.271 |
| ZSS sign | 0.091 | 0.068 | 0.285 | 0.202 |
| ZSS ver. | 1.305 | 0.970 | 4.489 | 3.082 |
| ECMQV | 0.200 | 0.164 | 0.568 | 0.415 |
| SOK ($\mathbb{G}_1$) | 0.798 | 0.798 | 2.782 | 1.963 |
| SOK ($\mathbb{G}_2$) | 0.798 | 0.798 | 3.212 | 2.279 |




(a) ROM usage

(b) RAM usage

Figure 2.3: RAM and ROM usage for ECC and PBC protocols

about 40,000 times, assuming a 3.31 mA current under 3.6 V, which are required to obtain the desired clock [46].

### 2.5.5 Impact of recent optimizations

Using the optimizations from [8] it is possible to speed up the prime pairing computation even more, but with a somewhat large memory cost. At the 80-bit level of security, using the lazy reduction for the entire extension tower leads to a 7–8% speed up in the pairing computation, but increases RAM usage by 1 KB (a 33% increase) and ROM usage by 0.9 KB. Using the compressed squaring optimization for the final exponentiation leads to a further 4% speed up, but increases RAM usage by another 1 KB and ROM usage by 1.4 KB. At the 128-bit level of security, the lazy reduction optimization leads to a 5% speed up (which is smaller than the 80-bit level since the reduction here is comparatively faster due to the sparser prime), increasing RAM usage by 1.5 KB (a 33% increase) and ROM usage by 1.2 KB; while the compressed squaring leads to a further 7% speed up, increasing RAM usage by 1.2 KB and ROM usage by 1.4 KB.

## 2.6 Related Work

One of the first ECC implementations for the MSP430 is the work of Guajardo et al. [66], which achieves a random point multiplication in 0.425 s (when scaled to 8 MHz) on the curve secp128r1 (64 bits of security), using the binary point multiplication algorithm and the 16-bit hardware multiplier. For comparison, our timing for the same algorithm is 0.581 s on the curve secp160r1, which offers 80 bits of security.

The work NanoECC [150] presents a point multiplication in 0.72 s and 1.04 s on prime (secp160r1) and binary (K-163) curves, respectively, at the 80-bit level of security. Unfortunately it is not clear which point multiplication algorithm was used in these timings. For comparison, our timings are 0.525 s and 0.256 s, respectively, using 4NAF/4TNAF. They also provide timings for the 7.4 MHz ATmega: 1.27 s and 2.16 s respectively.

In [60], it is described how to use the multiply and accumulate (MAC) operation of the MSP430 hardware multiplier to improve the speed of both ECC and PBC when using prime curves. At the 80-bit level of security, our pairing over the BN curve offers a 25% gain in comparison to their pairing over a MNT curve (which offers roughly 70 bits of security). This result may seem surprising since the BN curve uses a $\mathbb{G}_T$ with 1920-bit order, much more than the 1024 bits required at this level of security. However, it is explained by the many existent optimizations tailored for BN curves. At the 128-bit level of security, we have obtained a 30% improvement in comparison their work in the pairing computation over the BN curve. This saving is explained by the sparse prime reduction

optimization and the new formula for the final exponentiation from [64].

TinyPBC [124] provides timings for the $\eta_T$ pairing over a supersingular 271-bit curve, which offers roughly 70 bits of security. The pairing timing is 1.27 s for the 8 MHz MSP430, 1.90 s for the 7.4 MHz ATmega and 0.14 s for the 13 MHz ARM PXA27x. For comparison, our 80-bit security pairing timing is 2.586 s. It is possible to notice that the binary pairing scales very poorly with the level of security.

The work Secure-TWS [125] focuses on digital signatures at the 80-bit level of security and presents a small subset of the results of this paper. The 7.4 MHz ATmega timings are 0.710 s for ZSS signature over prime BN curve, 0.680 s for ECDSA signature over the secp160k1 curve and 0.370 s over the K-163 curve.

## 2.7 Conclusion

Even though it is challenging, the implementation of cryptography for wireless sensor networks is viable. In this work, the best known timings for ECC and PBC in the MSP430 family of microcontrollers were presented, including results for the MSP430X extension of the family and using the new 32-bit hardware multiplier featured in some MSP430X models. Specifically, we have obtained a prime field multiplication that is 12% and 18% faster for 160- and 256-bit prime fields, respectively, by taking advantage of the sparse prime reduction. Our efficient implementation leads to a 25–30% speedup in the pairing computation compared to the best known timings published. We note that the sparse prime reduction can also be applied in other 8- and 16-bit platforms, and also at larger levels of security. Additionally, we were able to improve the timings of ECDSA and ZSS signatures using the GLV method, describing a method for decomposing the multiplier that avoids divisions and a method for using GLV in conjunction with the Comb point multiplication.

It was shown that the MSP430X extension provides a performance gain of roughly 15% due to faster instructions (mainly when writing to memory) and we were able to take advantage of the 32-bit multiplier present in some of the MSP430X microcontrollers in order to obtain a 20–30% improvement in protocol timings. Finally, we remark that on a 25 MHz MSP430F5529 with full clock speed it is possible to compute a 80-bit security SOK key agreement in 0.8 s, a 128-bit SOK key agreement in less than 2.5 s, and an ECDSA signature in under 150 ms.

### Acknowledgements

We would like to thank the São Paulo Research Foundation (FAPESP), which supported author Conrado Gouvêa under grant 2010/15340-3. We also thank Diego Aranha for the

help with the implementation.

# Chapter 3

# High Speed Implementation of Authenticated Encryption for the MSP430X Microcontroller

Conrado P. L. Gouvêa and Julio López

**Abstract**

Authenticated encryption is a symmetric cryptography scheme that provides both confidentiality and authentication. In this work we describe an optimized implementation of authenticated encryption for the MSP430X family of microcontrollers. The CCM, GCM, SGCM, OCB3, Hummingbird-2 and MASHA authenticated encryption schemes were implemented at the 128-bit level of security and their performance was compared. The AES accelerator included in some models of the MSP430X family is also studied and we explore its characteristics to improve the performance of the implemented modes, achieving up to 10 times of speedup. The CCM and OCB3 schemes were the fastest when using the AES accelerator while MASHA and Hummingbird-2 were the fastest when using only software.

## 3.1 Introduction

Constrained platforms such as sensor nodes, smart cards and radio-frequency identification (RFID) devices have a great number of applications, many of which with security

Conrado P. L. Gouvêa and Julio López. High speed implementation of authenticated encryption for the MSP430X microcontroller. In *LATINCRYPT 2012*, volume 7533 of *Lecture Notes in Computer Science*, pages 288–304. Springer Berlin / Heidelberg, 2012, with kind permission from Springer Science and Business Media.

requirements that require cryptographic schemes. The implementation of such schemes in these devices is very challenging since it must provide high speed while consuming a small amount of resources (energy, code size and RAM). In this scenario, symmetric cryptography becomes an essential tool in the development of security solutions, since it can provide both confidentiality and authenticity after being bootstrapped by some protocol for key agreement or distribution. Encryption and authentication can be done through generic composition of separate methods; however, the study of an alternative approach named authenticated encryption (AE) has gained popularity.

Authenticated encryption provides both confidentiality and authenticity within a single scheme. It is often more efficient than using separate methods and usually consumes a smaller amount of resources. It also prevents common critical mistakes when combining encryption and authentication such as not using separate keys for each task. There are many AE schemes; see e.g. [90] for a non-exhaustive list. Some AE schemes are built using a block cipher, in this case, they are also called AE modes. In this work, we follow the approach from [90] and compare the Counter with CBC-MAC (CCM) mode [157], the Galois/Counter Mode (GCM) [109] and the Offset Codebook (OCB3) mode [90]. We have also implemented the Sophie Germain Counter Mode [136], the Hummingbird-2 cipher [49] and the MASHA cipher [84]. The CCM mode and GCM have been standardized by the National Institute of Standards and Technology (NIST); CCM is used for Wi-Fi WPA2 security (IEEE 802.11i) while GCM is used in TLS, IPSec and NSA Suite B, for example. The recently proposed OCB3 mode is the third iteration of the OCB mode and appears to be very efficient in multiple platforms. The SGCM is a variant of GCM and was proposed to be resistant against some existing attacks against GCM while being equally or more efficient; we have implemented it in order to check this claim and compare it to GCM. The Hummingbird-2 cipher (which may be referred to as HB2 in this work) is specially suited for 16-bit platforms and was implemented in order to compare it to the other non-specially suited modes. The MASHA cipher is based on a stream cipher and claims to fill the gap for authenticated encryption algorithms based on stream ciphers which achieve a good balance between security and performance.

The goal of this work is to provide an efficient implementation and comparison of the aforementioned AE schemes (CCM, GCM, SGCM, OCB3, Hummingbird-2, and MASHA) for the MSP430X microcontroller family from Texas Instruments. This family is an extension of the MSP430 which have been used in multiple scenarios such as wireless sensor networks; furthermore, some microcontrollers of this family feature an AES accelerator module which can encrypt and decrypt using 128-bit keys. Our main contributions are: (i) to study (for the first time, to the best of our knowledge) the efficient usage and impact of this AES accelerator module in the implemented AE schemes; (ii) to describe a high speed implementation of those AE schemes for the MSP430X, achieving performance 10

times faster for CCM using the AES accelerator instead of AES in software; (iii) to describe an efficient implementation of AES for 16-bit platforms; (iv) to show that CCM is the fastest of those schemes whenever a non-parallel AES accelerator is available; and (v) and to provide a comparison of the six AE schemes, with and without the AES accelerator. We remark that the results regarding the efficient usage of the AES accelerator can be applied to other devices featuring analogue accelerators, such as the AVR XMEGA.

This paper is organized as follows. In Section 3.2, the MSP430X microcontroller family is described. Section 3.3 offers an introduction to AE. Our implementation is described in Section 3.4, and the obtained results are detailed in Section 3.5. Section 3.6 provides concluding remarks.

## 3.2 The MSP430X Family

The MSP430X family is composed by many microcontrollers which share the same instruction set and 12 general purpose registers. Although it is essentially a 16-bit architecture, its registers have 20 bits, supporting up to 1 MB of addressing space. Each microcontroller has distinct clock frequency, RAM and flash sizes.

Some MSP430X microcontrollers (namely the CC430 series) have an integrated radio frequency transceiver, making them very suitable for wireless sensors. These models also feature an AES accelerator module that supports encryption and decryption with 128-bit keys only. The study of this accelerator is one key aspect of this study and for this reason we describe its basic usage as follows. In order to encrypt a block of 16 bytes, a flag must be set in a control register to specify encryption and the key must be written sequentially (in bytes or words) in a specific memory address. The input block must then be written, also sequentially, in another memory address. After 167 clock cycles, the result is ready and must be read sequentially from a third address. It is possible to poll a control register to check if the result is ready. Further blocks can be encrypted with the same key without writing the key again. The decryption follows the same procedure, but it requires 214 clock cycles of processing. It is worth noting that these memory read and writes are just like regular reads and writes to the RAM, and therefore the cost of communicating with the accelerator is included in our timings.

## 3.3 Authenticated Encryption

An authenticated encryption scheme is composed of two algorithms: authenticated encryption and decryption-verification (of integrity). The authenticated encryption algorithm is denoted by the function $\mathcal{E}_K(N, M, A)$ that returns $(C, T)$, where $K \in \{0,1\}^k$ is

the $k$-bit key, $N \in \{0,1\}^n$ is the $n$-bit nonce, $M \in \{0,1\}^*$ is the message, $A \in \{0,1\}^*$ is the associated data, $C \in \{0,1\}^*$ is the ciphertext and $T \in \{0,1\}^t$ is the authentication tag. The decryption-verification algorithm is denoted by the function $\mathcal{D}_K(N, C, A, T)$ that returns $(M, V)$ where $K, N, C, A, T, M$ are as above and $V$ is a boolean value indicating if the given tag is valid (i.e. if the decrypted message and associated data are authentic).

Many AE schemes are built using a block cipher such as AES. Let $E_K(B)$ denote the block cipher, where the key $K$ is usually the same used in the AE mode and $B \in \{0,1\}^b$ is a $b$-bit message (a *block*). The inverse (decryption) function is denoted $D_K(B')$ where $B'$ is also a block (usually from the ciphertext). The CCM, GCM, SGCM and OCB3 are based on block ciphers, while HB2 and MASHA are not.

It is possible to identify several properties of AE schemes; we offer a non-exhaustive list. The *number of block cipher calls* used in the scheme is an important metric related to performance. A scheme is considered *online* if it is able to encrypt a message with unknown length using constant memory (this is useful, for example, if the end of the data is indicated by a null terminator or a special packet). Some schemes *only use the forward function* of the underlying block cipher ($E_K$), which reduces the size of software and hardware implementations. A scheme *supports preprocessing of static associated data (AD)* if the authentication of the AD depends only on the key and can be cached between different messages being sent (this is useful for a header that does not change). Some schemes are *covered by patents*, which usually discourages its use. A scheme is *parallelizable* if it is possible to process multiple blocks (or partially process them) in a parallel manner. Some schemes support *processing regular messages and AD in any order*, while some schemes require the processing of AD before the message, for example. The properties of the AE schemes implemented in this work are compared in Table 3.1.

**Remarks about security.** The weak key attack against GCM, pointed out by the author of SGCM [136], has probability $n/2^{128}$ of working, where $n$ is the number of blocks in the message; this is negligible unless the message is large. There are related key attacks against Hummingbird-2 [35,159] which, while undesirable, can be hard to apply in practice since keys are (ideally) random. Finally, there is a key-recovery attack in the multi-user setting [36] that can be applied to all schemes in this paper; however, they can be avoided by using random nonces.

## 3.4 Efficient Implementation

We have written a fast software implementation of the AE schemes in the C language, with critical functions written in assembly. The target chip was a CC430F6137 with 20 MHz clock, 32 KB flash for code and 4 KB RAM. The compiler used was the IAR Embedded

Table 3.1: Comparison of implemented AE schemes

| Property | CCM | (S)GCM | OCB3 | HB2 | MASHA |
|---|---|---|---|---|---|
| Block cipher calls[a] | $2m + a + 2^b$ | $m$ | $m + a + 1^b$ | — | — |
| ... in key setup | 0 | 1 | 1 | — | — |
| Online | No | Yes | Yes | Yes[c] | Yes |
| Uses only $E_K$ | Yes | Yes | No | — | — |
| Prepr. of static AD | No | Yes | Yes | No | N/A |
| Patent-free | Yes | Yes | No | No | No |
| Parallelizable | No | Yes | Yes | No | No |
| Standardized | Yes | (No) Yes | No | No | No |
| Input order | AD first | AD first | Any | AD last | N/A |

[a] $m, a$ are the number of message and AD blocks, respectively
[b] May have an additional block cipher call
[c] AD size must be fixed

Workbench version 5.30. For the AE modes based on block ciphers, we have used the AES with 128-bit keys both in software and using the AES accelerator. Our source code is available[1] to allow reproduction of our results.

The interface to the AES accelerator was written in assembly, along with a function to xor two blocks and another to increment a block.

### 3.4.1 CCM

The CCM (Counter with CBC-MAC) mode [157] essentially combines the CTR mode of encryption with the CBC-MAC authentication scheme. For each message block, a counter is encrypted with the block cipher and the result xored to the message to produce the ciphertext; the counter is then incremented. The message is also xored to an "accumulator" which is then encrypted; this accumulator will become the authentication tag after all blocks are processed.

Its implementation was fairly straightforward, employing the assembly routines to xor blocks and increment the counter.

### 3.4.2 GCM

The GCM (Galois/Counter Mode) [109] employs the arithmetic of the finite field $\mathbb{F}_{2^{128}}$ for authentication and the CTR mode for encryption. For each message block, GCM encrypts the counter and xors the result into the message to produce the ciphertext; the

[1] `http://conradoplg.cryptoland.net/software/authenticated-encryption-for-the-msp430/`

counter is then incremented. The ciphertext is xored into an accumulator, which is then multiplied in the finite field by a key-dependent constant $H$. The accumulator is used to generated the authentication tag.

In order to speed up the GCM mode, polynomial multiplication was implemented in unrolled assembly with the López-Dahab (LD) [104] algorithm using 4-bit window and two lookup tables; it is described for reference in Appendix 3.6, Algorithm 3.3. The first precomputation lookup table holds the product of $H$ and all 4-bit polynomials. Each of the 16 lines of the table has 132 bits, which take 9 words. This leads to a table with 288 bytes. The additional lookup table (which can be computed from the first one, shifting each line 4 bits to the left) allows the switch from three 4-bit shifts of 256-bit blocks to a single 8-bit shift of a 256-bit block, which can be computed efficiently with the `swpb` (swap bytes) instruction of the MSP430.

### 3.4.3 SGCM

The SGCM (Sophie Germain Counter Mode) [136] is a variant of GCM that is not susceptible to weak key attacks that exist against GCM. While these attacks are of limited nature, the author claims that they should be avoided. It has the same structure as GCM, but instead of the $\mathbb{F}_{2^{128}}$ arithmetic, it uses the prime field $\mathbb{F}_p$ with $p = 2^{128} + 12451$.

Arithmetic in $\mathbb{F}_p$ can be carried out with known algorithms such as Comba multiplication. We follow the approach in [63] which takes advantage of the multiply-and-accumulate operation present in the hardware multiplier of the MSP430 family, also taking advantage of the 32-bit multiplier present in some MSP430X devices, including the CC430 series.

### 3.4.4 OCB3

The OCB3 (Offset Codebook) mode [90] also employs the $\mathbb{F}_{2^{128}}$ arithmetic (using the same reduction polynomial from GCM), but in a simplified manner: it does not require full multiplication, but only multiplication by powers of $z$ (the variable used in the polynomial representation of the field elements). For each $i$-th message block, OCB3 computes the finite field multiplication of a nonce/key-dependent constant $L_0$ by the polynomial $z^j$, where $j$ is the number of trailing zeros in the binary representation of the block index $i$; the result is xored into an accumulator $\Delta$. This accumulator is xored to the message, encrypted, and the result is xored back with $\Delta$ to generate the ciphertext. The message block is xored into another accumulator $Y$, which is used to generate the tag.

A lookup table with 8 entries (128 bytes) was used to hold the some precomputed values of $L_0 \cdot z^j$. Two functions were implemented in assembly: multiplication by $z$ (using

left shifts) and the function used to compute the number of trailing zeros (using right shifts).

### 3.4.5 Hummingbird-2 (HB2)

The Hummingbird-2 [49] is an authenticated encryption algorithm which is not built upon a block cipher. It processes 16-bit blocks and was specially designed for resource-constrained platforms. The small block size is achieved by maintaining an 128-bit internal state that is updated with each block processed. Authenticated data is processed after the confidential data by simply processing the blocks and discarding the ciphertext generated. The algorithm is built upon the following functions for encryption:

$$\begin{aligned} S(x) &= S_4(x[0..3]) \mid (S_3(x[4..7]) \ll 4) \\ &\quad \mid (S_2(x[8..11]) \ll 8) \mid (S_1(x[12..15]) \ll 12) \\ L(x) &= x \oplus (x \lll 6) \oplus (x \lll 10) \\ f(x) &= L(S(x)) \\ \texttt{WD16}(x, a, b, c, d) &= f(f(f(f(x \oplus a) \oplus b) \oplus c) \oplus d)\,; \end{aligned}$$

where $S_1, S_2, S_3, S_4$ are S-boxes and $\lll$ denotes the circular left shift of a 16-bit word. For each 16-bit message block, HB2 calls `WD16` four times, using as inputs different combinations of the message, state and key.

We have unrolled the `WD16` function. The function $f$ is critical since it is called 16 times per block and must be very efficient; our approach is to use two precomputed lookup tables $f_L$, $f_H$ each one with 256 2-byte elements, such that $f(x) = f_L[x \;\&\; \texttt{0xFF}] \oplus f_H[(x \;\&\; \texttt{0xFF00}) \gg 8]$. These tables are generated by computing $f_L[x] \leftarrow L(S_4(x[0..3]) \mid (S_3(x[4..7]) \ll 4))$ for every byte $x$ and $f_H[x] \leftarrow L((S_2(x[8..11]) \ll 8) \mid (S_1(x[12..15]) \ll 12))$ also for every byte $x$. This optimization does not apply for $f^{-1}(x)$ since the inverse S-boxes are applied after the shifts in $L^{-1}(x)$ . In this case, we have used precomputed lookup tables $L_L, L_H$ such that $L(x) = L_L[x \;\&\; \texttt{0xFF}] \oplus L_H[(x \;\&\; \texttt{0xFF00}) \gg 8]$. These are computed as $f_L[x] \leftarrow L(x[0..7]), f_H[x] \leftarrow L(x[8..15] \ll 8)$ for every byte $x$. The four 4-bit inverse S-boxes have been merged in two 8-bit inverse S-boxes $S_L^{-1}, S_H^{-1}$ such that $S^{-1}(x) = S_L^{-1}(x[0..7]) \mid (S_H^{-1}(x[8..15]) \ll 8)$.

### 3.4.6 MASHA

MASHA [84] is an authenticated encryption algorithm based on a stream cipher. Stream ciphers are interesting since they are often more efficient than block ciphers. However, many stream ciphers which also provide authentication either have security issues (e.g.

Phelix) or performance issues. The MASHA authors propose the algorithm in order to attempt to fill this gap.

Our implementation was based on C source provided by the designers. We have changed it to reduce code size and memory footprint. The code stores the linear shift registers in circular buffers in order to avoid the actual shifts. The scheme requires multiplication, in $\mathbb{F}_{2^8}$, by four distinct constants. These are precomputed in a 256-element table which stores the multiplication of all bytes by these constants. Two such tables are required for each of the two distinct fields used by MASHA, totaling 2 KB. Since this is already large, we chose to use a byte-oriented approach for the `MixColumns` step instead of the 16-bit tailored code we will describe below. Therefore, the total space for the precomputed values becomes 2.75 KB.

### 3.4.7 Improving AES for 16-bit

We have used a software implementation of AES in order to perform comparisons with the hardware accelerator. Our implementation was based on the byte-oriented version from [57], but we have modified it to take advantage of the 16-bit platform. The first change was to improve the `AddRoundKey` function (which simply computes the xor of 128-bit blocks) in order to xor 16-bit words at a time. The second change was to improve the use of lookup tables as follows.

As it is well known, the input and output blocks of the AES can be viewed as $4 \times 4$ matrices in column-major order whose elements are in $\mathbb{F}_{2^8}$; and the AES function `SubBytes`, `ShiftRows` and `MixColumns` steps can be combined in a single one. In this step, the column $j$ of the result matrix can be computed as

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = S[a_{0,j}] \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \oplus S[a_{1,j-1}] \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \oplus S[a_{2,j-2}] \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \oplus S[a_{3,j-3}] \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix},$$

where $e$ is the output matrix, $a$ is the input matrix, $S$ is the forward S-box, $k$ is the round key matrix, and matrix indices are computed modulo four. Inspired by the 32-bit optimization of using four precomputed tables with 256 elements of with 4-byte each (totaling 4 KB), we employ the following tables:

$$T_0[a] = \begin{bmatrix} S[a] \cdot 02 \\ S[a] \end{bmatrix}, T_1[a] = \begin{bmatrix} S[a] \\ S[a] \cdot 03 \end{bmatrix}, T_2[a] = \begin{bmatrix} S[a] \cdot 03 \\ S[a] \cdot 02 \end{bmatrix}, T_3[a] = \begin{bmatrix} S[a] \\ S[a] \end{bmatrix}.$$

They consume 2 KB, half the size of the 32-bit version, providing a good compromise between the 8-bit and 32-bit oriented implementations. These tables allow the computation

of column $e_j$ as

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \end{bmatrix} = T_0[a_{0,j}] \oplus T_2[a_{1,j-1}] \oplus T_1[a_{2,j-2}] \oplus T_3[a_{3,j-3}]\,,$$
$$\begin{bmatrix} e_{2,j} \\ e_{3,j} \end{bmatrix} = T_1[a_{0,j}] \oplus T_3[a_{1,j-1}] \oplus T_0[a_{2,j-2}] \oplus T_2[a_{3,j-3}]\,.$$

### 3.4.8 Using the AES accelerator

As previously mentioned, the AES encryption and decryption using the AES hardware accelerator requires waiting for 167 and 214 cycles, respectively, before reading the results. The key to an efficient implementation using the module is to use this "delay slot" to carry out other operations that do not depend on the result of the encryption/decryption.

For example, in the CCM mode, the counter incrementation and the xor between the message and the accumulator can be carried out while the counter is being encrypted: the counter is written to the AES accelerator, the counter is incremented, we then wait for the result of the encryption and xor the result to the message when it is ready. In CCM it is also possible to generate the ciphertext (xor the encrypted result and the message) while the accumulator is being encrypted. In the GCM mode, it is possible to increment the counter while the counter is being encrypted. In the OCB3 mode, the xor between the message and the accumulator $Y$ can be carried out while the message, xored to $\Delta$, is being encrypted. For reference, these computations which can be carried out in the delay slot are marked in the algorithms of Appendix 3.6.

## 3.5 Results

The performance of the implemented AE schemes was measured for the authenticated encryption and decryption-verification of messages with 16 bytes and 4 KB, along with the Internet Performance Index (IPI) [109], which is a weighted timing for messages with 44 bytes (5%), 552 bytes (15%), 576 bytes (20%), and 1500 bytes (60%). For each message size, we have measured the time to compute all nonce-dependent values along with time for authenticated encryption and decryption-verification with 128-bit tags (except MASHA, which uses 256-bit tags). The derivation of key-dependent values is not included. For OCB3, it was assumed that the block cipher call in `init_ctr` was cached.

The timings were obtained using a development board with a CC430F6137 chip and are reported on Table 3.2; this data can also be viewed as throughput in Figures 3.1 and 3.2, considering a 20 MHz clock. The number of cycles taken by the algorithms was measured using the built-in cycle counter present in the CC430 models, which can be read

Table 3.2: Timings of implemented AE schemes for different message lengths, in cycles per byte

| | Using AES accelerator | | | Using AES in software | | |
|---|---|---|---|---|---|---|
| Scheme | 16 bytes | IPI | 4 KB | 16 bytes | IPI | 4 KB |
| *Encryption* | | | | | | |
| CTR[a] | 26 | 23 | 23 | 195 | 194 | 193 |
| CCM | 116 | 38 | 36 | 778 | 381 | 375 |
| GCM | 426 | 183 | 180 | 696 | 320 | 314 |
| SGCM | 242 | 89 | 87 | 567 | 254 | 250 |
| OCB3 | 144 | 39 | 38 | 469 | 209 | 205 |
| HB2[b] | | | | 569 | 200 | 196 |
| MASHA[b] | | | | 3,014 | 182 | 152 |
| *Decryption* | | | | | | |
| CTR[a] | 26 | 23 | 23 | 195 | 194 | 193 |
| CCM | 129 | 47 | 46 | 781 | 380 | 375 |
| GCM | 429 | 183 | 180 | 699 | 319 | 314 |
| SGCM | 243 | 89 | 87 | 571 | 254 | 250 |
| OCB3 | 217 | 48 | 46 | 510 | 245 | 242 |
| HB2[b] | | | | 669 | 297 | 292 |
| MASHA[b] | | | | 3,016 | 182 | 151 |

[a] Non-authenticated encryption scheme included for comparison
[b] Does not use AES

in the IAR debugger. Stack usage was also measured using the debugger. Code size was determined from the reports produced by the compiler, adding the size for text (code) and constants.

**Using the AES accelerator.** First, we analyze the results using the AES accelerator, for IPI and 4 KB messages. The GCM performance is more than 5 times slower than the other schemes; this is due to the complexity of the full binary field multiplication. The SGCM is more than 50% faster than GCM, since the prime field arithmetic is much faster on this platform, specially using the 32-bit hardware multiplier. Still, it is slower than the other schemes. Both CCM and OCB3 have almost the same speed, with CCM being around 4% faster. This is surprising, since that OCB3 essentially outperforms CCM in many platforms [90]. The result is explained by the combination of two facts: the hardware support for AES, which reduces the overhead of an extra block cipher call in CCM; and the fact that the AES accelerator does not support parallelism, which prevents

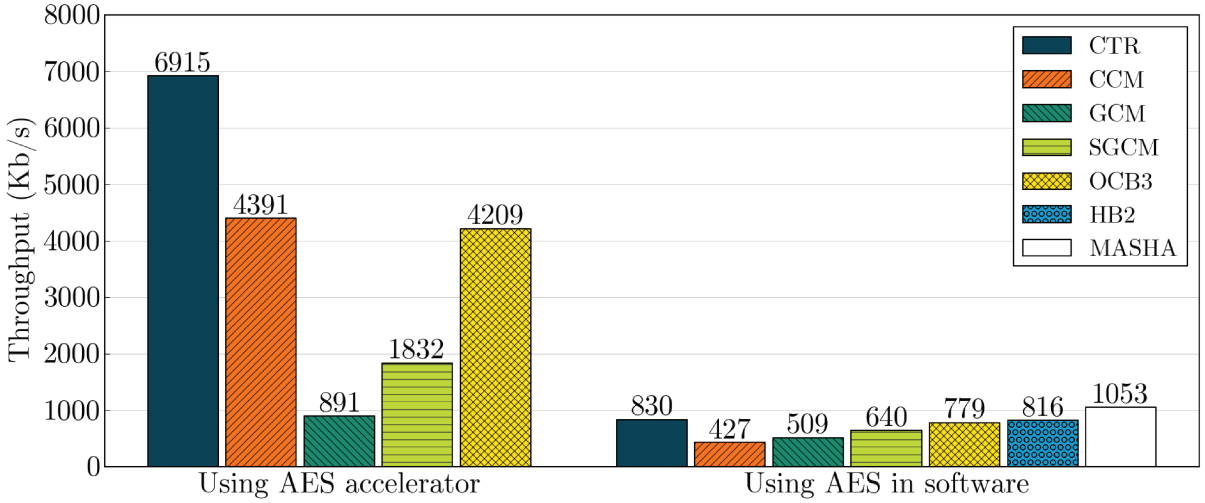



Figure 3.1: Encryption throughput in Kbps of CTR and AE schemes for 4 KB messages at 20 MHz

OCB3 from taking advantage of its support for it. We have measured that the delay slot optimization improves the encryption speed of GCM, SGCM and OCB3 by around 12% and CCM by around 24%.

**Using the AES in software.** We now consider the performance using the software AES implementation, for large messages. For reference, the block cipher takes 180 cycles per byte to encrypt and 216 cycles per byte to decrypt. The CCM mode becomes slower due to the larger overhead of the extra block cipher call. The GCM is still slower than OCB3 due to its expensive field multiplication. The SGCM is also faster than GCM, but the improvement is diluted to 20% with the software AES. The MASHA cipher is the fastest, followed by Hummingbird-2, which is 22% slower. Interestingly, Hummingbird-2 fails to outperform AES in CTR mode, which is surprising since it is specially tailored for the platform (of course, it must be considered that it provides authentication while AES-CTR by itself does not).

**AES accelerator vs. AES in software.** Using the AES accelerator, it is possible to encrypt in the CTR mode approximately 8 times faster than using AES in software; and it is possible to encrypt with CCM approximately 10 times faster for encryption and 8 times faster for decryption. The AES accelerator speedup for GCM, SGCM and OCB3 is smaller (around 1.7, 2.8, and 5.4, respectively), due to the larger software overhead.

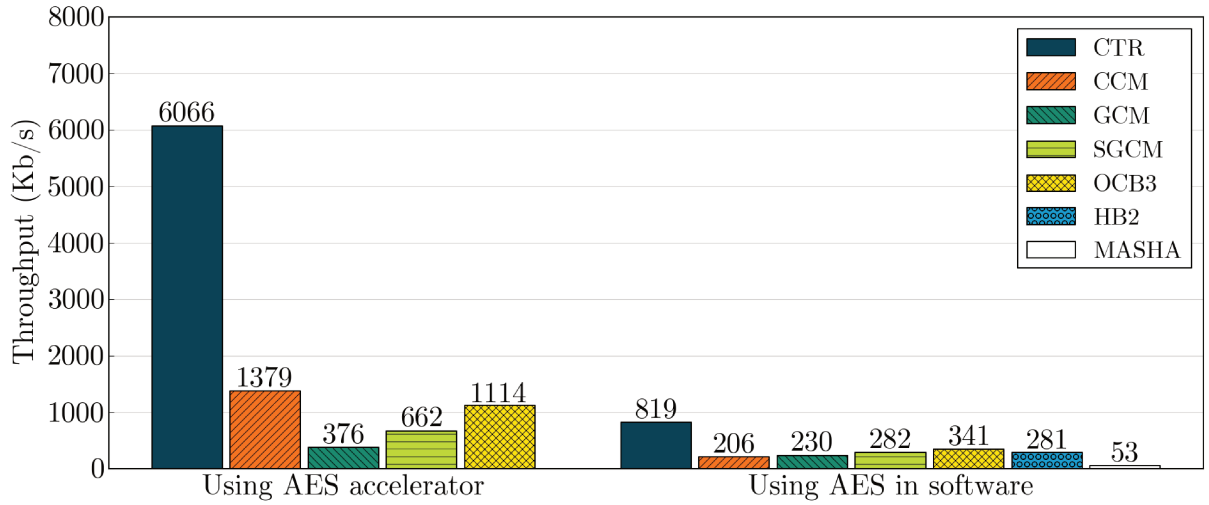



Figure 3.2: Encryption throughput in Kbps of CTR and AE schemes for 16-byte messages at 20 MHz

**Encryption vs. decryption.** When considering the usage of the AES accelerator, GCM has roughly the same performance in encryption and decryption, since the algorithm for both is almost equal; the same applies for SGCM. For both CCM and OCB3, decryption is around 25% and 20% slower, respectively. This is explained by the differences in the data dependencies of the decryption, which prevents the useful use of the delay slot, and that $D_K$ (used by OCB3) is slower than $E_K$ in the AES accelerator. Considering now the usage of the AES in software, encryption and decryption have the same performance in CCM and GCM (since there is no delay slot now) as well as in MASHA. However, decryption is almost 18% slower for OCB3, since the underlying block cipher decryption is also slower than the encryption. The decryption in Hummingbird-2 is almost 50% slower due to the $f^{-1}(x)$ function not being able to be fully precomputed, in contrast to $f(x)$. It is interesting to note that the decryption timings are often omitted in the literature, even though they may be substantially different from the encryption timings.

**Performance for small messages.** The timings for 16-byte messages are usually dominated by the computation of nonce-dependent values. The CCM using software AES has the second worst performance since all of its initialization is nonce-dependent (almost nothing is exclusively key-dependent) and it includes two block cipher calls. When using the AES accelerator, this overhead mostly vanishes, and CCM becomes the faster scheme. The nonce setup of GCM is very cheap (just a padding of the nonce) while the nonce setup of OCB3 requires the left shift of an 192-bit block by 0–63 bits. Still, the GCM

Table 3.3: ROM and RAM (stack) usage of AE schemes, in bytes. When using software AES, 2,904 additional ROM bytes are required for CCM, GCM and SGCM and 5,860 bytes for OCB3

| | CTR | CCM | GCM | SGCM | OCB3 | HB2 | MASHA |
|---|---|---|---|---|---|---|---|
| ROM | 130 | 1,094 | 4,680 | 2,172 | 1,724 | 3,674 | 5,602 |
| RAM | 100 | 258 | 886 | 322 | 538 | 196 | 499 |

performance for 16-byte messages is worse than OCB3 since it is still dominated by the block processing. Hummingbird-2 loses to OCB3 due to its larger nonce setup and tag generation. The greatest surprise is the MASHA performance which is almost four times slower than CCM, making it the slowest scheme for small messages. This result is explained by the fact that its nonce setup and tag generation are very expensive, requiring more than 20 state updates each (which take roughly the same time as encrypting ten 128-bit blocks).

**Further analysis.** In order to evaluate our AES software implementation, consider the timings from [42] (also based on [57]) which achieved 286 Kbps at 8 MHz in the ECB mode. Scaling this to 20 MHz we get 716 Kbps, while our ECB implementation achieved 889 Kbps. We conclude that our 16-bit implementation is 24% faster then the byte-oriented implementation.

Table 3.3 lists the ROM and RAM usage for programs implementing AE schemes for both encryption and decryption, using the AES accelerator. The reported sizes refer only to the code related to the algorithms and excludes the benchmark code. We recall that the MSP430X model we have used features 32 KB of flash for code and 4 KB RAM. The code for GCM is large due to the unrolled $\mathbb{F}_{2^{128}}$ multiplier, while the code for CCM is the smallest since it mostly relies on the block cipher. The RAM usage follows the same pattern: GCM has the second largest usage, since it has the largest precomputation table; the Hummingbird-2 cipher (followed by CCM) has the smallest RAM usage since it requires no runtime precomputation at all. The MASHA cipher requires the largest code space, due to the many precomputed tables used; this can be reduced by sacrificing speed. When using the software AES implementation, 2,904 additional ROM bytes are required for CCM, GCM and SGCM (which use $E_K$ only) and 5,860 additional ROM bytes are required for OCB3.

### 3.5.1 Related work

A commercial 128-bit AES implementation for the MSP430 [75] achieves 340 cycles per byte for encryption and 550 cpb for decryption, in ECB mode, using 2536 bytes. Our implementation provides 180 cpb and 216 cpb, respectively, but uses 5860 bytes. With space-time tradeoffs, it should be feasible to achieve similar results, but we have not explored them.

Simplicio Jr. et al. [146] have implemented EAX, GCM, LetterSoup, OCB2 and CCFB+H for the MSP430, using Curupira as the underlying block cipher. The EAX mode is [21] is described as an "cleaned-up" CCM and has similar performance. The authors report the results in milliseconds, but do not state the clock used. Assuming a 8 MHz clock, their timings (in cycles per byte, considering their timings for 60-byte messages and our timings for 16-byte messages) are 1,733 cpb for EAX, 5,133 cpb for GCM, 1,680 cpb for LetterSoup, 1,506 cpb for OCB2 and 2,266 cpb for CCFB+H with 8-byte tag. Our CCM is 2.2 times faster than their EAX, while our GCM is 7.3 times faster, and our OCB3 3.2 times faster than their OCB2. This difference can probably be explained by the fact that the authors have not optimized the algorithms for performance.

In [42], the encryption performance using the AES module present in the CC2420 transceiver is studied, achieving 110 cycles per byte. This is still 5 times slower than our results for the CTR mode, probably because the CC2420 is a peripheral and communicating with it is more expensive.

The Dragon-MAC [99] is based on the Dragon stream cipher. Its authors describe an implementation for the MSP430 that achieves 21.4 cycles per byte for authenticated encryption (applying Dragon then Dragon-MAC), which is faster than all timings in this work. However, it requires 18.9 KB of code. Our CCM implementation using the AES accelerator is 1.7 times slower, but 11 times smaller; while our HB2 is 9.2 times slower and 5.1 times smaller.

The Hummingbird-2 timings reported for the MSP430 in its paper [49] are about 6% and 2% faster for encryption and decryption than the timings we have obtained. However, the authors do not describe their optimization techniques, nor the exact MSP430 model used and their timing methodology, making it difficult to explain their achieved speed. However, we believe that our implementation is good enough for comparisons. Furthermore, by completely unrolling the encryption and decryption functions, we were able to achieve timings 3% and 4% faster than theirs, increasing code size by 296 and 432 bytes, respectively.

## 3.6 Conclusion and Future Work

The CCM and OCB3 modes were found to provide similar speed results using the AES accelerator, with CCM being around 5% faster. While OCB3 is the fastest scheme in many platforms, we expect CCM to be faster whenever a non-parallel AES accelerator is available. This is the case for the MSP430X models studied and is also the case for other platforms, for example, the AVR XMEGA microcontroller with has an analogue AES module.

The CCM appears to be the best choice for MSP430X models with AES accelerator considering that it also consumes less code space and less stack RAM. If one of the undesirable properties of CCM must be avoided (not being online, lack of support for preprocessing of static AD), a good alternative is the EAX mode [21] and should have performance similar to CCM. Considering software-only schemes, it is harder to give a clear recommendation: SGCM, OCB3 and HB2 provide good results, with distinct advantages and downsides. The GCM mode, even though it has many good properties, does not appear to be adequate in software implementation for resource-constrained platforms since it requires very large lookup tables in order to be competitive.

Some other relevant facts we have found are that Hummingbird-2 is slower than AES; that SGCM is 50% faster than GCM when using the AES accelerator and 20% when not; and that OCB3 and Hummingbird-2 in particular have a decryption performance remarkably slower than encryption (18% and 50% respectively). MASHA has great speed for large enough messages (29% faster than the second fastest, HB2) but very low performance for small messages (almost 4 times slower than the second slowest, CCM). For this reason, we believe there is still the need for a fast, secure and lightweight authenticated encryption scheme based on a stream cipher.

For future works it would be interesting to implement and compare lightweight encrypt-and-authenticate or authenticated encryption schemes such as LetterSoup [145] and Rabbit-MAC [152] for the MSP430X. Another possible venue for research is to study the efficient implementation of authenticated encryption using the AES accelerator featured in other platforms such as the AVR XMEGA and devices based on the ARM Cortex such as the EFM32 Gecko, STM32 and LPC1800.

# Appendix: Algorithms

Algorithm 3.1 presents CCM, where the function `format` computes a header block $B_0$ (which encodes the tag length, message length and nonce), the blocks $A_1, \ldots, A_a$ (which encode the length of the associated data along with the data itself) and the blocks $M_1, \ldots, M_m$ which represent the original message. The function `init_ctr` returns the

**Algorithm 3.1** CCM encryption

**Input:** Message $M$, additional data $A$, nonce $N$, key $K$
**Output:** Ciphertext $C$, authentication tag $T$ with $t$ bits

1: $B_0, A_1, \ldots, A_a, M_1, \ldots, M_m \leftarrow \texttt{format}(N, A, M)$
2: $Y \leftarrow E_K(B_0)$
3: **for** $i \leftarrow 1$ **to** $a$ **do**
4: $\quad Y \leftarrow E_K(A_i \oplus Y)$
5: $J \leftarrow \texttt{init\_ctr}(N)$
6: $S_0 \leftarrow E_K(J)$
7: $J \leftarrow \texttt{inc}(J)$
8: **for** $i \leftarrow 1$ **to** $m$ **do**
9: $\quad U \leftarrow E_K(J)$
10: $\quad J \leftarrow \texttt{inc}(J)$ ▷ delay slot
11: $\quad S \leftarrow M_i \oplus Y$ ▷ delay slot
12: $\quad Y \leftarrow E_K(S)$
13: $\quad C_i \leftarrow M_i \oplus U$ ▷ delay slot
14: $T \leftarrow Y[0..t-1] \oplus S_0[0..t-1]$

initial counter based on the nonce. The function `inc` increments the counter.

Algorithm 3.2 describes GCM, where the function `init_ctr` initializes the counter and the function `inc_ctr` increments the counter. The operation $A \cdot B$ denotes the multiplication of $A$ and $B$ in $\mathbb{F}_{2^{128}}$. The mode benefits from precomputed lookup tables since the second operand is fixed for all multiplications (lines 6, 14 and 16 from Algorithm 3.1). The LD multiplication with two tables, used in the field multiplication, is described in Algorithm 3.3.

OCB3 is described in Algorithm 3.4, where the function `init_delta` derives a value from the nonce and it may require a block cipher call, as explained later. The function $\texttt{ntz}(i)$ returns the number of trailing zeros in the binary representation of $i$ (e.g. $\texttt{ntz}(1) = 0$, $\texttt{ntz}(2) = 1$). The function `getL(`$L_0$`, `$x$`)` computes the field element $L_0 \cdot z^x$ and can benefit from a precomputed lookup table. Notice that the multiplication by $z$ is simply a left shift of the operand by one bit, discarding the last bit and xoring the last byte of the result with 135 (which is the representation of $z^7 + z^2 + z^1 + 1$) if the discarded bit was 1. The function `hash` authenticates the additional data and is omitted for brevity.

**Algorithm 3.2** GCM encryption

**Input:** Message $M$, additional data $A$, nonce $N$, key $K$
**Output:** Ciphertext $C$, authentication tag $T$ with $t$ bits

1: $A_1, \ldots, A_a \leftarrow A$
2: $M_1, \ldots, M_m \leftarrow M$
3: $H \leftarrow E_K(0^{128})$
4: $Y \leftarrow 0^{128}$
5: **for** $i \leftarrow 1$ **to** $a$ **do**
6: $\quad Y \leftarrow (A_i \oplus Y) \cdot H$
7: $J \leftarrow \texttt{init\_ctr}(N)$
8: $S_0 \leftarrow E_K(J)$
9: $J \leftarrow \texttt{inc}(J)$
10: **for** $i \leftarrow 1$ **to** $m$ **do**
11: $\quad U \leftarrow E_K(J)$
12: $\quad J \leftarrow \texttt{inc}(J)$ ▷ delay slot
13: $\quad C_i \leftarrow M_i \oplus U$
14: $\quad Y \leftarrow (C_i \oplus Y) \cdot H$
15: $L \leftarrow [len(A)]_{64} \,||\, [len(M)]_{64}$
16: $S \leftarrow (L \oplus Y) \cdot H$
17: $T \leftarrow (S \oplus S_0)[0..t-1]$

**Algorithm 3.3** López-Dahab multiplication in $\mathbb{F}_{2^{128}}$ for 16-bit words and 4-bit window, using 2 lookup tables.

**Input:** $a(z) = a[0..7], b(z) = b[0..7]$
**Output:** $c(z) = c[0..15]$

1: Compute $T_0(u) = u(z)b(z)$ for all polynomials $u(z)$ of degree lower than 4.
2: Compute $T_1(u) = u(z)b(z)z^4$ for all polynomials $u(z)$ of degree lower than 4.
3: $c[0..15] \leftarrow 0$
4: **for** $k \leftarrow 1$ **down to** $0$ **do**
5: $\quad$ **for** $i \leftarrow 0$ **to** $7$ **do**
6: $\quad\quad u_0 \leftarrow (a[i] \gg (8k)) \bmod 2^4$
7: $\quad\quad u_1 \leftarrow (a[i] \gg (8k+4)) \bmod 2^4$
8: $\quad\quad$ **for** $j \leftarrow 0$ **to** $8$ **do**
9: $\quad\quad\quad c[i+j] \leftarrow c[i+j] \oplus T_0(u_0)[j] \oplus T_1(u_1)[j]$
10: $\quad$ **if** $k > 0$ **then**
11: $\quad\quad c(z) \leftarrow c(z)z^8$
**return** $c$

**Algorithm 3.4** OCB3 mode encryption

**Input:** Message $M$, additional data $A$, nonce $N$, key $K$
**Output:** Ciphertext $C$, authentication tag $T$ with $t$ bits

1: $A_1, \ldots, A_a \leftarrow A$
2: $M_1, \ldots, M_m \leftarrow M$
3: $L_* \leftarrow E_K(0^{128})$
4: $L_\$ \leftarrow L_* \cdot z$
5: $L_0 \leftarrow L_\$ \cdot z$
6: $Y \leftarrow 0^{128}$
7: $\Delta \leftarrow \texttt{init\_delta}(N, K)$
8: **for** $i \leftarrow 1$ **to** $m$ **do**
9: $\quad \Delta \leftarrow \Delta \oplus \texttt{getL}(L_0, \texttt{ntz}(i))$
10: $\quad U \leftarrow E_K(M_i \oplus \Delta)$
11: $\quad Y \leftarrow Y \oplus M_i$ ▷ delay slot
12: $\quad C_i \leftarrow U \oplus \Delta$
13: $\Delta \leftarrow \Delta \oplus L_\$$
14: $F \leftarrow E_K(Y \oplus \Delta)$
15: $G \leftarrow \texttt{hash}(K, A)$
16: $T \leftarrow (F \oplus G)[0..t-1]$

# Chapter 4

# Fast Software Polynomial Multiplication on ARM Processors using the NEON Engine

Danilo Câmara, Conrado P. L. Gouvêa, Julio López and Ricardo Dahab

## Abstract

Efficient algorithms for binary field operations are required in several cryptographic operations such as digital signatures over binary elliptic curves and encryption. The main performance-critical operation in these fields is the multiplication, since most processors do not support instructions to carry out a polynomial multiplication. In this paper we describe a novel software multiplier for performing a polynomial multiplication of two 64-bit binary polynomials based on the `VMULL` instruction included in the NEON engine supported in many ARM processors. This multiplier is then used as a building block to obtain a fast software multiplication in the binary field $\mathbb{F}_{2^m}$, which is up to 45% faster compared to the best known algorithm. We also illustrate the performance improvement in point multiplication on binary elliptic curves using the new multiplier, improving the performance of standard NIST curves at the 128- and 256-bit levels of security. The impact on the GCM authenticated encryption scheme is also studied, with new speed records. We present timing results of our software implementation on the ARM Cortex-A8, A9 and A15 processors.

Danilo Câmara, Conrado P. L. Gouvêa, Julio López, and Ricardo Dahab. Fast software polynomial multiplication on ARM processors using the NEON engine. In *Security Engineering and Intelligence Informatics*, volume 8128 of *Lecture Notes in Computer Science*, pages 137–154. Springer Berlin / Heidelberg, 2013, with kind permission from Springer Science and Business Media.

## 4.1 Introduction

Mobile devices such as smartphones and tablets are becoming ubiquitous. While these devices are relatively powerful, they still are constrained in some aspects such as power consumption. Due to the wireless nature of their communication, it is very important to secure all messages in order to prevent eavesdropping and disclosure of personal information. For this reason, the research of efficient software implementation of cryptography in those devices becomes relevant. Both public key and symmetric cryptography are cornerstones of most cryptographic solutions; in particular, the public-key elliptic curve schemes and the symmetric authenticated encryption schemes are often used due to their high efficiency. Elliptic curve schemes include the well known Elliptic Curve Digital Signature Algorithm (ECDSA) and the Elliptic Curve Diffie Hellman (ECDH) key agreement scheme; while the Galois/Counter Mode (GCM) is an important example of authenticated encryption scheme which is included in many standards such as IPSec and TLS.

A significant portion of mobile devices uses processors based on the 32-bit RISC ARM architecture, suitable for low-power applications due to its relatively simple design, making it an appropriate choice of target platform for efficient implementation. Many ARM processors are equipped with a NEON engine, which is a set of instructions and large registers that supports operations in multiple data using a single instruction. Thus, our objective is to provide an efficient software implementation of cryptography for the ARM architecture, taking advantage of the NEON engine. We have aimed for standard protection against basic side-channel attacks (timing and cache-leakage). Our main contributions are: (i) to describe a new technique to carry out polynomial multiplication by taking advantage of the `VMULL` NEON instruction, achieving a binary field multiplication that is up to 45% faster than a state-of-the-art LD [104] multiplication also using NEON; (ii) using the new multiplier, to achieve speed records of elliptic curve schemes on standard NIST curves and of authenticated encryption with GCM; (iii) to offer, for the first time in the literature, comprehensive timings for four binary NIST elliptic curves and one non-standard curve, on three different ARM Cortex processors. With this contributions, we advance the state of the art of elliptic curve cryptography using binary fields, offering an improved comparison with the (already highly optimized) implementations using prime fields present in the literature. Our code will be available[1] to allow reproduction of results.

**Related work.**

Morozov et al. [119] have implemented ECC for the OMAP 3530 platform, which features a 500 MHz ARM Cortex-A8 core and a DSP core. Taking advantage of the XORMPY instruction of the DSP core, they achieve 2,106 µs in the B-163 elliptic curve and 7,965 µs

[1] `http://conradoplg.cryptoland.net/ecc-and-ae-for-arm-neon/`

in the B-283 curve to compute a shared key, which should scale to 1,053 and 3,982 Kcycles respectively.

Bernstein and Schwabe [26] have described an efficient implementation of non-standard cryptographic primitives using the NEON engine on a Cortex-A8 at the 128-bit security level, using Montgomery and Edwards elliptic curves over the prime field $\mathbb{F}_{(2^{255}-19)}$. The primitives offer basic resistance against side-channel attacks. They obtain 527 Kcycles to compute a shared secret key, 368 Kcycles to sign a message and 650 Kcycles to verify a signature.

Hamburg [69] has also efficiently implemented non-standard cryptographic primitives on a Cortex-A9 *without* NEON support at the 128-bit security level, using Montgomery and Edwards Curves over the prime field $\mathbb{F}_{(2^{252}-2^{232}-1)}$, with basic resistance against side-channel attacks. He obtains 616 Kcycles to compute a shared key, 262 Kcycles to sign a message and 605 Kcycles to verify a signature.

Faz-Hernández et al. [52] have targeted the 128-bit security level with a GLV-GLS curve over the prime field $\mathbb{F}_{(2^{127}-5997)^2}$, which supports a four dimensional decomposition of the scalar for speeding up point multiplication. The implementation also provides basic resistance against side-channel attacks. They have obtained 417 and 244 Kcycles for random point multiplication on the Cortex-A9 and A15 respectively; 172 and 100 Kcycles for fixed point multiplication and 463 and 266 Kcycles for simultaneous point multiplication.

Krovetz and Rogaway [90] studied the software performance of three authenticated encryption modes (CCM, GCM and OCB3) in many platforms. In particular, they report 50.8 cycles per byte (cpb) for GCM over AES with large messages using the Cortex-A8; an overhead of 25.4 cpb over unauthenticated AES encryption.

Polyakov [131] has contributed a NEON implementation of GHASH, the authentication code used by GCM, to the OpenSSL project. He reports a 15 cpb performance on the Cortex-A8.

**Paper structure.**

This paper is organized as follows. In Section 4.2 we describe the ARM architecture. In Section 4.3, the binary field arithmetic is explained, along with our new multiplier based on the the `VMULL` instruction. Section 4.4 describes the high-level algorithms used and Section 4.5 presents our results. Finally, concluding remarks are given in Section 4.6.

## 4.2 ARM Architecture

The ARM is a RISC architecture known for enabling the production of low-power processors and is widely spread in mobile devices. It features a fairly usual instruction set with

some interesting characteristics such as integrated shifts, conditional execution of most instructions, and optional update of condition codes by arithmetic instructions. There are sixteen 32-bit registers (`R0`–`R15`), thirteen of which are general-purpose. The version 7 of the ARM architecture has added an advanced Single Instruction, Multiple Data (SIMD) extension referred as "NEON engine", which is composed of a collection of SIMD instructions using 64- or 128-bit operands and a bank of sixteen 128-bit registers. These are named `Q0`–`Q15` when viewed as 128-bit, and `D0`–`D31` when viewed as 64-bit. There are many CPU designs based on the ARM architecture such as the ARM7, ARM9, ARM11 and the ARM Cortex series. In this work, we used three ARM Cortex devices, which we now describe.

**Cortex-A8.** The ARM Cortex-A8 processor is a full implementation of the ARMv7 architecture including the NEON engine. Compared to previous ARM cores the Cortex-A8 is dual-issue superscalar, achieving up to twice the instructions executed per clock cycle. Some pairs of NEON instructions can also be dual-issued, mainly a load/store or permutation instruction together with a data-processing instruction. Its pipeline has 13 stages followed by 10 NEON stages; its L2 cache is internal. The Cortex-A8 is used by devices such as the iPad, iPhone 4, Galaxy Tab, and Nexus S.

**Cortex-A9.** The ARM Cortex-A9 shares the same instruction set with the Cortex-A8, but it features up to four cores. It no longer supports NEON dual-issue and its L2 cache is external. However, it supports out-of-order execution of regular ARM instructions and register renaming, and has a 9–12 stage pipeline (more for NEON, we were unable to find how many). Devices that feature the Cortex-A9 include the iPad 2, iPhone 4S, Galaxy S II, and Kindle Fire.

**Cortex-A15.** Implements the ARMv7 architecture, provides dual-issue and out-of-order execution for most NEON instructions and can feature up to four cores. Its pipeline is wider, with 15 to 25 stages. The Cortex-A15 is present in devices such as the Chromebook, Nexus 10, and Galaxy S4.

**Instructions.**

We highlight the NEON instructions which are important in this work, also illustrated in Figure 4.1. The `VMULL` instruction is able to carry out several multiplications in parallel; the `VMULL.P8` version takes as input two 64-bit input vectors A and B of eight 8-bit binary polynomials and returns a 128-bit output vector C of eight 16-bit binary polynomials, where the $i$-th element of $C$ is the multiplication of the $i$-th elements from each input.

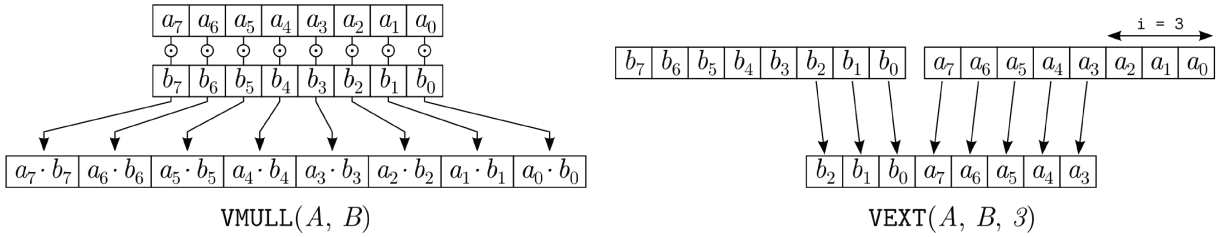



Figure 4.1: Main NEON instructions in this work: `VMULL.P8` (shortened as `VMULL`) and `VEXT`. Each square is 8 bits; rectangles are 16 bits.

The `VEXT` instruction, for two 64-bit registers and an immediate integer $i$, outputs a 64-bit value which is the concatenation of lower $8i$ bits of the first register and the higher $64 - 8i$ bits of the second register. Note that if the inputs are the same register then the `VEXT` instruction computes right bit rotation by multiples of 8 bits. The instruction also supports 128-bit registers, with similar functionality.

## 4.3 Binary Field Arithmetic

Binary field arithmetic is traditionally implemented in software using polynomial basis representation, where elements of $\mathbb{F}_{2^m}$ are represented by polynomials of degree at most $m-1$ over $\mathbb{F}_2$. Assuming a platform with a $W$-bit architecture ($W = 32$ for ARM), a binary field element $a(z) = a_{m-1}z^{m-1} + \cdots + a_2z^2 + a_1z + a_0$ may be represented by a binary vector $a = (a_{m-1}, \ldots, a_2, a_1, a_0)$ of length $m$ using $t = \lceil m/W \rceil$ words. Remaining $s = Wt - m$ bits are left unused.

Multiplication in $\mathbb{F}_{2^m}$ (*field multiplication*) is performed modulo $f(z) = z^m + r(z)$, an irreducible binary polynomial of degree $m$. This multiplication can be carried out in two steps: first, the polynomial multiplication of the operands; second, the polynomial reduction modulo $f(z)$. The basic method for computing the polynomial multiplication of $c(z) = a(z)b(z)$ is to read each $i$-th bit of $b(z)$ and, if it is 1, xor $a(z) \ll i$ into an accumulator. However, since the left-shifting operations are in general expensive, faster variations of this method have been developed. One of the fastest known methods for software implementation is the López-Dahab (LD) algorithm [104], which processes multiple bits in each iteration. We have implemented it using the NEON engine, taking advantage of the `VEXT` instruction and larger number of registers.

While the LD algorithm is often the fastest in many platforms, the presence of the `VMULL.P8` NEON instruction has the potential to change this landscape. However, it is not obvious how to build a $n$-bit polynomial multiplier for cryptographic applications ($n \geq 128$) using the eight parallel 8-bit multiplications provided by `VMULL.P8`. Our solution is a combination of the Karatsuba algorithm and a multiplier based on `VMULL.P8`

which we have named the Karatsuba/NEON/`VMULL` multiplier (KNV), described below.

### 4.3.1 New Karatsuba/NEON/VMULL (KNV) Multiplier

Our new approach was to built a 64-bit polynomial multiplier, which computes the 128-bit product of two 64-bit polynomials. This multiplier was then combined with the Karatsuba algorithm [83] in order to provide the full multiplication.

The 64-bit multiplier was built using the `VMULL.P8` instruction (`VMULL` for short) as follows. Consider two 64-bit polynomials $a(z)$ and $b(z)$ over $\mathbb{F}_2$ represented as vectors of eight 8-bit polynomials:

$$A = (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0); \quad B = (b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0).$$

To compute the polynomial multiplication $c(z) = a(z) \cdot b(z)$ (represented as a vector $C$), the schoolbook method would require sixty-four 8-bit multiplications with every $(a_i, b_j)$ combination, where each product is xored into an accumulator in the appropriate position. In our proposal, these multiplications can be done with eight executions of `VMULL` by rearranging the inputs. Let $\ggg$ denote a circular right shift; compute $A_1 = A \ggg 8$, $A_2 = A \ggg 16$, $A_3 = A \ggg 24$, $B_1 = B \ggg 8$, $B_2 = B \ggg 16$, $B_3 = B \ggg 24$ and $B_4 = B \ggg 32$ using `VEXT`. This results in:

$$\begin{aligned}
A_1 &= (a_0, a_7, a_6, a_5, a_4, a_3, a_2, a_1); & B_1 &= (b_0, b_7, b_6, b_5, b_4, b_3, b_2, b_1);\\
A_2 &= (a_1, a_0, a_7, a_6, a_5, a_4, a_3, a_2); & B_2 &= (b_1, b_0, b_7, b_6, b_5, b_4, b_3, b_2);\\
A_3 &= (a_2, a_1, a_0, a_7, a_6, a_5, a_4, a_3); & B_3 &= (b_2, b_1, b_0, b_7, b_6, b_5, b_4, b_3);\\
& & B_4 &= (b_3, b_2, b_1, b_0, b_7, b_6, b_5, b_4).
\end{aligned}$$

Now compute these `VMULL` products:

$$\begin{aligned}
D = \texttt{VMULL}(A, B) &= (a_7b_7, a_6b_6, a_5b_5, a_4b_4, a_3b_3, a_2b_2, a_1b_1, a_0b_0);\\
E = \texttt{VMULL}(A, B_1) &= (a_7b_0, a_6b_7, a_5b_6, a_4b_5, a_3b_4, a_2b_3, a_1b_2, a_0b_1);\\
F = \texttt{VMULL}(A_1, B) &= (a_0b_7, a_7b_6, a_6b_5, a_5b_4, a_4b_3, a_3b_2, a_2b_1, a_1b_0);\\
G = \texttt{VMULL}(A, B_2) &= (a_7b_1, a_6b_0, a_5b_7, a_4b_6, a_3b_5, a_2b_4, a_1b_3, a_0b_2);\\
H = \texttt{VMULL}(A_2, B) &= (a_1b_7, a_0b_6, a_7b_5, a_6b_4, a_5b_3, a_4b_2, a_3b_1, a_2b_0);\\
I = \texttt{VMULL}(A, B_3) &= (a_7b_2, a_6b_1, a_5b_0, a_4b_7, a_3b_6, a_2b_5, a_1b_4, a_0b_3);\\
J = \texttt{VMULL}(A_3, B) &= (a_2b_7, a_1b_6, a_0b_5, a_7b_4, a_6b_3, a_5b_2, a_4b_1, a_3b_0);\\
K = \texttt{VMULL}(A, B_4) &= (a_7b_3, a_6b_2, a_5b_1, a_4b_0, a_3b_7, a_2b_6, a_1b_5, a_0b_4).
\end{aligned}$$

These vectors of eight 16-bit polynomials contain the product of every $(a_i, b_j)$ combination, as required. We now need to xor everything into place. Let $L = E + F$, $M = G + H$ and

**Algorithm 4.1** Computation of $L$ and $(P_0 + P_1) \ll 8$ from $A$ and $B$

**Input:** 64-bit registers `ad` (holding $A$), `bd` (holding $B$) and `k48` (holding the constant `0x0000FFFFFFFFFFFF`)

**Output:** 128-bit register `tq (th|tl)` (holding $(P_0 + P_1) \ll 8$)

```
1: vext.8    tl, ad, ad, $1
2: vmull.p8  tq, tl, bd
3: vext.8    ul, bd, bd, $1
4: vmull.p8  uq, ad, ul
5: veor      tq, tq, uq
6: veor      tl, tl, th
7: vand      th, th, k48
8: veor      tl, tl, th
9: vext.8    tq, tq, tq, $15
```

$N = I + J$. Let $k_i$ be the $i$-th element of vector $K$ and analogously to $L$, $M$ and $N$. Now, compute:

$$P_0 = (0,\ 0,\ 0,\ 0,\ \ell_7,\ 0,\ 0,\ 0); \quad P_4 = (0,\ 0,\ 0,\ 0,\ n_7,\ n_6,\ n_5,\ 0);$$
$$P_1 = (0,\ \ell_6,\ \ell_5,\ \ell_4,\ \ell_3,\ \ell_2,\ \ell_1,\ \ell_0); \quad P_5 = (0,\ 0,\ 0, n_4,\ n_3,\ n_2,\ n_1, n_0);$$
$$P_2 = (0,\ 0,\ 0,\ 0, m_7, m_6,\ 0,\ 0); \quad P_6 = (0,\ 0,\ 0,\ 0,\ k_7,\ k_6,\ k_5, k_4);$$
$$P_3 = (0,\ 0, m_5, m_4, m_3, m_2, m_1, m_0); \quad P_7 = (0,\ 0,\ 0,\ 0,\ k_3,\ k_2,\ k_1, k_0).$$

The final result is obtained with:

$$C = A \cdot B = D + (P_0 + P_1) \ll 8 + (P_2 + P_3) \ll 16 + (P_4 + P_5) \ll 24 + (P_6 + P_7) \ll 32.$$

The expansion of the above equation produces the same results of the schoolbook method for multiplication, verifying its correctness. The whole process is illustrated in Figure 4.2, and Algorithm 4.6 in the Appendix lists the assembly code for reference. The partial results $(P_0 + P_1) \ll 8$, $(P_2 + P_3) \ll 16$ or $(P_4 + P_5) \ll 24$ can each be computed from $L$, $M$ or $N$ with four instructions (two xors, one mask operation and one shift). The partial result $(P_6 + P_7) \ll 32$ can be computed from $K$ with three instructions (one xor, one mask operation and one shift). To clarify our approach, we list the assembly code used in the computation of $L$ and $(P_0 + P_1) \ll 8$ from $A$ and $B$ in Algorithm 4.1 and describe it below.

In Algorithm 4.1, the 128-bit NEON register `tq` can be viewed as two 64-bit registers such that `tq` = `th`||`tl` where `tl` is the lower part and `th` is the higher part; the same applies to other registers. In line 1, the `VEXT` instruction concatenates the lower 8 bits of $A$ with the higher $(64 - 8) = 56$ bits of $A$, resulting in the value $A_1$ being stored in `tl`. Line 2 computes $F = \texttt{VMULL}(A_1, B)$ in the `tq` register. Lines 3 and 4 compute $B_1$ and then

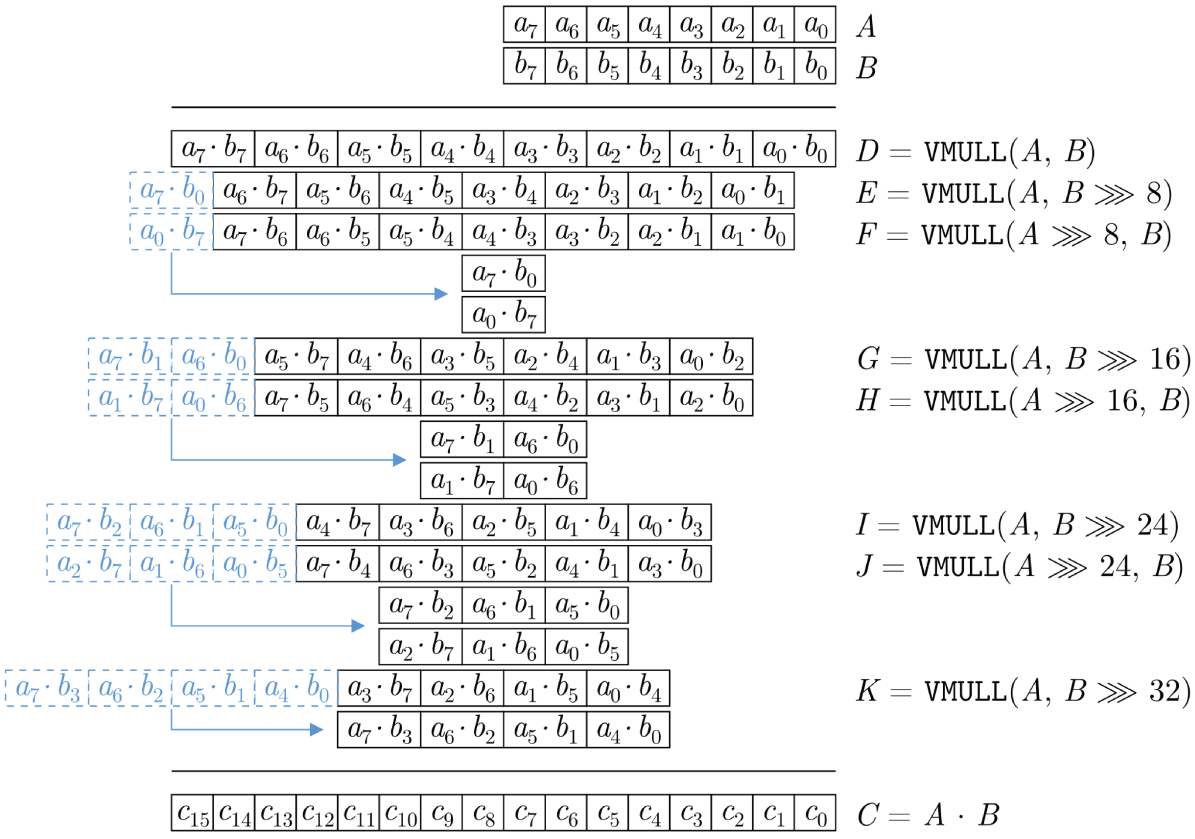



Figure 4.2: The 64 × 64-bit polynomial multiplier using `VMULL`

$E = \texttt{VMULL}(A, B_1)$ in the `uq` register, while line 5 computes $L = E + F$ in the `tq` register. Observe that the result we want, $(P_0 + P_1)$, can be viewed as $(0, \ell_6, \ell_5, \ell_4, \ell_3 + \ell_7, \ell_2, \ell_1, \ell_0)$. The straightforward way to compute $(P_0 + P_1)$ from $L$ would be to use a mask operation to isolate $\ell_7$, xor it to `tq` in the appropriate position and do another mask operation to clear the highest 16 bits. However, we use another approach which does not need a temporary register, described as follows. In line 6, we xor the higher part of `tq` into the lower part, obtaining $(\ell_7, \ell_6, \ell_5, \ell_4, \ell_3 + \ell_7, \ell_2 + \ell_6, \ell_1 + \ell_5, \ell_0 + \ell_4)$. Line 7 uses a mask operation to clear the higher 16 bits of `tq`, which now holds $(0, \ell_6, \ell_5, \ell_4, \ell_3 + \ell_7, \ell_2 + \ell_6, \ell_1 + \ell_5, \ell_0 + \ell_4)$. In line 8, the higher part of `tq` is again xored into the lower part, resulting in the expected $(0, \ell_6, \ell_5, \ell_4, \ell_3 + \ell_7, \ell_2, \ell_1, \ell_0)$ which is finally shifted 8 bits to the left with the `VEXT` instruction in line 9.

### 4.3.2 Additional Binary Field Operations

Squaring a binary polynomial corresponds to inserting a 0 bit between every consecutive bits of the input, which often requires precomputed tables. The `VMULL` instruction can improve squaring since, when using the same 64-bit value as the two operands, it computes the 128-bit polynomial square of that value.

Multiplication and squaring of binary polynomials produce values of degree at most $2m - 2$, which must be reduced modulo $f(z) = z^m + r(z)$. Since $z^m \equiv r(z) \pmod{f(z)}$, the usual approach is to multiply the upper part by $r(z)$ using shift and xors. For small polynomials $r(z)$ it is possible to use the `VMULL` instruction to carry out multiplication by $r(z)$ with a special $8 \times 64$-bit multiplier; this was done for $\mathbb{F}_{2^{128}}$ $(r(z) = z^7 + z^2 + z + 1)$ and $\mathbb{F}_{2^{251}}$ $(r(z) = z^7 + z^4 + z^2 + 1)$. Reduction in $\mathbb{F}_{2^{283}}$ takes advantage of the factorization of $r(z) = z^{12} + z^7 + z^5 + 1 = (z^7 + 1)(z^5 + 1)$ as described in [6]. For $\mathbb{F}_{2^{571}}$, $r(z) = z^{10} + z^5 + z^2 + 1$, and its reduction is computed with the usual shifts and xors.

Field inversion is commonly carried out with the well-known extended Euclidean algorithm, but it does not take constant time and may be vulnerable to side channel attacks. For this reason, we have used the Itoh-Tsujii algorithm [77], which is an optimization of inversion through Fermat's little theorem $(a(x)^{-1} = a(x)^{2^m - 2})$. The algorithm uses a repeated field squaring operation $a(x)^{2^k}$ for some values of $k$; we have implemented a special function where field squaring is completely done using NEON instruction and registers using the same techniques described for squaring and reduction, but avoiding reads and writes to memory.

## 4.4 Algorithms

The KNV multiplier was used as the building block for a implementation of Elliptic Curve Cryptography (ECC) and of authenticated encryption (AE), which we now describe together with our implementation of side-channel resistance.

### 4.4.1 Side-Channel Resistance

Side-channel attacks [89] are a serious threat for cryptographic implementations; different attacks require different levels of protection. Here we consider the basic level of resistance which avoids: branching on secret data, algorithms with timings dependent on secret data, and accessing table indexes with secret indices.

The building block of a side-channel resistant (SCR) implementation can be considered the "select" operation $t \leftarrow \textsc{Select}(a, b, v)$, which copies $a$ into $t$ if the bit $v$ is 0 or copies $b$ if $v$ is 1. This operation can be implemented without branching as described in [91] and listed for reference in Algorithm 4.3 in the Appendix. In ARM assembly, SELECT can be implemented easily since most instructions can be made conditional to a previous register comparison. However, a faster approach is to use the NEON instruction `VBIT Qd, Qn, Qm` (bitwise insert if false) — it inserts each bit in `Qn` into `Qd` if the corresponding bit in `Qm` is 1, otherwise it leaves the corresponding bit in `Qd` unchanged. If the $m$ value from Algorithm 4.3 is stored in `Qm`, then `VBIT` is precisely the SELECT operation restricted to the case where $t$ and $a$ refer to the same location (which is often the case).

Some of the algorithms we will describe use precomputed tables to improve performance. However, looking up a table entry may leak its index through side-channels, since it affects the contents of the processor cache. For this reason, we employ a side-channel resistant table lookup. We follow the strategy found in the source code of [26], listed for reference in Algorithm 4.4 in the Appendix, where $s$ can be computed without branches by copying the sign bit of $r$ (e.g. in the C language, convert $r$ to unsigned and right shift the result in order to get the highest bit). We have implemented the SCR table lookup for elliptic curve points entirely in assembly with the `VBIT` instruction. It is possible to hold the entire $t$ value (a point) in NEON registers, without any memory writes except for the final result.

### 4.4.2 Elliptic Curve Cryptography

Elliptic Curve Cryptography is composed of public key cryptographic schemes using the arithmetic of points on elliptic curves over finite fields, and it uses shorter keys at the same security level in comparison to alternative public-key systems such as RSA and DSA. Two types of fields are mainly used: prime fields (with $p$ elements, where $p$ is prime) and

binary fields (with $2^m$ elements for some $m$). While prime fields are used more often (and most literature on ECC for ARM uses them), we decided to study the efficiency of ECC using binary fields with our KNV multiplier.

Four standardized curves for Elliptic Curve Cryptography (ECC) [121] were implemented: the random curves B-283 and B-571 which provide 128 and 256 bits of security respectively; and the Koblitz curves K-283 and K-571 which provide the same bits of security respectively. A non-standard curve over $\mathbb{F}_{2^{251}}$ [24] ("B-251", roughly 128 bits of security) was also implemented, due to its high efficiency.

The main algorithm in ECC is the point multiplication, which often appears in three different cases: the random point multiplication $kP$ ($k$ terms of the elliptic point $P$ are summed), where the point $P$ is not known in advance; the fixed point multiplication $kG$, where $G$ is fixed; and the simultaneous point multiplication $kP + \ell G$ where $P$ is random and $G$ is fixed. In the random point case, we chose the Montgomery-LD multiplication [103] which offers high efficiency and basic side-channel resistance (SCR) without precomputed tables. In the fixed point case, the signed multi-table Comb method is employed [69], with side-channel resistant table lookups. It uses $t$ tables with $2^{w-1}$ points. For simultaneous point multiplication, we have used the interleaving method [56, 114] of $w$-(T)NAF. It employs two window sizes: $d$ for the fixed point (requiring a precomputed table with $2^{d-2}$ elements) and $w$ for the random point (requiring a on-the-fly table with $2^{w-2}$ elements). SCR is not required in this case since the algorithm is only used for signature verification, whose inputs are public.

The main advantage of Koblitz curves is the existence of specialized algorithms for point multiplication which take advantage of the efficient endomorphism $\tau$ present in those curves [147]. However, we have not used these algorithms since we are not aware of any SCR methods for recoding the scalar $k$ into the representation required by them. Therefore, the only performance gain in those curves were obtained using a special doubling formula with two field multiplications; see Algorithm 4.2. Montgomery-LD also requires one less multiplication per iteration in Koblitz curves.

We have selected the three following well known ECC protocols. The Elliptic Curve Digital Signature Algorithm (ECDSA) requires a fixed point multiplication for signing and a simultaneous point multiplication for verification. The Elliptic Curve Diffie-Hellman (ECDH) [15] is a key agreement scheme which requires a random point multiplication, and the Elliptic Curve Schnorr Signature (ECSS) [141] is similar to ECDSA but does not require an inversion modulo the elliptic curve order.

**Algorithm 4.2** Our proposed point doubling on the Koblitz curve $E_a$: $y^2 + xy = x^3 + ax^2 + 1$, $a \in \{0, 1\}$ over $\mathbb{F}_{2^m}$ using LD projective coordinates

**Input:** Point $P = (X_1, Y_1, Z_1) \in E_a(\mathbb{F}_{2^m})$
**Output:** Point $Q = (X_3, Y_3, Z_3) = 2P$
1: $S \leftarrow X_1 Z_1$
2: $T \leftarrow (X_1 + Z_1)^2$
3: $X_3 \leftarrow T^2$
4: $Z_3 \leftarrow S^2$
5: **if** $a = 0$ **then**
6: $\quad Y_3 \leftarrow ((Y_1 + T)(Y_1 + S) + Z_3)^2$
7: **else**
8: $\quad Y_3 \leftarrow (Y_1(Y_1 + S + T))^2$
9: **return** $(X_3, Y_3, Z_3)$

### 4.4.3 Inversion modulo the elliptic curve order

When signing, the ECDSA generates a random secret value $k$ which is multiplied by the generator point; this requires side-channel resistance since if $k$ leaks then it is possible to compute the signer's private key. However, an often overlooked point is that ECDSA also requires the inversion of $k$ modulo the elliptic curve order $n$. This is usually carried out with the extended Euclidean algorithm, whose number of steps are input-dependent and therefore theoretically susceptible to side-channel attacks. While we are not aware of any concrete attacks exploiting this issue, we are also not aware of any arguments for the impossibility of such an attack. Therefore, we believe it is safer to use a SCR inversion.

The obvious approach for SCR inversion would be to use Fermat's little theorem ($a^{-1} \equiv a^{n-2} \pmod{n}$), which would require a very fast multiplier modulo $n$ to be efficient. However, we have found a simple variant of the binary extended Euclidean algorithm by Niels Möller [115] which takes a fixed number of steps. For reference, it is described in Algorithm 4.5 in the Appendix, where branches are used for clarity and can be avoided with SELECT. The algorithm is built entirely upon four operations over integers with the same size as $n$: addition, subtraction, negation and right shift by one bit. These can be implemented in assembly for speed; alternatively the whole algorithm can be implemented in assembly in order to avoid reads and writes by keeping operands ($a$, $b$, $u$ and $v$) in NEON registers. We have followed the latter approach for fields at the 128-bit level of security, and the former approach for the 256-bit level, since the operands are then too big to fit in registers.

Interestingly, implementing this algorithm raised a few issues with NEON. The right shift and SELECT can be implemented efficiently using NEON; however, we had to resort to regular ARM instructions for addition and subtraction, since it is difficult to handle

carries with NEON. This requires moving data back and forth from NEON to ARM registers; which can be costly. In the Cortex A8, since the NEON pipeline starts after the ARM pipeline, a move from NEON to ARM causes a 15+ cycles stall. The obvious approach to mitigate this would be to move from NEON to ARM beforehand, but this is difficult due to the limited number of ARM registers. Our approach was then to partially revert to storing operands in memory since it becomes faster to read from cached memory than to move data between NEON and ARM. In the Cortex A9 we followed the same approach, but with smaller gains, since the ARM and NEON pipelines are partly parallel and moving from NEON to ARM is not that costly (around 4 cycles of latency). However, the Cortex A15 is much more optimized in this sense and our original approach of keeping operands in registers was faster.

### 4.4.4 Authenticated Encryption

An authenticated encryption (AE) symmetric scheme provides both encryption and authentication using a single key, and is often more efficient and easy to employ than using two separate encryption and authentication schemes (e.g. AES-CTR with HMAC). The Galois/Counter Mode (GCM) [109] is an AE scheme which is built upon a block cipher, usually AES. It was standardized by NIST and is used in IPSec, SSH and TLS. For each message block, GCM encrypts it using the underlying block cipher in CTR mode and xors the ciphertext into an accumulator, which is then multiplied in $\mathbb{F}_{2^{128}}$ by a key-dependent constant. After processing the last block, this accumulator is used to generate the authentication tag.

We have implemented the $\mathbb{F}_{2^{128}}$ multiplication using the same techniques described above; modular reduction took advantage of the `VMULL` instruction since $r(z)$ in this field is small. We remark that our implementation does not uses precomputed tables (as it is often required for GCM) and is side-channel resistant (if the underlying block cipher also is). For benchmarking, we have used an assembly implementation of AES from OpenSSL without SCR; however this is not an issue since we are more interested in the overhead added by GCM to the plain AES encryption.

## 4.5 Results

To evaluate our software implementation, we have used a DevKit8000 board with an 600 MHz ARM Cortex-A8 processor, a PandaBoard board with a 1 GHz ARM Cortex-A9 processor and an Arndale board with a 1.7 GHz ARM Cortex-A15 processor. We have used the GCC 4.5.1 compiler. Our optimized code is written in the C and assembly languages using the RELIC library [7]. Each function is benchmarked with two nested

Table 4.1: Our timings in cycles for binary field arithmetic

| Algorithm/Processor | | $\mathbb{F}_{2^{251}}$ | $\mathbb{F}_{2^{283}}$ | $\mathbb{F}_{2^{571}}$ |
|---|---|---|---|---|
| Multiplication (LD) | A8 | 671 | 1,032 | 3,071 |
| | A9 | 774 | 1,208 | 3,140 |
| | A15 | 412 | 595 | 1,424 |
| Multiplication (KNV) | A8 | 385 | 558 | 1,506 |
| | A9 | 491 | 701 | 1,889 |
| | A15 | 317 | 446 | 1,103 |
| Squaring (Table) | A8 | 155 | 179 | 349 |
| | A9 | 168 | 197 | 394 |
| | A15 | 128 | 151 | 282 |
| Squaring (VMULL) | A8 | 57 | 53 | 126 |
| | A9 | 63 | 59 | 146 |
| | A15 | 43 | 42 | 99 |
| Inversion (Itoh-Tsujii) | A8 | 18,190 | 20,777 | 90,936 |
| | A9 | 19,565 | 22,356 | 97,913 |
| | A15 | 13,709 | 16,803 | 71,220 |

loops with $n$ iterations each; inside the outer loop, an input is randomly generated; and the given operation is executed $n$ times in the inner loop using this input. The total time taken by this procedure, given by the `clock_gettime` function in nanoseconds, is divided by $n^2$ in order to give the final result for the given operation. We chose $n = 1024$ for measuring fast operations such as finite field arithmetic, and $n = 64$ for the slower operations such as point multiplication.

Table 4.1 presents the timings of field operations used in ECC. Our new Karatsuba/NEON/`VMULL` (KNV) multiplication gives a up to 45% improvement compared to the LD/NEON implementation. For field squaring, we have obtained a significant improvement of up to 70% compared to the conventional table lookup approach. The very fast squaring made the Itoh-Tsujii inversion feasible.

Timings for ECC protocols are listed in Table 4.2, while Figure 4.3 plots the 128-bit level timings to aid visualization. Compared to the LD/NEON multiplier with table-based squaring, the KNV multiplication with `VMULL`-based squaring improved the point multiplication by up to 50%. ECDSA is 25–70% slower than ECSS due to the SCR modular inversion required.

When limited to standard NIST elliptic curves, our ECDH over K-283 is 70% faster

Table 4.2: Our timings in $10^3$ cycles for elliptic curve protocols

| Algorithm/Processor | | B-251 | B-283 | K-283 | B-571 | K-571 |
|---|---|---|---|---|---|---|
| ECDH Agreement | A8 | 657 | 1,097 | 934 | 5,731 | 4,870 |
| | A9 | 789 | 1,350 | 1,148 | 7,094 | 6,018 |
| | A15 | 511 | 866 | 736 | 4,242 | 3,603 |
| ECDSA Sign | A8 | 458 | 624 | 606 | 2,770 | 2,673 |
| | A9 | 442 | 612 | 602 | 2,880 | 2,816 |
| | A15 | 233 | 337 | 330 | 1,740 | 1,688 |
| ECSS Sign | A8 | 270 | 389 | 371 | 1,944 | 1,846 |
| | A9 | 285 | 414 | 404 | 2,137 | 2,073 |
| | A15 | 186 | 270 | 263 | 1,264 | 1,212 |
| ECDSA Verify | A8 | 943 | 1,397 | 791 | 6,673 | 3,069 |
| | A9 | 1,100 | 1,644 | 887 | 8,171 | 3,581 |
| | A15 | 715 | 1,064 | 583 | 4,882 | 2,237 |
| ECSS Verify | A8 | 933 | 1,337 | 735 | 6,338 | 3,064 |
| | A9 | 1,086 | 1,572 | 827 | 7,776 | 3,602 |
| | A15 | 715 | 1,022 | 546 | 4,623 | 2,228 |

than the results of Morozov et al. [119]. Considering non-standard curves, we now compare our binary B-251 to the prime curves in the state of the art; this is also shown in Table 4.3. On the A8, compared to Bernstein and Schwabe's [26], our key agreement is 25% slower; our signing is 26% faster; and our verification is 43% slower. On the A9, compared to Faz-Hernández et al. [52], our random point multiplication is 88% slower, our fixed point multiplication is 53% slower; and our simultaneous point multiplication is 132% slower. On the A15, also compared to Faz-Hernández et al. [52], our random point multiplication is 108% slower, our fixed point multiplication is 72% slower; and our simultaneous point multiplication is 162% slower. We remark that this is a comparison of our implementation of binary elliptic curves with the state-of-the-art prime elliptic curve implementations, which are very different. In particular, note that the arithmetic of prime curves can take advantage of native $32 \times 32$-bit and $64 \times 64$-bit multiply instructions.

For the GCM authenticated encryption scheme, we have obtained 38.6, 41.9 and 31.1 cycles per byte for large messages, for the A8, A9 and A15 respectively; a 13.7, 13.6 and 9.2 cpb overhead to AES-CTR. Our A8 overhead is 46% faster than the timing reported by Krovetz and Rogaway's [90] and 8.6% faster than [131].

It is interesting to compare the timings across Cortex processors. The A9 results are often slower than the A8 results: while the A9 improved performance of regular ARM

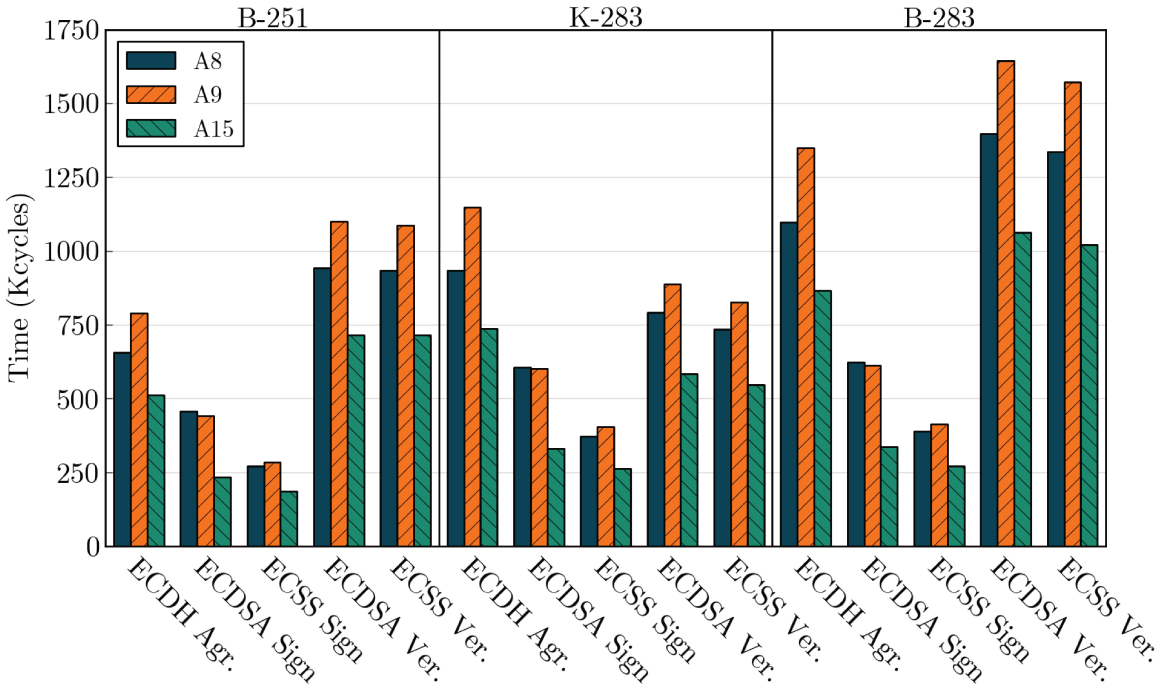



Figure 4.3: Our timings for ECC algorithms at the 128-bit level of security

code, the lack of partial dual issue in NEON caused a visible drop in performance in NEON-based code, which is our case. (The exception is ECDSA signing where the A8 currently does not have much advantage in the modular inversion, which dilutes any savings in the point multiplication.) On the other hand, the return and expansion of NEON dual issue in A15 caused great performance gains (up to 40%).

## 4.6 Conclusions and Future Work

In this paper we have introduced a new multiplier for 64-bit binary polynomial multiplication using the `VMULL` instruction, part of the NEON engine present in many ARM processors in the Cortex-A series. We then explain how to use the new multiplier to improve the performance of finite field multiplication in $\mathbb{F}_{2^m}$. We have also shown the performance gains by the new multiplier in elliptic curve cryptography and authenticated encryption with GCM. We were unable to break speed records for non-standard elliptic curves, but we believe this work offers a useful insight in how binary curves compare to prime curves in ARM processors. For standard curves we were able to improve the state of the art, as well for the GCM authenticated encryption scheme.

An interesting venue for future research is on the implementation of standard prime

Table 4.3: Our best ECC timings (on the non-standard elliptic curve B-251 over binary field) compared to state-of-the-art timings using non-standard elliptic curves over prime fields, at the 128-bit level of security, in $10^3$ cycles

| Algorithm/Processor | | Ours | [26] | [69] | [52] |
|---|---|---|---|---|---|
| Key Agreement | A8 | 657 | 527 | | |
| | A9 | 789 | | 616 | 417 |
| | A15 | 511 | | | 244 |
| Sign | A8 | 270 | 368 | | |
| | A9 | 285 | | 262 | 172 |
| | A15 | 186 | | | 100 |
| Verify | A8 | 933 | 650 | | |
| | A9 | 1,086 | | 605 | 463 |
| | A15 | 715 | | | 266 |

curves for ARM, which seems to be lacking in the literature. In addition, the arrival of ARMv8 processors in the future (including the Cortex A53 and A57) may provide great speed up to binary ECC, since the architecture will provide two instructions for the full 64-bit binary multiplier (PMULL and PMULL2) and will double the number of NEON registers [10].

# Reference Algorithms

Listed below are algorithms for reference and the full code of our multiplier.

**Algorithm 4.3** Branchless select, described by Emilia Käsper in [91]

**Input:** $W$-bit words $a$, $b$, $v$, with $v \in \{0, 1\}$
**Output:** $b$ if $v$, else $a$

1: **function** SELECT($a, b, v$)
2: $m \leftarrow$ TWOSCOMPLEMENT($-v, W$) ▷ convert $-v$ to $W$-bit two's complement
3: $t \leftarrow (m \,\&\, (a \oplus b)) \oplus a$
4: **return** $t$

**Algorithm 4.4** SCR table lookup, contained in the source code of Bernstein and Schwabe's [26]

**Input:** array $a$ with $n$ elements of any fixed type, desired index $k$, $0 \leq k < n$
**Output:** $a[k]$

```
1: function CHOOSE(a, n, k)
2:     t ← a[0]
3:     for i ← 1 to n − 1 do
4:         r ← (i ⊕ k) − 1
5:         s ← (r < 0)                          ▷ s holds whether i is equal to k
6:         t ← SELECT(t, a[i], s)
7:     return t
```

**Algorithm 4.5** SCR modular inversion algorithm by Niels Möller in the Nettle library [115]

**Input:** integer $x$, odd integer $n$, $x < n$
**Output:** $x^{-1} \pmod{n}$

```
1: function MODINV(x, n)
2:     (a, b, u, v) ← (x, n, 1, 1)
3:     ℓ ← ⌊log2 n⌋ + 1                         ▷ number of bits in n
4:     for i ← 0 to 2ℓ − 1 do
5:         odd ← a & 1
6:         if odd and a ≥ b then
7:             a ← a − b
8:         else if odd and a < b then
9:             (a, b, u, v) ← (b − a, a, v, u)
10:        a ← a ≫ 1
11:        if odd then u ← u − v
12:        if u < 0 then u ← u + n
13:        if u & 1 then u ← u + n
14:        u ← u ≫ 1
15:    return v
```

**Algorithm 4.6** Our proposed ARM NEON 64-bit binary multiplication $C = A \cdot B$ with 128-bit result

**Input:** 64-bit registers `ad` (holding $A$), `bd` (holding $B$), `k16` (holding the constant `0xFFFF`), `k32` (holding `0xFFFFFFFF`), `k48` (holding the constant `0xFFFFFFFFFFFF`).

**Output:** 128-bit register `rq (rh|rl)` (holding A).

Uses temporary 128-bit registers `t0q (t0h|t0l)`, `t1q (t1h|t1l)`, `t2q (t2h|t2l)`, `t3q (t3h|t3l)`.

```
 1: vext.8   t0l, ad, ad, $1                                  ▷ A1
 2: vmull.p8 t0q, t0l, bd                                ▷ F = A1*B
 3: vext.8   rl, bd, bd, $1                                   ▷ B1
 4: vmull.p8 rq, ad, rl                                  ▷ E = A*B1
 5: vext.8   t1l, ad, ad, $2                                  ▷ A2
 6: vmull.p8 t1q, t1l, bd                                ▷ H = A2*B
 7: vext.8   t3l, bd, bd, $2                                  ▷ B2
 8: vmull.p8 t3q, ad, t3l                                ▷ G = A*B2
 9: vext.8   t2l, ad, ad, $3                                  ▷ A3
10: vmull.p8 t2q, t2l, bd                                ▷ J = A3*B
11: veor     t0q, t0q, rq                                ▷ L = E + F
12: vext.8   rl, bd, bd, $3                                  ▷ B3
13: vmull.p8 rq, ad, rl                                  ▷ I = A*B3
14: veor     t1q, t1q, t3q                               ▷ M = G + H
15: vext.8   t3l, bd, bd, $4                                 ▷ B4
16: vmull.p8 t3q, ad, t3l                                ▷ K = A*B4
17: veor     t0l, t0l, t0h                  ▷ t0 = (L) (P0 + P1) << 8
18: vand     t0h, t0h, k48
19: veor     t1l, t1l, t1h                 ▷ t1 = (M) (P2 + P3) << 16
20: vand     t1h, t1h, k32
21: veor     t2q, t2q, rq                                ▷ N = I + J
22: veor     t0l, t0l, t0h
23: veor     t1l, t1l, t1h
24: veor     t2l, t2l, t2h                 ▷ t2 = (N) (P4 + P5) << 24
25: vand     t2h, t2h, k16
26: veor     t3l, t3l, t3h                 ▷ t3 = (K) (P6 + P7) << 32
27: vmov.i64 t3h, $0
28: vext.8   t0q, t0q, t0q, $15
29: veor     t2l, t2l, t2h
30: vext.8   t1q, t1q, t1q, $14
31: vmull.p8 rq, ad, bd                                  ▷ D = A*B
32: vext.8   t2q, t2q, t2q, $13
33: vext.8   t3q, t3q, t3q, $12
34: veor     t0q, t0q, t1q
35: veor     t2q, t2q, t3q
36: veor     rq, rq, t0q
37: veor     rq, rq, t2q
```

# Chapter 5

# Secure-TWS: Authenticating Node to Multi-user Communication in Shared Sensor Networks

Leonardo B. Oliveira, Aman Kansal, Conrado P. L. Gouvêa, Diego F. Aranha, Julio López, Bodhi Priyantha, Michel Goraczko, Feng Zhao

## Abstract

Recent works have shown the usefulness of network and application layer protocols that connect low-power sensor nodes directly to multiple applications and users on the Internet. We propose a security solution for this scenario. While previous works have provided security support for various communication patterns in sensor networks, such as among nodes, from nodes to a base station, and from users to nodes, the security of communication from sensor nodes to multiple users has not been sufficiently addressed. Specifically, we explore this design space and develop a security solution, named Secure-TWS, for efficient authentication of data sent by a resource constrained sensor node to multiple users, using digital signatures. We investigate the resource overheads in communication and computation of four suitable signature schemes – the Elliptic Curve Digital Signature Algorithm (ECDSA), the (elliptic curve) Schnorr signature, and the Boneh-Lynn-Shacham (BLS) and Zhang-Safavi-Naini-Susilo (ZSS) short signature schemes. We implement these schemes on two popular sensor node architectures (based on AVR ATmega128L and MSP430 processors with 802.15.4 radios) and experimentally characterize relevant trade-offs.

Leonardo B. Oliveira, Aman Kansal, Conrado P. L. Gouvêa, Diego F. Aranha, Julio López, Bodhi Priyantha, Michel Goraczko, and Feng Zhao. Secure-TWS: Authenticating node to multi-user communication in shared sensor networks. *The Computer Journal*, 55(4):384–396, 2012.

## 5.1 Introduction

This paper describes the implementation of a security solution for sensor nodes that are shared by multiple users. Shared sensor nodes are useful in many scenarios such as when a common sensing substrate is used by multiple applications. While the sharing of sensors over the Internet is not new [81], recent works have demonstrated the usefulness of methods that connect low-power sensor nodes directly to applications, locally and over the Internet, without intermediary gateways, such as to improve interoperability, deployment re-use, and reduce costs [132]. Network protocols at the IP layer [47, 74] and the application layer [132] have shown energy efficient methods to enable this mode of operation. Figure 5.1 shows these communication scenarios.

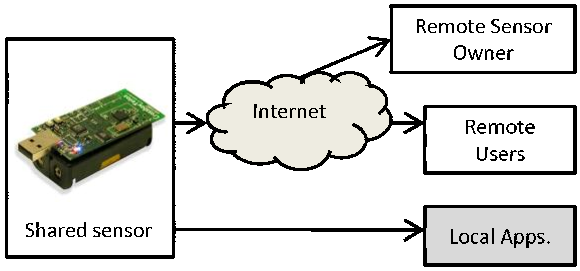



Figure 5.1: Shared sensor scenario for which security solution is implemented.

Implementing such a direct connection from sensor nodes to multiple clients in practice further entails providing *end-to-end security* for data communication. The need for security directly from the source node also arises in scenarios where the sensor node is deployed by one entity but it sends its data over a network device provided by an untrusted entity. For instance, a power metering node for real time demand-response pricing may be deployed by the utility company (shown as Remote Sensor Owner in the figure) in a home and the node may use the home Internet router supplied by the home owner. The utility company trusts only the sensor node but not the intermediate network device. The sensor node must supply authenticated data directly, rather than relying on the Internet router. The data may also be accessed by other interested clients such as the home user locally, or the building landlord remotely, who may all want authenticated data. Our implementation complements the network layer and application layer protocols available for these scenarios with support for security.

The fundamental communication pattern here is from a sensor node to multiple users. A first requirement for security is that multiple users must be able to authenticate the data they receive from a sensor node. Authentication is arguably the most important security property in securing WSN communication [105]. We describe our implementation to achieve authentication and provide experimental evaluations that help decide among key

design choices involved. The design is optimized to provide authentication for node to multi-user communication from resource-constrained sensor nodes. We explore the design space and show that a digital signature-based approach is the most efficient choice for this scenario, especially since the resource-constrained sensors may not be able to establish shared keys with a changing set of multiple users. We also compare different digital signature schemes for authenticating node to multi-user communication. The goal of the paper is not to invent new cryptography primitives but rather to close the gap between research and practice by evaluating certain key deign choices in realizing the security implementation for a shared sensor scenario.

To sum up, our key contributions are:

1. We demonstrate how node-to-multiuser communication can be authenticated in WSN's, as well as identify and evaluate the design choices involved.

2. We provide a comparison between Boneh-Lynn-Shacham (BLS) [30] and Zhang-Safavi-Naini-Susilo (ZSS) [158] short signature schemes with the Elliptic Curve Digital Signature Algorithm (ECDSA) and (elliptic curve) Schnorr signature in the context of WSN's (note that ours is the first implementation of short signature schemes on sensor platforms).

3. We provide an authenticated web service communication solution, named Secure Tiny Web Service or *Secure-TWS*, for shared and interoperable sensor nodes, based on integration of our security implementation with the Tiny Web Service stack from [132] along with its underlying IP stack [47].

The remainder of this work is organized as follows. Section 5.2 discuss the overall design motivation for the Secure-TWS security implementation along with its usage setup. A detailed discussion of the implementation issues that were evaluated in our experiments is provided in Section 5.3 and the evaluation results are discussed in Section 5.4. Related work and conclusions appear in Sections 5.5 and 5.6, respectively.

## 5.2 Authentication for Shared Sensor Scenario

Authentication consists of two properties: i) *source authentication*, which ensures a receiver that the message did in fact originate from the claimed sender; and ii) *data authentication*, that guarantees a receiver that the message received is "fresh" (i.e. it is not a replay attack) and its content was not changed since it left the sender. A second aspect of data security is privacy, typically ensured using data encryption. The implementation presented in this paper does not address the privacy aspect; privacy might not even be needed for many sensors shared over the Internet.

In this section, we identify the key security primitives that are applicable for providing authentication and select a small number of appropriate design options that need to be evaluated for enabling efficient implementations. The system to provide the required security features is described. The selected design options and parameters are then discussed and evaluated with their implementations in subsequent sections.

### 5.2.1 Design Space

Authentication can be achieved using one of two types of security primitives: *using message authentication codes (MAC's)* and using *digital signatures*.

#### Using Message Authentication Codes

Methods that use MAC's begin with an initialization step that involves distributing the shared keys in a secure manner. Once keys are distributed, authentication in a pairwise communication pattern is straightforward: sender uses the shared key to generate a message authentication code that may be verified by the receiver. Computation overhead for symmetric key cryptography is very low [129] and many embedded processors have hardware support for it, such as a hardware implementation of AES.

The use of MAC's for a node to multi-user scenario is possible, such as demonstrated in $\mu$TESLA [129] and LEAP [161]. The $\mu$TESLA scheme was described for authentication of messages from a base station to multiple nodes and the LEAP scheme for single hop communication. Both schemes use a one-way key chain (a sequence of keys $k_1, \ldots k_n$, where $k_{i+1}$ is generated from $k_i$ by applying a one-way hash function $f()$, i.e., $k_{i+1} = f(k_i)$) to achieve authentication. All nodes who are supposed to receive the authenticated messages must be supplied with a group key $k_n$ in a secure manner. This group key is common to all nodes. In $\mu$TESLA all nodes also have to maintain a synchronization with the sender regarding which key in the sequence is currently valid. Clearly, these requirements are not well-suited for our scenario where a sensor node is shared over the Internet by a changing set of multiple clients.

The special case of one-to-one communications when the connection is between a resource-constrained sensor node and a single client connected via the Internet was implemented in [67] using SSL, and named Sizzle. The conventional Public Key Cryptography (PKC) mechanisms used in SSL for exchange of shared keys were replaced with a resource efficient version of PKC, based on Elliptic Curve Cryptography (ECC), leading to a lighter version of SSL suitable for resource-constrained sensor nodes. Applying Sizzle directly for node to multi-user communication communication is inefficient as the sensor node would then have to run the SSL handshake with every one of its clients on the Internet, including short term and transient ones. Also, the data would have to be sent

multiple times to each client since a different session key would be used by each client, and no multi-cast or sharing of the same data message would be feasible.

### Digital Signature-Based Authentication

The alternative option of using digital signatures leads to certain advantages over the approach of using MAC's in this node to multi-user scenario:

1. When signatures are used, the node is not required to establish shared secret keys with each client who wishes to receive authenticated data.

2. Shared keys need not be managed or stored at the resource-constrained node.

3. The same authenticated message can be sent to multiple users and forwarded or multi-cast to other users retaining its authentication properties. This allows the network layer to optimize service to multiple simultaneous clients using multi-cast without requiring the sensor node to send multiple packets.

The design options for implementing digital signatures for sensor networks are discussed next.

A digital signature allows a sender (signatory) to generate a signature on a message. The receiver can verify the authenticity of the signature to ensure that the message indeed originated from the claimed sender and has not been modified since. The signature is generated using a private key known only to the sender and verified using a public key known publicly to everyone including the receivers. An adversary cannot forge a sender's signature without the sender's private key. Implementing digital signatures thus involves providing two components – a method to obtain [public, private] key pairs used for signing and verification, and efficient computation of the signatures.

The [public, private] key pairs can be obtained either using certificate-based schemes or certificate-free schemes. In certificate-based schemes, the key pair is associated with a particular user by a mutually trusted entity, sometimes referred to as a certification authority (CA). The trusted entity signs a user's credentials using its own private key and everyone who trusts this entity can associate the given public key to that user.

Among certificate-free schemes, many are based on identity-based methods [144]. In this approach, some unique information that correctly identifies the user (such as an email address or an IP address) is used to derive their public key. No certification is thus needed to bind the public key to the user. The keys are generated by an unconditionally trusted authority (TA) for each user. While the identity-based scheme has lower overheads in managing the issuance and verification of certificates, it has the drawback that the TA in this case knows everyone's private keys and can impersonate any user. While such

a trusted authority is easy to implement within a single sensor network, such as when all nodes fully trust the base station [124], it is not easy to provide in an Internet-based sharing scenario with many different types of users.

A certificate-free scheme that does not require a TA has been proposed in [3], named *certificateless*. However this scheme has a very high computation complexity and is not suitable for resource-constrained sensor nodes. On the other hand, CA's are already a part of the existing infrastructure. Note that CA's are easier to provide in the Internet since the users only trust the CA to reliably bind public keys to themselves, and do not have to allow the CA to be able to impersonate as themselves. Hence, in our implementation, we use the certificate-based scheme.

While obtaining the key pair is a one time overhead, the second component, the computation using the keys for signing and verification, is involved in every message exchange. We now consider the computation and communication overheads of digital signatures, for optimizing the design of this step. The signature computation choices available are described below.

**Digital Signature Algorithm (DSA):** DSA is a commonly used certificate-based signature scheme specified by NIST in FIPS 186-3. Its security relies on the Discrete Logarithm Problem (DLP) and the best known algorithm to solve it has a subexponential running time. As as result, the parameters for DSA are rather large, making it ill suited for constrained environments such as sensor nodes.

**Elliptic Curve DSA (ECDSA):** ECDSA is the elliptic curve analogue of DSA. However, its security relies on the Elliptic Curve DLP which the best known algorithm to solve it runs in fully exponential time. As consequence, one can use smaller parameters with the same security level of DSA. In addition, small key sizes offer potential reduction in processing power, memory, bandwidth, and energy. In TinyECC [100], for example, is shown that the generation of signatures using ECDSA on an MSP430-based sensor platform, at the 80-bit security level, is only 1.6s. ECDSA's signature length, however, is as long as DSA's. For instance, the signature schemes DSA-1024 and ECDSA-160, at 80-bit security level, produce a 320-bit signature.

**Schnorr Signature:** This signature scheme can be instantiated with the same efficient parameters as ECDSA, producing signatures of the same bit-length. The main difference between both is that Schnorr signatures do not require the computation of modular inverses for generating signatures. It is considered the simplest digital signature scheme to be provably secure in a random oracle model and is covered by U.S. Patent 4,995,082, which has already expired.

| Signature Scheme | Computation | | Communication (bytes) |
|---|---|---|---|
| | Generation | Verification | |
| **Identity-based** [18] | $2S$ | $1S + 1P$ | 40 |
| **Certificateless** [3] | $1S + 1P$ | $4P$ | 40 |
| **DSA** | $1E$ | $2E$ | 40 |
| **ECDSA** | $1S$ | $2S$ | 40 |
| **Schnorr** | $1S$ | $2S$ | 40 |
| **BLS** | $1S$ | $2P$ | 20 |
| **ZSS** | $1S$ | $1S + 1P$ | 20 |

Table 5.1: Signature schemes's requirements and costs

**Boneh-Lynn-Shacham (BLS) Scheme:** BLS [30] is a certicate-based signature generation and verification scheme that relies on pairings [139]. It has the advantage that its signature bit-length is half that of DSA's and ECDSA's for RSA-1024 security level. For this reason it is also referred to as a short signature scheme. Its computation overhead is asymmetric: heavier computation is needed on the receiver side but this is not a major concern for the scenario of interest.

**Zhang-Safavi-Naini-Susilo (ZSS) Scheme:** ZSS [158] is another certificate-based signature scheme relying on pairings. In comparison with BLS, ZSS requires smaller computation overhead under the same random oracle hardness assumption. The ZSS scheme is closely related to a scheme independently proposed by Boneh and Boyen (BB) [27], but proven secure in a slightly weaker complexity assumption.[1]

The resource overheads for the above options are listed in Table 5.1. The exact cost may vary by specific implementation; the table accounts for the fundamental steps involved, using specific examples. Examples of identity-based and certificateless schemes are also included for completeness, though they are clearly not strong contenders for our design choices.

The communication overhead is quantified using the extra bits needed for security, in addition to the data and protocol headers. The computation overhead is listed in terms of the computationally intensive operations involved. A *scalar* or *point multiplication* (denoted using $S$ in the table) is a multiplication operation of a point, $\mathbf{P}$, on the elliptic curve by a scalar, $k$, to obtain $\mathbf{Q} = k\mathbf{P}$. This represents $\mathbf{P}$ added to itself $k$ times where the addition is as defined in the elliptic curve group. A *pairing* (denoted as $P$) is a

[1]It is worth noting that BB has one variant that does not rely on the *random oracle model* of a hash function at the expense of increasing the signature length. We have not employed this one for efficiency purposes.

computable, non-degenerate function that has a special property known as bilinearity. Variants of the Tate pairing [139] are used in our implementation. An *exponentiation* (denoted $E$ in the table) is a modular exponentiation. i.e., a computation of the form $a^b \bmod c$.

Note that in the above schemes point multiplication works with 160-bit parameters while the pairings and exponentiation work with much larger parameters, making the point multiplication here relatively less computationally intensive than the other two operations. The parameter sizes come from the underlying problems of the cryptosystems, for achieving RSA-1024 equivalent security.

### 5.2.2 Authentication Setup

Based on the discussion of design options above, we can now select the overall authentication procedure for the security scenario of interest. To summarize, a signature-based scheme is preferred over MAC-based ones. Among signature-based schemes, a method based on certificates is preferred over certificate-free ones. Such an authentication scheme is implemented in Secure-TWS. It operates in shared wireless sensors as shown in Figure 5.2 and described below. Note that the procedure does not involve any processing at the sensor that is specific to the client identity and the approach is hence scalable to any number of short term or long term users of the sensor node.

*Initialization:* Prior to deployment, such as when the application code is loaded into the sensor node, a [public,private] key pair, $[p_{pub}(i), p_{pvt}(i)]$, is generated for each node $x(i)$. The private key $p_{pvt}(i)$ is loaded onto the node. The private key may optionally be stored in a sensor node management database by the sensor owner but these need not be shared with the certification authority or anyone else. The public key, $p_{pub}(i)$, should be provided to a certification authority. The CA will verify the deploying entity's identity and issue a certificate that binds $p_{pub}(i)$ to $x(i)$. Subsequently, the public key and the certificate are uploaded to key servers from where clients may download them directly as needed. This helps alleviating the need for nodes to transmit public keys and certificates themselves.

*Connection Setup:* Client applications that wish to use data from a sensor node $x(i)$ may establish a connection to the sensor node using the Tiny Web Services stack [132] included in Secure-TWS, or a lower layer Internet Protocol stack [47,74]. Multi-cast may be used to support multiple simultaneous clients when available.

*Authenticated Data Access:* The client now downloads the sensor node's public key, $p_{pub}(i)$ and the corresponding certificate from a key server. The CA's signature on the certificate is verified by the client. Node $x(i)$ signs the application layer data using one of the certificate-based signature generation schemes and sends it to the client. The clients

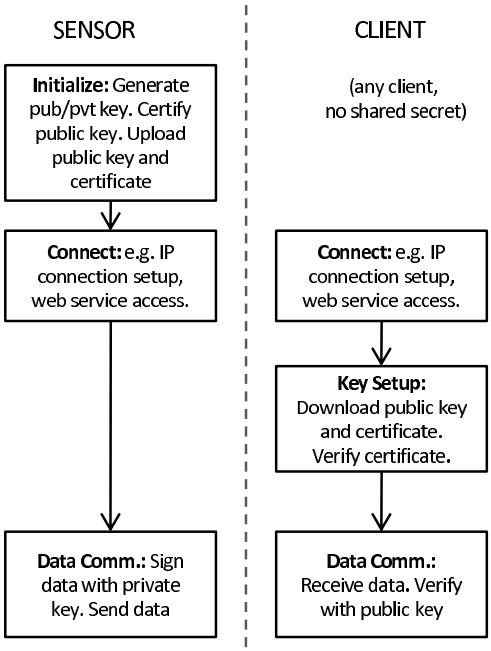



Figure 5.2: Overview of the authentication procedure implemented in Secure-TWS.

can authenticate the data using signature verification based on $p_{pub}(i)$.

### 5.2.3 Key Design Choices

The certificate-based digital signature scheme in the Secure-TWS setup described above could be based on either the DSA, ECDSA, BLS, or ZSS algorithms. DSA is clearly worse than ECDSA for sensor nodes due to its large parameter sizes and then we compare ZSS, BLS, and ECDSA. The computation performed at the sensor node only involves signature generation since the verification is performed at the client which is not resource-constrained. Considering the computation overhead of signature generation in Table 5.1, we see that all three schemes involve one point multiplication. ZSS and BLS do have a lower communication overhead. Considering the table, therefore, complexity, i.e, the order of computation alone, one may expect that ZSS and BLS are better. However in actual implementation, it turns out that exact computation costs vary greatly among the three choices due to the specific computation and parameter values involved. The choice between these three options is explored in greater depth in the next section using actual implementations of the schemes on two common sensor platforms.

## 5.3 Secure-TWS Implementation

We now discuss in depth the design parameters and resource overheads that affect the choice of digital signature schemes for shared sensor node usage. We also highlight specific implementation issues we faced in implementing these security primitives on two commonly used sensor platforms.

### 5.3.1 Platform and Software

We utilize two commonly used low-power processors to implement the Secure-TWS authentication solution: the MSP430-F2418 (16-bit 16MHz core, 8KB RAM, 116KB ROM) and AVR ATmega128L (8-bit, 7.3728MHz core, 4KB RAM, 128KB ROM). The former is present on the mPlatform [107] and the latter on the MICA mote family of nodes [73]. A block diagram of the software implementation is shown in Figure 5.3.

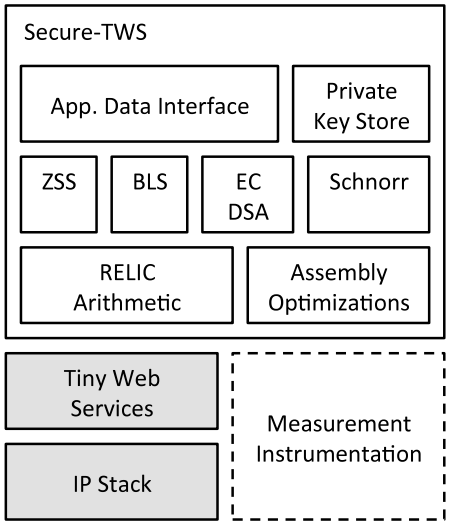



Figure 5.3: Block diagram of the security solution implementation.

As shown, the Secure-TWS implementation has been integrated with the Tiny Web Service [132] stack along with its IP stack [47]. This integration has been tested for the MSP430 processor only.

The key elements of the implementation are the ZSS, BLS, and ECDSA signature generation methods. Note that only one of these is required in a specific instantiation and we only compile the software to produce the complete system containing one of these three schemes. Assembly language optimizations have been included to improve the computation performance of signature generation.

The finite field arithmetic and big number arithmetic used for the ECC arithmetic is implemented using the RELIC library from [7]. RELIC is a publicly available C library that implements all the arithmetic primitives required in our implementation and has support for several popular platforms including the ones used in our implementation.

Note that the private key is stored locally on the node as it is used for signature generation. If the node is compromised, this key may be stolen. However, the key pair for the node is exclusive to that node itself and there is no shared key that may compromise other nodes if this node is compromised.

```
// Application data collection and
processing
...

// generate signature and signed
message
ec_sign(msg,sig); //generate
signature
strcpy(sigmsg,msg); //copy orig msg
strcat(sigmsg,sig); //append
signature

// prepare to send a signed message
s->send_data_ptr = (char *) sigmsg;
s->send_data_len = sizeof(sigmsg)-1;
```

Figure 5.4: Secure-TWS interface usage example

The application data interface uses C function calls. It allows using our implementation easily with the underlying Tiny Web Services stack [132]. Sample usage is shown in Figure 5.4, with simply a function call made for signature generation, followed by commands to send the message.

Next, we consider the choice of parameter values required for the various security modules in the implementation.

### 5.3.2 Parameter Choices

In certain sensor network scenarios, where security is limited to within the network, a 64-bit security level has sometimes been used [129] to help reduce security overheads. However, in our scenario, since the authentication must interface with the Internet, we choose to use a 80-bit security level (or 1024-bit RSA) for greater security.

The choice of finite fields and elliptic curves are crucial for the overall performance of ECDSA and pairing-based schemes. The most commonly finite fields used in ECC schemes are the prime field $\mathbb{F}_p$ and the binary field $\mathbb{F}_{2^m}$. For some field operations such as squaring, square-root, addition and reduction, binary fields present some computational advantage over prime fields. On the other hand, for some computational platforms, prime fields can take advantage of a native multiplication instruction, for accelerating a field multiplication. Both curves over prime or binary fields offer some optimizations for point multiplication; the special family of binary curves, called Koblitz curves [87], have the property that a point multiplication can be accelerated by exploiting the Frobenius endomorphism $\pi : (x, y) \rightarrow (x^2, y^2)$. For example, the $w$TNAF [147] method for point multiplication, replaces the computation of point doubling $2P$ by $\pi(P)$, a much faster operation; on the other hand, for certain ordinary elliptic curves defined over prime fields ($p > 3$), that have an efficiently-computable endomorphism, the technique of Gallant, Lambert and Vanstone (GLV) [56] can be used to speed point multiplication on these curves. We describe this method in Section 5.3.4 with a small modification to make it faster in embedded architectures with expensive division instructions.

For our scenario, we obtain a good performance using standardized curves for the software implementation of ECDSA at the 80-bit security level. Our implementation was based on the binary Koblitz curve `sect163k1` and on the prime curve `secp160k1` defined by [148]. The selection between primary and binary fields for the pairing-based schemes, however, depends on numerous aspects and is not so straightforward. The verification process in BLS and ZSS requires the computation of pairings. This restricts the choice of parameters to pairing-friendly curves:

- When using binary fields, the only pairing friendly elliptic curves known are supersingular, whose embedding degree are at most $k = 4$. This requires a 353-bit binary field in order to provide the required 80-bit level of security, which in turn increases the signature size to 353 bits. We conclude that curves over binary fields are not adequate for implementing BLS or ZSS in this scenario since they provide worst performance (which we have demonstrated experimentally) while requiring larger signatures; this was also pointed out in [30]. Implementations of pairing-based key agreement schemes have suggested using a 271-bit binary field [124], but due to Coppersmith's attack [40], the more conservative choice of 353 bits is a better suggestion for attaining the 80-bit security level.

- When using prime fields, there is a choice between supersingular and ordinary curves. Supersingular prime curves are limited to an embedding degree of $k = 2$ (which is too low for the 80-bit level and would require a 512-bit prime field); therefore, an ordinary prime curve was chosen. There are multiple families of pairing-

friendly ordinary prime curves, and for the required security level the MNT family with $k = 6$ seems the most appropriate. However, there are some disadvantages with this family — namely, it is a sparse family, which prevents the use of prime modulus with a special form; and it provides quadratic twists only [113]. An alternative is the BN family, which is specially appropriate for the 128-bit level of security due to its embedding degree $k = 12$ and sextic twists [20]. While it seems ill-suited for the 80-bit level of security, it actually provides a faster implementation in this scenario due to many optimizations tailored for the family [128]. In particular, the use of sparse primes greatly improves its performance, as will be described later. We use the BN curve $y^2 = x^3 + 3$ with parameter $x = 2^{38} + 2^5 + 2^4 + 1$ and the Optimal Ate pairing [154].

### 5.3.3 Algorithmic Choices

As mentioned in Table 5.1, elliptic curve scalar point multiplication is the most expensive operation in ECDSA, BLS and ZSS signature generation. However, the computation complexity of this operation varies widely between the three candidates.

**ECDSA and Schnorr:** As mentioned before, there is no restriction on curve selection, so the most efficient curves can be used. For this reason, Koblitz binary curves are employed due to the fast point multiplication using the $w$TNAF method. Furthermore, the point $P$ to be multiplied is a fixed public parameter that is known *a priori*. In this case, point multiplication can be accelerated significantly using pre-computation at the expense of some storage overhead. When using the $w$TNAF method, for example, $2^{w-2} - 1$ points are precomputed offline, where $w$ is the number of bits processed at once. In our ECDSA and Schnorr implementation, we have used $w = 5$ resulting in 7 precomputed points stored in ROM. Each precomputed value requires 44 bytes for storage resulting a storage overhead of 308 bytes. Such an overhead is acceptable in most situations given the ROM sizes of current sensor platforms.

**BLS:** In BLS, on the other hand, the point $P$ value is dependent of the message being signed and thus assumes a different value each time a signature is generated. Thus, for the random point multiplication, we have used a right-to-left $w$NAF method, with $w = 4$, requiring the online precomputation of 3 points. The reason for this was that right-to-left approaches are faster when point doubling is cheaper, as it is the case with the employed supersingular binary curve. Over a binary field, it is also possible to use an optimization know as short exponent [88]: by using a private key $k$ with 160 bits instead of the full 353 bits, the security provided stays the

same but the multiplication $kP$ is much faster. Point multiplication in the prime pairing-friendly curve was implemented according to the GLV method [56], with the resulting simultaneous multiplication computed as the standard left-to-right interleaving of 4-width NAFs [70].

**ZSS:** In this scheme, the point multiplication again involves a fixed basis. However, this point multiplication is computed in the same pairing-friendly curves required by BLS, providing a performance middle-ground between the schemes above. A *comb* approach [98] with 8 precomputed points stored in ROM was used for computing the fixed point multiplication in the binary case, while a combination of the *comb* approach and the GLV method was employed in the prime case. The implementation could use more precomputation to accelerate this operation, but for fairness, the schemes must be evaluated with the same memory footprint.

### 5.3.4 Strategies for Point Multiplication

In the next section, we briefly describe our simple modifications to the conventional approaches for computing point multiplication in the pairing-friendly curves employed.

#### The Right-to-left approach

Left-to-right window-based approaches for computing scalar multiplications $kP$ require some amount of precomputation in the form of small multiples of a point $P$ depending on the window length $w$. The $w$NAF approach, for example, requires recoding the integer $k$ such that it can be represented in the digit set $\{3, 5, \ldots, (2^{w-1}-1)P\}$ and also computing the set of small multiples $\{3P, 5P, \ldots, (2^{w-1}-1)P\}$. These precomputed points must be represented in affine coordinates to allow faster mixed-coordinate addition and minimize storage overhead [70]. A straightforward way to compute this set of multiples is to double $P$ as $2P$ and successively add $2P$ as $3P = P + 2P, 5P = 3P + 2P, \ldots, (2^{w-1}-1)P = (2^{w-1}-3)P + 2P$, requiring in total one point doubling and $2^{w-2}-1$ point additions. In order to minimize the number of field inversions computed in this precomputation stage, curve arithmetic can be done in projective coordinates and the affine coordinates can be recovered through an expensive simultaneous inversion [116].

If the elliptic curve supports a fast inversion-free doubling algorithm in affine coordinates, as the pairing-friendly supersingular binary curves discussed, this can be avoided with a simple alternate strategy: a set of accumulators $\{Q_1, Q_3, Q_5, \ldots, Q_{(2^{w-1}-1)}\}$ are used to accumulate point $2^iP$ in the $i$-th iteration of the algorithm. The final result can then be recovered as $Q = \sum_{j=1}^{(2^{w-1}-1)} jQ_j$. This strategy will be faster whenever these final $(2^{w-1}-1)$ point additions are faster than the simultaneous inversion operation and also allow further savings depending on how efficient is doubling a point in affine coordinates.

**The GLV method**

For the ECDSA, BLS and ZSS signatures, it is possible to exploit the efficient endomorphism $\phi\colon E \to E$ such that $\phi(P) = \lambda P$ for some $\lambda$. Using it, the point multiplication $kP$ can be written as $k = k_0 + k_1\phi(P)$, which can be computed more efficiently using an interleaving method. In order to split $k$, it is necessary to solve the equation $(k, 0) = \beta_1 v_1 + \beta_2 v_2$ for $\beta_1, \beta_2$ where $v_1 = (v_{10}, v_{11})$ and $v_2 = (v_{20}, v_{21})$ are short vectors precomputed as described in [56]. We can then write

$$\begin{aligned} d &= v_{1,0}v_{21} - v_{11}v_{2,0} \\ \beta_1 &= k\frac{v_{21}}{d} \\ \beta_2 &= -k\frac{v_{11}}{d} \end{aligned}$$

and then round $\beta_1, \beta_2$ to the nearest integer. However, this approach requires a long division by $d$, which is costly. In order to avoid it, simply precompute $c_1 = \frac{2^t v_{21}}{d}$ and $c_2 = \frac{2^t v_{11}}{d}$, rounded to the nearest integer, where $t$ is the number of bits of the curve order plus one. Then, for example, one can compute $\beta_1 = \frac{kc_1}{2^t}$, since division by $2^t$ is cheap. The last bit discarded in the right shift by $t$ decides if $\beta_1$ should be rounded up or not. This precomputation technique was hinted at in [70], but not actually described.

While the application of the GLV method for the ECDSA and BLS protocols is straightforward, it is not that clear for the ZSS protocol, since it requires a fixed point multiplication, usually computed with the *comb* method [98]. It can be done, however, by an adaptation of the two-table variant of the *comb* method. In this approach, the multiplier $k$ is written as $k = k_0 + 2^{\frac{|k|}{2}}k_1$ and certain multiples of the fixed point $P$ are precomputed in one table and multiples of $2^{\frac{|k|}{2}}P$ are precomputed in a second table. This approach is faster since $kP$ can then be computed with an interleaving method, halving the number of point duplications, but it requires twice the storage size. However, when the GLV method is available, write $k = k_0 + k_1\phi(P)$ as usual and the elements of the second table can be computed from the first table using the endomorphism as needed. Therefore, we achieve a faster multiplication with the same storage overhead.

### 5.3.5 Implementation on the ATmega128 8-bit processor

The MICAz Mote sensor node is equipped with an ATmega128 8-bit processor clocked at 7.3728MHz. The program code is stored in a 128KB EEPROM chip and data memory is provided by a 4KB RAM chip [73]. The ATmega128 processor is a typical RISC architecture with 32 registers, but six of them are special pointer registers. Since at least one register is needed to store temporary results or data loaded from memory, 25 registers are generally available for arithmetic. The instruction set is also reduced, as

only 1-bit shift/rotate instructions are natively supported. Bitwise shifts by arbitrary amounts can then be implemented with combinations of shift/rotate instructions and other instructions. In particular, shifts by 1, 4 and 7 bits can be implemented very efficiently [9]. The processor pipeline has two stages and memory instructions always cause pipeline stalls. Arithmetic instructions with register operands cost 1 cycle and memory instructions or memory addressing cost 2 processing cycles [12]. Minimizing the number of executed memory operations in low-level arithmetic is thus an evident necessity.

For the standardized binary field $\mathbb{F}_{2^{163}}$ used in the ECDSA algorithm, we followed the polynomial-basis implementation described in [9]. For the binary field $\mathbb{F}_{2^{353}}$ used in BLS/ZSS, we have selected the square-root friendly [13] trinomial $f(x) = x^{353} + x^{69} + 1$. This pentanomial has two important features: modular reduction only requires shifts by 1, 4 or 7 bits which are fast in this platform; square-root extraction does not require expensive shifts in processors with word size of 8 or 16 bits. The implementation of this field closely follows [124], with the difference that the multiplier now first features an instance of the Karatsuba algorithm before the direct López-Dahab method can be used. This was required due to the increase in parameter size from 271 to 353 bits, quickly exhausting the number of registers available for performing arithmetic.

For the prime curve used in BLS/ZSS, multiplication and squaring were implemented with a hybrid comba approach as described in [143]. This method was specifically designed to be implemented in embedded architectures with expensive memory acess. The multiple-precision multiplication required for computing Montgomery reduction [117] was specially optimized to take into account the sparse form of the prime modulus, where 6 of the 20 bytes required to store it are zero, allowing the elimination of several word multiplications inside the reduction algorithm.

### 5.3.6 Implementation on the MSP430 16-bit processor

The mPlatform provides an MSP430F2418 16-bit processor clocked at 16 MHz. It contains 116KB of program flash memory and 8KB of RAM. The MSP430 family provides 12 general purpose registers and a small instruction set with 27 instructions including 1-bit-only shifts (it is possible to use up to 4-bit shifts, but with the same speed of 4 distinct shifts). In particular, 15-bit shifts can be implemented with the left-shift/rotate-through-carry instructions. Operands may be located in registers or in memory. Since there is no cache, determining the number of cycles taken by each instruction is simple (with a few exceptions): one cycle to fetch the instruction, one cycle to fetch each offset word (if any), one cycle for each memory read and one cycle for each memory write. Small constants ($-1$, 0, 1, 2, 4 and 8) are generated by using some special registers and do not require offset words when used.

In the binary curve used for ECDSA, the standardized pentanomial $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$ was used. Multiplication in $\mathbb{F}_{2^{163}}$ was implemented with the LD algorithm, while squaring was implemented with a 512-byte lookup table storing the square of all 8-bit polynomials in ROM. Field elements are stored as an array of $n = 11$ 16-bit digits. For the binary curve used in BLS/ZSS, we have chosen the slightly different trinomial $f(x) = x^{353} + x^{95} + 1$ which allows reduction with only 1- and 2-bit shifts. Multiplication was implemented with one level of Karatsuba, as described for the ATmega128.

For the prime curve used in BLS/ZSS, multiplication was implemented with the Comba multiplication algorithm, using the MAC operation of the MSP430 hardware multiplier as described in [60]. Modular reduction was implemented with the Montgomery algorithm (also using the MAC operation), which avoids costly divisions. It was also possible to speed up the reduction by exploiting the format of the prime modulus, since two of its 16-bit digits are zero. Since the Montgomery reduction has the same structure of a multiple precision multiplication (with one of the operands being the prime modulus), it is possible to skip the steps involving multiplication by those zero digits.

## 5.4 Evaluation and Results

This section presents the performance and resource overheads of the Secure-TWS solution implemented using the above parameters and compares these overheads for ZSS, BLS, ECDSA, and Schnorr signature schemes on the MSP and AVR platforms. Specifically, the measurements evaluate i) storage, ii) computation, and iii) communication overheads.

### 5.4.1 Storage

Table 5.2 summarizes storage requirements for the signature schemes. In general, ZSS requires more ROM and global RAM, while BLS requires more RAM from the stack.

Note that virtually all the RAM memory is allocated from the stack, which means that once cryptographic operations are completed the memory becomes available for the sensor applications. Since this RAM is only used before sending a message, the memory is available for data collection and processing operations at other times for the applications.

The ZSS protocol requires the largest global RAM space due to the precomputation table for the fixed point multiplication. As a tradeoff, this table could be moved to ROM. This also applies to the precomputed constants from the GLV method, which is the reason ZSS requires more global RAM than ECDSA. The BLS protocol uses more stack RAM due to the precomputation table used in random point multiplication.

The ZSS protocol also requires more ROM in comparison to BLS due to the inversion modulo $n$ present in the signature; this routine is fairly large and is not required for BLS.

| Memory | Algorithm | MSP | | AVR | |
|---|---|---|---|---|---|
| | | **Prime** | **Binary** | **Prime** | **Binary** |
| **RAM (global)** | ZSS | 1.804 | 2.002 | 2.073 | 1.968 |
| | BLS | 1.292 | 0.818 | 1.239 | 1.412 |
| | ECDSA | 1.390 | 1.260 | 1.361 | 1.492 |
| | Schnorr | 1.390 | 1.260 | 1.361 | 1.492 |
| **RAM (stack)** | ZSS | 1.438 | 2.596 | 1.471 | 2.280 |
| | BLS | 2.086 | 2.830 | 1.942 | 2.533 |
| | ECDSA | 1.923 | 1.834 | 1.783 | 1.326 |
| | Schnorr | 2.010 | 1.860 | 1.811 | 1.356 |
| **ROM** | ZSS | 32.3 | 28.3 | 39.2 | 35.7 |
| | BLS | 30.3 | 25.2 | 37.6 | 30.7 |
| | ECDSA | 27.4 | 27.3 | 36.9 | 34.5 |
| | Schnorr | 26.9 | 27.4 | 36.1 | 33.6 |

Table 5.2: Secure-TWS's memory overheads (KB).

| Field | Algorithm | MSP | | AVR | |
|---|---|---|---|---|---|
| | | Time (ms) | Energy (mJ) | Time (ms) | Energy (mJ) |
| prime | ZSS | 229 | 6.8 | 710 | 18.0 |
| | BLS | 302 | 9.0 | 1130 | 25.7 |
| | ECDSA | 134 | 4.0 | 680 | 17.5 |
| | Schnorr | 121 | 3.6 | 620 | 14.9 |
| binary | ZSS | 703 | 20.9 | 2490 | 59.8 |
| | BLS | 391 | 11.6 | 1180 | 28.3 |
| | ECDSA | 128 | 3.8 | 370 | 8.9 |
| | Schnorr | 114 | 3.4 | 330 | 7.9 |

Table 5.3: Secure-TWS's computations costs.

The same effect appears when comparing ECDSA and Schnorr.

### 5.4.2 Computation and Communication

Signature schemes based on ECC are often referred to as having similar computation requirements for signature generation. Theoretically, this is not incorrect: the computationally intensive step that both schemes require is a point multiplication which in practice incurs costs of the same order of magnitude. In practice, however, costs of the same order are not necessarily equivalent, especially in resource-constrained sensor platforms. Having implemented schemes on a real sensor platform allows measuring these differences in a more precise manner.

Table 5.3 shows the computation costs for ZSS, BLS, ECDSA, and Schnorr signature schemes. In our implementation, ZSS is faster than BLS due to the fixed point multiplication using precomputation. ZSS is not faster than ECDSA since the latter i) makes use of a special prime enabling fast reduction in the prime case and ii) uses a smaller finite field in the binary case. Also, concerning ECDSA, its is worth noting that the binary case faster than the prime case – due to the Koblitz curve optimizations –, but BLS and ZSS are faster in the prime field – due to the smaller field required. The binary ZSS is much slower than the binary BLS because it can not use the short exponent optimization. Schnorr is slightly faster than ECDSA because the computation of a modular inversion is not required. Also note that the results for the MSP and for the AVR are not equivalent, since there are differences in clock speed and word size that favors the former – MSP is usually 3 or 4 times faster than AVR.

Table 5.4, in turn, shows how signature length affects the communication energy cost of transmitting the signature. The cost is higher for binary fields since the signature size depends on the field size, and pairing-based protocols need a larger field in the binary case

| Field | Algorithm | MSP | | AVR | |
|---|---|---|---|---|---|
| | | **Bit-length** | **Energy (mJ)** | **Bit-length** | **Energy (mJ)** |
| **prime** | ZSS | 161 | 0.15 | 161 | 0.14 |
| | BLS | 161 | 0.15 | 161 | 0.14 |
| | ECDSA | 320 | 0.23 | 320 | 0.21 |
| | Schnorr | 320 | 0.23 | 320 | 0.21 |
| **binary** | ZSS | 354 | 0.24 | 354 | 0.23 |
| | BLS | 354 | 0.24 | 354 | 0.23 |
| | ECDSA | 320 | 0.23 | 320 | 0.21 |
| | Schnorr | 320 | 0.23 | 320 | 0.21 |

Table 5.4: Communication signature lengths and energy consumption for signature schemes implemented in Secure-TWS

due to the small embedding degree of the supersingular binary curves [30]. It is worth noting that the term "short signatures" is only applicable to ZSS and BLS when those are implemented under prime fields – in fact, under binary fields, ZSS' and BLS' signature sizes are larger even than ECDSA's.

Finally, note that the energy consumption does not depend solely on the signature length because of existing radio start-up energy costs. Radio costs are discussed in more detail in [106].

### 5.4.3 Combined Resource Overhead

Overall costs are shown in Table 5.5. As can be seen from its data, the computation energy dominates, while the size of the signature is not a significant issue.

In fact, the cost of computation is so much higher that even when the transmission of the signed data to multiple recipients and over multiple wireless hops is considered, the one time cost of computing the signature is still dominant for a reasonable number of clients.

Based on these comparisons, it seems that Secure-TWS should be used with Schnorr when only time and overall energy efficiency is required. Whenever there is the additional requirement of interoperability, ECDSA must rather be considered since it has also performed well on the evaluated scenarios and is a standardized algorithm.

It is worth noting that ZSS and BLS may also be a good choice despite their results in the evaluated scenarios. They indeed have smaller signatures and then are more appropriate for scenarios where communication dominates energy consumption (e.g. for underwater sensor networks [41]). Besides, BLS is able to aggregate signatures [29] and then is also more adequate for environments where receivers are storage-constrained.

| Field | Algorithm | MSP | AVR |
|---|---|---|---|
| | | Energy (mJ) | Energy (mJ) |
| prime | ZSS | 6.95 | 18.14 |
| | BLS | 9.15 | 25.84 |
| | ECDSA | 4.23 | 17.71 |
| | Schnorr | 3.83 | 15.12 |
| binary | ZSS | 21.14 | 60.03 |
| | BLS | 11.84 | 28.53 |
| | ECDSA | 4.03 | 9.11 |
| | Schnorr | 3.63 | 8.15 |

Table 5.5: Energy overhead of different signature schemes in SecureTWS.

## 5.5 Related Work

Several works have addressed the problem of providing security in wireless sensor networks. Communication in WSNs exhibits a number of different patterns. To be effective and efficient, a solution needs to be tailored to the particular communication pattern at hand. This has lead to several methods for security for different scenarios:

1. Node to node [2] (e.g. [45, 51, 101, 126, 129, 161] among others);
2. Node to multiple nodes within same sensor network (e.g. [160]);
3. User to node (e.g. [129])
4. Node to user (e.g. [67]).
5. User to multiple nodes (e.g. [129]);

In this work we consider a communication pattern from a sensor node to multiple Internet connected users.

This has not been sufficiently addressed before. The most closely related works are those in one-to-many communication within sensor networks. Majority of these proposals make use of authenticated broadcasts, based on symmetric cryptosystems, such as $\mu$TESLA [129]. The $\mu$TESLA approach has been studied and improved for specific contexts in follow up works (e.g [44, 102, 105, 122]). A slightly different approach, specifically targeted to local broadcasts, was proposed by Zhu *et al.* [161]. As discussed in Section 5.2.1, those strategies, albeit very effective for the scenarios they are designed for, are not adequate for authenticating node to multi-user interactions.

[2] Sometimes, this problem has been addressed indirectly, i.e., by providing a key agreement protocol. Keys established can further be employed to generate message authentication codes

The authentication in the reverse direction, from a user to multiple sensor nodes has also been considered before. Ren *et al.* [133] have combined Merkle trees, Bloom filters, and PKC-based signature schemes for this purpose.

Our implementation makes use of digital signature schemes based on PKC. PKC has already been shown to be feasible in resource-constrained sensor nodes [68,100,108,151]). For instance, Gura *et al.* [68] reported results for ECC and RSA primitives on the ATmega128L and demonstrated the advantages of ECC. We use ECC-based PKC in our implementation. The ECC implementation in [68] is based upon arithmetic in prime finite fields. Malan *et al.* [108], on the other hand, presented the first ECC implementation over binary fields for sensor nodes. We have used prime and binary fields in our implementation, as discussed in Section 5.3.2.

Liu *et al.* [100] have previously demonstrated the ECDSA signature scheme in resource-constrained sensor node platforms. We compare the ECDSA performance to BLS, demonstrate its use for a shared sensor authentication scenario, and integrate it with end-to-end network and application layer protocols from [47, 132].

One of the signature schemes we have used in our proposal is the BLS, which is based on Pairing-Based Cryptography (PBC), a relatively recent addition to ECC. PBC has previously been proposed for use in WSNs [124, 151] but it has a high computation overhead of several seconds. Szczechowiak *et al.* [151] developed an implementation of pairings over binary and prime fields. Their implementation uses the Karatsuba's multiplication method and takes 10.96s on an ATmega128L-based platform. This performance has further been improved in [124], achieving the $\eta_T$ pairing computation in 5.5s on the ATmega128L by using López-Dahab field multiplication [104]. The same $\eta_T$ has been implemented in the work of Ishiguro et al [76] using ternary fields and evaluates pairings in 5.79s. These works are complimentary to our implementation as in our approach, the pairings are not required to be computed on the resource-constrained sensors, thus avoiding the high computation overhead of PBC on the sensors.

## 5.6 Conclusion

We developed a solution, Secure-TWS, for authenticated communication for the scenario of sensor nodes that are shared as a common deployed substrate among multiple applications and users both locally and through the Internet. Our implementation was tested with existing web service layer and IP layer implementations for resource-constrained sensor nodes, thus providing a useful system that can be used for deployment scenarios where authentication is necessary.

We also discussed the numerous design choices that were considered in our implementation. We found that a digital signature-based approach is the most efficient choice

for this type of interaction and then compared different digital signature schemes for realizing it. The ZSS, BLS, ECDSA, and Schnorr schemes were the most desirable ones and were implemented on two popular sensor platforms. This implementation provides one of the first experimental characterization of ZSS's and BLS's resource overheads on resource-constrained sensor nodes. The performance resource overheads were experimentally measured and the factors involved in the choice among these schemes were discussed. The implementation clearly shows that for this authentication scenario, computation costs largely dominate over communication, as opposed to symmetric key-based schemes where communication is the dominant cost.

The comparison also showed that, among the three schemes, Schnorr is fastest on the AVR and MSP430 processors. In addition to providing a practically usable system, the implementation effort has also provided valuable insights into relevant challenges and design choices.

While this implementation has addressed the authentication problem, other security aspects such as privacy may also be relevant in certain shared sensor node scenarios and end-to-end solutions that are integrated with sensor node network stack implementations. These may be addressed in future work.

## Acknowledgments

The authors are grateful to Piotr Szczechowiak, Michael Scott, and Fredrik Österlind for useful discussions and helpful tips from their implementation experiences.

# Chapter 6

# Conclusions

The secure and efficient implementation of cryptographic schemes is an important area of research and is critical for the successful deployment of cryptographic software in the real world. Security against side-channel attacks is increasingly being considered essential due to many existing feasible attacks. Efficiency is still important — despite the high performance of modern CPUs — for multiple reasons such as: cryptographic schemes are required in low-cost (and therefore low-performance) equipment such as smartcards and wireless sensors; modern devices have limited battery life, and faster algorithms can consume less energy; and high traffic servers may need to execute cryptographic schemes as fast as possible to keep up with the number of client requests.

In the previous chapters, contributions to this subject were presented as a collection of papers on which the author participated. These contributions were in Elliptic Curve Cryptography, Pairing-Based Cryptography and authenticated encryption; and were concretely implemented in the C and assembly languages and are available online.

In Chapter 2, the implementation of Elliptic Curve Cryptography (ECC) and Pairing-Based Cryptography (PBC) for the MSP430 microcontroller was presented. New optimizations for the finite field arithmetic were developed, taking advantage of the hardware multiplier. The new MSP430X extension of the architecture was also studied and employed to successfully improve protocol timings. A technique for combining the GLV method and the Comb point multiplication was also described, leading to improved timings for ECDSA and ZSS signing. The timings obtained for digital signature, short signature and key agreement are still the state of the art. Timings at 25 MHz include 0.144 s for ECDSA signature, 0.271 s for ECDSA verification and 2.279 s SOK key agreement at the 128-bit security level.

The implementation of authenticated encryption (CCM, GCM, SGCM, OCB3, MASHA, Hummingbird-2) for the same MSP430 was then described in Chapter 3, establishing a new state of the art. The AES accelerator present in some MSP430 devices was studied

and used to greatly improve timings. To allow a fair comparison, an efficient software implementation of the AES block cipher was also developed, focusing on 16-bit platforms. Some interesting findings include CCM being faster than other schemes with the AES accelerator, SGCM being faster than GCM; plain AES being faster than Hummingbird-2; and the low performance of MASHA for small messages. With the AES accelerator, throughputs obtained at 20 MHz include 6,915 Kb/s for AES-CTR, 4,391 Kb/s for CCM, 891 Kb/s for GCM; without the accelerator, 830 Kb/s for AES-CTR, 427 Kb/s for CCM, 509 Kb/s for GCM.

In Chapter 4, the ARM Cortex series of processors was the target platform; they are widely used in mobile devices such as smartphones and tablets. The NEON engine — a collection of registers and instructions supporting operations on multiple data with a single instruction — was studied and its `VMULL` instruction was explored in order to develop an efficient binary polynomial multiplier. This multiplier was then used to implement both ECC and authenticated encryption with the GCM scheme. Improvement of the state of the art was obtained for ECC over standard curves and for GCM, but not for non-standard curves, where implementations over prime fields still hold speed records. Nevertheless, it was possible to provide an interesting comparison between the two types of fields. The work also paves the way for an efficient implementation on the new ARMv8 architecture which will include a NEON instruction for polynomial multiplication; significant speed improvements for both ECC and GCM are expected.

In Chapter 5, a concrete application of digital signatures (both regular and short, pairing-based) was presented: Secure-TWS, a security solution which enables wireless sensors to send data to clients through the web in an authenticated manner. Two platforms were targeted: the AVR Atmel and MSP430 microcontrollers; I was responsible for the latter implementation. Short and regular signatures were compared in terms of performance and energy consumption. It was concluded that short signatures are not better in this particular scenario since the energy saved in the transmission of the shorter signature is less than the additional energy required by the short signature computation. However, the paper remarks that short signatures are still useful in scenarios where communication is very expensive, such as underwater sensor networks.

All code developed in this work is or will be made available in the RELIC library [7], which is partly maintained by the author. This allows the reproduction of results by other researchers.

**Future work.** The efficient and secure implementation of cryptographic schemes should be a relevant topic for the foreseeable future. However, a complex topic is: which cryptographic schemes should be implemented and deployed, and with which parameters? Recently, documents on the programs carried out by the National Security Agency (NSA)

leaked by a former contractor, Edward Snowden, confirmed the suspicion that the agency enforced the inclusion of a backdoored pseudo-random number generator into one of the NIST standards [14]. Along with many other leaks, this weakened the trust into american cryptographic standards, which in turn is increasing the adoption of alternative schemes and parameters. An example is Curve25519 [23], an elliptic curve which supports usual public-key schemes which can be implemented in a secure and efficient manner, and which is now the standard algorithm of the popular remote access software OpenSSH. This rapidly changing landscape must be followed and studied in order to determine safe and efficient schemes for future use.

# Bibliography


[1] Leonard Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *SFCS '79: Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, pages 55–60. IEEE Computer Society, 1979.

[2] Essame Al-Daoud, Ramlan Mahmod, Mohammad Rushdan, and Adem Kilicman. A new addition formula for elliptic curves over GF($2^n$). *IEEE Transactions on Computers*, 51(8):972–975, 2002.

[3] Sattam S. Al-Riyami and Kenneth G. Paterson. Certificateless public key cryptography. In Chi-Sung Laih, editor, *Proceedings of the 9th International Conference on the Theory and Application of Cryptology and Information Security ASIACRYPT'03*, pages 452–473, Taipei,Taiwan, November 2003. Springer.

[4] Nadhem AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering, and Jacob C.N. Schuldt. On the security of RC4 in TLS. In *Proceedings of the 22nd USENIX Security Symposium*, 2013.

[5] Nadhem J. AlFardan and Kenneth G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols, 2013. `http://www.isg.rhul.ac.uk/tls/TLStiming.pdf`.

[6] Diego F. Aranha, Armando Faz-Hernández, Julio López, and Francisco Rodríguez-Henríquez. Faster implementation of scalar multiplication on Koblitz curves. In Alejandro Hevia and Gregory Neven, editors, *Progress in Cryptology — LATINCRYPT 2012*, volume 7533 of *Lecture Notes in Computer Science*, pages 177–193. Springer Berlin / Heidelberg, 2012.

[7] Diego F. Aranha and Conrado P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. `http://code.google.com/p/relic-toolkit/`.

[8] Diego F. Aranha, Koray Karabina, Patrick Longa, Catherine Gebotys, and Julio López. Faster explicit formulas for computing pairings over ordinary curves. In

*Advances in Cryptology — EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 48–68. Springer Berlin / Heidelberg, 2011.

[9] Diego F. Aranha, Leonardo B. Oliveira, Julio López, and Ricardo Dahab. Efficient implementation of elliptic curve cryptography in wireless sensors. *Advances in Mathematics of Communications*, 4(2):169–187, 2010.

[10] ARM Limited. ARMv8 instruction set overview, 2012.

[11] Christophe Arène, Tanja Lange, Michael Naehrig, and Christophe Ritzenthaler. Faster computation of the Tate pairing. *Journal of Number Theory*, 131(5):842–857, 2011.

[12] Atmel. *8 bit AVR Microcontroller ATmega128(L) manual*, 2467m-avr-11/04 edition, November 2004.

[13] Roberto M. Avanzi. Another look at square roots (and other less common operations) in fields of even characteristic. In C. M. Adams, A. Miri, and M. J. Wiener, editors, *Selected Areas in Cryptography*, pages 138–154, Berlin, Heidelberg, March 2007. Springer-Verlag.

[14] James Ball, Julian Borger, and Glenn Greenwald. Revealed: how US and UK spy agencies defeat internet privacy and security. Guardian Weekly, September 2013. `http://gu.com/p/3thvv`.

[15] Elaine Barker, Don Johnson, and Miles Smid. NIST SP 800-56A: Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography, March 2007.

[16] Elaine Barker and Allen Roginsky. Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. NIST Special Publication 800-131A, 2011. `http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf`.

[17] Paulo S. L. M. Barreto, Steven Galbraith, Colm Ó hÉigeartaigh, and Michael Scott. Efficient pairing computation on supersingular abelian varieties. *Designs, Codes and Cryptography*, 42(3):239–271, 2007.

[18] Paulo S. L. M. Barreto, Benoît Libert, Noel Mccullagh, and Jean-Jacques Quisquater. Efficient and provably-secure identity-based signatures and signcryption from bilinear maps. In Bimal K. Roy, editor, *Advances in Cryptology - ASIACRYPT 2005*, pages 515–532, Chennai, December 2005. Springer-Verlag.

[19] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. On the selection of pairing-friendly groups. In *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 17–25. Springer Berlin / Heidelberg, 2004.

[20] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer Berlin / Heidelberg, 2006.

[21] Mihir Bellare, Phillip Rogaway, and David Wagner. The EAX mode of operation. In *Fast Software Encryption*, volume 3017 of *Lecture Notes in Computer Science*, pages 389–407. Springer Berlin / Heidelberg, 2004.

[22] Daniel J. Bernstein. A software implementation of NIST P-224. In *Presentation at the 5th Workshop on Elliptic Curve Cryptography (ECC 2001)*, 2001.

[23] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In *Public Key Cryptography - PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer Berlin / Heidelberg, 2006.

[24] Daniel J. Bernstein. Batch binary Edwards. In *Advances in Cryptology — CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 317–336. Springer Berlin / Heidelberg, 2009.

[25] Daniel J. Bernstein and Tanja Lange. Faster addition and doubling on elliptic curves. In *Advances in Cryptology — ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 29–50. Springer Berlin / Heidelberg, 2008.

[26] Daniel J. Bernstein and Peter Schwabe. NEON crypto. In *Cryptographic Hardware and Embedded Systems — CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 320–339. Springer Berlin / Heidelberg, 2012.

[27] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21:149–177, 2008.

[28] Dan Boneh and Matt Franklin. Identity-based encryption from the Weil pairing. In *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer Berlin / Heidelberg, 2001.

[29] Dan Boneh and Craig Gentry. Aggregate and verifiably encrypted signatures from bilinear maps. In *Proceedings of Eurocrypt 2003*, pages 416–432, Warsaw, Poland, May 2003. Springer-Verlag.

[30] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004.

[31] Billy B. Brumley and Nicola Tuveri. Remote timing attacks are still practical. In *Computer Security — ESORICS 2011*, volume 6879 of *Lecture Notes in Computer Science*, pages 355–371. Springer Berlin / Heidelberg, 2011.

[32] "Bushing", Hector Martin "marcan" Cantero, Segher Boessenkool, and Sven Peter. PS3 epic fail, 2010. `http://events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf`.

[33] CAESAR committee. CAESAR: Competition for authenticated encryption: Security, applicability, and robustness. `http://competitions.cr.yp.to/caesar.html`, 2013.

[34] Certicom Research. SEC 2: Recommended elliptic curve domain parameters version 1.0, 2000. `http://www.secg.org/`.

[35] Qi Chai and Guang Gong. A cryptanalysis of HummingBird-2: The differential sequence analysis. Cryptology ePrint Archive, Report 2012/233, 2012. `http://eprint.iacr.org/`.

[36] Sanjit Chatterjee, Alfred Menezes, and Palash Sarkar. Another look at tightness. In *Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 293–319. Springer Berlin / Heidelberg, 2012.

[37] Danilo Câmara, Conrado P. L. Gouvêa, Julio López, and Ricardo Dahab. Fast software polynomial multiplication on ARM processors using the NEON engine. In *Security Engineering and Intelligence Informatics*, volume 8128 of *Lecture Notes in Computer Science*, pages 137–154. Springer Berlin / Heidelberg, 2013.

[38] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *Cryptography and Coding*, volume 2260 of *Lecture Notes in Computer Science*, pages 360–363. Springer Berlin / Heidelberg, 2001.

[39] Paul G. Comba. Exponentiation cryptosystems on the IBM PC. *IBM Systems Journal*, 29(4):526–538, 1990.

[40] Don Coppersmith. Fast evaluation of logarithms in fields of characteristic two. *IEEE Transactions on Information Theory*, 30(4):587–594, 1984.

[41] Jun-Hong Cui, Jiejun Kong, Mario Gerla, and Shengli Zhou. Challenges: Building scalable mobile underwater wireless sensor networks for aquatic applications. *IEEE Network, Special Issue on Wireless Sensor Networking*, 20(3):12–18, 2006.

[42] Shammi Didla, Aaron Ault, and Saurabh Bagchi. Optimizing AES for embedded devices and wireless sensor networks. In *Proceedings of the 4th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, pages 4:1–4:10, 2008.

[43] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[44] Qi Dong, Donggang Liu, and Peng Ning. Pre-authentication filters: providing dos resistance for signature-based broadcast authentication in sensor networks. In *WiSec'08: 1st ACM Conference on Wireless Network Security*, pages 2–12, New York, NY, USA, March 2008. ACM.

[45] Wenliang Du, Jing Deng, Yunghsiang S. Han, Pramod K. Varshney, Jonathan Katz, and Aram Khalili. A pairwise key pre-distribution scheme for wireless sensor networks. *ACM Transactions on Information and System Security*, 8(2):228–58, 2005.

[46] Karel Dudacek and Vlastimil Vavricka. Experimental evaluation of the MSP430 microcontroller power requirements. In *The International Conference on "Computer as a Tool" — EUROCON, 2007*, pages 400–404, 2007.

[47] Adam Dunkels. Full TCP/IP for 8-bit architectures. In *MobiSys '03: Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 85–98, San Franciso, CA, May 2003. ACM.

[48] Régis Dupont and Andreas Enge. Provably secure non-interactive key distribution based on pairings. *Discrete Applied Mathematics*, 154(2):270–276, 2006.

[49] Daniel Engels, Markku-Juhani O. Saarinen, and Eric M. Smith. The Hummingbird-2 lightweight authenticated encryption algorithm. In *RFID Security and Privacy*, volume 7055 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2011.

[50] Joakim Eriksson, Adam Dunkels, Niclas Finne, Fredrik Österlind, and Thiemo Voigt. MSPsim – an extensible simulator for MSP430-equipped sensor boards. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session*, January 2007.

[51] Laurent Eschenauer and Virgil D. Gligor. A key management scheme for distributed sensor networks. In *9th ACM conf. on Computer and communications security (CCS'02)*, pages 41–47, Washington, DC, November 2002. ACM.

[52] Armando Faz-Hernández, Patrick Longa, and Ana H. Sánchez. Efficient and secure algorithms for GLV-based scalar multiplication and their implementation on GLV-GLS curves. In *RSA Conference Cryptographers' Track (CT-RSA 2014)*, 2014.

[53] Kenny Fong, Darrel Hankerson, Julio López, and Alfred Menezes. Field inversion and point halving revisited. *IEEE Transactions on Computers*, 53(8):1047–1059, 2004.

[54] Laura Fuentes-Castañeda, Edward Knapp, and Francisco Rodríguez-Henríquez. Faster hashing to $\mathbb{G}_2$. In *Selected Areas in Cryptography — SAC 2011*, 2011.

[55] David Galindo, Rodrigo Roman, and Javier Lopez. A killer application for pairings: Authenticated key establishment in underwater wireless sensor networks. In *Cryptology and Network Security*, volume 5339 of *Lecture Notes in Computer Science*, pages 120–132. Springer Berlin / Heidelberg, 2008.

[56] Robert P. Gallant, Robert J. Lambert, and Scott A. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 190–200. Springer Berlin / Heidelberg, 2001.

[57] Brian Gladman. AES and combined encryption/authentication modes. `http://gladman.plushost.co.uk/oldsite/AES/`, 2008.

[58] Faruk Göloğlu, Robert Granger, Gary McGuire, and Jens Zumbrägel. On the function field sieve and the impact of higher splitting probabilities: Application to discrete logarithms in $\mathbb{F}_{2^{1971}}$ and $\mathbb{F}_{2^{3164}}$. Cryptology ePrint Archive, Report 2013/074, 2013. `http://eprint.iacr.org/`.

[59] Conrado P. L. Gouvêa and Julio López. High speed implementation of authenticated encryption for the MSP430X microcontroller. In *LATINCRYPT 2012*, volume 7533 of *Lecture Notes in Computer Science*, pages 288–304. Springer Berlin / Heidelberg, 2012.

[60] Conrado P. L. Gouvêa and Julio López. Software implementation of pairing-based cryptography on sensor networks using the MSP430 microcontroller. In *Progress in Cryptology — INDOCRYPT 2009*, volume 5922 of *Lecture Notes in Computer Science*, pages 248–262. Springer Berlin / Heidelberg, 2009.

[61] Conrado P. L. Gouvêa and Julio López. Implementação em software de criptografia assimétrica para redes de sensores com o microcontrolador MSP430. In *Anais do X Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 419–432, 2010.

[62] Conrado P. L. Gouvêa and Julio López. High speed implementation of authenticated encryption for the MSP430X microcontroller. In *ECRYPT Workshop on Lightweight Cryptography*, 2011.

[63] Conrado P. L. Gouvêa, Leonardo B. Oliveira, and Julio López. Efficient software implementation of public-key cryptography on sensor networks using the MSP430X microcontroller. *Journal of Cryptographic Engineering*, 2(1):19–29, 2012.

[64] Robert Granger and Michael Scott. Faster squaring in the cyclotomic subgroup of sixth degree extensions. In *Public Key Cryptography — PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 209–223. Springer Berlin / Heidelberg, 2010.

[65] Johann Großschädl. TinySA: A security architecture for wireless sensor networks. In *Proceedings of the 2006 ACM CoNEXT conference*, page 55. ACM New York, 2006.

[66] Jorge Guajardo, Rainer Blümel, Uwe Krieger, and Christof Paar. Efficient implementation of elliptic curve cryptosystems on the TI MSP430x33x family of microcontrollers. In *Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 365–382. Springer Berlin / Heidelberg, 2001.

[67] Vipul Gupta, Michael Wurm, Yu Zhu, Matthew Millard, Stephen Fung, Nils Gura, Hans Eberle, and Sheueling Chang Shantz. Sizzle: A standards-based end-to-end security architecture for the embedded internet. *Pervasive Mob. Comput.*, 1(4):425–445, December 2005. Also appeared in PERCOM'05.

[68] Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and Sheueling Chang Shantz. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In *Cryptographic Hardware and Embedded Systems — CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 925–943. Springer Berlin / Heidelberg, 2004.

[69] Mike Hamburg. Fast and compact elliptic-curve cryptography. Cryptology ePrint Archive, Report 2012/309, 2012. `http://eprint.iacr.org/`.

[70] Darrel Hankerson, Alfred Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, 2004.

[71] Florian Hess. Pairing lattices. In *Pairing-Based Cryptography — Pairing 2008*, volume 5209 of *Lecture Notes in Computer Science*, pages 18–38. Springer Berlin / Heidelberg, 2008.

[72] Florian Hess, Nigel Smart, and Frederik Vercauteren. The Eta pairing revisited. *IEEE Transactions on Information Theory*, 52(10):4595–4602, 2006.

[73] Jason L. Hill and David E. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, 2002.

[74] Jonathan W. Hui and David E. Culler. IP is dead, long live IP for wireless sensor networks. In *Proceedings of the 6th international Conference on Embedded Networked Sensor Systems ACM Sensys'08*, Raleigh, North Carolina, USA, November 2008. ACM.

[75] Institute for Applied Information Processing and Communication. Crypto software for microcontrollers - Texas Instruments MSP430 microcontrollers. `http://jce.iaik.tugraz.at/sic/Products/Crypto_Software_for_Microcontrollers/Texas_Instruments_MSP430_Microcontrollers`, 2012.

[76] Tsukasa Ishiguro, Masaaki Shirase, and Tsuyoshi Takagi. Efficient implementation of pairings on sensor nodes. In *Applications of Pairing-Based Cryptography – NIST*, pages 96–106, 2008.

[77] Toshiya Itoh and Shigeo Tsujii. A fast algorithm for computing multiplicative inverses in GF($2^m$) using normal bases. *Information and Computation*, 78(3):171 – 177, 1988.

[78] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. *Journal of Cryptology*, 17(4):263–276, 2004.

[79] Antoine Joux. Discrete logarithms in $GF(2^{6168})$ [$=GF((2^{257})^{24})$]. NMBRTHRY mailing list, 2013. `https://listserv.nodak.edu/cgi-bin/wa.exe?A2=NMBRTHRY;49bb494e.1305`.

[80] Antoine Joux. A new index calculus algorithm with complexity $L(1/4 + o(1))$ in very small characteristic. Cryptology ePrint Archive, Report 2013/095, 2013. `http://eprint.iacr.org/`.

[81] Aman Kansal, Suman Nath, Jie Liu, and Feng Zhao. SenseWeb: An infrastructure for shared sensing. *IEEE MultiMedia*, 14(4):8–13, 2007.

[82] Koray Karabina. Squaring in cyclotomic subgroups. *Mathematics of Computation*, 82(281):555–579, 2013.

[83] Anatolii A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, 7:595, 1963.

[84] Shinsaku Kiyomoto, Matt Henricksen, Wun-She Yap, Yuto Nakano, and Kazuhide Fukushima. Masha — low cost authentication with a new stream cipher. In *Information Security*, volume 7001 of *Lecture Notes in Computer Science*, pages 63–78. Springer Berlin / Heidelberg, 2011.

[85] Miroslav Knežević, Frederik Vercauteren, and Ingrid Verbauwhede. Faster interleaved modular multiplication based on Barrett and Montgomery reduction methods. *IEEE Transactions on Computers*, 59(12):1715–1721, 2010.

[86] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[87] Neal Koblitz. CM-curves with good cryptographic properties. In *11th Annual International Cryptology Conference (CRYPTO 1991)*, volume 576 of *LNCS*, pages 279–287, Santa Barbara, California, August 1991. Springer.

[88] Neal Koblitz and Alfred Menezes. Pairing-based cryptography at high security levels. In *Cryptography and Coding*, volume 3796 of *Lecture Notes in Computer Science*, pages 13–36. Springer Berlin / Heidelberg, 2005.

[89] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology — CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer Berlin / Heidelberg, 1996.

[90] Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In *Fast Software Encryption*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327. Springer Berlin / Heidelberg, 2011.

[91] Emilia Käsper. Fast elliptic curve cryptography in OpenSSL. In *Financial Cryptography and Data Security*, volume 7126 of *Lecture Notes in Computer Science*, pages 27–39. Springer Berlin / Heidelberg, 2012.

[92] Adam Langley. Overclocking SSL, 2010. `http://www.imperialviolet.org/2010/06/25/overclocking-ssl.html`.

[93] Laurie Law, Alfred Menezes, Minghua Qu, Jerry Solinas, and Scott Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28:119–134, 2003.

[94] Eunjeong Lee, Hyang-Sook Lee, and Cheol-Min Park. Efficient and generalized pairing computation on abelian varieties. *IEEE Transactions on Information Theory*, 55(4):1793–1803, 2009.

[95] Hendrik W. Lenstra Jr. Factoring integers with elliptic curves. *The Annals of Mathematics*, 126(3):649–673, 1987.

[96] Stephen Lichtenbaum. Duality theorems for curves over $p$-adic fields. *Inventiones Mathematicae*, 7(2):120–136, 1969.

[97] Chae Lim. A new method for securing elliptic scalar multiplication against side-channel attacks. In *Information Security and Privacy*, volume 3108 of *Lecture Notes in Computer Science*, pages 289–300. Springer Berlin / Heidelberg, 2004.

[98] Chae Hoon Lim and Pil Joong Lee. More flexible exponentiation with precomputation. In *Advances in Cryptology — CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 95–107. Springer Berlin / Heidelberg, 1994.

[99] Shu Yun Lim, Chuan Chin Pu, Hyo Taek Lim, and Hoon Jae Lee. Dragon-MAC: Securing wireless sensor networks with authenticated encryption. Cryptology ePrint Archive, Report 2007/204, 2007. `http://eprint.iacr.org/`.

[100] An Liu and Peng Ning. TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. In *IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks*, pages 245–256, Washington, DC, USA, 2008. IEEE Computer Society.

[101] Donggang Liu, Peng Ning, and Rongfang Li. Establishing pairwise keys in distributed sensor networks. *ACM Trans. on Info. and System Security*, 8(1):41–77, 2005. Also in ACM CCS'03.

[102] Donggang Liu, Peng Ning, Sencun Zhu, and Sushil Jajodia. Practical broadcast authentication in sensor networks. In *MOBIQUITOUS '05: 2nd Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 118–132, Washington, DC, July 2005. IEEE Computer Society.

[103] Julio López and Ricardo Dahab. Fast multiplication on elliptic curves over GF($2^m$) without precomputation. In *Cryptographic Hardware and Embedded Systems*, volume 1717 of *Lecture Notes in Computer Science*, page 724. Springer Berlin / Heidelberg, 1999.

[104] Julio López and Ricardo Dahab. High-speed software multiplication in $\mathbb{F}_{2^m}$. In *Progress in Cryptology — INDOCRYPT 2000*, volume 1977 of *Lecture Notes in Computer Science*, pages 93–102. Springer Berlin / Heidelberg, 2000.

[105] Mark Luk, Adrian Perrig, and Bram Whillock. Seven cardinal properties of sensor network broadcast authentication. In *SASN '06: Proceedings of the fourth ACM workshop on Security of ad hoc and sensor networks*, pages 147–156, New York, NY, USA, October 2006. ACM.

[106] Dimitrios Lymberopoulos, Nissanka B. Priyantha, Michel Goraczko, and Feng Zhao. Towards energy efficient design of multi-radio platforms for wireless sensor networks. In *IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks*, pages 257–268, Washington, DC, USA, April 2008. IEEE Computer Society.

[107] Dimitrios Lymberopoulos, Nissanka B. Priyantha, and Feng Zhao. mPlatform: a reconfigurable architecture and efficient data sharing mechanism for modular sensor nodes. In *IPSN '07: 6th international conference on Information processing in sensor networks*, pages 128–137, New York, NY, USA, 2007. ACM.

[108] David J. Malan, Matt Welsh, and Michael D. Smith. Implementing public-key infrastructure for sensor networks. *ACM Transactions on Sensor Networks*, 4(4):22:1–22:23, September 2008. Also in SECON'04.

[109] David McGrew and John Viega. The security and performance of the Galois/Counter Mode (GCM) of operation. In *Progress in Cryptology — INDOCRYPT 2004*, volume 3348 of *Lecture Notes in Computer Science*, pages 377–413. Springer Berlin / Heidelberg, 2005.

[110] Alfred J. Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, 1993.

[111] Victor S. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology — CRYPTO'85 Proceedings*, pages 417–426, 1986.

[112] Victor S. Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 17:235–261, 2004.

[113] Atsuko Miyaji, Masaki Nakabayashi, and Shunzou Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 84(5):1234–1243, 2001.

[114] Bodo Möller. Algorithms for multi-exponentiation. In *Selected Areas in Cryptography*, volume 2259 of *Lecture Notes in Computer Science*, pages 165–180. Springer Berlin / Heidelberg, 2001.

[115] Niels Möller. Nettle, low-level cryptographics library. Nettle Git repository, 2013. `http://git.lysator.liu.se/nettle/nettle/blobs/9422a55130ba65f73a053f063efa6226f945b4f1/sec-modinv.c#line67`.

[116] P. Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation*, 48:243–264, 1987.

[117] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[118] Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264, 1987.

[119] Sergey Morozov, Christian Tergino, and Patrick Schaumont. System integration of elliptic curve cryptography on an OMAP platform. In *2011 IEEE 9th Symposium on Application Specific Processors (SASP)*, pages 52–57. IEEE, 2011.

[120] National Institute of Standards and Technology. Recommendation for key management, March 2007. `http://www.itl.nist.gov`.

[121] National Institute of Standards and Technology. FIPS 186-4: Digital signature standard (DSS), July 2013. `http://www.itl.nist.gov`.

[122] Peng Ning, An Liu, and Wenliang Du. Mitigating DoS attacks against broadcast authentication in wireless sensor networks. *ACM Transactions on Sensor Networks*, 4(1):1–35, 2008.

[123] Yasuyuki Nogami, Masataka Akane, Yumi Sakemi, Hidehiro Kato, and Yoshitaka Morikawa. Integer variable $\chi$-based Ate pairing. In *Pairing-Based Cryptography — Pairing 2008*, volume 5209 of *Lecture Notes in Computer Science*, pages 178–191. Springer Berlin / Heidelberg, 2008.

[124] Leonardo B. Oliveira, Diego F. Aranha, Conrado P. L. Gouvêa, Michael Scott, Danilo F. Câmara, Julio López, and Ricardo Dahab. TinyPBC: Pairings for authenticated identity-based non-interactive key distribution in sensor networks. *Computer Communications*, 34(3):485–493, 2011.

[125] Leonardo B. Oliveira, Aman Kansal, Conrado P. L. Gouvêa, Diego F. Aranha, Julio López, Bodhi Priyantha, Michel Goraczko, and Feng Zhao. Secure-TWS: Authenticating node to multi-user communication in shared sensor networks. *The Computer Journal*, 55(4):384–396, 2012.

[126] Leonardo B. Oliveira, Hao Chi Wong, Marshall Bern, Ricardo Dahab, and Antonio A. F. Loureiro. SecLEACH – a random key distribution solution for securing clustered sensor networks. In *5th IEEE International Symposium on Network Computing and Applications (NCA'06)*, July 2006. p. 145-154.

[127] Colin Percival. Cache missing for fun and profit. `http://www.daemonology.net/papers/cachemissing.pdf`, 2005.

[128] Geovandro Pereira, Marcos A. Simplício, Jr., Michael Naehrig, and Paulo S. L. M. Barreto. A family of implementation-friendly BN elliptic curves. *Journal of Systems and Software*, 84:1319–1326, August 2011.

[129] Adrian Perrig, Robert Szewczyk, J. D. Tygar, Victor Wen, and David E. Culler. SPINS: Security protocols for sensor networks. *Wireless Networks*, 8(5):521–534, 2002. Also in MobiCom'01.

[130] John M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, 1978.

[131] Andy Polyakov. The OpenSSL project. OpenSSL Git repository, 2013. `http://git.openssl.org/gitweb/?p=openssl.git;a=blob;f=crypto/modes/asm/ghash-armv4.pl;h=d91586ee2925bb695899b17bb8a7242aa3bf9150;hb=9575d1a91ad9dd6eb5c964365dfbb72dbd3d1333#l35`.

[132] Nissanka Bodhi Priyantha, Aman Kansal, Michel Goraczko, and Feng Zhao. Tiny Web Services: Design and implementation of interoperable and evolvable sensor networks. In *Proceedings of 6th ACM Conference on Embedded Networked Sensor Systems (Sensys'08)*, pages 253–266, Raleigh, NC, November 2008.

[133] Kui Ren, Wenjing Lou, and Yanchao Zhang. Multi-user broadcast authentication in wireless sensor networks. In *SECON'07 4th Sensor, Mesh and Ad Hoc Communications and Networks*, pages 223–232, Kiev, Ukraine, July 2007. IEEE.

[134] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[135] Juliano Rizzo and Thai Duong. Practical padding oracle attacks. In *Proceedings of the 4th USENIX conference on Offensive technologies, WOOT*, volume 10, pages 1–8, 2010.

[136] Markku-Juhani O. Saarinen. SGCM: The Sophie Germain counter mode. Cryptology ePrint Archive, Report 2011/326, 2011. `http://eprint.iacr.org/`.

[137] Markku-Juhani O. Saarinen. Cycling attacks on GCM, GHASH and other polynomial MACs and hashes. In *Fast Software Encryption*, volume 7549 of *Lecture Notes in Computer Science*, pages 216–225. Springer Berlin / Heidelberg, 2012.

[138] Markku-Juhani O. Saarinen. Related-key attacks against full Hummingbird-2. In *Fast Software Encryption*. Springer Berlin / Heidelberg, 2013. `http://eprint.iacr.org/`.

[139] Ryuichi Sakai, K. Ohgishi, and Masao Kasahara. Cryptosystems based on pairing. In *The 2000 Symposium on Cryptography and Information Security, Okinawa, Japan*, 2000.

[140] Sam Schillace. Default https access for Gmail, 2010. `http://gmailblog.blogspot.com.br/2010/01/default-https-access-for-gmail.html`.

[141] Claus P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[142] Michael Scott, Naomi Benger, Manuel Charlemagne, Luis J. Dominguez Perez, and Ezekiel J. Kachisa. On the final exponentiation for calculating pairings on ordinary elliptic curves. In *Pairing-Based Cryptography — Pairing 2009*, volume 5671 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2009.

[143] Michael Scott and Piotr Szczechowiak. Optimizing multiprecision multiplication for public key cryptography. Cryptology ePrint Archive, Report 2007/299, 2007. `http://eprint.iacr.org/`.

[144] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer Berlin / Heidelberg, 1985.

[145] Marcos A. Simplicio Jr., Pedro D'Aquino F. F. S. Barbuda, Paulo S. L. M. Barreto, Tereza C. M. B. Carvalho, and Cintia B. Margi. The MARVIN message authentication code and the LETTERSOUP authenticated encryption scheme. *Security and Communication Networks*, 2(2):165–180, 2009.

[146] Marcos A. Simplicio Jr., Bruno T. de Oliveira, Paulo S. L. M. Barreto, Cintia B. Margi, Tereza C. M. B. Carvalho, and Mats Naslund. Comparison of authenticated-encryption schemes in wireless sensor networks. In *2011 IEEE 36th Conference on Local Computer Networks (LCN)*, pages 450–457, 2011.

[147] Jerome A. Solinas. Efficient arithmetic on Koblitz curves. *Designs, Codes and Cryptography*, 19(2):195–249, 2000.

[148] Standards for Efficient Cryptography Group. Sec 2: Recommended elliptic curve domain parameters. SECG2, 2000.

[149] Piotr Szczechowiak, Anton Kargl, Michael Scott, and Martin Collier. On the application of pairing based cryptography to wireless sensor networks. In *Proceedings of the second ACM conference on Wireless network security*, pages 1–12. ACM New York, 2009.

[150] Piotr Szczechowiak, Leonardo B. Oliveira, Michael Scott, Martin Collier, and Ricardo Dahab. NanoECC: Testing the limits of elliptic curve cryptography in sensor networks. In *Wireless Sensor Networks*, volume 4913 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2008.

[151] Piotr Szczechowiak, Leonardo B. Oliveira, Michael Scott, Martin Collier, and Ricardo Dahab. NanoECC: Testing the limits of elliptic curve cryptography in sensor networks. In *European conference on Wireless Sensor Networks (EWSN'08)*, volume 4913, pages 305–320, Bologne/Italy, February 2008. Springer-Verlag.

[152] R. Tahir, M.Y. Javed, and A.R. Cheema. Rabbit-MAC: Lightweight authenticated encryption in wireless sensor networks. In *Information and Automation, 2008. ICIA 2008. International Conference on*, pages 573–577, 2008.

[153] John Tate. WC-groups over p-adic fields. *Séminaire Bourbaki*, 1556, 1957.

[154] Frederik Vercauteren. Optimal pairings. *IEEE Transactions on Information Theory*, 56(1):455–461, 2010.

[155] Damian Weber and Thomas Denny. The solution of McCurley's discrete log challenge. In *Advances in Cryptology — CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 458–471. Springer Berlin / Heidelberg, 1998.

[156] André Weil. On the Riemann hypothesis in function-fields. In *Proceedings of the National Academy of Sciences*, volume 27, pages 345–347, 1941.

[157] Doug Whiting, Russ Housley, and Niels Ferguson. Counter with CBC-MAC (CCM), 2002. `http://csrc.nist.gov/groups/ST/toolkit/BCM/index.html`.

[158] Fangguo Zhang, Reihaneh Safavi-Naini, and Willy Susilo. An efficient signature scheme from bilinear pairings and its applications. In *Public Key Cryptography — PKC 2004*, volume 2947 of *Lecture Notes in Computer Science*, pages 277–290. Springer Berlin / Heidelberg, 2004.

[159] Kai Zhang, Lin Ding, and Jie Guan. Cryptanalysis of Hummingbird-2. Cryptology ePrint Archive, Report 2012/207, 2012. `http://eprint.iacr.org/`.

[160] S. Zhu, S. Xu, S. Setia, and S. Jajodia. Establishing pairwise keys for secure communication in ad hoc networks: A probabilistic approach. In *11th IEEE Inter'l Conference on Network Protocols (ICNP'03)*, pages 326–335, Atlanta, Nov 2003. IEEE.

[161] Sencun Zhu, Sanjeev Setia, and Sushil Jajodia. LEAP: efficient security mechanisms for large-scale distributed sensor networks. In *10th ACM conference on Computer and communication security (CCS'03)*, pages 62–72, New York, NY, USA, October 2003. ACM Press.